Міжнародні олімпіади з інформатики

А. М. Гуржій В. В. Бондаренко О. В. Співаковський
Ш. І. Ягіяєв

А.М. Гуржій, В.В. Бондаренко, О.В. Співаковський, Ш.І. Ягіяєв.
Міжнародні олімпіади з інформатики - Херсон: Айлант. - 2005. -233 с.: іл.

Рецензенти:
М.І. Жалдак - завідувач кафедрою інформаційних технологій НПУ ім. М.П. Драгоманова, доктор педагогічних наук, професор, академік АПН України
Н.В. Морзе - проректор з наукової роботи Академії праці і соціальних відносин федерації профспілок України, доктор педагогічних наук, професор

Рекомендовано Міністерством освіти і науки України (Лист Міністерства освіти і науки України №5/1946 від 05.05.2005 р.)

У посібнику описані правила проведення олімпіад з інформатики, організаційні питання проведення змагань, механізми та принципи відбору задач. Розглянуто і проаналізовано завдання Міжнародних олімпіад з інформатики різних років, приведено рекомендації до розв'язування олімпіадних задач та їх рішення. Посібник орієнтований на учнів старших класів загальноосвітніх навчальних закладів, студентів, та вчителів з інформатики.

Зміст

Однією з найважливіших особливостей XXI століття є перехід розвинутих країн світу від постіндустріального до інформаційного суспільства. Інформатика є однією з основних галузей сучасної науки. Розвиток обчислювальної техніки та новітніх інформаційних технологій визначатиме рівень розвитку сучасної науки та суспільства вцілому.

Вирішального значення для економічної конкурентноздатності України, як держави, забезпечення її інтелектуальної самостійності й власного місця в сучасному світі набувають наукові й технічні знання, високі моральні якості особистості, її інтелектуальний та творчий потенціал.

Міркуючи про шляхи підвищення ефективності навчально-виховного процесу з інформатики, не можна обійти стороною роль олімпіад у вирішенні цієї проблеми.

Ідея проведення Міжнародних олімпіад з інформатики була висловлена на 24-й конференції ЮНЕСКО болгарським делегатом професором Сендовим у жовтні 1987 року в Парижі. У травні 1989 року ЮНЕСКО організувала і провела першу таку олімпіаду в болгарському місті Правець. Уже тоді в ній взяли участь представники 13 країн.

Кожного року збірна команда України, яка формується з урахуванням результатів Всеукраїнської олімпіади з інформатики та за результатами відбіркових зборів, приймає участь у Міжнародних олімпіадах з інформатики та займає достойне місце серед найсильніших команд світу.

Високих результатів, за словами наукового керівника команди, академіка АПН України А.М.Гуржія, команда України досягла у 2003 році на XV Міжнародній олімпіаді з інформатики, яка проходила у м. Кеноша (США). Богдан Яковенко виборов для України першу золоту медаль, Олександр Галкін та Юрій Знов'як отримали бронзові медалі.

На XVI Міжнародній олімпіаді 2004 року, що проходила у Греції (м. Афіни), Україну представляли Сергій Гончаренко, Андрій Гриненко, Юрій Знов'як, Дмитро Кордубан, які отримали срібну та дві бронзові медалі.

І знов впевнена перемога на XVII Міжнародній олімпіаді 2005 ро-

ку у Польщі, де Юрій Знов'як та Андрій Гриненко вибороли золоті медалі.

Предметна олімпіада з програмування проводиться з метою пропаганди сучасних методів і прийомів програмування, підвищення до них інтересу в учнів та студентів, а також виявлення і заохочення учнів та студентів, що домоглися кращих результатів у цій області. Вона сприяє розвиткові навичок самостійної роботи, творчого підходу до розв'язання поставлених задач, нетривіальності мислення, виховує в учасників прагнення до досягнення високих результатів.

Тільки шляхом розв'язання складних нетривіальних задач олімпіадного типу можна реалізувати світоглядну функцію інформатики, пов'язану з розкриттям ролі інформаційних процесів у природі, техніці, суспільстві, значенням нових інформаційних технологій для розвитку продуктивних сил суспільства, зміни характеру праці людини.

Наша книга адресована учням, вчителям, викладачам вищих навчальних закладів і може бути покладена в основу спеціальних курсів, використана у гуртковій та факультативній роботі, а також при самостійній підготовці учнів та студентів до олімпіад з інформатики.

Збірник містить задачі Міжнародних олімпіад з інформатики з 1989 по 2005 рік. До кожної задачі наведені вказівки до розв'язку.

Результати участі у Міжнародних олімпіадах з інформатики збірної команди України:

Олімпіада	Рік	Країна проведення	Медалі збірної України
I	1989	Болгарія	1 золота; 1срібна
II	1990	СРСР	2 золоті; 1 бронзова
III	1991	Греція	3 срібні
IV	1992	Німеччина	1 срібна; 3 бронзові
V	1993	Аргентина	
VI	1994	Швеція	1 срібна; 1 бронзова
VII	1995	Нідерланди	2 бронзові
VIII	1996	Угорщина	1 срібна; 1 бронзова
IX	1997	ПАР	2 бронзові
X	1998	Португалія	1 срібна; 2 бронзові
XI	1999	Турція	2 бронзові
XII	2000	Китай	2 срібні; 1 бронзова
XIII	2001	Фінляндія	4 бронзові
XIV	2002	Південна Корея	1 срібна; 2 бронзові
XV	2003	США	1 золота; 1 бронзова
XVI	2004	Греція	1 срібна; 2 бронзові
XVII	2005	Польща	2 золоті; 1 срібна; 1 бронзова

Частина I

Завдання міжнародних олімпіад

Розділ 1. Болгарія'1989

1.1 Завдання олімпіади

1.1.1 «AB»

Дана послідовність з $2N$ комірок ($N \leq 5$). Дві суміжні з них — порожні, а інші містять $N-1$ символів `A` та $N-1$ символів `B`. Приклад для $N = 5$:

A	B	B	A			A	B	A	B

Правило переміщення Вміст будь-яких двох сусідніх непорожніх комірок, зберігаючи їх порядок, може бути переміщено в порожні комірки.
Мета Використовуючи правило переміщення, досягнути конфігурації, в якій всі символи `A` розміщенні лівіше всіх символів `B`. Місцезнаходження порожніх комірок після переміщення не має значення.
Завдання Написати програму, яка:

1. Зчитує з клавіатури початкову конфігурацію у вигляді послідовності символів `A`, `B` та нулів для порожніх комірок, та моделює переміщення.
2. Для заданої початкової конфігурації визначає один з можливих планів переміщення, за допомогою якого можна досягнути мети, чи повідомляє, що такого плану не існує. План переміщення повинен містити початкову конфігурацію, проміжні конфігурації після кожного кроку, а також заключну конфігурацію.
3. Знаходить план досягнення мети за мінімальне число кроків.

Результати Навести по меншій мірі хоча б один розв'язок для прикладу, наведеного вище.

Розділ 2. CPCP'1990

2.1 Завдання першого туру

2.1.1 «Гра 14»

Задана таблиця розміром 4×4, в кожній клітині якої, крім двох, зберігається одне з чисел від 1 до 14 (всі числа різні). Ті дві клітини, що залишилися — порожні. (Приклад — таблиця 1.)

7	3	5	14
	4	9	13
1		2	10
11	8	12	6

Таблиця 1

1	2	3	4
5	6	7	8
9	10	11	12
13	14		

Таблиця 2

Правила переміщення Число з будь-якої клітини може бути переміщено по горизонталі чи по вертикалі в будь-яку порожню сусідню клітину. Клітина, в якій раніше містилося число, стає порожнею.
Мета Необхідно за допомогою вказаного правила виконати по крокам перетворення довільної вихідної таблиці в кінцеву таблицю 2.
Завдання Написати програму, яка:

1. Виконує введення з клавіатури вихідної таблиці і вивід її на екран (пусті клітини можуть бути закодовані нулями).
2. Виконує перетворення введеної таблиці в таблицю 2;
3. На кожному кроці виводить на екран зліва варіант таблиці до цього кроку, справа — варіант таблиці після кроку і вказує номер кроку (1, 2, 3 і т. д.) так, що в кінці роботи програми буде показано загальна кількість зроблених кроків.
4. Мінімізувати число кроків, необхідних для розв'язання задачі.

2.2 Завдання другого туру

2.2.1 «Картинна галерея»

В картинній галереї кожний сторож працює на протязі деякого неперервного відрізка часу. *Розкладом охорони* називається множина пар $[T_1(i), T_2(i)]$ — моментів початку і кінця чергування i-го сторожа з інтервалу $[0; EndTime]$. Для заданого розкладу охорони потрібно:

1. Перевірити, чи в будь-який момент часу в галереї знаходиться не менше двох сторожів. Якщо умова 1 не виконується, то:
2. Перелічити усі інтервали часу з недостатньою охороною (менше двох сторожів);
3. Додати найменше число сторожів з заданою, однаковою для всіх довжиною чергування, щоб отримати правильний розклад (що задовільняє умову 1);
4. Перевірити, чи можна уникнути додавання нових сторожів, якщо дозволяється переміщувати час чергування кожного сторожа з збереженням часу його чергування;
5. При задовільній відповіді на пункт 4 скласти розклад з найменшим числом переміщень.

Вхідні дані Моменти часу задаються в цілих хвилинах. $EndTime$ — момент закінчення роботи охорони (момент початку — 0); N — число сторожів; $T_1(i)$, $T_2(i)$, $i = 1, \ldots, N$ — моменти початку і кінця чергування i-го сторожа. $Length$ — довжина чергування кожного додаткового сторожа.

Вихідні дані:

- відповідь на пункт 1 в форматі «так/ні»;
- при відповіді «ні» на пункт 1 — список пар (k, l) — початків і кінців всіх інтервалів, що мало охороняються, з вказанням числа сторожів у кожному (0 чи 1);
- число додаткових сторожів і моменти початку і кінця чергування кожного додаткового сторожа;
- відповідь на пункт 4 в формі «так/ні»; якщо «так», то номера сторожів, зміна яких зміщується, і значення зміщень;
- відповідь на пункт 5 — найменше число сторожів, зміна яких зміщується, їх номера і значення зміщень.

Примітка Програма повинна допускати незалежне тестування пунктів 3, 4, 5.

Розділ 3. Греція'1991

3.1 Завдання першого туру

3.1.1 «Матриця 5×5»

Пронумерувати позиції в матриці розміром 5×5 наступним чином. Якщо номер i $(1 \leq i \leq 25)$ відповідає в матриці позиції з координатами (x, y), то номер $i+1$ може відповідати позиції з координатами (z, w), що обчислюється за одним з наступних правил:

(1) $(z, w) = (x \pm 3, y)$;

(2) $(z, w) = (x, y \pm 3)$;

(3) $(z, w) = (x \pm 2, y \pm 2)$.

Завдання

(A) Написати програму, яка послідовно нумерує позиції матриці 5×5 при заданих координатах позицій, з номером 1 (результати повинні бути виведені у вигляді заповненої матриці).

(B) Обчислити число всіх можливих розташуань номерів для всіх початкових позицій, що знаходяться в правому верхньому трикутнику матриці, включаючи її головну діагональ.

Приклад Якщо у якості початкової позиції в матриці вибрана позиція з координатами $(2, 2)$, то на наступному кроці координати позиції з номером 2 у відповідності з наведеними правилами можуть бути $(2, 5)$, $(5, 2)$ або $(4, 4)$.

```
    1   2   3   4   5
  +---+---+---+---+---+
1 :   :   :   :   :   :
  +---+---+---+---+---+
2 :   : 1 :   :   : * :
  +---+---+---+---+---+
3 :   :   :   :   :   :
  +---+---+---+---+---+
4 :   :   :   : * :   :
  +---+---+---+---+---+
5 :   : * :   :   :   :
  +---+---+---+---+---+
```

Примітка Буде оцінюватися чи схожий ваш вивід на наведений приклад.

3.2 Завдання другого туру

3.2.1 «S-терми»

S-терм — це послідовність символів `S` і дужок, що визначається рекурсивно наступним чином:

- символ `S` є S-термом;
- якщо M і N — S-терми, то вираз (MN) є також S-термом.

Приклад S-терма: `((((SS)(SS))S)(SS))`. Праві дужки не несуть інформації і можуть опускатись. В цьому випадку наведений вище S-терм виглядатиме так: `((((SS(SSS(SS`.

Завдання

1. Напишіть процедуру `gensterm` для породження S-термів. Вона повинна для заданого n заповнювати n текстових файлів (n — довжина S-терма, що дорівнює числу символів `S`), кожний з яких містить усі S-терми довжини $n = 1, 2, \ldots, n$ відповідно. Всередині файлу S-терми відокремлюються символом `;` (крапка з комою). В кінці кожного файлу повинен знаходитися символ `.` (крапка).

   Напишіть програму, яка за заданим цілим n ($n \leq 10$) виконує описану вище процедуру і видає на дисплей всі згенеровані S-терми.

Розглянемо обчислення S-термів. Єдине алгебраїчне правило (*S-правило*), яке може бути використано, полягає в наступному: будь-який підтерм S-терма, який має вигляд $(((\mathtt{S}A)B)C)$, де A, B, C — S-терми, може бути переписаний як $((AC)(BC))$, тобто

$$Context_1(((\mathtt{S}A)B)C)Context_2 \to Context_1((AC)(BC))Context_2$$

Застосування цього правила до S-терму називається *редукцією* S-терма. Можливі різні способи (стратегії) вибору підтермів для використання S-правила. Послідовне застосування S-правила до S-терму до тих пір, доки це можливо, називається *нормалізацією*. Приклад ланцюга редукції S-терма:

$$\mathtt{((((SS)(SS))S)(SS))} \to \mathtt{(((SS)((SS)((SS)S))(SS))} \to$$
$$\mathtt{((S(SS))(((SS)S)(SS)))} \to \mathtt{((S(SS))((S(SS))(S(SS))))}$$

2. Запропонуйте ефективну структуру даних для представлення *S*-термів, що полегшує використання S-правила. Напишіть дві процедури: readterm и printterm. Перша з них перетворює S-терми в вашу структуру даних з форми, що породжується процедурою gensterm; друга перетворює S-терми з вашої структури у форму, що породжується процедурою gensterm. Ваша програма повинна демонструвати ці перетворення.

3. Напишіть процедуру reduce, що виконує один шаг редукції у відповідності з S-правилом над заданим підтермом S-терма в вашому представленні. Програма повина демонструвати це.

4. Напишіть процедуру normalize, яка в заданому S-термі повинна послідовно вибирати підтерми і застосовувати S-правило до тих пір, поки подальші редукції стануть неможливими, або число шагів не досягне деякого максимума, наприклад 30. Ваша програма повинна демонструвати це.

5. Об'єднайте всі процедури в одну програму, яка:

   a) запитує у користувача довжину n,

   b) породжує за допомогою процедури gensterm S-терми заданої довжини,

   c) перетворює всі S-терми у ваше представлення,

   d) нормалізує їх (якщо це можливо),

   e) виводить в якості результату нормалізовані S-терми,

f) виводить послідовно число шагів редукції, здійснених над кожним S-термом, або повідомлення `not normalized`, якщо нормалізація потребує більше 30 кроків,

g) Виводить число ненормалізованих термів і загальне число всіх S-термів заданої довжини n.

Розділ 4. Німеччина'1992

4.1 Завдання першого туру

4.1.1 «Острова у морі»

Море представлене матрицею $N \times N$. Кожний острів — це символ `*` (зірочка) в матриці. Задача полягає в побудові карти островів тільки по кодованій інформації про горизонтальний та вертикальний розподіл островів.

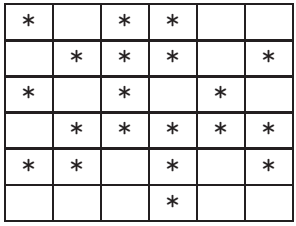
Числа праворуч від кожного рядка показують порядок та розміри груп островів в цих рядках. Наприклад, `1 2` в першому рядку означає, що цей рядок містить групу з одного острова, за якою йде група з двох островів з морем довільної довжини ліворуч та праворуч від кожної групи островів. Подібним чином послідовність `1 1 1` під першим стовпчиком означає, що цей стовпчик містить три групи з одним островом в кожній і т.д.

Завдання Реалізувати програму, яка повторює такі кроки, поки заданий вхідний файл, що містить задані блоки інформації, не буде прочитано повністю.

1. Читати наступний блок інформації з вхідного `ASCII`-файлу (структуру даних в цьому файлі дивитесь далі в прикладах) та вивести його на екран. Кожен блок інформації складається з розміру квадратної матриці, за яким ідуть обмеження по рядках та обмеження по стовпчиках. Кожне обмеження для одного рядка або стовпчика міститься в окремому рядку файла як послідовність чисел, розділених пропусками, що закінчується нулем.

2. Побудувати карту (або всі карти, якщо можлива не єдина відповідь, див. приклад 4) та вивести її/їх на екран.

3. Записати карту(и) в кінець вихідного `ASCII`-файлу. Кожний острів повинен бути відображеним зірочкою, за якою йде пропуск. Кожне порожнє місце повинно бути відображене двома пропусками. Різні карти, що задовольняють однаковим умовам, повинні бути розділені



Мал. 4.1: Кодування карти

порожнім рядком. Якщо не існує карти, що задовольняє умові, вивести рядок з текстом `no map` (немає карт). Розв'язки для різних блоків інформації треба розділити рядком з текстом `next problem` (наступна задача).

Технічні обмеження N не може бути менше 1 та більше 8.

Приклади

Дані
6 1 2 0 3 1 0 1 1 1 0 5 0 2 1 1 0 1 0 1 1 1 0 1 2 0 4 0 2 3 0 2 0 1 2 0

Дані	Розв'язок
2 0 0 2 0 2 0	Не існує карти яка б задовольняла цим обмеженням.

Дані
4 0 1 0 2 0 0 0 1 0 2 0 0

Розв'язок
1 2 3 4 1: 2: * 3: * * 4:

Дані
2 1 0 1 0 1 0 1 0

Розв'язок
Існує дві різні карти, що задовольняють наведеним обмеженням.

4.2 Завдання другого туру

4.2.1 «Альпіністи»

Клуб альпіністів має P членів з номерами від 1 до P. Всі альпіністи сходять на гору з однаковими швидкостями; швидкість підйому та спуску не відрізняються. i-й альпініст витрачає $c(i)$ одиниць ресурсів на день та може нести не більше ніж $s(i)$ таких одиниць. Всі $c(i)$ та $s(i)$ — цілі числа.

Припустимо, що альпініст з достатньою кількістю ресурсів досягає вершини за N днів. Гора може бути такою високою, що жоден альпініст не зможе нести всі ресурси, необхідні для сходження та спуску. Таким чином, група альпіністів стартує одночасно в одному місці. Альпініст, який спускається не досягнувши вершини, повинен віддати іншим скалолазам решту своїх ресурсів, що не знадобляться йому для спуску; вони повинні мати змогу взяти ці ресурси.

Задача полягає в складанні розкладу для клубу альпіністів. Хоча б один альпініст повинен досягти вершини. Всі альпіністи повинні повернутися до пункта старту.

Завдання Реалізувати програму, яка повинна:

1. Читати з клавіатури ціле число днів N, необхідне для досягнення вершини, кількість P альпіністів у клубі, та числа $s(i)$ і $c(i)$ для всіх i від 1 до P. Можна вважати вхідні дані цілими. Якщо дані не мають змісту, їх слід проігнорувати та повторити запит.

2. Спробувати відшукати розклад сходження на гору. Визначити можливу групу $a(1), \ldots, a(k)$ альпіністів, що мають взяти участь в підйомі, та кількість ресурсів $M(j)$ для кожного $j = 1, \ldots, k$, які альпіністи $a(j)$ несуть від старту. Повідомити, якщо для комбінації N, $s(j)$, $c(j)$ розклад не існує.

3. Вивести таку інформацію на екран:

   (a) K — кількість альпіністів, які приймуть участь у сходженні,

   (b) загальну кількість необхідних ресурсів,

   (c) номери альпіністів $a(1), \ldots, a(k)$,

   (d) для всіх $a(j)$, $j = 1, \ldots, k$ — початкові кількості $M(j)$ ресурсів, які альпіністи $a(j)$ несуть від старту,

   (e) день $D(j)$, коли альпініст $a(j)$ почне спускатися, для кожного $j = 1, \ldots, k$.

4. Розклад є оптимальним, якщо:

(a) кількість альпіністів, що беруть участь у сходженні, мінімальна, та

(b) серед всіх розкладів для груп, що задовільняють умові мінімальності кількості альпіністів 4a, загальні витрати ресурсів мінімальні.

Спробуйте знайти відповідь близьку до оптимальної.

Технічні обмеження Програма повинна повторити запит даних в разі, якщо $N < 1$ або $N > 100$. P має бути не менше ніж 1 та не більше ніж 20. Недодатні добові витрати та запаси ресурсів не мають змісту, це також повинно перевірятися.

Приклади Діалог програми повинен мати вигляд як на мал. 4.2.

```
Дні для досягнення вершини: 4
Кількість членів клубу: 5
Максимальний ресурс для альпініста 1: 7
Добові витрати для альпініста 1: 1
Максимальний ресурс для альпініста 2: 8
Добові витрати для альпініста 2: 2
Максимальний ресурс для альпініста 3: 12
Добові витрати для альпініста 3: 2
Максимальний ресурс для альпініста 4: 15
Добові витрати для альпініста 4: 3
Максимальний ресурс для альпініста 5: 7
Добові витрати для альпініста 5: 1
<результати>
Потрібно 2 альпіністи, загальна кількість ресурсів дорівнює 10.
Підуть альпіністи 1,5
Альпініст 1 візьме 7 одиниць ресурсу та піде на спуск через 4 днів
Альпініст 5 візьме 3 одиниць ресурсу та піде на спуск через 1 днів
Спланувати інші клуби(т/н)? т
Дні для досягнення вершини: 2
Кількість членів клубу: 1
Максимальний ресурс для альпініста 1: 3
Добові витрати для альпініста 1: 1
Сходження неможливе
Спланувати інші клуби(т/н)? н
До побачення!
```

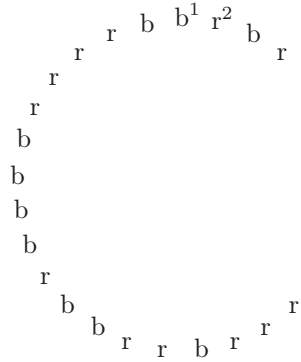
Мал. 4.2: Приклад діалогу задачі 4.2.1

Розділ 5. Аргентина'1993

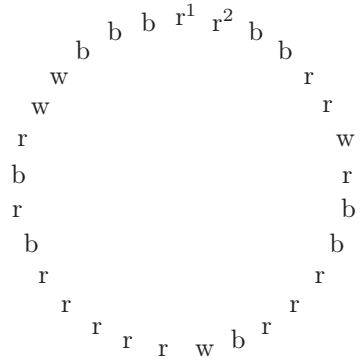
5.1 Завдання першого туру

5.1.1 «Буси»

Розглянемо буси, що складаються з n бусинок ($n \le 100$) деякі з яких червоні, деякі — сині, а інші — білі. Бусинки різних кольорів розташовані довільним чином. На малюнках наведено приклад для $n = 29$:



Мал. 5.1.1



Мал. 5.1.1

`r` — червона бусинка
`b` — синя бусинка
`w` — біла бусинка

На малюнках цифрами відмічено бусинки, що вважаються першими та другими у своїх ланцюжках.

Конфігурація бус на рис. 1 може бути представлена як рядок, що складається з символів `b` та `r` наступним чином:

```
brbrrrbbbrrrrrbrrbbrbbbbbrrrrb
```

Припустимо що необхідно розірвати буси, та знімати бусинки одного кольору з першого з кінців доти, доки не зустрінеться бусинка іншого кольору. Теж саме треба проробити і з іншим кінцем (бусинки, зняті з різних кінців можуть бути різного кольору). Необхідно знайти таку точку розриву

бус, щоб можна було зняти максимальну можливу кількість бусинок з обох кінців.

Наприклад, для бус зображених на мал. 1, 8 бусинок може бути знято, якщо точка розриву буде між бусинками 9 та 10, або між 24 та 25.

В деяких бусах, були додані білі бусинки (як це зображено на рис. 2). При стягуванні бусинок, білі бусинки, що зустрілися, можуть при необхідності прийматися за червоні або за сині. Рядок, який представляє таку конфігурацію буде містити символи `r`, `b` та `w`.

Завдання Напишіть програму, яка:

1. Зчитує конфігурацію з вхідного `ASCII`-файлу `NECKLACE.DAT`, в кожному рядку якого задано конфігурацію бус. Записує цю інформацію у вихідний `ASCII`-файл `NECKLACE.SOL`.

2. Для кожної конфігурації бус визначає максимальну кількість M знятих бусинок, разом з точкою розриву.

3. Записує у вихідний файл `NECKLACE.SOL`, кількість M та точку розриву. Розв'язки для різних конфігурацій повинні бути відокремлені порожнім рядком.

Приклад вхідних та вихідних даних

NECKLACE.DAT
brbrrrbbbrrrrrbrrbbrbbbbrrrrb bbwbrrrwbrbrrrrrb

NECKLACE.SOL
brbrrrbbbrrrrrbrrbbrbbbbrrrrb 8 between 9 and 10 bbwbrrrwbrbrrrrrb 10 between 16 and 17

5.1.2 «Компанії»

Деякі компанії є співвласниками інших компаній, якщо вони придбали частину акцій цих компаній. Наприклад, компанія «Форд» володіє 12% компанії «Мазди». Вважають, що компанія A *контролює* компанію B якщо виконується хоча б одна з наступних вимог:

a) $A = B$

b) A володіє $> 50\%$ акцій компанії B

c) A контролює k ($k > 1$) компаній, $C(1), \ldots, C(k)$, так що: $C(i)$ володіє $x(i)\%$ компанії B для $1 \le i \le k$ та $x(1) + \ldots + x(k) > 50\%$.

За даним набором трійок (i, j, p) які означають, що компанія i володіє $p\%$ компанії j, обчислити всі пари (h, s) такі що компанія h контролює компанію s. Кількість компаній обмежена 100.

Завдання Напишіть програму яка:

1. Читає з вхідного `ASCII`-файлу `COMPANY.DAT`, послідовність трійок (i, j, p), де i, j і p — цілі додатні числа. У файлі можуть знаходитися дані для декількох тестів. Окремі тестові блоки відокремлені порожнім рядком.

2. Знаходить усі такі пари (h, s), що компанія h контролює компанію s.

3. Записує в вихідний `ASCII`-файл `COMPANY.SOL`, усі знайдені пари (h, s), в яких h відрізняється від s. Пари (h, s) повинні бути записані в послідовних рядках, та впорядковані за зростанням h. Розв'язки для різних тестів повинні бути відокремлені порожнім рядком.

Приклад вхідних та вихідних даних

COMPANY.DAT
2 3 25
1 4 36
4 5 63
2 1 48
3 4 30
4 2 52
5 3 30
1 2 30
2 3 52
3 4 51
4 5 70
5 4 20
4 3 20

COMPANY.SOL
4 2
4 3
4 5
2 3
2 4
2 5
3 4
3 5
4 5

5.1.3 «Прямокутники»

N прямокутників різного кольору накладаються на білий прямокутний аркуш паперу, з розмірами a см. у ширину та b см. у довжину. Всі прямокутники повністю розміщуються на аркуші паперу, та їхні сторони паралельні границям аркуша. У результаті з'являються фігури різних кольорів. Два прямокутника одного кольору вважаються частиною однієї фігури, якщо вони мають принаймні одну спільну точку. В іншому випадку, вони вважаються різними фігурами.

Необхідно обчислити площу кожної з таких однокольорових фігур. a, b — парні додатні цілі числа, які не перевищують 30.

Вважається, що початок системи координат знаходиться у центрі аркуша, та його вісі паралельні границям аркуша.

Вхідний ASCII-файл RECTANG.DAT містить декілька наборів тестових даних. Числа a, b і N, розділені пропуском, знаходяться у першому рядку кожного тестового набору.

У кожному з наступних N рядків знаходяться:

- цілочисельні координати лівої нижньої вершини прямокутника,
- після цього слідують цілочисельні координати правої верхньої вершини прямокутника,
- потім — колір прямокутника, заданий цілим числом від 1 до 64. Білий колір представлений числом 1.

Порядок рядків відповідає порядку в якому прямокутники було покладено на аркуш паперу. Різні тестові блоки будуть відокремлюватися порожнім рядком.

Завдання Напишіть програму яка:

1. Зчитує наступний блок даних з RECTANG.DAT.
2. Обраховує площу кожної фігури.
3. Записує у вихідний ASCII-файл RECTANG.SOL, колір та площу кожної фігури, як це продемонстровано на прикладі, в порядку зростання кольору. Розв'язки для різних тестових блоків повинні відокремлюватися порожнім рядком.

Приклад вхідних та вихідних даних

RECTANG.DAT

```
20 12 5
-7 -5 -3 -1 4
-5 -3 5 3 2
-4 -2 -2 2 4
2 -2 3 -1 12
3 1 7 5 1

30 30 2
0 0 5 14 2
-10 -7 0 13 15
```

RECTANG.SOL

```
1 172
2 47
4 12
4 8
12 1

1 630
2 70
15 200
```

5.2 Завдання другого туру

5.2.1 «Канадські авіалінії»

Ви перемогли у змаганні, організованому Канадськими авіалініями. Головний приз — подорож по Канаді. Подорож починається з самого західного міста, що обслуговується авіакомпанією, та проходить лише у східному напрямку доки не буде досягнуте саме східне місто. Після цього подорож проходить у зворотньому напрямку, з сходу на захід, доки не буде досягнуте місто, з якого вона почалася. Жодне місто не може бути відвідано більше ніж один раз, за виключенням початкового міста, де потрібно побувати рівно двічі (на початку та в кінці подорожі). Вам не дозволяється користуватися послугами будь-якої іншої авіакомпанії, або будь-якого іншого виду транспорту.

За списком міст, що обслуговуються авіакомпанією, та списком прямих рейсів між парами міст необхідно знайти маршрут, що містить якомога більше міст, та задовольняє описаним вимогам. Маршрут повинен починатися з першого міста у списку, включати останнє, та завершуватися у першому.

У вхідному ASCII-файлу `C:\IOI\ITIN.DAT` може бути записано декілька наборів тестових даних.

- Перший рядок містить додатне ціле число N — кількість міст ($N \leq 100$), що обслуговуються авіакомпанією, та додатне ціле число V — кількість рейсів між цими містами.
- Кожен з наступних N рядків задає назву міста, що обслуговується авіакомпанією. Міста впорядковані з заходу на схід, тобто i-те місто західніше j-го тоді і тільки тоді коли $i < j$ (Немає двох різних міст, розташованих на одному мередіані). Назва кожного міста це рядок, що складається, якнайбільше з 15 цифр та/або символів латинського алфавіту. Наприклад: `AGR34` або `BEL4`. (В назві міста не може бути пропусків).
- Кожен з наступних V рядків містить дві назви міст, що розділені пропуском. Пара $city_1$ $city_2$ означає що існує прямий рейс з міста $city_1$ у $city_2$ і навпаки з $city_2$ у $city_1$.

Різні набори тестових даних відокремлені порожнім рядком. Після останнього набору даних порожнього рядка немає.

Вхідні дані — коректні, їх перевірка не вимагається.

Розв'язок знайдений для кожного набору даних повинен бути записаний у вихідний ASCII-файл `C:\IOI\ITIN.SOL` наступним чином:

- В першому рядку розв'язку повинна міститися загальна кількість міст у тестовому наборі даних.

- У другому рядку число M — кількість різних міст, які містить знайдений маршрут.

- В наступних $M + 1$ рядку назви міст, у порядку, в якому вони були відвідані.

Перше місто та останнє у списку повинні співпадати. Необхідно знайти тільки один розв'язок. У випадку, коли тестового набору даних неможливо знайти розв'язок, тільки два рядки повинні бути записані у вихідний файл. Перший рядок з загальною кількістю міст, та другий рядок з повідомленням `NO SOLUTION`.

Приклад вхідних та вихідних даних

ITIN.DAT
8 9 Vancouver Yellowknife Edmonton Calgary Winnipeg Toronto Montreal Halifax Vancouver Edmonton Vancouver Calgary Calgary Winnipeg Winnipeg Toronto Toronto Halifax Montreal Halifax Edmonton Montreal Edmonton Yellowknife Edmonton Calgary

ITIN.SOL
8 7 Vancouver Edmonton Montreal Halifax Toronto Winnipeg Calgary Vancouver

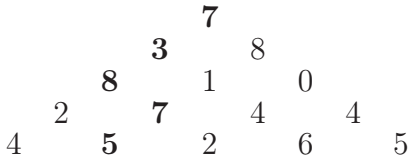
ITIN.DAT
5 5 C1 C2 C3 C4 C5 C5 C4 C2 C3 C3 C1 C4 C1 C5 C2

ITIN.SOL
5 NO SOLUTION

Розділ 6. Швеція'1994

6.1 Завдання першого туру

6.1.1 «Трикутник»



На малюнку зображено числовий трикутник. Написати програму, яка визначає максимальну суму чисел розташованих на шляху, який починається з верхнього числа та закінчується на будь-якому з чисел основи трикутника.

- На кожному кроці можна рухатися по діагоналі вниз та ліворуч або по діагоналі вниз та праворуч.
- Кількість рядків у трикутнику > 1 та ≤ 100.
- Усі числа у трикутнику — цілі в інтервалі від 0 до 99 включно.

На прикладі зображеному вище максимальна сума досягається на шляху 7, 3, 8, 7, 5, та дорівнює 30.

Вхідні дані Дані о кількості рядків у трикутнику знаходяться в першому рядку вхідного файлу `INPUT.TXT`. Далі записана інформація про рядки трикутника.

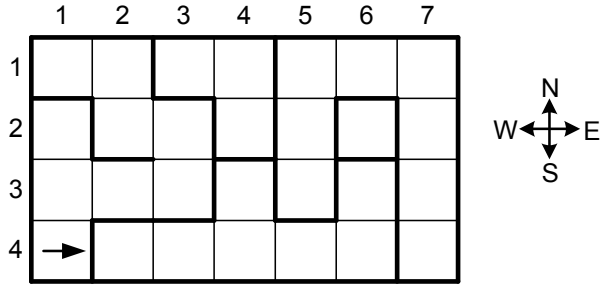
Вихідні дані Найбільша можлива сума повинна бути записана у вихідному файлі з ім'ям `OUTPUT.TXT`.

Приклад вхідних та вихідних даних

INPUT.TXT
5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5

OUTPUT.TXT
30

6.1.2 «Замок»



На малюнку зображено карту замка. Напишіть програму, що визначає:

1. Кількість кімнат у замку.
2. Площу найбільшої кімнати.
3. Стіну, яку потрібно знищити, щоб отримати кімнату якомога більшої площі.

Замок умовно розділений на $m \times n$ ($m \leq 50$, $n \leq 50$) квадратних клітин. Навколо кожної клітини може бути від 0 до 4 стін включно.

Вхідні дані Карта замку зберігається у файлі `INPUT.TXT` у формі чисел, по одному для кожної клітини.

- Перший рядок файлу містить кількість клітин у напрямку з півночі на південь, а другий — кількість клітин у напрямку з заходу на схід.
- У наступних рядках кожна клітина описується числом p ($0 \leq p \leq 15$). Число p задає розташування стін навколо клітини і дорівнює сумі чисел, заданих наступним чином: 1 (якщо є стіна на заході), 2 (якщо є стіна на півночі), 4 (якщо є стіна сході) і 8 (якщо є стіна на півдні). Внутрішні стіни задані двічи. Наприклад: стіна з півдня у клітині $1,1$ також задана як стіна з півночі у клітині $2,1$.
- У замку обов'язково є хоча б дві кімнати.

Вихідні дані В файл `OUTPUT.TXT` програма повинна записати три рядка:

1. У першому — кількість кімнат.
2. У другому — площа (кількість клітин) найбільшої кімнати;
3. У третьому — пропозиція щодо стіни яку треба знищити: спочатку — координати (рядок та стовбчик) клітини біля цієї стіни, потім — напрямок за компасом (латинську букву), що вказує з цієї клітини на стіну, що знищується. Можливі декілька розв'язків, але вам потрібно вказати будь-який з них.

Приклад вхідних та вихідних даних

INPUT.TXT
4 7 11 6 11 6 3 10 6 7 9 6 13 5 15 5 1 10 12 7 13 7 5 13 11 10 8 10 12 13

OUTPUT.TXT
5 9 4 1 E

6.1.3 «Прості»

1	1	3	5	1
3	3	2	0	3
3	0	3	2	3
1	4	0	3	3
3	3	3	1	1

На малюнку зображено квадрат. Кожен рядок, кожен стовпчик та дві діагоналі можна розглядати як п'ятизначне просте число. Рядки читаються зліва направо. Стовбчики читаються зверху вниз. Обидві діагоналі читаються зліва направо. Напишіть програму, яка знаходить такі квадрати, з використанням даних з файлу `INPUT.TXT`. Усі прості числа по горизонталі, вертикалі та діагоналі повинні мати однакові суми цифр (в прикладі — 11). Цифра у верхньому лівому куті квадрату задана заздалегіть (у прикладі — 1). Просте число може з'являтися в одному і тому ж квадраті декілька разів. Якщо є декілька розв'язків, необхідно знайти їх усі. П'ятизначне просте число починається не з нуля, тобто `00003` не є п'ятизначним простим числом.

Вхідні дані Програма читає вхідні дані з файлу `INPUT.TXT`. Перше число — сума цифр кожного з простих чисел, друге — цифра в лівому верхньому куту квадрата. Файл складається з двох рядків. Для кожного набору тестових даних існує хоча б один розв'язок.

Вихідні дані В файл `OUTPUT.TXT` програма повинна записати 5 рядків для кожного отриманого розв'язку, кожний з яких містить п'ятизначне просте число.

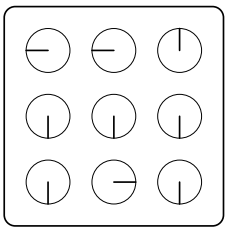
Приклад вхідних та вихідних даних

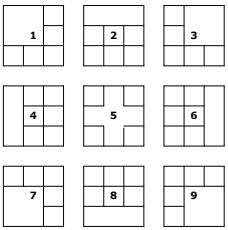
INPUT.TXT
11 1

OUTPUT.TXT
11351 14033 30323 53201 13313 11351 33203 30323 14033 33311 13313 13043 32303 50231 13331

6.2 Завдання другого туру

6.2.1 «Годинник»





У матриці розміром 3×3 розміщені 9 циферблатів з заданим положенням стрілок (див. приклад на мал. 1). Мета — повернути усі циферблати, встановивши їх на 12 годин за найменьшу кількість кроків.

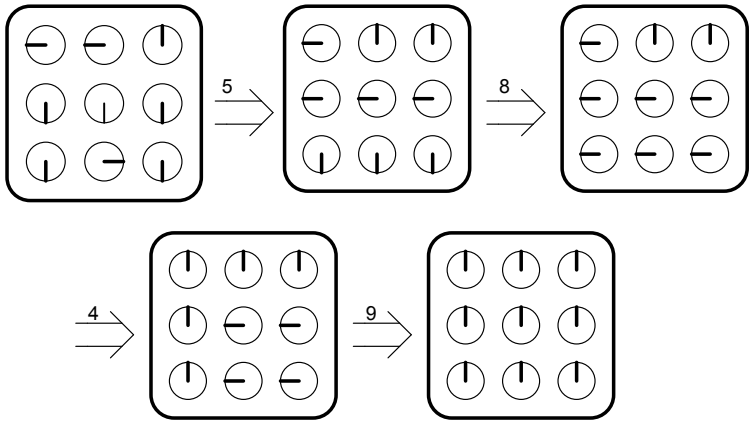
Можливі 9 різних способів повертання циферблатів. Ці способи зображено на мал. 2. Кожний такий спосіб задається числом від 1 до 9, та повертання підмножини циферблатів, що виділені сірим кольором на мал. 2,

на 90° за годинниковою стрілкою. Переведення стрілок з початкового положення у кінцеве відбувається покроково.

Напишіть програму, що визначає найкоротшу послідовність кроків.

Вхідні дані Прочитайте дев'ять цифр з файлу `INPUT.TXT`, в якому задано початкове розташування циферблатів. Положення стрілки на циферблаті задається числом від 0 до 3: 0 = 12 годин, 1 = 3 години, 2 = 6 годин, 3 = 9 годин.

Вихідні дані Запишіть у вихідний файл `OUTPUT.TXT` шукану мінімільну послідовність кроків (номерів способів), що встановлюють усі циферблати на 12 годин. Якщо для набору тестових даних існує декілька оптимальних розв'язків, достатньо знайти тільки один з них.



Приклад вхідних та вихідних даних

INPUT.TXT	OUTPUT.TXT
3 3 0 2 2 2 2 1 2	5849

6.2.2 «Автобуси»

Чоловік прийшов на автостанцію о 12^{00} і знаходився на ній до 12^{59}. На станції зупиняються автобуси декількох маршрутів. Чоловік записав час прибуття кожного автобусу. Ці дані — відомі. Визначіть за такими даними розклад з найменшою кількістю автобусних маршрутів, які можуть зупинятися на станції, щоб виконувалися наступні вимоги:

- Автобуси одного маршруту прибувають з рівномірним інтервалом (через однакові проміжки часу) з 12^{00} до 12^{59} на протязі всієї години.
- Час задається у хвилинах цілими числами від 0 до 59.
- Автобуси кожного з маршрутів зупиняються з 12^{00} до 12^{59} по меньшій мірі 2 рази.
- Кількість маршрутів у тестових прикладах не перевищує 17.
- Декілька автобусних маршрутів можуть мати однаковий час прибуття і/або інтервал. Для кожного маршруту треба вивести час прибуття першого автобуса та інтервал руху цього маршруту.

Вхідні дані Вхідний файл `INPUT.TXT` містить число n ($n \leq 300$), що відповідає кількості автобусів, що прибули та були записані. Далі слідує рядок з часом прибуття автобусів (у хвилинах).

Вихідні дані Запишіть результат у файл `OUTPUT.TXT`. Кожен його рядок повинен містити дані для одного маршруту. В рядку повинні бути час прибуття першого автобуса цього маршруту, та інтервал часу в хвилинах.

Якщо для набору тестових даних можливо знайти декілька оптимальних розв'язків, достатньо вказати один з них.

Приклад вхідних та вихідних даних

INPUT.TXT
17 0 3 5 13 13 15 21 26 27 ↓ 29 37 39 39 45 51 52 53

OUTPUT.TXT
0 13 3 12 5 8

6.2.3 «Круги»

Задан круг, розділений на сектори. Відомі три числа: k ($0 < k \leq 20$), n ($n \leq 6$) та m ($m \leq 20$), де n — кількість секторів. Потрібно розмістити в секторах додатні цілі числа $\geq k$. Коли сектори заповнені числами, ви можете отримувати з них нові числа за одним з наступних правил:

- взяти число з одного сектора, або
- взяти число, що дорівнює сумі чисел у двох, або більше, суміжних секторах.

З цих нових чисел необхідно скласти скінченну послідовність цілих чисел, що ідуть підряд, починаючи з m: $(m, m+1, m+2, \ldots, i)$.

Ваша задача — написати програму, що визначає для заданих n, m і k числа в секторах так, щоб найбільший член i, отриманної нерозривної

послідовності був максимальним для усіх послідовностей, означених таким чином.

Приклад на мал. 1 показує, як отримати всі числа від 2 до 21 для деяких чисел у секторах. Числа в секторах, виділених сірим, складаються, щоб отримати члени шуканої послідовності.

Вхідні дані Вхідний файл `INPUT.TXT` містить три цілих числа (n, m і k).

Вихідні дані Вихідний файл `OUTPUT.TXT` повинен містити:

- Найбільше число (i) у нерозривній послідовності, яке може бути отримано з чисел у секторах

- Всі набори чисел у секторах, з яких можна отримати нерозривну послідовність від m до i. В кожному рядку треба вивести один набір чисел у секторах. Кожний такий набір — список чисел, що починається з найменшого (яке може бути не єдиним)

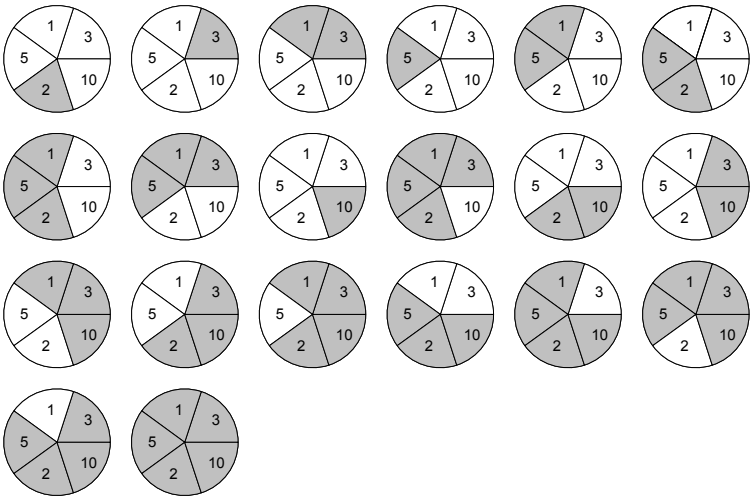
Наприклад, `2 10 3 1 5` не є розв'язком, так як починається не з найменшого числа. Зверніть увагу, що `1 3 10 2 5` та `1 5 2 10 3` вважаються різними розв'язками і повинні бути виведені.

Якщо найменше число зустрічається декілька разів, необхідно вивести всі можливі комбінації, наприклад, `1 1 2 3`, `1 2 3 1`, `1 3 2 1` та `1 1 3 2`.

Приклад вхідних та вихідних даних

INPUT.TXT
5 2 1

OUTPUT.TXT
21 1 3 10 2 5 1 5 2 10 3 2 4 9 3 5 2 5 3 9 4



Розділ 7. Нідерланди'1995

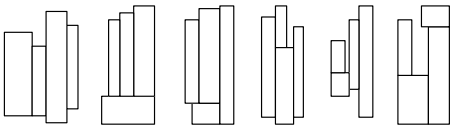
7.1 Завдання першого туру

7.1.1 «Упаковка прямокутників»

Дано чотири прямокутники. Знайдіть найменший прямокутник, який їх покриває, тобто у якому можуть розміститися не перекриваючи один одного задані чотири прямокутники. Під найменшим ми розуміємо прямокутник найменшої площі.

Сторони всіх чотирьох прямокутників повинні бути паралельними відповідним сторонам прямокутника, що їх покриває. Мал. 1 зображує шість способів сумісного розташування чотирьох прямокутників. Ці 6 прикладів вичерпують усі можливі структури, оскільки будь-яка інша може бути отримана з деякої наведеної поворотами та/або відображеннями.

Можуть існувати покриваючі прямокутники з різними довжинами сторін, що задовольняють умовам і мають мінімальну площу. Ви повинні отримати всі такі прямокутники, що покривають задані.



Вхідні дані Вхідний файл `INPUT.TXT` складається з чотирьох рядків. Кожен рядок описує один вихідний прямокутник двома додатніми цілими числами: довжинами сторін прямокутника. Кожна сторона прямокутника має довжину не менше ніж 1 і не більше ніж 50.

Вихідні дані Вихідний файл `OUTPUT.TXT` повинен містити на один рядок більше, ніж існує розв'язків. Перший рядок повинен містити одне ціле число: мінімальну площу шуканого прямокутника (підзадача A). Кожен з наступних рядків повинен містити один розв'язок, описаний числами p і q — довжинами сторін шуканого прямокутника; при цьому повинна виконуватись умова $p \le q$ (підзадача B). Ці рядки повинні бути впорядковані за зростанням p; усі вони повинні бути різними.

Приклад вхідних та вихідних даних

INPUT.TXT
1 2 2 3 3 4 4 5

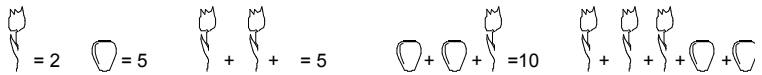
OUTPUT.TXT
40 4 10 5 8

7.1.2 «Торгівельні знижки»

В магазині кожен товар має ціну. Наприклад, ціна однієї квітки дорівнює 2 ICU (ICU — валютні одиниці інформатики), а ціна однієї вази дорівнює 5 ICU. Щоб привабити більше покупців, магазин ввів знижки.

Знижка полягає в тому, щоб продавати набір однакових або різних товарів за зниженою ціною. Приклади: три квітки за 5 ICU замість 6, або дві вази разом з однією квіткою за 10 ICU замість 12.

Напишіть програму, що обчислює найменшу ціну, яку покупець має сплатити за задані покупки. Оптимальний розв'язок повинен бути отриманий за допомогою знижок. Набір товарів, який треба купити, не можна доповнювати нічим, навіть якщо б це знизило загальну вартість набора. Для описаних вище цін і знижок найменша ціна за три квітки і дві вази дорівнює 14 ICU: дві вази і одна квітка продаються за зниженою ціною за 10 ICU, і дві квітки — за звичайною ціною — 4 ICU.



Вхідні дані Вхідні дані містяться в двох файлах: `INPUT.TXT` та `OFFER.TXT`. Перший файл описує покупки («кошик з покупками»). Другий файл описує знижки. В обох файлах містяться тільки цілі числа.

Перший рядок файлу `INPUT.TXT` містить кількість b різних видів товару в кошику ($0 \le b \le 5$). Кожен з наступних b рядків містить значення c, k і p. Значення c — унікальний код товару ($1 \le c \le 999$). Значення k задає, скільки одиниць товару знаходиться в кошику ($1 \le k \le 5$). Значення p задає звичайну (без знижок) ціну одиниці товару ($1 \le p \le 999$). Зверніть увагу, що загальна кількість b товарів у кошику може бути не більше ніж $5 \times 5 = 25$ одиниць.

Перший рядок файлу `OFFER.TXT` містить кількість s можливих знижок ($0 \le s \le 99$). Кожен з наступних s рядків описує одну знижку, визначаючи набір товарів і загальну вартість набору. Перше число n в такому рядку визначає кількість різних видів товару в наборі ($1 \le n \le 5$). Наступні n пар чисел (c, k) вказують, що k одиниць товару з кодом c включені до набору для знижки ($1 \le k \le 5$, $1 \le c \le 999$). Останнє число в рядку p визначає знижену вартість набору ($1 \le p \le 9\,999$). Вартість набору менше сумарної вартості окремих одиниць товарів в наборі.

Вихідні дані Запишіть до вихідного файлу `OUTPUT.TXT` один рядок з найменшою можливою сумарною вартістю покупок, заданих у вхідному файлі.

Приклад вхідних та вихідних даних

INPUT.TXT
2 7 3 2 8 2 5

OFFER.TXT
2 1 7 3 5 2 7 1 8 2 10

OUTPUT.TXT
14

7.1.3 «Клієнт і сервер»

Є два користувача, у кожного з яких є комп'ютер. Комп'ютери ідентифікуються іменами: CLIENT(1) та CLIENT(2). Два комп'ютера підключені до одного або більше принтерів з іменами SERVER(1), SERVER(2), тощо. Друк з обох комп'ютерів може виконуватись тільки послідовно. Щоб скоординувати взаємодію комп'ютерів з принтером використовується об'єкт SEMAPHORE (семафор).

Об'єкт SEMAPHORE Кожен принтер має семафор, що відноситься до нього. Семафор знаходиться в одному з станів: S1 або S2. Коли принтер вільний для друку, семафор знаходиться у стані S1. Коли принтер друкує, семафор знаходиться у стані S2.

Семафор може здійснювати два типи переходів між станами S1⇒S2 та S2⇒S1. Коли користувач надсилає комп'ютеру завдання для друку, комп'ютер надсилає семафору повідомлення `Are_you_open?`. Якщо семафор знаходиться у стані S1, то його стан змінюється на S2, та семафор надсилає повідомлення `Open` тому комп'ютеру, який надіслав повідомлення `Are_you_open?`. Якщо семафор знаходиться у стані S2, то він повертає пові-

домлення `Closed`. Після завершення друку принтер надсилає повідомлення `Ready` своєму семафору. Після отримання повідомлення `Ready` семафор змінює свій стан на S1.

Тип об'єкту SEMAPHORE В документі 1 (Documentation 1) ви знайдете специфікацію типу об'єкту SEMAPHORE. Специфікація містить можливі ідентифікатори та стан об'єкту SEMAPHORE, список приоритетів (Priority list) діаграму з'єднань (Communication diagram), діаграму переходу станів діаграму (State transition diagram), процедури прийому діаграму (Receive procedures) для повідомлень: Ready та Are˙you˙open?. В процедурах Receive procedures описано, як семафор відкликається на повідомлення.

Список приоритетів (Priority list) необхідний, тому що повідомлення, що отримуються семафором одночасно, можуть бути оброблені тільки послідовно. Список приоритетів (Priority list) В документі 1 вказує, наприклад, що кожне повідомлення SERVER має більш високий приоритет, ніж повідомлення CLIENT, а повідомлення SERVER(2) більш високий, ніж повідомлення SERVER(3).

Тип об'єкту CLIENT В документі 2 ви знайдете специфікацію типу об'єкту CLIENT. Об'єкт типу CLIENT знаходиться в одному з трьох станів: SA, SB або SC. Клієнт знаходиться в стані SA, якщо цей клієнт не звертался до сервера і сервери не друкують завдання для цього клієнту. Клієнт знаходиться в стані SB, якщо він хоче отримати доступ до сервера, но не може (клієнт може отримати доступ до сервера лише за допомогою семафору). Клієнт знаходиться у стані SC, якщо сервер виконує завдання на друк для даного клієнта.

Об'єкт CLIENT може змінювати стан трьома способами: SA⇒SB, SB⇒SC, SC⇒SA

Коли об'єкт CLIENT знаходиться в стані SB, він може отримати повідомлення `Closed` від семафору. Після отримання цього повідомлення клієнт очікує на протязі періоду `Waiting_Period`, перед тим, як знову надіслати семафору повідомлення `Are_you_open?`. Коли семафор надсилає клієнту повідомлення `Open`, цей клієнт змінює свій стан на SC та одночасно надсилає завдання на друк серверу за допомогою повідомлення `S_Job`. Після завершення роботи сервер одночасно надсилає два повідомлення: повідомлення `Ready` своєму семафору і повідомлення `C_Ready` об'єкту CLIENT. Після цього об'єкт CLIENT змінює свій стан з SC на SA.

Принтери можна вважати ідеальними, тобто вони завершують кожне завдання на друк. Наприклад, ніколи не закінчується папір.

Communication (З'єднання) В документі 3 ви знайдете в діаграмі структури з'єднань (Communication Structure Diagram) всі типи повідомлень, які можуть пересилатися між різними типами об'єктів. У списку повідомлень (Message List) ви знайдете специфікацію кожного типу повідомлення. Кожне повідомлення має ідентифікатор, ім'я відправника, ім'я одержувача та, в деяких випадках, якісь зміст.

Коли відправник у момент часу t надсилає повідомлення з ідентифікатором A, то одержувач у момент часу $t+1$ буде обробляти повідомлення з використанням процедури Receive Procedure з ім'ям A.

Якщо декілька відправників у момент часу t надсилають повідомлення одному й тому ж одержувачу, то одержувач обробляє усі ці повідомлення в момент $t+1$ в тому ж порядку, в якому відправники розташовані у списку приоритетів (Priority List) даного одержувача.

Підзадача A В початковий момент часу ($t = 0$) локальна мережа задається наступними об'єктами:

- Object: CLIENT(1), `Client.State`=SA, `Waiting_Period`=2, `Number_of_Servers`=1
- Object: CLIENT(2), `Client.State`=SA, `Waiting_Period`=1, `Number_of_Servers`=1
- Object: SERVER(1)
- Object: SEMAPHORE(1), `Semaphore.State`=S1

У цій мережі були надіслані, зокрема, наступні повідомлення:

- в момент $t = 1$: CLIENT(1) надіслав повідомлення з ідентифікатором `Are_you_open?`;
- в момент $t = 2$: CLIENT(2) надіслав повідомлення з ідентифікатором `Are_you_open?`;
- в момент $t = 4$: SERVER(1) надіслав повідомлення з ідентифікатором `Ready`;
- в момент $t = 5$: CLIENT(1) надіслав повідомлення з ідентифікатором `Are_you_open?`.

У документі 4 для об'єктів SEMAPHORE(1), CLIENT(1) та CLIENT(2) у вигляді таблиці розкладу до моменту $t = 6$ відображено, які повідомлення об'єкти надсилали, які повідомлення об'єкти отримували, а також у яких станах об'єкти знаходяться, або в які стани вони перейшли.

Питання A.1 Що трапиться, якщо у доповнення к вищесказаному у момент часу $t = 4$ об'єкт CLIENT(1) отримає повідомлення `C_Job`?

Запишіть свою відповідь у документі 5 (відповідь складайте згідно документу 4)

Питання A.2 Що трапиться, якщо в доповнення к вищесказаному в момент $t = 4$ об'єкт CLIENT(2) отримає повідомлення `C_Job` замість об'єкту CLIENT(1)?

Запишіть свою відповідь у документі 5 (відповідь складайте згідно документу 4)

Питання A.3 Заповніть в документі 4 таблицю розкладу до моменту $t = 13$ включно, якщо відомо, що:

- в момент $t = 8$: SERVER(1) надіслав повідомлення з ідентіфікатором `Ready`;
- в момент $t = 10$: CLIENT(1) отримав повідомлення з ідентифікатором `C_Job`; в момент $t = 12$: SERVER(1) надіслав повідомлення з ідентифікатором `Ready`.

Підзадача B Розглянемо розширену локальну мережу. Тепер вона включає дві пари semaphore-server: (SEMAPHORE(1), SEMAPHORE(2), SERVER(1), SERVER(2)).

Для кожного об'єкту CLIENT значення `Number_of_Servers` дорівнює 2. Для того щоб використовувати обидва принтери, необхідно внести зміни в описання об'єкту CLIENT, яке наведене в документі 2. В документі 6 наведені змінені процедури прийому повідомлень `Wait` та `C_Job`, а також опис станів об'єктів в момент $t = 0$.

З врахуванням сказангого, у вказані моменти часу об'єкти приймають наступні повідомлення:

- в момент $t = 0$: CLIENT(1) отримав повідомлення з ідентифікатором `C_Job` від користувача;
- в момент $t = 0$: CLIENT(2) отримав повідомлення з ідентифікатором `C_Job` від користувача;
- в момент $t = 4$: SEMAPHORE(1) отримав повідомлення з ідентифікатором `Ready`.

Що трапиться в заданій розширеній мережі при вказаних змінах у описі об'єкту CLIENT? Оберіть вірну відповідь з наведених у документі 6 варіантів.

Підзадача C З'ясувалося, що здійснені у підзадачі B зміни у опис об'єкту CLIENT є неефективними для роширенної мережі з більш ніж одною парою semaphore-server.

Питання С.1 Внести зміни в опис об'єкту типу CLIENT (см. документ 2) так, щоб розширена локальна мережа з підзадачи B з більш ніж одною парою semaphore-server могла функційонувати наступним чином:

- Об'єкт CLIENT(i) міг використовувати будь-які сервери у мережі, при цьому в кожний момент часу він повинен мати не більше одного завдання, що виконується. Кожне завдання друкується лише один раз.

- Об'єкт CLIENT(i) надсилає повідомлення з ідентифікатором `Are_you_open?` декільком семафорам доки він не отримає повідомлення `Open` чи не отримає визначене для даного об'єкту CLIENT порогову кількість повідомленнь `Closed` (див. документ 3).

- Коли об'єкт CLIENT(i) отримує відповідну кількість відповідей `Closed` він очікує на протязі періоду часу `Waiting_Period`, після чого знову розсилає серії повідомлень `Are_you_open?`.

Запишіть свій розв'язок в документі 7.

Питання С.2 Змініть опис типу об'єкту CLIENT так, щоб об'єкт CLIENT(i) міг мати також декілька завдань, що виконуються одночасно різними серверами. Однак, кількість завдань на друк у об'єкту CLIENT(i) не повинно перевищувати значення CLIENT(i).`Job_Maximum`. Запишіть свій розв'язок у документі 8.

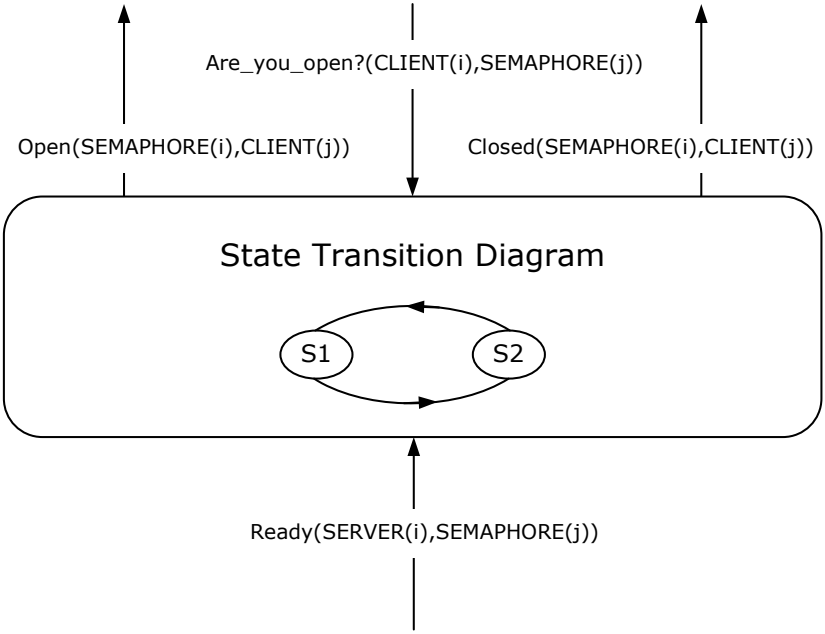
Документ 1 Object type: SEMAPHORE

Possible identifiers: SEMAPHORE(1),SEMAPHORE(2),...

State: S1,S2; initial state is S1

Priority List: SERVER(1),SERVER(2),...,CLIENT(1),CLIENT(2)...

Communication Diagram



Receive Procedure

```
procedure Are_you_open?(Client,Semaphore)
begin
        if State = S1
        then State ← S2
                Send("Open(Semaphore,Client)")
        else
        if State = S2
        then Send("Closed(Semaphore,Client)")
end


procedure Ready(Server,Semaphore)
begin
        State ← S1
end
```

Документ 2 Object type: CLIENT

Possible identifiers: CLIENT(1),CLIENT(2),CLIENT(3),...

State: SA,SB,SC; initial state is SA

Priority List: CLIENT,SERVER(1),SERVER(2),...,
SEMAPHORE(1),SEMAPHORE(2)..., USER(1),USER(2),...

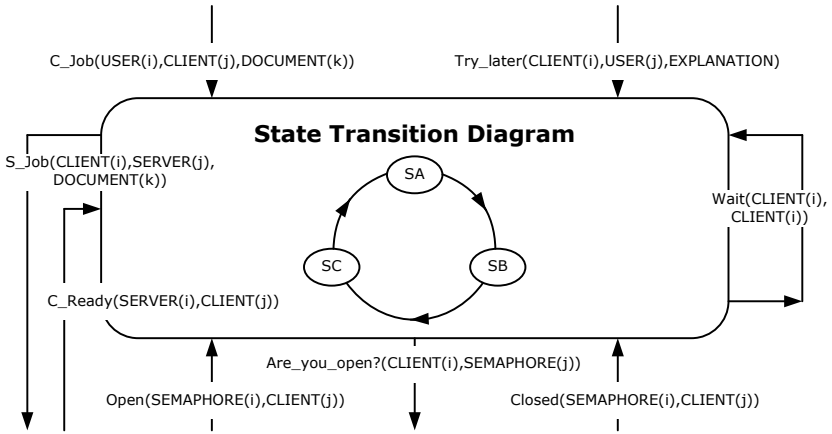
Countdown: $\{t \mid t \in \mathbb{N}\}$; initial value is 0

Waiting_Period: $\{t \mid t \in \mathbb{N} \text{ and } t > 0\}$

Semaphore_Index: $\{i \mid i \in 1, 2, \ldots, \text{Number\_of\_Servers}\}$

Number_of_Servers: $\{i \mid i \in \mathbb{N} \text{ and } i > 0\}$

Communication Diagram



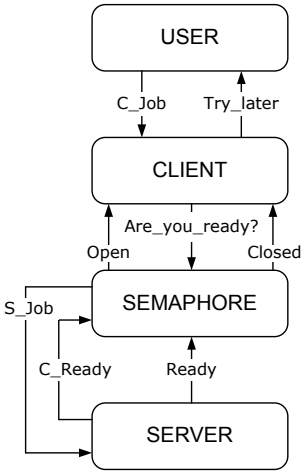
Receive Procedure

procedure C_Job(User,Client,Document)
begin
    if State = SA

```
        then State ← SB
             Send("Are_you_open?(Client,
             SEMAPHORE(Semaphore_Index)"))
        else
        if State = SB
        then Send("Try_later(Client,User,
                  Client_is_busy"))
        else
        if State = SC
        then Send("Try_later(Client,User,
                  All_Servers_are_busy"))
end

procedure Open(Semaphore,Client)
begin
        if State = SB
        then State ← SC
             Send("S_Job(Client,Server,
             Document)")
end

procedure Closed(Semaphore,Client)
begin
        Countdown ← Waiting_Period
        Send("Wait(Client,Client)")
end


procedure Wait(Client,Client)
begin
        Countdown ← Countdown - 1
        if Countdown > 0
        then Send("Wait(Client,Client)")
        else Send("Are_you_open?(Client,
                  SEMAPHORE(Semaphore_Index)"))
end


procedure C_Ready(Client,Client)
begin
        State ← SA
```

end

Документ 3



Message List $(i, j, k \in N)$

identifier	**sender**	**receiver**	**content**
`Are_you_open?`	CLIENT(i)	SEMAPHORE(j)	-
`C_Job`	USER(i)	CLIENT(j)	DOCUMENT(k)
`C_Ready`	SERVER(i)	CLIENT(j)	-
`Closed`	SEMAPHORE(i)	CLIENT(j)	-
`Open`	SEMAPHORE(i)	CLIENT(j)	-
`Ready`	SERVER(i)	SEMAPHORE(j)	-
`S_Job`	CLIENT(i)	SERVER(j)	DOCUMENT(k)
`Try_later`	CLIENT(i)	USER(j)	-
`Wait`	CLIENT(i)	CLIENT(j)	EXPLANATION

Документ 4

Question A.1
What would happened if CLIENT(1) receives a message "C_Job"at time 4?

Question A.2
What would happened if CLIENT(2) receives a message "C_Job"at time 4?

Документ 5

SEMAPHORE(1)

time	received messages	State/ Transition	sent messages
0		S1	
1		S1	
2	Are_you_open?	S1→S2	Open
3	Are_you_open?	S2	Closed
4		S2	
5	Ready	S2→S1	
6	Are_you_open? Are_you_open?	S1→S2 S2	Open Closed
7			
8			
9			
10			
11			
12			
13			

CLIENT(1)

time	received messages	State/ Transition	Count-down	sent messages
0		SA	0	
1	C_Job	SA→SB	0	Are_you_open?
2		SB	0	
3	Open	SB→SC	0	S_Job
4		SC	0	
5	C_Ready C_Job	SC→SA SA→SB	0 0	 Are_you_open?
6				
7				
8				
9				
10				
11				
12				
13				

CLIENT(2)

time	received messages	State/ Transition	Count-down	sent messages
0		SA	0	
1		SA	0	
2	C_Job	SA→SB	0	Are_you_open?
3		SB	0	
4	Closed	SB	0→1	Wait
5	Wait	SB	1→0	
6				
7				
8				
9				
10				
11				
12				
13				

Документ 6

The changed Receive Procedures: C_Job and Wait

```
procedure C_Job(User,Client)
begin
        if State = SA
        then State ← SB
                for Index←1 step 1 until Number_of_Servers
                    Send("Are_you_open?(Client, SEMAPHORE(Index)"))
        else
        if State = SB
        then Send("Try_later(Client,User,
                     Client_is_busy"))
        else
        if State = SC
        then Send("Try_later(Client,User,
                     All_Servers_are_busy"))
end


procedure Wait(Client,Client)
begin
        if State = SB
        then Countdown←Countdown - 1
                if Countdown > 0
                then Send("Wait(Client,Client)")
                else
                if Countdown < 0
                then Countdown ← 0
                else for Index←1 step 1 until Number_of_Servers
                          Send("Are_you_open?(Client, SEMAPHORE(Index)"))
end
```

At time 0 the situation in LAN is as follows:
Object: CLIENT(1), `Waiting_Period`=2, `Number_of_Servers=2`, Client.State=SA
Object: CLIENT(2), `Waiting_Period`=1, `Number_of_Servers`=2, Client.State=SA
Object: SEMAPHORE(1), Semaphore.State=S1
Object: SEMAPHORE(2), Semaphore.State=S1
As the following the following messages are received:
At time 0: CLIENT(1) receives a message `C_Job` of a user.
At time 0: CLIENT(2) receives a message `C_Job` of a user.
At time 4: SEMAPHORE(1) receives a message `Ready`.
What happens in the extended LAN according to the changed object CLIENT? Mark the correct answer.

(a) The print job of CLIENT(1) will be printed both on SERVER(1) and SERVER(2). The print job of CLIENT(2) will not be printed.

(b) The print job of CLIENT(1) will be printed once on SERVER(1). The print job of CLIENT(2) will be printed once on SERVER(2)

(c) The print job of CLIENT(1) will be printed once on SERVER(1). The print job of CLIENT(2) will be printed once on SERVER(1)

(d) The print job of CLIENT(1) will be printed once on SERVER(2). The print job of CLIENT(2) will be printed once on SERVER(2)

(e) The print job of CLIENT(1) will not be printed. The print job of CLIENT(2) will be printed on SERVER(1) and SERVER(2)

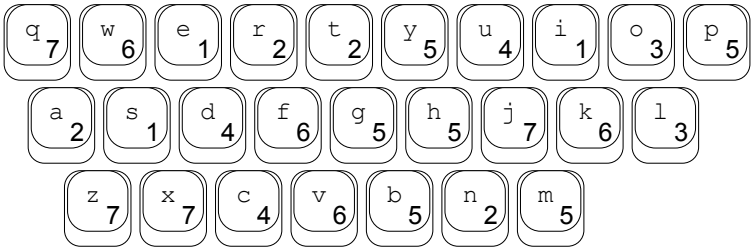
Сьомий та восьмий документи містять порожні форми для заповнення учасником олімпіади.

7.2 Завдання другого туру

7.2.1 «Словесна гра»

Словесні ігри популярні вдома та на телебаченні. У нашому варіанті гри кожна літера має ціну, і ви повинні скласти з літер одне чи більше слів, що дають максимальну сумарну вартість. До тих пір, доки це можливо, ви перебираєте усі відомі вам слова, іноді перевіряючи правопис, і обраховуючи їх вартості. Зрозуміло, що це краще робити за допомогою комп'ютера.

Дани ціни літер (див. мал. 1), список англійських слів і набір літер. Знайдіть у словнику слова або пари слів, з найбільшою сумарною вартістю, які можна скласти з заданого набору літер.



Вхідні дані Вхідний файл `INPUT.TXT` містить один рядок з маленькими латинськими літерами (від `a` до `z`) — допустимі літери. Рядок містить від трьох до сіми літер включно у довільному порядку. Файл-словник `WORDS.TXT` містить не більше ніж 40 000 слів. В кінці цього файлу знаходиться рядок з єдиним символом `.` (крапка). Кожний з наступних рядків містить слово, що складається від трьох до сіми маленьких латинських літер включно. Слова в файлі впорядковано за алфавітом.

Вихідні дані В перший рядок вихідного файлу `OUTPUT.TXT` ваша програма повинна записати найбільшу можливу сумарну вартість (підзадача A), а у наступних рядках — перелічити усі допустимі слова і/або пари слів з словника `WORDS.TXT` з такою вартістю (підзадача B). Кожне слово чи пару слів треба виводити виводіть у окремому рядку. Використовуйте ціни літер задані на мал. 1. Якщо з заданих літер можливо скласти пару, ці слова потрібно вивести, відокремивши їх пропуском. Не повторюйте пари; наприклад, `rag prom` і `prom rag` — одна пара. Вона повинна бути виведена один раз.

Примітка Файл `WORDS.TXT` не містить слів, що повторюються. Буква не може зустрічатися у обраному слові чи парі слів більшу кількість разів, ніж у файлі `INPUT.TXT`. Слово може утворювати пару саме з собою.

Приклад вхідних та вихідних даних

`WORDS.TXT`

```
profile
program
prom
rag
ram
rom
.
```

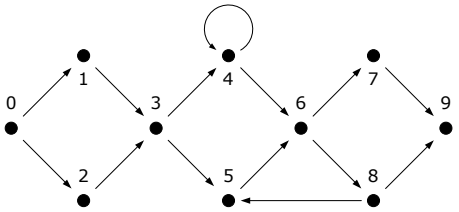
`INPUT.TXT`

```
prmgroa
```

`OUTPUT.TXT`

```
24
program
prom rag
```

7.2.2 «Вулична гонка»



На рисунку зображений приклад плану вулиць для гонки. Ви бачите точки, що помічені числами від 0 до N (де $N = 9$), а також стрілки, що сполучають їх. Точка 0 є стартовою; точка N — фінішною. Стрілками представлені вулиці з одностороннім рухом. Учасники гонки переміщуються від точки до точки по вулицям тільки у напрямку стрілок. В кожній точці учасник гонки може обрати будь-яку з стрілок, що виходять з неї.

Назвемо план вулиць гарним, якщо він має наступні властивості:

1. Кожна точка плану може буди досягнута з старту.
2. Фініш може бути досягнутий з будь-якої точки плану.
3. Немає стрілок, що виходять з фінішу.

Для досягнення фінішу учасник не зобов'язаний пройти через усі точки. Однак деякі точки неможливо обійти. Назвемо їх неминучими. В прикладі такими є точки 0, 3, 6, 9. Для заданого гарного плану ваша програма повинна визначити множину неминучих точок (за виключенням старту і фінішу), які повинні відвідати усі учасники (підзадача A). Припустимо, що гонка повинна проводитися на протязі двох послідовних днів. Для цієї цілі план повинен бути розбитий на два гарних плани, по одному на кожен день.

В перший день стартом є точка 0, а фінішем є деяка *точка розбиття*. У другий день старт знаходиться у цій точці розбиття, а фініш знаходиться у точці N. Для заданого гарного плану ваша програма повинна визначати множину усіх можливих точок розбиття (підзадача B). Точка S є точкою розбиття для гарного плану C, якщо S відрізняється від старту і фінішу C, і план розділяється на два гарних плани без загальних стрілок і з єдиною загальною точкою S. В прикладі точкою розбиття є точка 3.

Вхідні дані В файлі `INPUT.TXT` описаний гарний план, що містить не більше 50 точок, і не більше 100 стрілок. В файлі знаходиться $N + 1$ рядок. Перші N рядків містять кінцеві точки стрілок, що виходять відповідно з точок від 0 до $N - 1$. Кожний з цих рядків закінчується числом `-2`. В останньому рядку міститься число `-1`.

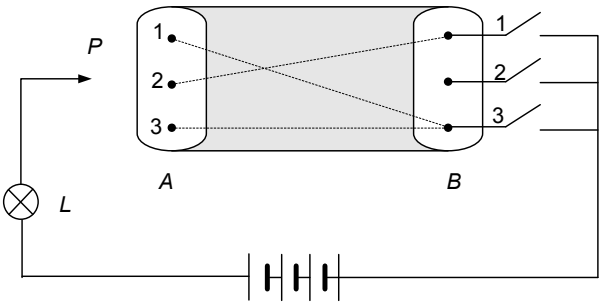
Вихідні дані Ваша програма повинна записувати два рядки в файл OUTPUT.TXT. Перший рядок повинен містити кількість неминучих точок в заданому плані, після чого в тому ж рядку повинні знаходитися номера цих точок у будь-якому порядку (підзадача A). Другий рядок повинен містити кількість точок розбиття в заданому плані, за яким в тому ж рядку повинні розміщуватися номера цих точок у будь-якому порядку (підзадача B).

Приклад вхідних та вихідних даних

INPUT.TXT
1 2 -2 3 -2 3 -2 5 4 -2 6 4 -2 6 -2 7 8 -2 9 -2 5 9 -2 -1

OUTPUT.TXT
2 3 6 1 3

7.2.3 «Дроти и вимикачи»



На мал. 1 зображено кабель, що містить 3 дроти, які сполучають кінці A та B. На кінці A знаходиться три дроти, що пронумеровані числами 1, 2 і 3. На кінці B дрота сполучені з занумерованими вимикачами: дроти 1 та 3 сполучаються з вимикачем 3, а дріт 2 з вимикачем 1.

В загальному випадку, кабель складається з m дротів ($1 \le m \le 90$), пронумерованих на кінці A числами від 1 до m. Кожний дрот сполучається рівно з одним вимикачем. К кожному вимикачу підходить нульова, або більша кількість дротів.

Виміри Напишіть програму, яка за допомогою декількох питань визначає, які дроти з якими вимикачами сполучені. Кожний вимикач може знаходитися в одному з двох станів: від або сполучає дрот (вимикач включений), або розриває (вимикач виключений). В початковий момент часу всі вимикачі виключені. В процесі вимірів провод може бути перевірений за допомогою щупа P: щуп сполучаєьться з дротом, що тестується, і лампочка L загорається в тому і тільки в тому разі, якщо цей дрот сполучений з одним з включених вимикачів на кінці B.

Програма повинна працювати у діалоговому режимі, при цьому введення і виведення здійснюються через стандартні пристрої (`standard input` і `standard output` відповідно), якими по замовченню є клавіатура та екран монітору. На початку програма зчитує один рядок, що містить кількість проводів m. Далі вона послідовно виводить команди, які бувають трьох видів.

Команда задається в окремому рядку і починаєть з імені команди — заглавної букви латинського алфавіту: `T` (`Test a wire` — перевірити дріт), `C` (`Change a switch` — змініти стан вимикача) і `D` (`Done` — кінець роботи). За командою `T` вказується номер дроту, за командою `C` — номер вимикача, за командою `D` — результат роботи програми — список з m номерів вимикачей, в якому i-ий елемент задає номер вимикача, до якого приєднується дріт з номером i.

Після видачі команд `T` або `C` програма повинна зчитати рядок, що містить відповідь на цей запит. На запит `T` видається відповідь `Y` (`Yes` — Так), якщо лампочка `L` загорілася, і відповідь `N` (`No` — Ні) у протилежному випадку. Запит `C` змінює стан відповідного вимикача на протилежний. На команду `C` видається відповідь `Y` (`Yes` — Так), якщо нове положення вимикача — «включений» , і відповідь `N` (`No` — Ні) якщо нове положення вимикача — «виключений». Відповідь на команду `C` видається тільки у якості підтверження. На команду `D` відповідь не видається.

Ваша програма повинна виводити послідовність з не більше ніж 900 команд, останньою з яких є команда `D`, що містить знайдену схему сполученнь.

Примітка Не підключайте до вашої програми модуль `CRT`!

Приклад вхідних та вихідних даних

Standard output
C 3 T 1 T 2 T 3 C 3 C 2 T 2 D 3 1 3

Standard input
3 Y Y N Y N Y N

Розділ 8. Угорщина'1996

8.1 Завдання першого туру

8.1.1 «Гра»

Розглянемо таку гру між двома гравцями. На гральній дошці записано послідовність додатніх цілих чисел. Гравці ходять по черзі. Хід полягає в тому, що гравець вибірає число, що розташоване на лівому, або на правому краю послідовності. Вибране число стирається з дошки. Гра закінчується, коли на дошці не залишиться чисел. Перший гравець виграє, якщо сума вибраних їм чисел не менша, ніж сума чисел вибраних другим гравцем. Другий гравець грає найкращим чином.

Перший гравець завжди починає гру.

Відомо, що якщо на початку гри на дошці записана парна кількість чисел, то у першого гравця є виграшна стратегія.

Завдання Треба написати програму, що реалізує виграшну стратегію першого гравця.

Відповіді другого гравця видаються заданою комп'ютерною програмою. Два гравця спілкуються через три процедури модуля PLAY, що є і у вашому розпоряджені. Ці процедури звуться StartGame (почати гру), MyMove (мій хід) та YourMove (твій хід). Перший гравець повинен ініціалізувати гру, викликавши процедуру StartGame без параметрів. Якщо перший гравець обирає число з лівого кінця, він викликає процедуру MyMove('L'). Подібним чином, викликаючи процедуру MyMove('R'), перший гравець надсилає другому повідомлення, яке значить що він вибирає число справа. Другий гравець, тобто комп'ютер, одразу відповідає і перший гравець може узнати його хід, викликавши процедуру YourMove(C), де C — змінна символьного типу (в мовах C та C++ ви повинні писати YourMove(&C)). Ця процедура визначає значення C, що дорівнює 'L' або 'R' в залежності від того, чи вибрав комп'ютер число ліворуч або праворуч.

Вхідні дані Перший рядок вхідних даних з іменем INPUT.TXT містить довжину початкової послідовності N (N — парне і $2 \leq N \leq 100$). Далі йдуть N рядків, в кохному з яких міститься одне число. Ці числа задають початкові значення, записані на дошці в порядку зліва направо. Числа на дошці не перевищують 200.

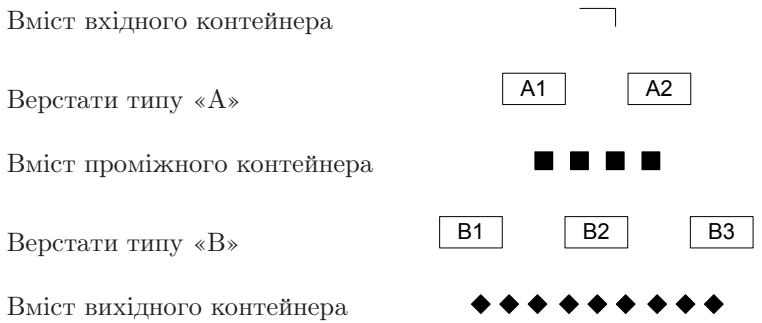
Вихідні дані Коли гра завершується, ваша програма повинна записати результат гри до файлу з іменем OUTPUT.TXT. Файл в першому рядку повинен містити два числа. Перше число дорівнює сумі чисел, вибраних першим гравцем, друге число — сумі чисел вибраних другим гравцем. Ваша програма обов'язково повинна зіграти з модулем PLAY, а вивід повинен відповідати зіграній грі.

Приклад вхідних та вихідних даних

INPUT.TXT
6 4 7 2 9 5 2

OUTPUT.TXT
15 14

8.1.2 «Роботи»



На фабриці діє виробнича лінія, що обробляє однакові деталі. Обробка кожної деталі вимагає двох операцій: спочатку виконується операція «A», а далі — операція «B». Є певна кількість верстатів для виконання кожної з операцій. На мал. 1 показано приклад організації виробничої лінії, що працює таким чином. Будь-який верстат, що виконує операцію «A», бере деяку деталь з вхідного контейнера, виконує операцію «B» та кладе деталь до вихідного контейнера. Всі верстати працюють паралельно та незалежно один від одного. Місткість кожного з контейнерів необмежена. Верстати мають різні часові характеристики обробки деталей — кожен верстат має свій час обробки.

Завдання Напишіть програму, що обчислює найменьший час закінчення операції «A» над всіма N деталями за умови, що час початку обробки дорівнює 0 (підзадача A).

Треба також обчислити мінімальний час закінчення обох операцій «A» та «B» над усіма N деталями (підзадача B).

Вхідні дані Файл `INPUT.TXT` складається з п'яти рядків і містить цілі додатні числа. В першому рядку знаходиться число N — загальна кількість деталей ($1 \le N \le 1\,000$). У другому рядку знаходиться число M_1 — загальна кількість верстатів, що виконують операцію «A» ($1 \le M_1 \le 30$). У третьому рядку містяться M_1 цілих чисел, що визначають часи обробки деталі відповідними верстатами. В четвертому та п'ятому рядках знаходяться відповідно число M_2 — загальна кількість верстатів, що виконують операцію «B» ($1 \le M_2 \le 30$), та M_2 цілих чисел, що визначають часи обробки деталі відповідними верстатами. Час обробки деталі на кожному верстаті включає час необхідний для переміщення деталі з контейнера до верстата і з верстата до контейнера. Час обробки для кожного верстата задається цілим числом від 1 до 20.

Вихідні дані Ваша програма повинна записати два рядка до файлу `OUTPUT.TXT`. Перший рядок повинен містити одне додатнє ціле число — розв'язок підзадачі «A». Другий рядок повинен містити розв'язок підзадачі «B».

Приклад вхідних та вихідних даних

INPUT.TXT
5 2 1 1 3 3 1 4

OUTPUT.TXT
3 5

8.1.3 «Мережа шкіл»

Певна кількість шкіл об'єднана комп'ютерною мережею. Між цими школами укладено угоди. Кожна школа має список шкіл–отримува-чів, яким вона розсилає програмне забезпечення щоразу, коли отримує безкоштовне програмне забезпечення з іншої школи або зовні мережі. При цьому, якщо школа B є в списку отримувачів школи A, школа A може і не бути в списку отримувачів школи B.

Завдання Треба написати програму, що обчислює мінімальну кількість шкіл, яким треба передати по копії нового програмного забезпечення, щоб розповсюдити його по всіх школах мережі згідно до угод (підзадача A).

Крім того, треба забезпечити можливість розсилки нового програмного забезпечення з будь-якої школи по всіх інших школах. Для цього можна розширювати списки розсилки отримувачів деяких шкіл, додаючи до них нові школи. Обчислить мінімальну сумарну кількість розширень списків, при яких програмне забезпечення з будь-якої школи досягло б усіх інших шкіл (підзадача B). Одне розширення відповідає додаванню однієї нової школи-отримувача до списку отримувачів однієї з шкіл.

Вхідні дані Перший рядок файлу `INPUT.TXT` містить ціле число N — кількість шкіл в мережі ($2 \le N \le 100$). Школи нумеруються першими N додатніми цілими числами. Кожен з наступних N рядків задає список отримувачів. $i+1$-й рядок містить номери отримувачів i-ї школи. Кожнен такий список закінчується нулем. Порожній список містить тільки `0`.

Вихідні дані Ваша програма повинна записати два рядка до файлу `OUTPUT.TXT`. Його перший рядок повинен містити одне додатнє ціле число — розв'язок підзадачі A. Другий рядок повинен містити розв'язок підзадачі B.

Приклад вхідних та вихідних даних

INPUT.TXT	OUTPUT.TXT
5 2 4 3 0 4 5 0 0 0 1 0	1 2

8.2 Завдання другого туру

8.2.1 «Сортування-3»

Сортування — це одна з найпоширеніших обчислювальних задач. Розглянемо спеціальну задачу сортування, де записи, що повинні бути відсортовані, можуть мати не більше трьох різних значень ключа. Це буває, наприклад, тоді, коли ми впорядковуємо медалістів змагань згідно з рангами медалей, коли першими йдуть золоті призери, слідом за ними — срібні, а останніми — бронзові.

В цій задачі три можливі значення ключа — цілі числа `1`, `2` і `3`. Необхідний порядок сортування — за неспаданням. Сортування повинно бути виконано послідовністю попарних перестановок записів. Операція попарної перестановки, що задається двома номерами позицій p і q, переставляє елементи послідовності, що знаходяться в позиціях p і q. Вам дано послідовність значень ключів.

Завдання Напишіть програму, що визначає мінімальну кількість операцій попарної перестановки, необхідних для того, щоб відсортувати послідовність (підзадача A). Крім того, побудуйте послідовність операцій попарної перестановки для відповідного сортування (підзадача B).

Вхідні дані Перший рядок файлу `INPUT.TXT` містить кількість записів N ($1 \le N \le 1\ 000$). Кожен з наступних N рядків — запис, що містить значення ключа.

Вихідні дані Запишіть в першому рядку файлу `OUTPUT.TXT` мінімальну кількість L операцій попарної перестановки, необхідну для сортування послідовності (підзадача A). Наступні L рядків повинні містити відповідну послідовність операцій попарної перестановки в порядку їх виконання. Кожен рядок містить одну операцію попарної перестановки, задану двома числами p і q — позиціями двох елементів, що переставляються (підзадача B). Позиції позначаються числами від 1 до N.

Приклад вхідних та вихідних даних

INPUT.TXT
9 2 2 1 3 3 3 2 3 1

OUTPUT.TXT
4 1 3 4 7 9 2 5 9

8.2.2 «Найдовший префікс»

Структура деяких біологічних об'єктів представляється послідовністю їх складових. Ці складові позначаються великими літерами. Біологи цікавляться розкладенням довгої послідовності на коротші. Ці короткі послідовності називаються примітивами. Кажуть, що послідовність S може бути утворена з даної множини примітивів P, якщо існують n примітивів $p_1, \ldots, p_n$ в P, таких, що їх конкатенація (зчеплення) $p_1, \ldots, p_n$ дорівнює S. При конкатенації примітиви $p_1, \ldots, p_n$ записуються послідовно без пропусків. Деяки примітиви можуть зустрічатися у конкатенації понад один раз, і не обов'язково всі примітиви повинні бути використані.

Наприклад, послідовність `ABABACABAAB` може бути утворена з множини примітивів {`A`, `AB`, `BA`, `CA`, `BBC`}.

Перші K символів рядка S будемо називати префіксом S довжини K.

Завдання Напишіть программу, яка для заданих множини примітивів P та послідовності T визначає довжину максимального префікса послідовності T, який може утворюватися з множини примітивів P.

Вхідні дані Вхідні дані знаходяться у двох файлах. Текстовий файл `INPUT.TXT` визначає множину примітивів P, а файл `DATA.TXT` містить послідовність T, яка потребує перевірки. Перший рядок файлу `INPUT.TXT` містить число N — кількість примітивів у множині P ($1 \le N \le 100$). Кожен примітив задано в двох послідовних рядках. Перший з них містить довжину L примітива ($1 \le L \le 20$). Другий містить рядок великих літер (від `A` до `Z`) довжиною L. Всі N примітивів різні. Кожен рядок файлу `DATA.TXT` містить велику літеру у першій позиції. Файл `DATA.TXT` закінчується рядком з символом `.` (крапка) в першій позиції. Довжина послідовності — як мінімум 1 і як максимум 500 000.

Вихідні дані Записати в перший рядок вихідного текстового файлу `OUTPUT.TXT` довжину максимального префікса послідовності T, який може утворитися з множини P.

Приклад вхідних та вихідних даних

INPUT.TXT
5 1 A 2 AB 3 BBC 2 CA 2 BA

DATA.TXT
A B A B A C A B A A B C B .

OUTPUT.TXT
11

8.2.3 «Магічні квадратики»

Після успіху головоломки «Кубик Рубіка», пан Рубік вигадав її плоский варіант «Магічні квадратики». Головоломка складається з 8 однакових квадратиків (див. мал. 1). Ці квадратики розфарбовані у 8 різних кольорів. Кольори позначаються цілими числами від 1 до 8, як показано на малюнку. Стан головоломки позначається послідовністю кольорів квадратиків, заданих за стрілкою годинника, починаючи з лівого верхнього квадрату, і закінчуючи правим нижнім. Наприклад, стан на мал. 1 визначається послідовністю (1, 2, 3, 4, 5, 6, 7, 8). Цей стан завжди є початковим.

Існують три базових перетворення головоломки. Ці перетворення позначаються літерами `A`, `B` та `C`.

- `A` — обмін верхнього та нижнього рядків прямокутника;
- `B` — один циклічний зсув прямокутника праворуч;
- `C` — поворот за стрілкою годинника чотирьох середніх квадратиків на 90°.

Відомо, що використовуючи ці базові перетворення можна отримати будь-яку задану кінцеву конфігурацію.

Результати базових перетворень початкового стану зображено на мал. 2. Числа, розташовані біля квадратиків, позначають положення квадратиків. Якщо квадратик в положенні p містить число (колір) і, це означає, що

в результаті відповідного перетворення початкового стану той квадратик, який знаходився спочатку в положенні i (і має колір i), пересувається в положення p.

Завдання Напишіть програму, що знаходить послідовність базових перетворень, що переводить початковий стан (див. мал. 1) в заданий кінцевий стан (підзадача A).

Два додаткових бали за кожен тест будуть присуджуватися в тому разі, якщо довжина послідовності базових перетворень не перевищіть 300 (підзадача B).

Вхідні дані Файл вхідних даних `INPUT.TXT` містить 8 чисел, які знаходяться в першому рядку файла. Ці числа задають кінцевий стан головоломки.

Вихідні дані В першому рядку файла `OUTPUT.TXT` повинне міститися число L — довжина послідовності базових перетворень. Наступні L рядків повинні містити задану послідовність перетворень. В кожному рядку в першій позиції повинна міститися одна з літер `A`, `B` або `C`, що позначає відповідне перетворення.

Додаткові засоби В ваше розпорядження надається програма `MTOOL.EXE`, яка міститься в каталозі задачі і дозволяє вам «пограти» з магічними квадратиками. Програма `MTOOL.EXE` має такий формат виклика `MTOOL.EXE INPUT.TXT OU` де формати файлів `INPUT.TXT` і `OUTPUT.TXT` повинні відповідати наведеним вимогам. За допомогою цієї програми ви зможете експерементувати з кінцевим станом і послідовністю перетворень.

	1	2	3	4
A	8	7	6	5
	1	2	3	4

	1	2	3	4
B	4	1	2	3
	5	8	7	6

	1	2	3	4
C	1	7	2	4
	8	6	3	5

Приклад вхідних та вихідних даних

INPUT.TXT
2 6 8 4 5 7 3 1

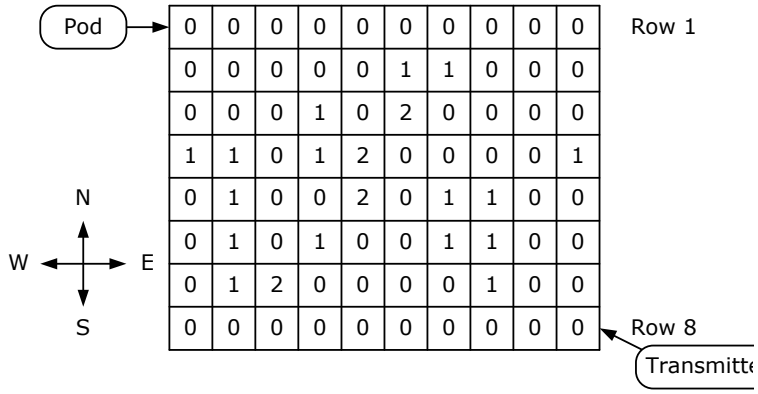
OUTPUT.TXT
7
B
C
A
B
C
C
B

Розділ 9. ПАР'1997

9.1 Завдання першого туру

9.1.1 «Марсохід»

У майбутній експедиції на поверхню Марсу виконає посадку спеціальний *апарат*. Апарат містить марсоходи, що будуть рухатися до *модуля*, який приземлився неподалеку. Модуль буде повернуто на Землю по закінченню експедиції. На шляху до модуля марсоходи збиратимуть зразки гірських порід. Зразок може бути підібран лише один раз тим марсоходом, який вперше відвідав цей участок поверхні. Після цього зразок не може бути знову взятий з цього участку, але інші марсоходи можуть проходити по цьому участку.



Марсоходи не можуть рухатися по нерівним участкам поверхні.

Марсохід сконструйований так, що він може рухатися лише на південь або на схід на шляху від апарату до повертаємого модулю. На одному участку одночасно можуть знаходитися декілька марсоходів.

Зверніть увагу, що якщо марсохід не може досягнути модуля, усі зібрані їм зразки втрачаються назавжди.

Завдання Визначити індивідуальні траєкторії руху марсоходів, щоб максимізувати свої бали, які залежать від кількості зібраних та доставлених зразків і кількості марсоходів, що досягли модуля.

Вхідні дані Поверхня планети між містом посадки апарату і місцезнаходженням модуля задається у вигляді прямокутної сітки розміру $P \times Q$. Місце посадки апарату знаходиться в позиції $(1, 1)$, а місце модуля — в позиції (P, Q). Тип участка поверхні визначається наступним чином:

Рівний участок: `0`
Нерівний участок: `1`
Участок з зразком: `2`

Структура вхідного файлу:

КількістьМарсоходів
P
Q
$(X_1Y_1)(X_2Y_1)(X_3Y_1)\ldots(X_{P-1}Y_1)(X_PY_1)$
$(X_1Y_2)(X_2Y_2)(X_3Y_2)\ldots(X_{P-1}Y_2)(X_PY_2)$
$(X_1Y_3)(X_2Y_3)(X_3Y_3)\ldots(X_{P-1}Y_3)(X_PY_3)$
$\ldots$
$(X_1Y_{Q-1})(X_2Y_{Q-1})(X_3Y_{Q-1})\ldots(X_{P-1}Y_{Q-1})(X_PY_{Q-1})$
$(X_1Y_Q)(X_2Y_Q)(X_3Y_Q)\ldots(X_{P-1}Y_Q)(X_PY_Q)$

де P і Q — розміри сітки ($P \le 128$, $Q \le 128$), КількістьМарсоходів — ціле число, яке $< 1\,000$, що дорівнює кількості марсоходів, які були спущені на поверхню.

Вихідні дані Вихідний файл задається у вигляді послідовності рядків, що визначають рух марсоходів до місцезнаходження модуля. Кожен рядок містить номер марсохода, то числа `0` або `1`, де `0` означає рух на південь, а `1` — на схід.

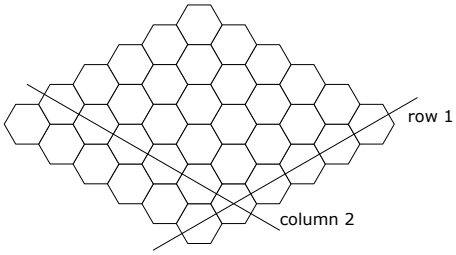
Оцінки Бали підраховуються на основі кількості зібраних та доставлених зразків, з врахуванням кількості марсоходів, які досягли, або не досягли модуля.

- За розв'язок з неправильним ходом нараховується 0 балів. Неправильним вважається хід, коли марсоход рухається по нерівному участку, або коли він виходить за границі сітки.
- Бали = (кількість зібраних і доставлених до модуля зразків + кількість досягнувших модуля марсоходів − кількість марсоходів, що не досягнули модуля) як % максимально можливого балу розв'язку.
- Можна отримати максимальну оцінку 100% і мінімальну — 0%.

9.1.2 «Гра гекс»

Мета гри для першого гравця — з'єднати своїми фішками стовпчик з номером 1 з стовпчиком з номером N.

Правила гекса Гекс — стратегічна гра двох гравців на ромбовидній дошці, що складена з $N \times N$ шестикутників, як зображено на малюнку для $N = 6$.



1. Два гравці — це ваша програма та бібліотека журі.
2. Ваша програма завжди робить перший хід.
3. Гравці по черзі встановлюють фішки на клітини дошки.
4. Фішку можна ставити на будь-яку вільну клітину.
5. Два шестикутника суміжні, якщо у них є спільна сторона (ребро).
6. Фішки одного гравця (как для учасника, так і для бібліотеки журі), що стоять на суміжних шестикутниках, зв'язані.
7. Зв'язність транзитивна (і комутативна), тобто якщо фішка hex_1 зв'язана з фішкою hex_2, і фішка hex_2 зв'язана з фішкою hex_3, то hex_3 зв'язана з hex_1, і навпаки: hex_1 зв'язана з hex_3.

Завдання

- Потрібно написати програму, що грає у гекс.
- Мета першого гравця (вашої програми) — зв'язати своїми фішками стовпчик (column) з номером 1 з стовпчиком з номером N.
- Другий гравець (програма журі) намагається зв'язати своїми фішками рядок (row) з номером 1 з рядком з номером N.
- Якщо ваша програма грає оптимально, то вона завжди виграє.

Вхід та вихід Ваша програма не повинна читати та записувати ніякі файли. Ваша програма не повинна вводити дані з клавіатури і виводити дані на екран. Вона буде отримувати дані як значення функцій з бібліотеки журі (hex library). Бібліотека сама створить файл `HEX.OUT`; ви повинні ігнорувати його вміст.

На початку гри ваша програма отримує гральну дошку, на якій, можливо, вже розміщені фішки, причому таким чином, як це могло трапитися в ході гри; в цій позиції перший гравець може досягнути виграшу. Для того, щоб визначити розміри дошки та положення фішек на ній, ваша програма може використовувати функції бібліотеки журі GetMax (получити максимум) та LookAtBoard (подивитися на дошку).

На початку гри на дошці знаходиться однакова кількість фішек вашої програми і бібліотеки журі.

Обмеження

1. Розмір дошки завжди знаходиться у межах від 1 до 20 включно.
2. Ваша програма повинна зробити до 200 ходів до завершення гри. Уся гра повинна завершитися за 40 секунд. Гарантується, що бібліотека журі витрачає на усі свої ходи в сумі не більше 20 секунд.

Бібліотека Ви повинні підключити до своєї програми бібліотеку з назвою HexLib. Приклади бібліотек для кожної мови програмування, які демонструють як працює бібліотека знаходяться в каталозі задачі. Ці файли мають назви `TESTHEX.CPP`, `TESTHEX.C`, `TESTHEX.PAS` і `TESTHEX.BAS`. Якщо ви використовуєте `QuickBasic` ви повинні підключити бібліотеку так:

```
QB /L HEXLIB
```

Бібліотека HexLib містить наступні функції і процедури (відповідно для `Pascal`, `C`/`C++` і `Basic`):

```
function LookAtBoard (row, column: integer): integer;
int LookAtBoard (int row, int column);
declare function LookAtBoard cdecl (byval x as integer,
                                    byval y as integer)
```

Ця функція приймає наступні значення:

-1, якщо $row < 1$ або $row > N$ або $column < 1$ або $column > N$;
0, якщо на заданому полі немає ніякої фішки;
1, якщо на цьому полі знаходиться фішка вашої програми (1-го гравця);
2, якщо на цьому полі знаходиться фішка, що належить бібліотеці журі (2-го гравця);

```
procedure PutHex (row, column: integer);
void PutHex (int row, int column);
declare sub PutHex cdecl (byval x as integer, byval y as integer)
```

Ця процедура встановлює фішку вашої програми в поле з заданими номерами рядка та стовпчика, якщо це поле вільне.

```
function GameIsOver: integer;
int GameIsOver (void);
declare function GameIsOver cdecl ()
```

Ця функція приймає одне з наступних цілочесельних значень:

0, гра ще не закінчена.
1, всі поля дошки зайняті фішками.
2, ваша програма перемогла.
3, бібліотека журі перемогла.

```
procedure MakeLibMove;
void MakeLibMove(void);
declare sub MakeLibMove cdecl ()
```

Ця процедура дозволяє бібліотеці журі визначити її наступний хід, та помістити його на дошці. Зміну позиції на дошці можна визначити за допомогою LookAtBoard та інших функцій.

```
function GetRow: integer;
int GetRow (void);
declare function GetRow cdecl ()
```

Ця функція приймає значення номера рядка останньої фішки, яку розмістила бібліотека журі, або -1, якщо бібліотека фішку не встановлювала. Ця функція при послідовних викликах повертає одне і теж саме значення, доки ваша програма ще раз не викличе процедуру MakeLibMove.

```
function GetColumn: integer;
int GetColumn (void);
declare function GetColumn cdecl ()
```

Ця функція приймає значення номера стовпчика останньої фішки, яку розмістила бібліотека журі, або `-1`, якщо бібліотека фішку не встановлювала. Ця функція при послідовних викликах повертає одне і теж саме значення, доки ваша програма ще раз не викличе процедуру MakeLibMove.

```
function GetMax: integer;
int GetMax (void);
declare function GetMax cdecl ()
```

Значення цієї функції завжди рівно N — розміру дошки.

Оцінки

- Якщо ваша програма виграла гру, вона отримує повний бал за тест.
- Якщо ваша програма програла гра, вона отримує 20% від повного балу за тест.

  Если Ваша программа проиграла игру, ей ставится 20
- Якщо ваша програма припиняє роботу, не дограв ши до кінця, або не вкладається у відведений час, вона отримує 0 балів за тест.

9.1.3 «Ненажерлива Шонгололо»

Шонгололо — зулуська назва багатоножки. Вона має довге гладке чорне тіло з великою кількістю ніг.

Шонгололо проїдає їстівний фрукт, який для даної задачі може розглядатися як прямокутний паралелепіпед з цілочисельними розмірами: L (довжина), W (ширина) і H (висота).

Завдання Треба написати програму, яка максимізує число блоків одиничного розміру, з'їдених Шонгололо без порушення заданих обмежень. Програма повинна виводити дії, які Шонгололо здійснює в процесі поїдання блоків.

Шонгололо починає процес поїдання за межами фрукта. Перший блок, який повинна з'їсти Шонгололо, має координати $(1, 1, 1)$ і після цього вона пересувається в цей блок. Вона зупиняється, коли більше немає блоків, які можна з'їсти, і блоків, в які можна пересунутися.

Обмеження

1. Шонгололо займає в процесі поїдання рівно один порожній блок.
2. За один раз Шонгололо може з'їсти тільки один заповнений блок.
3. Шонгололо не може пересуватися у позицію, де вона вже була раніше (тобто рухатися назад або перетинати свій путь).
4. Шонгололо не може пересунутися в блок, що не був з'їденим, або за межі фрукту.
5. Шонгололо може пересуватися в блоки, або з'їдати блоки, що мають загальну грань з тим блоком, де зараз знаходиться Шонгололо. Вона може поїдати тільки ті блоки, що не мають загальних граней з порожними (вже з'їденими) блоками, не враховуючи блок, в якому вона знаходиться у поточний момент.

Вхідні дані Вхідними даними для програми є три цілих числа, які задають довжину (L), ширину (W) та висоту (H) фрукта.

Кожне з трьох цілих чисел записане у окремому рядку. Числа знаходяться в інтервалі від 1 до 32 включно.

Вихідні дані Вихідний файл повинен містити рядки, які починаються з літери `E` (поїдати), або з літери `M` (пересуватися). За літерою повинні слідувати три цілих числа, які задають координати блоку, який Шонгололо їсть, або в який пересувається.

Оцінки

- Якщо Шонгололо порушує задані вимоги, ваш розв'язок отримує 0 балів.
- Загальна кількість балів — відсоткове відношення кількості з'їдених блоків у вашому розв'язку до числа блоків у розв'язку журі.
- Ітогова оцінка не може перевищувати 100%.

Приклад вхідних та вихідних даних

TOXIC.DAT
2 3 2

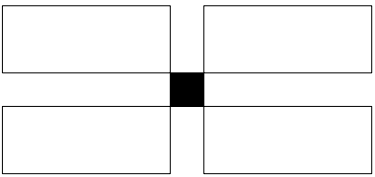
TOXIC.OUT
E 1 1 1
M 1 1 1
E 2 1 1
E 1 1 2
E 1 2 1
M 1 2 1
E 1 3 1
M 1 3 1
E 2 3 1
E 1 3 2
M 1 3 2

9.2 Завдання другого туру

9.2.1 «Розмітка карт»

Уявіть собі, що ви — допомічник картографа. Вам дано важке завдання по нанесенню назв міст на нову карту.

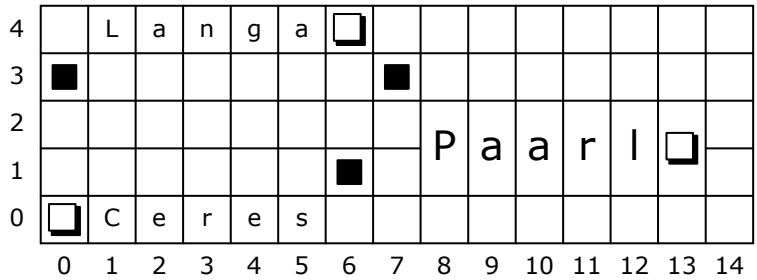
Карту можна уявити у вигляді сітки, що складається з 1000×1000 комірок. Місцезнаходження міста визначається однією коміркою на карті. Назви міст повинні бути розміщені на карті в прямокутниках, що складаються з комірок. Такі прямокутники називаються *покажиками*.



Розміщення покажчиків повинно задовольняти наступним обмеженням:

1. Покажчик міста повинен знаходитися в одій з чотирьох позицій відносно позиції міста, як зображено на мал. 1.
2. Покажчики не повинні перекривати один одного.
3. Покажчики не повинні перекривати позиції міст.
4. Покажчики не повинні виходити за межі карти.

Кожен покажчик містить усі букви назви міста і один пропуск. Для кожної назви міста ширина та висота його літер задана; пропуск має такі ж самі розміри, як і літера.



❑ представляє пропуск

■ представляє позицію міста

Найлівіший стовпчик карти має номер 0 по горизонталі, а найнижчий рядок карти має номер 0 по вертикалі. На мал. 2 зображено нижню ліву частину карти для міст `Langa` в позиції $(0, 3)$, `Ceres` в позиції $(6, 1)$ і `Paarl` в позиції $(7, 3)$. Усі покажчики розміщені коректно, але існують і інші правильні розміщення.

Завдання Ваша програма повинна для кожного міста ввести його координати на карті, розміри букв назви міста, та саму назву. Після цього ваша програма повинна розмістити на карті якомога більше покажчиків без порушення вищезазначених обмежень і вивести положення покажчиків, які розміщені на карті.

Вхідні дані Вхідний файл починається з рядка, що містить ціле число N, яке задає кількість міст на карті. Після цього для кожного міста слідує рядок, що містить:

- координату міста по горизонталі (X),
- координату міста по вертикалі (Y),
- ширину (W) кожної літери у назві міста у вигляді цілого числа,
- висоту (H) кожної літери у назві міста у вигляді цілого числа,
- саму назву міста.

Назва міста складається з одного слова. Кількість міст не перевищує 1 000. Довжина назви міста не перевищує 200 літер.

Вихідні дані Ваша програма повинна вивести N рядків. Кожен рядок повинен містити координату по горизонталі і координату по вертикалі верхньої лівої комірки покажчика міста. Якщо ваша програма не може розмістити покажчик на карті, то вона повинна вивести `-1 -1` у відповідному рядку.

Ці рядки повинні бути виведені в тому ж порядку, в якому задані міста у вхідному файлі. Два числа в рядку повинні бути розділені одним пропуском.

Оцінки Для набора даних кожного тесту кількість балів буде наступною:

- Відсоткове відношення кількості назв міст, які розмістить на карті ваша програма, до кількості назв міст у розв'язку наукового комітету.
- Мінімальна кількість балів рівна 0%, максимальна — 100%.
- 0 балів, якщо будь-який покажчик порушує обмеження.
- 0 балів, якщо покажчики не відповідають заданим містам.

Приклад вхідних та вихідних даних

MAPS.DAT
3 0 3 1 1 Langa 6 1 1 1 Ceres 7 3 1 2 Paarl

MAPS.OUT
1 4 0 0 8 2

9.2.2 «Розпізнання символів»

Завдання полягає у написанні програми, що розпізнає символи.

Уточнения Образ кожного символа складається з 20 рядків по 20 цифр у кожному. Значення цифр — `0` або `1`. На мал. 1а зображено спосіб представлення образів символів у файлі.

В файлі `FONT.DAT` зберігається образи 27 символів у наступному порядку:

```
?abcdefghijklmnopqrstuvwxyz
```

де ? відповідає символу пропуска.

Файл `IMAGE.DAT` містить один чи декілька образів символів, можливо, спотворених. Образ символа може бути спотворений наступним чином:

- Максимум один рядок може бути продубльований, при цьому дубль розміщується за вихідним рядком.

- Максимум один рядок може бути пропущений.
- Деяки `0` можуть бути замінені на `1`.
- Деяки `1` можуть бути замінені на `0`.

Жоден образ символа не може бути спотворений так, щоб одночасно один рядок був продубльований і один пропущений.

Ніякий образ символа не може бути спотворений більш ніж на 30% замінами `0` та `1` у порівнянні з його вихідним представленням.

У випадку дублювання рядку, він та його дубль можуть після цього бути спотворені, при чому по-різному.

Завдання Написати програму для розпізнання послідовності з одного або більше символів, представлених у файлі `IMAGE.DAT`, використовуючи образи символів, які задано у файлі `FONT.DAT`.

Розпізнайте образ символу, обрав такий вихідний образ, для отримання якого потрібна мінімальна загальна кількість змін `0` та `1` у спотвореному образі, враховуючи найкраще припущення щодо дублювання або пропуску рядка. У випадку дублювання рядка підраховуйте заміни тільки в меньш спотвореному з двох результуючих рядків. Усі образи симолів, що використовуються при тестуванні, гарно написана програма зможе розпізнати.

Для кожного набору тестових даних існує лише один оптимальний розв'язок.

Правильний розв'язок буде використовувати усі дані, представлені у вхідному файлі `IMAGE.DAT`.

Вхідні дані Обидва вхідні файли починаються з цілого числа N ($19 \leq N \leq 1\ 200$), яке задає кількість рядків, що слідують далі, як це зображено на прикладі.

$$\begin{array}{|l|}\hline N \\ (\text{цифра}_1)(\text{цифра}_2)(\text{цифра}_3)\ldots(\text{цифра}_{20}) \\ (\text{цифра}_1)(\text{цифра}_2)(\text{цифра}_3)\ldots(\text{цифра}_{20}) \\ \vdots \\ \hline \end{array}$$

Кожен рядок містить рівно 20 символів без пропусків.

Файл `FONT.DAT` описує образи символів і завжди містить рівно 541 рядок. Файл `FONT.DAT` може відрізнятися для різних тестів.

Вихідні дані Ваша програма повинна створити файл `IMAGE.OUT`, та записато до нього рядок розпізнаних символів у форматі `ASCII` без роздільників. Якщо ваша програма не може розпізнати окремой символ, вона повинна вивести символ `?` у відповідній позиції.

Увага Наведений вище вихідний формат не відповідає загальним вихідним форматам, описаним у правилах проведення олімпіади.

Оцінки Бали обчислюються у відповідності з відсотком коректно розпізнаних образів символів.

Приклад вхідних та вихідних даних

Початок `FONT.DAT`

```
540
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000011100000000000
00000111111011000000
00001111111001100000
00001110001100100000
00001100001100010000
00001100000100010000
00000100000100010000
00000010000000110000
00000010000001110000
00001111111111110000
00001111111111110000
00001111111111000000
00001000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

`IMAGE.DAT`

```
19
00000000000000000000
00000000000000000000
00000000000000000000
00000011100000000000
00100111011011000000
00001111111001100000
00001110001100100000
00001100001100010000
00001100000100010000
00000100000100010000
00000010000000110000
00001111011111110000
00001111111111110000
00001111111111000000
00001000010000000000
00000000000000000000
00000000000001000000
00000000000000000000
00000000000000000000
```

`IMAGE.OUT`

```
a
```

9.2.3 «Складування контейнерів»

Компанія транспортних перевезень «Нептун» володіє складом контейнерів. Контейнери приймаються на склад, зберігаються там певний період часу, та відправляються зі складу.

Контейнери надходять на склад для зберігання кожну годину. Термін збереження на складі — ціла додатня кількість годин. Коли контейнер надходить, в його документах вказано очікуваний час відправки. Перший контейнер надходить у час рівний 1. У дійсності може трапитися, що контейнер необхідно буде відправити раніше чи пізніше зазначеного у документах часу в межах ± 5 годин.

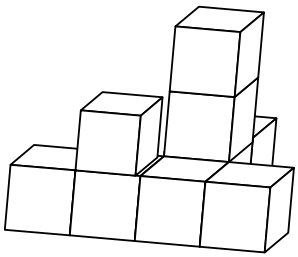
В цій задачі час представлено у годинах цілими додатніми числами, що не перевищують 150.

Кран, що працює у приміщенні складу, пересуває контейнери до складу (під час прийому), зі складу (під час відправки), а іноді переставляє їх всередені складу. Кран може працювати тільки у межах складу.

Завдання Вам треба написати програму з гарною стратегією прийому, зберігання та відправки контейнерів. Гарна стратегія мінімізує загальну кількість пересувань крану.

Склад має форму прямокутного паралелепіпеда. Його довжина (X), ширина (Y) та висота (Z) відомі вашій програмі.

Кожен контейнер являє собою куб розміром $1 \times 1 \times 1$. Контейнери складаються в стопки (стеки) один на одного (див. мал.). Кран може брати лише верхній контейнер з будь-якої стопки.



Пересування контейнера з одного місця до іншого — це одне пересування крана. Всі пересування крана відбуваються між прийомом та відправленням контейнерів. Пересування контейнера краном з одного місця до іншого відбувається миттєво.

Якщо склад заповнений, ваша програма повинна відмовитися приймати нові контейнери. Ваша програма може виявитися меньш ефективною або

неспроможною продовжувати роботу, якщо склад майже повністю заповнений. Ваша програма може відмовитися приймати контейнери у будь-який момент.

Вхідні дані Від вашої програми вимагається взаємодіяти з бібліотекою, яка буде надавати дані, і якій ваша програма буде повідомляти свої дії та передавати повідомлення. На початку роботи склад порожній.

При тестуванні вашої програми бібліотека буде певертати змістовні (розумні) значення для невеликого набору тестових даних.

Кожному контейнеру відповідає унікальне додатнє ціле число.

Ваша програма в будь-який час може викликати наступні функції.

```
int GetX();
function GetX: integer;
DECLARE FUNCTION GetX CDECL ()
```

Завжди приймає значення довжини складу (integer).

```
int GetY();
function GetY: integer;
DECLARE FUNCTION GetY CDECL ()
```

Завжди приймає значення ширини складу (integer).

```
int GetZ();
function GetZ: integer;
DECLARE FUNCTION GetZ CDECL ()
```

Завжди приймає значення висоти складу (integer).

X, Y, Z не перевищують 32.

Далі описані функції, які дають інформацію о послідовності дій (прибуття та відправка контейнерів). Приймання контейнера проходить точно у деяку годину, а запити на відправку надходять у проміжках між годинами. Таким чином, для обліку часу кожне прибуття контейнеру означає, що минула одна година.

```
int GetNextContainer();
function GetNextContainer: integer;
DECLARE FUNCTION GetNextContainer CDECL ()
```

Приймає додатнє ціле значення — номер наступного контейнера, який треба прийняти на збереження, або відправити. Якщо контейнерів більше немає як для прийому так і для відправки, функція приймає значення `0`, що означає, що ваша програма повинна закінчити роботу, навіть якщо на складі є контейнери.

```
int GetNextAction();
function GetNextAction: integer;
DECLARE FUNCTION GetNextAction CDECL ()
```

Приймає додатнє значення, що представляє дію, яку необхідно виконати: 1 — прийняти новий контейнер, 2 — відправити контейнер.

```
int GetNextStorageTime();
function GetNextStorageTime: integer;
DECLARE FUNCTION GetNextStorageTime CDECL ()
```

Приймає значення очікуваного часу у годинах (абсолютний час), коли контейнер необхідно відправити. Це потрібно для того, щоб ваша програма планувала свою роботу; у дійсності запит може прийти і в інший час, який може відрізнятися від плануємого не більше ніж на 5 годин. Ця функція приймає змістовне значення тільки якщо GetNextAction рівна 1.

Порядок визову цих трьох функцій не має значення.

Послідовні виклики GetNextContainer, GetNextAction і GetNextStorageTime завжди будуть повертати інформацію про один і той самий контейнер, доки не трапиться одна з наступних подій:

- черговий контейнер буде відхилений,
- прийнятий на зберігання,
- відправлений.

Після цього вказані функції будуть повідомляти інформацію про наступний контейнер.

Вихідні дані Кожен раз, коли ваша програма отримала необхідну їй інформацію про наступний контейнер, використовуйте наступні функції для проведення операцій на складі.

```
int MoveContainer(int x1, int y1, int x2, int y2);
function MoveContainer(x1, y1, x2, y2: integer): integer;
DECLARE FUNCTION MoveContainer CDECL (BYVAL x1 AS INTEGER,
                                      BYVAL y1 AS INTEGER,
                                      BYVAL x2 AS INTEGER,
                                      BYVAL y2 AS INTEGER)
```

Пересуває один контейнер зверху стопки з координатами (x_1, y_1) на верх стопки з координатами (x_2, y_2). Приймає значення 1, якщо операція виконана правильно, і 0, якщо операція виконана неправильно (тобто проведення операції неможливе).

```
void RefuseContainer();
procedure RefuseContainer;
DECLARE SUB RefuseContainer CDECL ()
```

Відмовляє у прийомі контейнера, що надійшов до складу.

```
void StoreArrivingContainer(int x, int y);
procedure StoreArrivingContainer(x, y: integer);
DECLARE SUB StoreArrivingContainer CDECL (BYVAL x AS INTEGER,
                                          BYVAL y AS INTEGER)
```

Приймає контейнер, що надійшов до складу, на зберігання наверху стопки з координатами (x, y).

```
void RemoveContainer(int x, int y);
procedure RemoveContainer(x, y: integer);
DECLARE SUB RemoveContainer CDECL (BYVAL x AS INTEGER,
                                   BYVAL y AS INTEGER)
```

Відправляє зі складу контейнер зверху стопки з координатами (x, y). x, y.

Якщо ваша програма не може нічого зробити, вона повинна закінчувати роботу.

Невірні переміщення ігноруються бібліотекою і не впливають на стан моделі і оцінку розв'язку.

Від вашої програми *не вимагається* виводити будь-що у файли. Бібліотека, з якою взаємодіє ваша програма, записує події у файл. Цей файл використовується при перевірці.

Порядок роботи Ваша програма повинна отримувати інформацію про контейнер, який потрібно обробити наступним. Після чого вона повинна перемістити контейнери краном, якщо це потрібно, та взяти на зберігання, відправити, або відмовитися приймати контейнер.

Бібліотека Вам дається бібліотека з ім'ям StackLib, яку ви повинні підключити до вашої програми.

Стандартні бібліотеки `C` і `C++` містять цю бібліотеку, та автоматично будуть підключатися до вашої програми, якщо ви підготуєте відповідний файл заголовку (header file).

Якщо ви використовуєте `QuickBasic`, ви повинні підключити бібліотеку таким чином:

```
QB /L STACKLIB
```

Приклади вихідних текстів представлено у каталозі задачі під назвами `TESTSTK.BAS`, `TESTSTK.PAS`, `TESTSTK.CPP` та `TESTSTK.C`.

Оцінки Програма буде перевірятися з різними наборами даних. Для кожного набору даних її продуктивність буде порівнюватися з найбільш ефективним розв'язком, відомим науковому комітету, при урахуванні наступних факторів:

- Загальна кількість переміщень крану вашою програмою.
- Штраф у 5 переміщень накладається за кожен контейнер, який програма відмовилась приймати на збереження.
- Штраф у 5 переміщень накладається за кожен не збережений та не відправлений контейнер (у випадку, якщо програма нормально завершується, а потрібні операції ще не закінчені).
- Загальний бал обраховується по відношенню до найкращого відомого научному комітету розв'язку.
- Якщо програма перевищує кількість операцій більше ніж у двічи, вона оцінюється 0 балами.
- Мінімальна оцінка — 0%, максимальна — 100%.

Розділ 10. Португалія'1998

10.1 Завдання першого туру

10.1.1 «Контакт»

Доктор Astro Insky працює в радіотелескопічному центрі. Нещодавно вона помітила дуже незвичне пульсуюче мікрохвильове випромінювання, що походить безпосередньо з центру галактики. Чи послані ці сигнали якою-небудь позаземною формою розумного життя? Або це просте серцебиття зірок?

Завдання Ви повинні допомогти Доктору Insky відшукати істину, створивши інструмент для аналізу бітових послідовностей у файлах, що вона записала. Доктор Insky хоче знайти бітові підпослідовності довжиною від A до B включно, що зустрічаються в записаній послідовності найбільш часто. Підпослідовності можуть перекриватися. Визначаються N найбільших різноманітних частот (тобто кількостей входжень). Враховуються підпослідовності, що зустрічаються хоча б один раз.

Вхідні дані Файл `CONTACT.IN` містить дані в такому форматі: перший рядок містить ціле A — мінімальну довжину підпослідовності. Другий рядок містить ціле B — максимальну довжину підпослідовності. Третій рядок містить ціле N — кількість різноманітних частот входження. Четвертий рядок містить послідовність символів `0` і `1`, обмежену символом `2`.

У прикладі потрібно відшукати десять (третій рядок) найбільших частот входження підпослідовностей довжиною від двох (перший рядок) до чотирьох (другий рядок) у послідовності (четвертий рядок файла)

```
0101001001000100011110110000101001100111100001001001111001000000 0
```

Зверніть увагу, що четвертий рядок вхідного файла розділений на кілька рядків для того, щоб поміститися на папері. У цьому прикладі підпослідовність `100` зустрічається 12 разів, а підпослідовність `1000` зустрічається 5 разів. Найбільше часто зустрічається підпослідовність `00`.

Вихідні дані Файл `CONTACT.OUT` містить не більш N рядків, кожна з який містить одну з N найбільших частот входження і відповідні їй підпослідовності. Рядки повинні розташовуватися в порядку спадання частот входження і мати такий вигляд:

частота підпосл. підпосл. ... підпосл.

де *частота* — це кількість входжень вказаних далі підпослідовностей. Підпослідовності в кожному рядку повинні розташовуватися в порядку спадання довжини. Підпослідовності однакової довжини повинні розташовуватися в порядку спадання їхнього числового значення. Якщо в послідовності зустрічаються менше N різноманітних частот, вихідний файл буде містити менше N рядків.

Обмеження Розмір вхідного файла не перевищує 2-х мегабайт. Параметри A, B і N мають такі обмеження: $0 < N \leq 20$, $0 < A \leq B \leq 12$.

Приклад вхідних та вихідних даних

`CONTACT.IN`

```
2
4
10
0101001001000100011110 ↓
1100001010011001111000 ↓
0100100111100100000002
```

`CONTACT.OUT`

```
23 00
15 10 01
12 100
11 001 000 11
10 010
8 0100
7 1001 0010
6 0000 111
5 1000 110 011
4 1100 0011 0001
```

10.1.2 «Лампи для свята»

Для освітлення заключного вечора IOI'98 є N кольорових ламп, пронумерованих від 1 до N. Чотири кнопки дозволяють керувати станами ламп. Натискання кнопок впливає на стан ламп у такий спосіб: кнопка 1 — змінює стан усіх ламп: ті, що були ввімкнені, стають вимкнинеми, ті, що були вимкнені — ввімкнинеми; кнопка 2 — змінює стан усіх ламп, що мають непарні номери; кнопка 3 — змінює стан усіх ламп, що мають парні номери; кнопка 4 — змінює стан усіх ламп, що мають номери, які обчисляються по формулі $3K + 1$ (де $K \leq 0$), тобто 1, 4, 7,... Мається лічильник C, що рахує сумарне число натискань кнопок. На початку вечора усі лампи були ввімкнені, а лічильник C був встановлений у нуль.

Завдання Задано значення лічильника C і інформація про остаточний стан деяких ламп. Напишіть програму для визначення усіх можливих різноманітних остаточних конфігурацій станів N ламп, щоб кожна конфігурація відповідала заданій інформації.

Вхідні дані Файл з ім'ям `PARTY.IN` містить чотири рядки, що визначають кількість ламп N, остаточне значення лічильника C, і стан деяких ламп в остаточній конфігурації. У першому рядку міститься число N, у другому рядку — остаточне значення лічильника C. Третій рядок містить список номерів ламп, про які відомо, що в остаточній конфігурації вони ввімкнені. Номера ламп у рядку відділений один від одного одним пропуском, і список закінчується числом `-1`. Четвертий рядок містить список номерів ламп, про які відомо, що в остаточній конфігурації вони вимкнені. Номера ламп у рядку відділений один від одного одним пропуском, і список закінчується числом `-1`.

На прикладі число ламп дорівнює 10 і зроблено тільки одне натискання. Лампа 7 — вимкнена в остаточній конфігурації.

Вихідні дані Файл `PARTY.OUT` повинен містити усі можливі різноманітні остаточні конфігурації (без повторень). Кожна можлива конфігурація повинна бути записана в окремому рядку. Конфігурації можуть бути перераховані в довільному порядку.

Кожен рядок містить N символів, де перший символ представляє стан лампи з номером 1, а останній символ представляє стан лампи з номером N. Символ `0` (нуль) означає, що відповідна лампа, вимкнена, а символ `1` (одиниця) означає, що лампа ввімкнена.

Очевидно, що для прикладу існує три можливі різноманітні остаточні конфігурації: всі лампи вимкнені; або лампи 1, 4, 7, 10 вимкнені, а лампи 2, 3, 5, 6, 8, 9 ввімкнені; або лампи 1, 3, 5, 7, 9 вимкнені, а лампи 2, 4, 6, 8, 10 ввімкнені.

Обмеження Параметри N і C мають наступні обмеження: $10 \leq N \leq 100$, $1 \leq C \leq 10\,000$ Кількість ламп, про які відомо, що в остаточній конфігурації вони ввімкнені, менше або дорівнює 2. Кількість ламп, про які відомо, що в остаточній конфігурації вони вимкнені, менше або дорівнює 2. Відомо, що існує хоча б одна остаточна конфігурація для кожного вхідного файла.

Приклад вхідних та вихідних даних

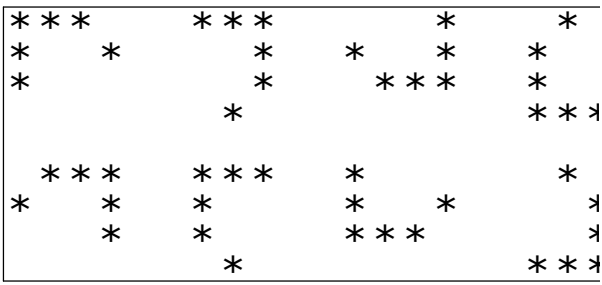
PARTY.IN
10 1 -1 7 -1

PARTY.OUT
0000000000 0110110110 0101010101

10.1.3 «Зоряна ніч»

Погляньте на нічне небо, де світяться зірки, об'єднані в сузір'я (кластери) різноманітних форм. Сузір'я — це непуста група поруч розташованих зірок, що мають сусідами зірки або по горизонталі, або по вертикалі, або по діагоналі. Сузір'я не може бути частиною іншого сузір'я.

На небі можуть бути однакові сузір'я. Два сузір'я є *однаковими*, якщо вони мають ту саму форму незалежно від їхньої орієнтації. У загальному випадку, число можливих орієнтацій для сузір'я дорівнює восьми, як показано на мал. 1.



Нічне небо представляється картою неба, що є двомірной матрицею, яка складаються з нулів і одиниць. Елемент цієї матриці містить цифру `1`, якщо це зірка, і цифру `0` — у противному випадку.

Завдання Для заданої карти неба відзначити всі сузір'я, використовуючи при цьому маленькі літери. Однакові сузір'я повинні бути відзначені однією і тією ж літерою; різні сузір'я повинні бути відзначені різними маленькими літерами.

Для того щоб відзначити сузір'я, необхідно замінити кожен символ `1` (одиниця) у сузір'ї відповідною маленькою літерою.

Вхідні дані Вхідній файл з ім'ям `STARRY.IN` містить у перших двох рядках відповідно ширину W і висоту H двомірної матриці, що відповідає карті неба. Далі в цьому файлі знаходяться H рядків, кожен з яких містить W символів.

На прикладі матриця, що відповідає карті неба, має ширину 23 і висоту 15. Даному вхідному файлу відповідає наступне зображення карти неба.

Вихідні дані Файл `STARRY.OUT` містить ту ж саму матрицю карти неба, що і файл `STARRY.IN`, тільки сузір'я в ній відзначені так, як описано в розділі «Завдання».

На прикладі вказано один з можливих варіантів вихідного файла для поданого вхідного файла. Цьому вихідному файлу відповідає наступне зображення.

Обмеження

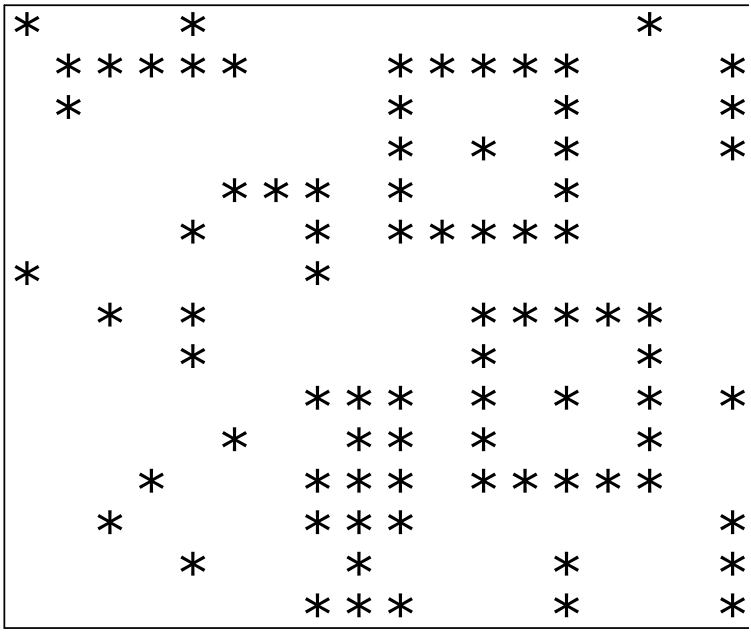
W — ширина карти неба, $0 \le W \le 100$

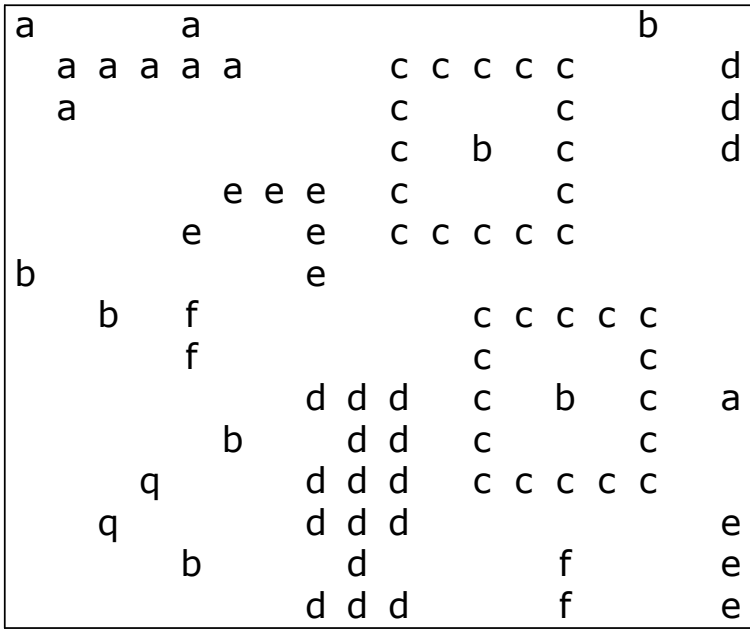
H — висота карти неба, $0 \le H \le 100$

$0 \le$ Число сузір'їв ≤ 500

$0 \le$ Число різноманітних сузір'їв ≤ 26 (a. . . z)

$1 \le$ Число зірок у кожному сузір'ї ≤ 160





Приклад вхідних та вихідних даних

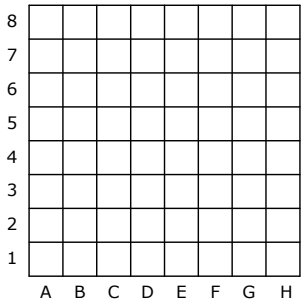
STARRY.IN
23 15 100010000000000010000000 011111000111110001011 01 010000000100001000111111 000000000101010001011 11 000001110100010000000000 000010010111110000000000 100000010000000000000000 001010000001111100100 00 000010000001000100111 11 000000011101010101000 10 000001001101000100000000 001000011100000001000000 000010001000010001001 01 000000011100010001110 00

STARRY.OUT
a000a0000000000b0000000 0aaaaa000ccccc000d0dd0d 0a0000000c000c000dddddd 000000000c0b0c000d0dddd 00000eee0c000c000000000 0000e00e0ccccc000000000 b000000e000000000000000 00b0f000000ccccc00a0000 0000f000000c000c00aaaaa 0000000ddd0c0b0c0a000a0 00000b00dd0c000c0000000 000g000ddd0ccccc0000000 00g0000ddd0000000e00000 0000b000d0000f000e00e0b 0000000ddd000f000eee000

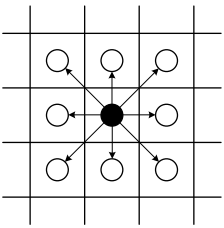
10.2 Завдання другого туру

10.2.1 «Камелот»

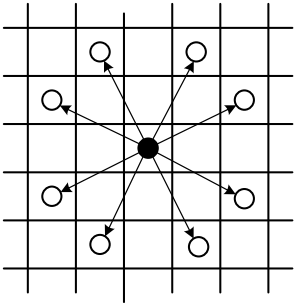
Давним-давно, король Артур і лицарі Круглого Столу звичайно збиралися на Новий рік, щоб відсвяткувати свою дружбу. У пам'ять про ці події придумали настільну гру Камелот для одного гравця, у якій фігура Короля і декілька фігур Лицарів довільно розташовуються в різноманітних клітинах дошки. Дошка має розмір 8×8 клітин (див. мал. 1).



Король може переміщатися в будь-яку суміжну клітину дошки як показано на мал. 2, при цьому не виходячи за межі дошки.



Лицар (як і шаховий кінь) може переміщатися як показано на мал. 3, при цьому не виходячи за межі дошки.



Під час гри гравець може помістити більше однієї фігури в одну клітину. Клітини вважаються достатньо великими, і не виникає перешкод для вільного переміщення фігур.

Гравцю необхідно так переміщати фігури Короля і Лицарів, щоб зібрати їх всіх у якійсь одній клітині за найменше число ходів. Ходи фігурами необхідно робити по правилах, описаним вище. Додатково до цього, якщо Король і один або більш Лицарів знаходяться в одній клітині, гравець може Короля й одного з Лицарів перемістити разом по правилах переміщення Лицаря і вважати це одним ходом.

Завдання Напишіть програму для обчислення мінімальної кількості ходів, необхідних для переміщення усіх фігур в одну клітину дошки.

Вхідні дані Файл `CAMELOT.IN` містить один рядок символів, що описує початкове розташування фігур на дошці. Рядок містить послідовність клітин дошки, перша з який — клітина Короля, інші — клітини Лицарів. Кожна клітина описується парою буква-цифра. Літера позначає горизонтальну, а цифра — вертикальну координату клітини дошки. Усі фігури на початку гри розташовані в різних клітинах.

На прикладі Король розташований у клітині `D4`. Чотири Лицарі в клітинах `A3`, `A8`, `H1`, і `H8`.

Вихідні дані Файл `CAMELOT.OUT` повинен містити єдиний рядок із додаьнім цілим числом, що позначає мінімальне число ходів гравця, необхідних для переміщення усіх фігур в одну клітину дошки.

Обмеження

$0 \leq$ кількість Лицарів ≤ 63

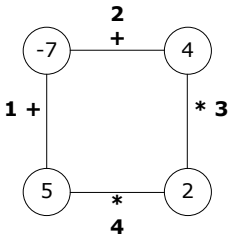
Приклад вхідних та вихідних даних

CAMELOT.IN
D4A3A8H1H8

CAMELOT.OUT
10

10.2.2 «Полігон»

Існує гра для одного гравця, що починається з завдання Полігона з N вершинами. Приклад графічного представлення Полігона показаний на мал. 1, де $N = 4$. Для кожної вершини Полігона задається значення — ціле число, а для кожного ребра — мітка операції + (додавання) або * (множення). Ребра Полігона пронумеровані від 1 до N.

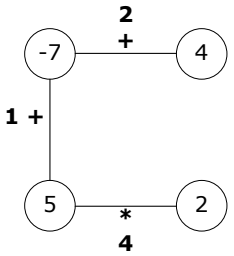


Першим ходом у грі видаляється одне з ребер.
Кожний наступний хід складається з таких кроків:

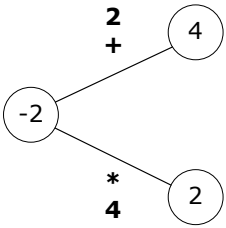
- вибирається ребро E та дві вершини V_1 і V_2, що з'єднані ребром E;
- ребро E і вершини V_1 і V_2 замінюються новою вершиною зі значенням, рівним результату виконання операції, визначеною міткою ребра E, над значеннями вершин V_1 і V_2.

Гра закінчується, коли більше немає жодного ребра. Результат гри — число, рівне значенню вершини, що залишилася.

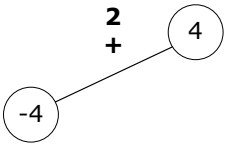
Приклад гри Розглянемо Полігон на мал. 1. Гравець почав гру з видалення ребра 3 (див. мал. 2).



Після цього, гравець вибирає ребро 1 (див. мал. 3),



Ребро 4



і, нарешті, ребро 2. Результатом гри буде число 0 (див. мал. 5).

Завдання Напишіть програму, що по заданому Полігону обчислює максимальне значення вершини що залишилася і виводить список усіх тих ребер, видалення яких на першому ході гри дозволяє одержати це значення.

Вхідні дані Файл `POLYGON.IN` описує Полігон із `N` вершинами. Файл містить два рядки. У першому рядку записане число `N`. Другий рядок містить мітки ребер із номерами $1, \ldots, N$, між якими записані через один пропуск значення вершин (перше з них відповідає вершині, суміжній ребрам 1 і 2, наступне — суміжній ребрам 2 і 3, і так далі, останнє — суміжній ребрам N і 1). Мітка ребра — це літера `t`, що відповідає операції додавання, або літера `x`, що відповідає операції множення.

Вхідний файл у прикладі описує Полігон, зображений на мал. 1. Другий рядок починається з мітки ребра 1.

Вихідні дані У першому рядку вихідного файла `POLYGON.OUT` програма повинна записати максимальне значення вершини що залишилася, яке може бути досягнуте для заданого Полігона. В другому рядку повинен бути записаний список усіх тих ребер, видалення яких на першому ході дозволяє одержати це значення. Номери ребер повинні бути записані в зростаючому порядку і відокремлюватися один від одного одним пропуском.

Вихідний файл для Полігона, зображеного на мал. 1, наведено у прикладі.

Обмеження
$3 \leq N \leq 50$

Для будь-якої послідовності ходів, значення вершин знаходяться в межах $[-32\ 768, 32\ 767]$.

Приклад вхідних та вихідних даних

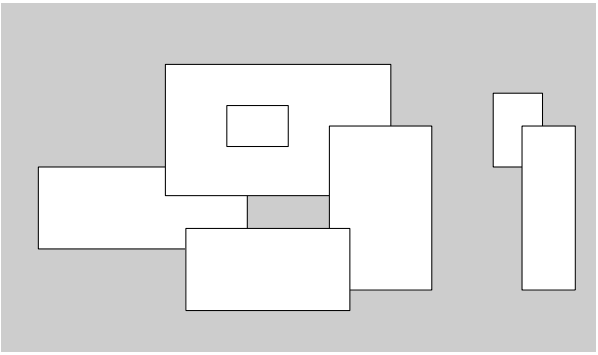
POLYGON.IN
4 t -7 t 4 x 2 x 5

POLYGON.OUT
33 1 2

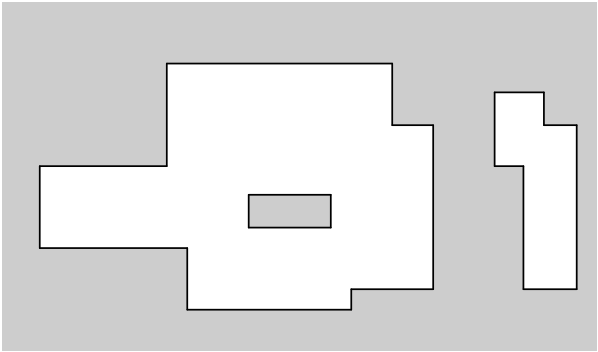
10.2.3 «Картинки»

На стіні наклеєна деяка кількість плакатів, фотографій і інших картинок прямокутної форми. Усі їх сторони або вертикальні, або горизонтальні. Кожен прямокутник може бути частково або цілком перекритий іншими прямокутниками. Периметром називається загальна довжина внутрішніх і зовнішніх границь фігур, отриманих шляхом накладення прямокутників.

Завдання Написати програму, що обчислює значення отриманого периметра. На мал. 1 показаний набір із 7 прямокутників.



Відповідні границі отриманої фігури показані на мал. 2.



Вершини всіх прямокутників мають цілочисельні координати.

Вхідні дані Перший рядок вхідного файла `PICTURE.IN` містить число прямокутників, наклеєних на стіні. У кожному наступному рядку наводяться цілочисельні координати лівої нижньої вершини і правої верхньої вершини кожного прямокутника. Значення цих координат дані як впорядковані пари, що складаються з X-координати та Y-координати.

Вхідний файл у прикладі відповідає прикладу, наведеному на мал. 1.

Вихідні дані Вихідний файл повинен містити тільки один рядок із невід'ємним цілим числом, що є периметром для заданих прямокутників.

Обмеження
$0 \leq$ число прямокутників $< 5\ 000$

Усі координати знаходяться в межах $[-10\ 000, 10\ 000]$, і кожен заданий прямокутник має додатню площу. Для збереження результату достатньо 32 бита.

Приклад вхідних та вихідних даних

PICTURE.IN
7 -15 0 5 10 -5 8 20 25 15 -4 24 14 0 -6 16 4 2 15 10 22 30 10 36 20 34 0 40 16

PICTURE.OUT
228

Розділ 11. Турція'1999

11.1 Завдання першого туру

11.1.1 «Квіткова крамничка»

Ви хочете оформити вітрину вашої квіткової крамнички щонайкраще. У вас є F букетів (кожен букет складається з квітів одного виду, усі квіти одного виду зібрані в одному букеті) і не менша кількість ваз, розташованих у ряд. Вази приклеєні до полиці і занумеровані зліва направо послідовно від 1 до V, де V — кількість ваз. Ваза 1 є крайньою лівою в ряду, а ваза V — крайньою правою. Букети можна поміщати у різні вази і кожен букет має унікальний номер у межах від 1 до F. Порядок розміщення букетів у вазах такий: букет i завжди міститься у вазі, що стоїть лівіше вази з букетом j, якщо $i < j$. Нехай, наприклад, у вас є букет азалій із номером 1, букет бегоній із номером 2 і букет гвоздик із номером 3. Усі букети повинні бути поміщені у вази. Букет азалій повинен бути поміщеним у вазу, розташовану лівіше вази з бегоніями, а букет бегоній — лівіше вази з гвоздиками. Якщо ваз буде більше, ніж букетів, то деякі вази залишаться порожніми. У вазі може бути не більше одного букета.

Кожна ваза має власні характеристики, так само як і квіти. Тому поміщений у вазу букет має деяку естетичну характеристику, виражену цілим числом. Значення естетичних характеристик наведені у таблиці нижче.

	1	2	3	4	5
1 (азалії)	7	23	-5	-24	16
2 (бегонії)	5	21	-4	10	23
3 (гвоздики)	-21	5	-4	-20	20

Порожня ваза має значення естетичної характеристики, рівне нулю.

Відповідно до цієї таблиці азалії прекрасно виглядають у другій вазі і жахливо у четвертій.

Для досягнення найкращого естетичного оформлення вітрини ви повинні максимізувати суму значень естетичних характеристик розміщень квітів у вазах, зберігаючи необхідне упорядкування букетів. Якщо існує більше одного розміщення, що мають максимальну суму, то ви повинні вивести тільки одне з них.

Обмеження

- $1 \le F \le 100$, де F — кількість букетів. Букети пронумеровані від 1 до F.
- $F \le V \le 100$, де V — кількість ваз.
- $-50 \le A_{ij} \le 50$, де A_{ij} — значення естетичної характеристики, отримане при поміщенні букета i у вазу j.

Вхідні дані Вхідний текстовий файл називається `FLOWER.INP`.

- перший рядок містить два числа: F, V.
- наступні F рядків: кожен з цих рядків містить V цілих чисел, A_{ij} є j-м числом у (i+1)-му рядку вхідного файла.

Вихідні дані Вихідний файл є файлом з ім'ям `FLOWER.OUT` і складається з двох рядків:

- перший рядок містить суму значень естетичних характеристик вашого розміщення квітів у вазах.
- другий рядок містить послідовність із F чисел, де k-те число визначає номер вази, у якій поміщено букет k.

Оцінки Ваша програма повинна виконуватися не більше 2-х секунд. Часткові розв'язки в кожному тесті оцінюватися не будуть.

Приклад вхідних та вихідних даних

FLOWER.INP
3 5 7 23 -5 -24 16 5 21 -4 10 23 -21 5 -4 -20 20

FLOWER.OUT
53 2 4 5

11.1.2 «Сховані коди»

Є множина кодових слів і текст. Текст містить повідомлення, створене з кодових слів, включених у нього деяким, можливо неоднозначним, способом. Кодові слова і текст містять тільки великі і малі літери англійського алфавіту. Велика і мала літери розрізняються. Довжина кодового слова визначається звичним чином: наприклад, довжина кодового слова `ALL` дорівнює 3.

Літери кодового слова зустрічаються в даному тексті не обов'язково послідовно. Наприклад, кодове слово `ALL` завжди буде присутнім у тексті, що містить підпослідовність виду `A`u`L`v`L`, де u і v є послідовностями літер (можливо порожніми). Будемо говорити, що `A`u`L`v`L` є покриваючою послідовністю для `ALL`. У загальному випадку, покриваюча послідовність для кодового слова визначається як підпослідовність тексту, у якої перша й остання літери збігаються відповідно з першою й останньою літерами кодового слова й існує спосіб відновити кодове слово вилученням, якщо необхідно, деяких літер підпослідовності. Зауважимо, що кодове слово може міститися в одній чи декількох покриваючих послідовностях, або не зустрічатися в тексті взагалі. Крім того, покриваюча послідовність може бути покриваючою послідовністю для більш ніж одного кодового слова.

Покриваюча послідовність визначається своєю початковою і кінцевою позиціями в тексті (позиціями першої й останньої літер відповідно). Перша літера тексту знаходиться в позиції 1. Будемо говорити, що покриваючі послідовності c_1 і c_2 не перекриваються, якщо початкова позиція c_1 більша ($>$) ніж кінцева позиція c_2 або навпаки. У іншому випадку ці дві послідовності перекриваються.

Розв'язання задачі полягає у виділенні схованого повідомлення з тексту. Розв'язком є множина елементів, кожний із який складається з кодового слова і покриваючої послідовності для цього кодового слова. При цьому повинні виконуватися наступні умови:

- покриваючі послідовності не перекривають одна одну;
- довжина покриваючої послідовності не перевищує 1 000;
- сума довжин кодових слів — максимальна (кожен елемент добавляє довжину кодового слова, яке він містить, до суми довжин).

При наявності більше одного розв'язку вам необхідно вказати тільки один з них.

Обмеження

- $1 \leq N \leq 100$, де N — кількість кодових слів.

- Максимальна довжина кодового слова — 100 літер.
- $1 \leq$ довжина заданого тексту $\leq 1\ 000\ 000$.

Гарантується, що в даному тексті

- Для будь-якої позиції в тексті кількість мінімальних справа покриваючих послідовностей, що містять цю позицію, не перевищує 2 500;
- кількість мінімальних справа покриваючих послідовностей не перевищує 10 000.

Будемо говорити, що покриваюча послідовність c для кодового слова w є мінімальною справа, якщо ніякий власний префікс c (власним префіксом c є будь-яка початкова підпослідовність c, така що вона не дорівнює c) не є покриваючою послідовністю для w. Наприклад, для кодового слова `ALL`, `AAALAL` є мінімальною справа покриваючою послідовністю, у той час як `AAALALAL` також є покриваючою послідовністю, але не є мінімальною справа.

Вхідні дані Є два вхідні текстових файли: `WORDS.INP` і `TEXT.INP`. Файл `WORDS.INP` містить список кодових слів, а файл `TEXT.INP` містить текст.

Перший рядок файла `WORDS.INP` містить значення N. Кожен з наступних N рядків містить кодове слово, що є послідовністю літер без пропусків між ними. Кожне кодове слово однозначно визначене своїм порядковим номером у файлі `WORDS.INP`: Цілі числа від 1 до N є номерами кодових слів.

Файл `TEXT.INP` складається з послідовності літер, що закінчується одним символом кінця рядка (end-of-line character), за яким іде символ кінця файла (end-of-file symbol). Цей файл не містить пропусків.

Рекомендації для програмуючих мовою `Pascal` Для забезпечення ефективності рекомендується описувати вхідні файли типу text.

Вихідні дані Вивід повинен проводитися в файл `CODES.OUT`.

Перший рядок повинен містити суму, яку обчислила ваша програма.

Кожен наступний рядок повинен містити один елемент вашого розв'язку. Рядок складається з трьох цілих чисел i, s, e, де i — номер кодового слова, що міститься в покриваючій послідовності, яка визначається початковою позицією s і кінцевою позицією e. Порядок виводу усіх рядків, починаючи з другого, не має значення.

Примітка Сховане повідомлення, що відповідає наведеному прикладу, виглядає так: `RuN RaBbit RuN`. (Іншим можливим повідомленням є `RuN HoBbit RuN`). Зверніть увагу, що повідомлення не повинно з'являтися у вихідному файлі.

Оцінки Ваша програма повинна виконуватися не більш 10 секунд. Часткові розв'язки в кожному тесті оцінюватися не будуть.

Приклад вхідних та вихідних даних

WORDS.INP
4 RuN RaBbit HoBbit StoP

TEXT.INP
StXRuYNvRuHoaBbvizXztNwRRuuNNP

CODES.OUT
12 2 9 21 1 4 7 1 24 28

11.1.3 «Підземне місто»

Ви опинилися в одному з підземних міст Кападокії. Блукаючи у темряві, ви випадково знаходите карту міста. На жаль, на карті не позначене те місце, де ви знаходитесь. Ваша задача — досліджуючи місто, визначити це місце.

Карта міста являє собою прямокутну сітку з одиничними квадратами. Кожен квадрат може бути або порожнім, позначеним літерою `O`, або частиною стіни, позначеним літерою `W`. На карті також показано напрямок на північ. На щастя, у вас у руці є компас, так що ви можете правильно орієнтуватися по карті. Спочатку ви знаходитеся на порожньому квадраті.

Дослідження міста починається з виклику процедури (функції) Start, що не має аргументів. При дослідженні міста ви можете використовувати процедури (функції) Look і Move.

Ви можете питати, викликаючи функцію з ім'ям Look(Dir), де Dir позначає напрямок дослідження. Воно може мати значення `N`, `S`, `E` або `W`, що позначають відповідно північ, південь, схід і захід. Нехай аргумент Dir має значення `N`. Тоді відповіддю буде літера `O`, якщо квадрат, що ви аналізуєте в північному напрямку, є порожнім. Якщо там стіна, то відповіддю буде літера `W`. Аналогічно, можна аналізувати й одержувати інформацію про інші сусідні квадрати

Ви можете переміщуватися в один із чотирьох сусідніх квадратів, викликаючи процедуру Move(Dir), де Dir позначає напрямок вашого руху. Ви можете переміщуватися тільки в порожні квадрати. Переміщення на клітину, зайняту стіною, є помилкою. З будь-якого порожнього квадрата можна потрапити в будь-який інший порожній квадрат.

Вам необхідно знайти координати порожнього квадрата, у якому ви виявили карту, викликаючи процедуру Look(Dir) мінімальну кількість разів. Коли ви, нарешті, визначите ці координати, ви повинні викликати процедуру (функцію) Finish(x,y), де x — горизонтальна (схід-захід), а y — вертикальна (південь-північ) координати цього порожнього квадрата.

Обмеження

- $3 \le U \le 100$, де U є ширина карти, тобто кількість квадратів карти у горизонтальному (захід-схід) напрямку.

- $3 \le V \le 100$, де V є висота карти, тобто кількість квадратів карти у вертикальному (південь-північ) напрямку.
- Місто оточене стінами, що позначені на карті.
- Південно-західний кут міста має координати $(1, 1)$, а північно-східний кут має координати (U, V).

Вхідні дані Вхідним файлом є текстовий файл з ім'ям `UNDER.INP`.

Перший рядок містить два числа: U, V.

Кожен з наступних V рядків містить інформацію про рядки карти. Рядок складається з U значень. При цьому x-ий символ рядка $(V - y + 2)$ вхідного файла задає координати позиції (x, y) на карті: це або літера `W`, що позначає стіну, або літера `0`, що позначає порожній квадрат. Рядки не можуть містити пропусків.

Вихідні дані Вихідний файл не повинен створюватися. Результат розв'язання задачі повідомляється викликом процедури (функції) `Finish(x,y)`.

Інструкція для програмуючих мовою `Pascal` У вихідний файл вашої програми включіть наступний рядок:

```
uses undertpu;
```

Цей модуль буде містити наступне:

```
procedure Start; { must be called first }
function Look (dir:char):char;
procedure Move (dir:char);
procedure Finish (x,y:integer); { must be called last }
```

Інструкція для програмуючих на `C`/`C++` У вихідному файлі вашої програми потрібен наступний рядок:

```
#include "under.h"
```

У `UNDER.H` будуть міститись наступні оголошення:

```
void Start (void); /* must be called first */
char Look (char);
void Move (char);
void Finish (int,int); /* must be called last */
```

Оцінки Ваша програма повинна виконуватися не більш 5 секунд.

Щоб одержати повний бал A за тест, потрібно, щоб x — кількість викликів функції Look, була менше або рівна числу M, встановленому тестуючою програмою. Зауважимо, що M вибирається таким, щоб бути більшим ($>$), ніж мінімум. Зокрема, M не залежить від того, обрано напрямок огляду по чи проти годинникової стрілки. Ви можете одержати частину балів, якщо число звернень до функції Look більше ($>$) M, але менше ($<$) подвоєного M. Бали, які ви одержите, обчислюються шляхом округлення першого десяткового знака того значення, яке визначається за наступною формулою:

$$\begin{array}{ll} A & \text{якщо } x \le M; \\ \frac{A(2M-x)}{M} & \text{якщо } M < x < 2M; \\ 0 & \text{якщо } x \ge 2M. \end{array}$$

Якщо програма працює з порушенням правил, то вона оцінюється в 0 балів. До порушень правил належить наступне:

- Звернення до бібліотечної процедури (функції) із неприпустимим аргументом, наприклад, із символом, що не позначає напрямок.
- Спроба рухатися на стіну.
- Невідповідність умовам задачі.

Приклад вхідних та вихідних даних

UNDER.INP
5 8 WWWWW WWWOW WWWOW WOOOW WOWOW WOOWW WWOOW WWWWW

Можлива послідовність викликів процедур, що закінчується коректним викликом процедури Finish:

```
Start()
Look('N')   'W'
Look('E')   'O'
Move('E')
Look('E')   'W'
Finish(3,5)
```

11.2 Завдання другого туру

11.2.1 «Світлофори»

Дорожній рух в місті Дингилвілі улаштовано незвичним способом. У місті є перехрестя і дороги, якими перехрестя пов'язані між собою. Будь-які два перехрестя можуть бути пов'язані не більш ніж однією дорогою. Не існує доріг, що з'єднують те саме перехрестя із самим собою. Час проїзду по шляху в обох напрямках однаковий. На кожному перехресті знаходиться один світлофор, що у кожний момент часу може бути або блакитним, або червоним. Колір кожного світлофора змінюється періодично: протягом деякого інтервалу часу він блакитний, а потім, протягом деякого іншого інтервалу — червоний. Рух по дорозі між будь-якими двома перехрестями дозволений тоді і тільки тоді, коли світлофори на обох перехрестях цієї дороги мають той самий колір у момент в'їзду на цю дорогу. Якщо транспортний засіб прибуває на перехрестя в момент переключення світлофора, то його поведінка буде визначатися новим світлом світлофора. Транспортні засоби можуть знаходитися в стані чекання на перехрестях. У вас є карта міста, що показує:

- час проходження кожної дороги (цілі числа),
- тривалість горіння кожного кольору для кожного світлофора (цілі числа),
- початковий колір і час горіння, що залишився, цього кольору (цілі числа) для кожного світлофора.

Ви повинні визначити шлях між двома заданими перехрестями, що дозволяє транспортному засобові проїхати від початкового перехрестя до кінцевого за мінімальний час із моменту старту. Якщо існують декілька таких шляхів, то ви повинні вивести тільки один із них.

Обмеження

- $2 \leq N \leq 300$, де N — кількість перехресть. Перехрестя нумеруються цілими числами від 1 до N.
- $1 \leq M \leq 14\,000$, де M — кількість шляхів.
- $1 \leq l_{ij} \leq 100$, де l_{ij} — час, необхідний для переміщення від перехрестя i до перехрестя j, використовуючи шлях, що з'єднують перехрестя i і j.
- $1 \leq t_{ic} \leq 100$, де t_{ic} — тривалість горіння цвіту c для світлофора на перехресті i. Індекс c приймає значення `B` для блакитного цвіту і `P` — для червоного (`B` і `P` — велики латинські букви).

- $1 \le r_{ic} \le t_{ic}$, де r_{ic} — час світіння, що залишився, початкового цвіту c на перехресті i.

Вхідні дані Вхідний файл є текстовим файлом з ім'ям `LIGHTS.INP`.

- Перший рядок містить два числа: номер початкового перехрестя і номер кінцевого перехрестя.
- Другий рядок містить два числа: N, M.
- Наступні N рядків містять інформацію про N перехресть. $(i+2)$-ий рядок вхідного файла містить інформацію про перехрестя i: C_i, r_{ic}, t_{iB}, t_{iP}, де C_i (`B` або `P`) — початковий колір світлофора на перехресті i.
- Наступні M рядків описують M доріг. Кожен рядок має вигляд: i, j, l_{ij}, де i і j — номери перехресть, що зв'язує ця дорога.

Вихідні дані Вихідний файл повинен мати ім'я `LIGHTS.OUT`.

Якщо шлях існує:

- Перший рядок повинен містити мінімальний час, необхідний для досягнення кінцевого перехрестя з початкового.
- Другий рядок повинен містити список перехресть, що відповідає знайденому мінімальному шляху. Ви повинні записати перехрестя в порядку їх проходження. Перше ціле повинно бути номером початкового перехрестя, останнє — кінцевого.

Якщо шлях не існує:

- Єдиний рядок повинен містити тільки ціле число `0`.

Оцінки Ваша програма повинна виконуватися не більше 2 секунд.

Часткові розв'язки в кожному тесті оцінюватися не будуть.

Приклад вхідних та вихідних даних

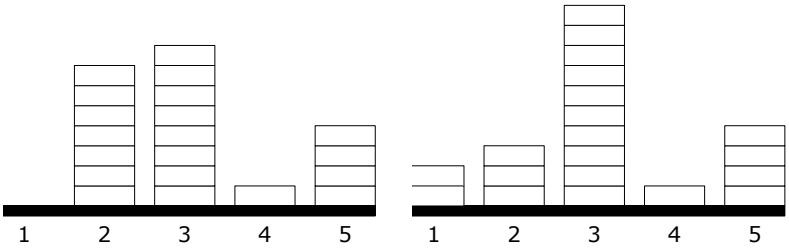
LIGHTS.INP
1 4 4 5 B 2 16 99 P 6 32 13 P 2 87 4 P 38 96 49 1 2 4 1 3 40 2 3 75 2 4 76 3 4 77

LIGHTS.OUT
127 1 2 4

11.2.2 «Вирівнювання»

Для гри в пасьянс задаються послідовно розташовані N стопок, у кожній з який знаходиться деяка кількість (можливо нульова) карток (див. мал. 1). Стопки занумеровані цілими числами від 1 до N. Для того, щоб зробити один хід, ви повинні вказати номер стопки, наприклад p, і назвати деяке число, наприклад m. Це відповідає тому, що m карток переміщаються зі стопки з номером p на кожну із сусідніх із нею стопок (див. мал. 2). Для стопки p сусідніми є стопки з номерами $p-1$ і $p+1$ при $1 < p < N$. Для $p = 1$ сусідньою є тільки стопка з номером 2, а для $p = N$ — стопка $N-1$. Для виконання одного ходу, у стопці p повинно знаходитися принаймні $2m$ карток, якщо вона має двох сусідів, і принаймні m карток, якщо є тільки один сусід.

Метою гри є «вирівнювання» усіх стопок, тобто потрібно зробити так, щоб у стопках знаходилася однакова кількість карток, при цьому затративши якомога меншу кількість кроків. Якщо існує більше одного способу, то ви повинні вивести тільки один із них.



Обмеження

- Гарантується, що можна зрівняти задані стопки карток не більш ніж за 10 000 кроків.
- $2 \le N \le 200$
- $0 \le C_i \le 2\,000$, де C_i — кількість карток у стопці i у момент початку гри ($1 \le i \le N$).

Вхідні дані Запровадження здійснюється з текстового файла з ім'ям `FLAT.INP`, що містить два рядки.

- Перший рядок: N
- Другий рядок: N цілих чисел C_i.

Вихідні дані Висновок провадиться у вихідний текстовий файл з ім'ям `FLAT.OUT`.

- Перший рядок: кількість кроків (назвемо його M).
- Кожен з наступних M рядків містить два цілих числа, що описують відповідний хід: p, m.

Порядок ходів у вихідному файлі повинний бути таким же, як і в грі. Так, Ваш перший хід буде записаний у другому рядку вихідного файла.

Оцінки Ваша програма повинна виконуватися не більш 3 секунд.

Щоб одержати повний бал A за тест, потрібно, щоб кількість кроків x була менше або дорівнювала числу B, установленому тестуючою програмою. Зверніть увагу, що B не обов'язково є мінімумом. Фактично B залежить від кількості ходів, зроблених відповідно до деякої простої стратегії без надлишкових ходів, і від середньої кількості карток у стопках. Ви можете одержати частину балів, призначених за тест. Бали, що ви одержите, обчислюються шляхом округлення обчисленого по такій формулі числа до найближчого цілого:

- A, якщо $x \leq B$
- $\frac{2A(\frac{3}{2}B-x)}{B}$, якщо $B < x < \frac{3}{2}B$
- 0, якщо $x \leq \frac{3}{2}B$

Приклад вхідних та вихідних даних

FLAT.INP
5 0 7 8 1 4

FLAT.OUT
5 5 2 3 4 2 4 3 1 4 2

11.2.3 «Смужка землі»

Мешканці Дингилвілля хочуть знайти місце для будівництва аеропорту. У них є карта місцевості. Карта являє собою прямокутну сітку з одиничними квадратами, кожний із який визначений своїми координатами (x, y), де x — горизонтальна (захід-схід), а y — вертикальна (південь-північ) координата. Для кожного квадрата на карті показана висота місцевості.

Потрібно знайти прямокутну область найбільшої площі, що складається з квадратів карти і задовольняє таким умовам:

- різниця між висотою найвищого і найнижчого квадрата цієї області менша або рівна заданому числу C

- ширина цієї області, тобто кількість квадратів уздовж напрямку захід-схід, більша 100.

Якщо існують більше однієї шуканої області, ви повинні вивести тільки одну з них.

Обмеження

- $1 \le U \le 700$, $1 \le V \le 700$, де U і V визначають розміри карти. Зокрема, U визначає кількість квадратів у напрямку захід-схід, а V — у напрямку південь-північ.

- $0 \le C \le 10$

- $-30\ 000 \le H_{xy} \le 30\ 000$, де ціле число H_{xy} — висота місцевості квадрата з координатами (x, y), $1 \le x \le U$, $1leqy \le V$.

- Квадрат у південно-західному куті карти має координати $(1, 1)$, а в північно-східному — координати (U, V).

Вхідні дані Вхідні дані вводяться з текстового файла з ім'ям `LAND.INP`.

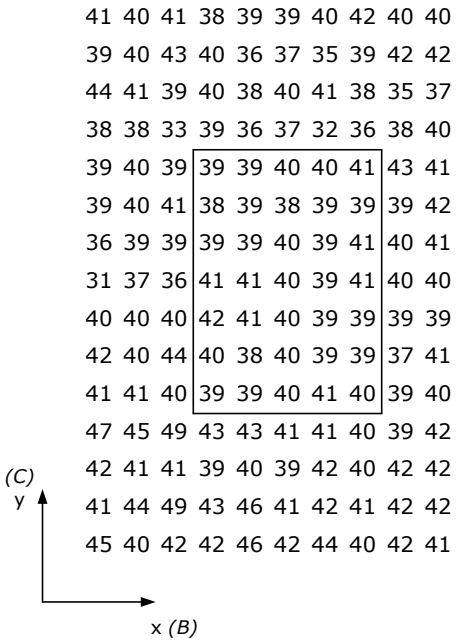
- Перший рядок містить три цілих числа: U, V і C.

- Кожен з наступних V рядків містить цілі числа Нху для $x = 1, \ldots, U$. Зокрема, H_{xy} є x-им числом у $(V - y + 2)$-му рядку вхідного файла.

Вихідні дані Вивід повинен здійснюється в текстовий файл з ім'ям `LAND.OUT`, що складається з одного рядка. У цьому рядку повинні бути чотири числа: X_{min}, Y_{min}, X_{max}, Y_{max}, де (X_{min}, Y_{min}) — координати південно-західного квадрата, а (X_{max}, Y_{max}) — координати північно-східного квадрата цієї області.

Оцінки Ваша програма повинна виконуватися не більш 60 секунд.

Часткові розв'язки в кожному тесті оцінюватися не будуть.


41 40 41 38 39 39 40 42 40 40
39 40 43 40 36 37 35 39 42 42
44 41 39 40 38 40 41 38 35 37
38 38 33 39 36 37 32 36 38 40
39 40 39 39 39 40 40 41 43 41
39 40 41 38 39 38 39 39 39 42
36 39 39 39 39 40 39 41 40 41
31 37 36 41 41 40 39 41 40 40
40 40 40 42 41 40 39 39 39 39
42 40 44 40 38 40 39 39 37 41
41 41 40 39 39 40 41 40 39 40
47 45 49 43 43 41 41 40 39 42
42 41 41 39 40 39 42 40 42 42
41 44 49 43 46 41 42 41 42 42
45 40 42 42 46 42 44 40 42 41
(C)
y
x (B)

Приклад вхідних та вихідних даних

LAND.INP
10 15 4 41 40 41 38 39 39 40 42 40 40 39 40 43 40 36 37 35 39 42 42 44 41 39 40 38 40 41 38 35 37 38 38 33 39 36 37 32 36 38 40 39 40 39 39 39 40 40 41 43 41 39 40 41 38 39 38 39 39 39 42 36 39 39 39 39 40 39 41 40 41 31 37 36 41 41 40 39 41 40 40 40 40 40 42 41 40 39 39 39 39 42 40 44 40 38 40 39 39 37 41 41 41 40 39 39 40 41 40 39 40 47 45 49 43 43 41 41 40 39 42 42 41 41 39 40 39 42 40 42 42 41 44 49 43 46 41 42 41 42 42 45 40 42 42 46 42 44 40 42 41

LAND.OUT
4 5 8 11

Розділ 12. Китай'2000

12.1 Завдання першого туру

12.1.1 «Паліндром»

Завдання Паліндром — це симетричний рядок, тобто він однаково читається як зліва направо, так і з права наліво. Ви повинні написати програму, яка за даним рядком визначить мінімальну кількість символів, які потрібно вставити в рядок для утворення паліндрома. Наприклад, вставкою двох символів рядок `Ab3bd` може бути перетворений в паліндром (`dAb3bAd` або `Adb3bdA`), а вставкою менше ніж двох символів паліндром в цьому прикладі отримати не можна.

Вхідні дані Вхідний файл має ім'я `PALIN.IN` і складається з двох рядків. Перший рядок містить одне ціле число — довжину N вхідного рядка, $3 \le N \le 5\,000$. Друга — рядок довжини N, яка складається з великих літер від `A` до `Z`, маленьких літер від `a` до `z` та цифр від `0` до `9`. Великі та маленькі літери вважаються різними.

Вихідні дані Вихідний файл має ім'я `PALIN.OUT` і складається з одного рядка. Цей рядок містить одне ціле, яке є шуканим мінімальним числом символів.

Приклад вхідних та вихідних даних

PALIN.IN
5 Ab3bd

PALIN.OUT
2

12.1.2 «Паркування»

Завдання Паркування біля Великої Стіни складається з довгого ряду місць, які повністю заповнені машинами. Один з країв вважається лівим, а другий — правим. Кожна з машин має свій тип. Типи перенумеровані натуральними числами і декілька машин можуть мати однаковий тип. Працівники вирішили впорядкувати машини в ряді так, щоб їх типи зростали зліва направо. Впорядкування складається з декількох етапів. На початку кожного етапу працівники одночасно виводять машини з їх місць, і потім ставлять машини на вільні місця, що утворилися як результат виїзду машин на цьому етапі, тобто міняють їх місцями. Кожен працівник пересуває тільки одну машину за етап, та деякі працівники можуть не приймати участі в пересуванні машин на етапі. Ефективність процесу впорядкування машин визначається кількістю етапів.

Припустимо, що на паркуванні знаходиться N машин і W працівників. Потрібно написати програму, яка знаходить таке впорядкування, при якому кількість етапів не перевищує значення $\frac{N}{W-1}$, заокругленого в більшу сторону, тобто $\left\lceil \frac{N}{W-1} \right\rceil$. Відомо, що мінімальна кількість етапів не перевищує $\left\lceil \frac{N}{W-1} \right\rceil$.

Розглянемо наступний приклад. На паркуванні знаходиться 10 машин типів `1`, `2`, `3`, `4` та четверо працівників. Початкове розташування машин зліва направо, задане їх типами — `2 3 3 4 4 2 1 1 3 1`.

Мінімальна кількість етапів — 3, і вони можуть бути виконані так, що розташування машин по типам після кожного з етапів наступне:

```
2 1 1 4 4 2 3 3 3 1      після першого етапу
2 1 1 2 4 3 3 3 4 1      після другого етапу
1 1 1 2 2 3 3 3 4 4      після третього етапу
```

Вхідні дані Вхідний файл має ім'я `CAR.IN`. Перший рядок вхідного файлу складається з трьох чисел. Перше — кількість машин N, $2 \le N \le 20\,000$. Друге — кількість типів машин M, $2 \le M \le 50$, тип машини — натуральне число від 1 до M. На паркуванні є хоча б одна машина кожного типу. Третє — кількість працівників W, $2 \le W \le M$. Другий рядок містить N цілих чисел, де i-те число є типом i-ої машини в ряді, рахуючи зліва направо.

Вихідні дані Вихідний файл має ім'я `CAR.OUT`. Перший рядок вихідного файла містить число R — кількість етапів. Далі йдуть R рядків, де $i+1$ рядок вихідного файла описує i-ий етап. Формат рядка, який описує етап, наступний. Перше число — кількість машин C, що пересуваються на даному етапі. Далі йдуть C пар чисел, що визначають пересування машин на даному етапі. Перше число пари — позиція, з якої машина пересувається, друге число пари — позиція, на яку машина пересувається. Позиція

машини є цілим числом від 1 до N та відраховується з лівого кінця ряду.

У випадку, коли існує декілька оптимальних розв'язків, видайте тільки одне з них.

Приклад вхідних та вихідних даних

CAR.IN
10 4 4 2 3 3 4 4 2 1 1 3 1

CAR.OUT
3 4 2 7 3 8 7 2 8 3 3 4 9 9 6 6 4 3 1 5 5 10 10 1

Оцінки Припустимо, що кількість етапів, виданих вашою програмою — R, а $\left\lceil \frac{N}{W-1} \right\rceil = Q$. Якщо деякий з R етапів видано некоректно, або після виконання всіх етапів типи машин не розташовані в зростаючому порядку, ви набираєте 0 балів за тест. В супротивному випадку, ваші бали будуть нараховані у відповідності до наступних правил:

$R \le Q$	100% балів
$R = Q + 1$	50% балів
$R = Q + 2$	20% балів
$R \ge Q$	0% балів

12.1.3 «Медіанна енергія»

Завдання В новому космічному експерименті приймають участь N об'єктів, занумерованих від 1 до N. Відомо, що N — непарне. Кожен об'єкт має унікальну, але невідому енергію Y, що виражається натуральним числом, $1 \le Y \le N$. Назвемо об'єктом з медіанною енергією такий об'єкт X, для якого кількість об'єктів з енергією, що менше ніж у X, дорівнює кількості об'єктів з енергією, що більше ніж у X. Потрібно написати програму, яка визначає об'єкт з медіанною енергією. Нажаль, порівнювати енергії можна тільки за допомогою пристрою, що для трьох різних об'єктів визначає, який з них має медіанну енергію.

Бібліотека Вам надається бібліотека `device` з трьома операціями:

- GetN, викликається тільки один раз на початку програми, не має аргументів, повертає значення N.
- Med3, викликається з трьома різними номерами об'єктів в якості аргументів. Повертає номер об'єкта, що має медіанну енергію.
- Answer, викликається тільки один раз в кінці програми. Єдиний аргумент — номер об'єкта X, що має медіанну енергію. Ця функція коректно завершує виконання вашої програми.

Бібліотека `device` генерує два текстових файла `MEDIAN.OUT` і `MEDIAN.LOG`. Перший рядок файла `MEDIAN.OUT` містить одне ціле число: номер об'єкта, переданого бібліотеці при виклику Answer. Другий рядок містить одне ціле число: кількість викликів Med3, яку було зроблено Вашою програмою. Діалог між вашою програмою і бібліотекою записується в файл `MEDIAN.LOG`.

Експериментування Ви можете проводити експерименти з бібліотекою, створивши текстовий файл `DEVICE.IN`. Файл має містити два рядка. Перший рядок повинен містити одне ціле число: кількість об'єктів N. Другий рядок повинен містити цілі числа від 1 до N в деякому порядку: i-е число є енергією об'єкта з номером i.

Приклад вхідних та вихідних даних

DEVICE.IN
5 2 5 4 3 1

Файл `DEVICE.IN`, зазначений вище, описує 5 об'єктів с їх енергіями:

Номер об'єкта:	1	2	3	4	5
Енергія:	2	5	4	3	1

Нижче наведена коректна послідовність з п'ятьма зверненнями до бібліотеки:

1. GetN (у `Pascal`) або `GetN()` (у `C`/`C++`) повертає 5.
2. Med3(1,2,3) повертає 3.
3. Med3(3,4,1) повертає 4.
4. Med3(4,2,5) повертає 4.
5. Answer(4)

Обмеження

- Кількість об'єктів N непарна та $5 \le N \le 1\ 499$.
- Для об'єкта з номером i виконується $1 \le i \le N$.
- Для енергії об'єкта Y виконується $1 \le Y \le N$ і всі енергії різні.
- В ході виконання програми дозволяється використовувати не більше 7 777 викликів функції Med3.
- Ваша програма не повинна читати або писати ніякі файли.

12.2 Завдання другого туру

12.2.1 «Поштові відділення»

Завдання Вздовж прямої дороги розташовані сала. Дорога подається цілочисельною віссю, а розташування кожного села задається одним цілим числом — координатою на цій осі. Ніякі дві села не мають однакових координат. Відстань між двома селами обчислюється як модуль різниці їх координат.

У деяких, не обов'язково всіх, селах будуть збудовані почтові відділення. Село та розташоване в ній поштове відділення мають одинакові координати. Почтові відділення необхідно розташувати в селах таким чином, щоб загальна сума відстаней від кожного села до найближчого до нього поштового відділення була мінімальною.

Напишіть програму, яка по заданим координатам сіл і кількості поштових відділень знаходить таке розташування поштових відділень по селам, при якому загальна сума відстаней від кожного села до його найближчого поштового відділення буде мінімальною.

Вхідні дані Вхідний файл має ім'я `POST.IN`. В першому рядку міститься два цілих числа: перше число — кількість сіл V, $1 \le V \le 300$, друге число — кількість поштових відділень P, $1 \le P \le 30$, $P \le V$. Другий рядок містить V цілих чисел в зростаючому порядку, що є координатами сіл. Для кожної координати X виконується $1 \le X \le 10\ 000$.

Вихідні дані Вихідний файл має ім'я `POST.OUT`. Перший рядок містить одно ціле число S — загальну суму відстаней від кожного села до його найближчого поштового відділення для розташування поштових відділень, що описане в другому рядку. Другий рядок містить P цілих чисел у зростаючому порядку. Ці числа є шуканими координатами поштових відділень. Якщо для заданого розташування сіл є декілька розв'язків, то програма повинна знайти один з них.

Приклад вхідних та вихідних даних

POST.IN
10 5 1 2 3 6 7 9 11 22 44 50

POST.OUT
9 2 7 22 44 50

Оцінки Якщо вихідний файл не відповідає описаним вимогам — оцінка 0. У протилежному випадку, оцінка буде обчислена по таблиці наступним чином: якщо знайдена сума — S, а мінімальна сума — S_{min}, то необхідно обчислити $q = S/(S_{min})$ і знайти відповідну оцінку c в нижньому рядку.

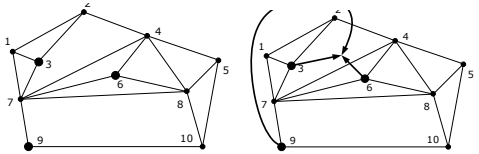
$q = S/(S_{min})$	$q = 1.0$	$1.0 < q \le 1.1$	$1.1 < q \le 1.15$	$1.15 < q \le 1.2$
c	10	5	4	3

$q = S/(S_{min})$	$1.2 < q \le 1.25$	$1.25 < q \le 1.3$	$1.3 < q$
c	2	1	0

12.2.2 «Стіни»

Завдання В деякій країні стіни збудовані таким образом, що кожна стіна з'єднує рівно два міста, і стіни не перетинають одна одну. Таким чином, країна розділена на окремі області. Щоб добратись із однієї області до іншої необхідно або пройти через місто, або перетнути стіну. Два довільних міста A и B можуть з'єднуватись не більш ніж однією стіною (один кінець стіни в місті A, а інший — в місті B). Більш того, завжди існує шлях із міста A у місто B, що проходить повз стін та через міста.

Існує клуб, члени якого мешкають в містах. В кожному місті може жити або один член клуба, або взагалі жодного. Інколи у членів клуба виникає бажання зустрітись всередині однієї з областей, але не в місті. Щоб попасти в потрібну область, кожен член клуба має перетнути деяку кількість стін, можливо рівне нулю. Члени клуба мандрують на велосипедах. Вони не бажають приїздити ні в одне місто з-за інтенсивного руху на міських вулицях. Вони також бажають перетнути мінімально можливу кількість стін, оскільки перетинати стіну з велосипедом досить важко. Виходячи з цього, необхідно знайти таку оптимальну область, щоб сумарна кількість стін, які потрібно перетнути членам клуба, яка називається сумою перетинів, була мінімальною із усіх можливих.



Міста занумеровані цілими числами від 1 до N, де N — кількість міст. На мал. 1 занумеровані вершини позначають міста, а лінії, що з'єднують вершини, позначають стіни. Припустимо, що членами клуба є три людини, які живуть в містах з номерами 3, 6 та 9. Тоді оптимальна область і відповідні маршрути руху членів клуба показані на мал. 2. Сума перетинів дорівнює 2, так як членам клуба треба перетнути дві стіни: людині із міста 9 треба перетнути стену між містами 2 та 4, людині із міста 6 потрібно перетнути стіну між містами 4 і 7, а людина із міста 3 не перетинає стін взагалі.

Потрібно написати програму, яка по заданій інформації про міста, області і місця мешкання членів клуба знаходить оптимальну область і мінімальну суму перетинів.

Вхідні дані Вхідний файл має ім'я `WALLS.IN`. Перший рядок містить одне ціле число: кількість областей M, $2 \le M \le 200$. Другий рядок містить одне ціле число: кількість міст N, $3 \le N \le 250$. Третій рядок містить одне ціле число: кількість членів клуба L, $1 \le L \le 30$, $L \le N$. Четвертий рядок містить L різних цілих чисел у зростаючому порядку: номера міст, де живуть члени клуба.

Решта файла містить $2M$ рядків, по парі рядків для кожної з M областей. Перші два рядка з них описують першу область, другі два рядка — другу область, і т.д. В кожній парі перший рядок містить кількість міст I на границі області, другий рядок містить номера цих I міст в порядку обходу границі області за годинниковою стрілкою. Єдиним виключенням є остання область — це розташована зовні область, що оточує всі міста і інші області, і для неї порядок слідування міст на границі задається против годинникової стрілки. Порядок опису областей у вхідному файлі задає цілі номера цим областям. Область, що описана першою у вхідному файлі має номер 1, описана другою — номер 2, і т.д. Зверніть увагу, що вхідний файл містить опис всіх областей, що утворені містами і стінами, включаючи зовнішню область.

Вихідні дані Вихідний файл має ім'я `WALLS.OUT`. Перший рядок цього файла містить одне ціле число: мінімальну суму перетинів. Другий рядок містить одне ціле число: номер оптимальної області. Якщо існує декілька різних розв'язків, вам потрібно видати лише одне з них.

Приклад вхідних та вихідних даних Наведені вхідний та вихідний файли відповідають розглянутому у тексті прикладу.

WALLS.IN
10 10 3 3 6 9 3 1 2 3 3 1 3 7 4 2 4 7 3 3 4 6 7 3 4 8 6 3 6 8 7 3 4 5 8 4 7 8 10 9 3 5 10 8 7 7 9 10 5 4 2 1

WALLS.OUT
2 3

12.2.3 «Конструювання з блоків»

Завдання Одиничним кубом назвемо куб $1 \times 1 \times 1$, вершини якого мають цілочисельні координати x, y, z. Два одиничних куба можуть утворювати новий об'єкт шляхом з'єднання їх по грані. Кубоїдом назвемо непусте з'єднання одиничних кубів (див. малюнок 1). Об'єм кубоїда дорівнює кількості одиничних кубів, із яких його складено. Блоком називається кубоїд з об'ємом не більше чотирьох. Будемо казати, що два блока мають однаковий тип, якщо один може бути отриманий із іншого за допомогою поворотів та зсувів блока (зауважимо, що відбиття робити заборонено). Всього існує 12 типів блоків (див. малюнок 2). Кольори об'єктів на малюнку зображені лише для ілюстрації структури об'єктів, і вони не несуть ніякого змістового навантаження.

Набір кубоїдів D назвемо декомпозицією кубоїда S, якщо об'єднання всіх кубоїдів із D дорівнює S, і ніякий одиничний куб із кубоїда S не належить одночасно двом різним кубоїдам із D.

Напишіть програму, яка по заданому опису типів блоків і кубоїда S визначає найменший за кількістю елементів набір блоків, що є декомпозицією кубоїда S. Вам потрібно видати тільки типи цих блоків. Кожен тип

має бути виведений стільки разів, скільки блоків цього типу зустрічається в декомпозиції.

Вхідні дані Розташування одиничного куба в просторі будемо задавати за допомогою цілочисельних координат x, y, z такої його вершини, для якої сума $x + y + z$ мінімальна.

Вхідний файл з описом типів блоків має ім'я `TYPES.IN`. Вміст цього файла однаковий для всіх тестів. Файл містить опис всіх 12 блоків, зображених на малюнку 2, відсортованих за номером типу. Кожен з блоків описується групою слідуючи один за одним рядків наступним чином. Перший рядок складається з цілого числа I, ідентифікуючого тип блока ($1 \le I \le 12$). Другий рядок містить об'єм V блока ($1 \le V \le 4$). Далі ідуть V рядків, що описують положення одиничних кубів, із яких складається блок. Кожен із цих V рядків складається із трьох цілих чисел x, y, z ($1 \le x, y, z \le 4$).

Вхідний файл, що описує кубоїд, має ім'я `BLOCK.IN`. Перший рядок файла містить об'єм V кубоїда ($1 \le V \le 50$). Наступні V рядків описують положення одиничних кубів, із яких складається кубоїд. Кожен з цих V рядків складається із трьох цілих чисел x, y, z ($1 \le x, y, z \le 7$).

Вихідні дані Вихідній файл має ім'я `BLOCK.OUT`. Перший рядок файла містить одне ціле число M, яке дорівнює кількості блоків в мінімальному наборе, що є декомпозицією заданого кубоїда. Другий рядок містить список номерів типів блоків, що містяться в наборі. Номери можуть іти в довільному порядку.

У випадку коли існує декілька розв'язків потрібно вивести тільки один з них.

TYPES.IN

```
1
1
1 1 1
2
2
1 1 1
1 2 1
3
3
1 1 1
1 2 1
1 3 1
4
3
1 1 1
1 2 1
1 1 2
5
4
1 1 1
1 2 1
1 3 1
1 4 1
6
4
1 1 1
1 2 1
1 1 2
1 2 2
7
4
1 1 1
1 2 1
1 1 2
1 1 3
8
4
1 1 1
1 2 1
1 3 1
1 2 2
9
4
1 2 1
1 3 1
1 1 2
1 2 2
```

BLOCK.IN

```
18
2 1 1
4 1 1
2 3 1
4 3 1
2 1 2
3 1 2
4 1 2
1 2 2
2 2 2
3 2 2
4 2 2
2 3 2
3 3 2
4 3 2
4 2 3
4 2 4
4 2 5
5 2 5
```

BLOCK.OUT

```
5
7 10 2 10 12
```

TYPES.IN
10 4 2 1 1 1 2 1 2 2 1 2 1 2 11 4 1 1 1 1 2 1 2 2 1 1 1 2 12 4 2 2 1 2 1 2 1 2 2 2 2 2

BLOCK.IN

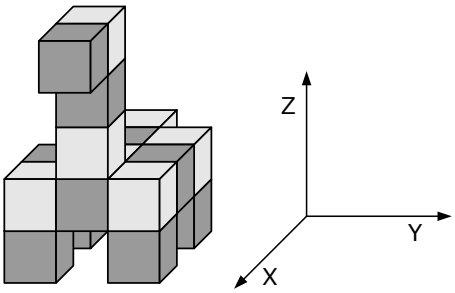
BLOCK.OUT

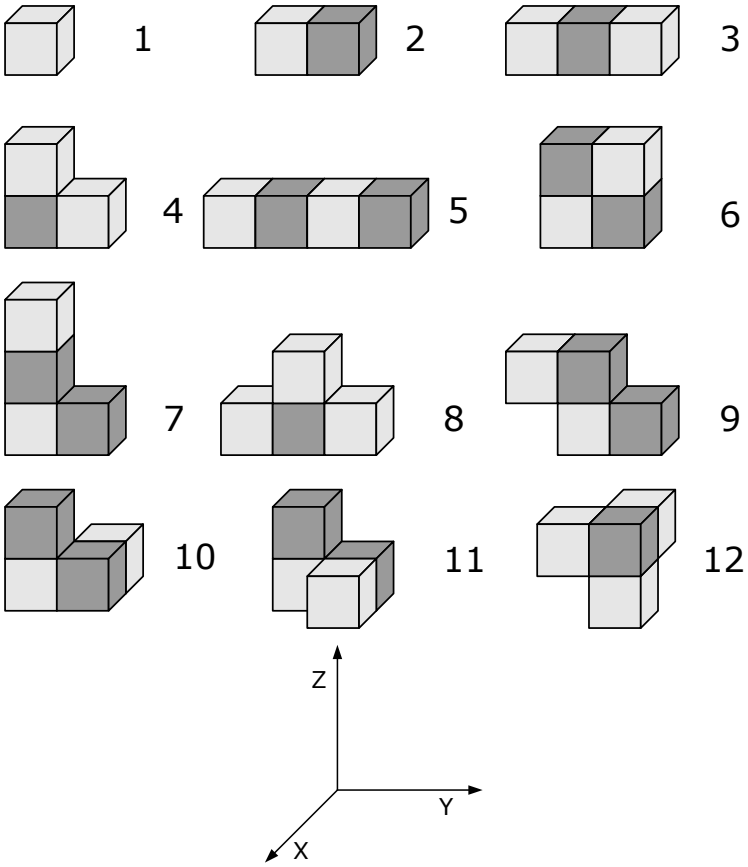
Зауваження

1. Вхідний файл `BLOCK.IN` описує кубоїд «кінь», зображений на малюнку 1.

2. Всі інші варіанти вмісту другого рядка вихідного файлу, що описує типи використаних блоків, наведені нижче:

```
2  7  10  11  12
2  7  11  11  12
4  4  7   10  11
4  4  9   10  11
```





Розділ 13. Фінляндія'2001

13.1 Завдання першого туру

13.1.1 «Мобільні телефони»

Припустимо, що в регіоні Тампере базові станції забезпечення мобільного телефонного зв'язку четвертого покоління діють наступним чином. Регіон поділено на квадрати. Квадрати утворюють матрицю розміра $S \times S$, рядки та стовпчики якої занумеровані від 0 до $S-1$. У кожному квадраті знаходи-

тся базова станція. Кількість працюючих мобільних телефонів у квадраті мож мінятися, так як телефони можеть перемещуватися з одного квадрата в інший, і телефони можуть включатися або виключатися. У деякі моменти часу кожна базова станція передає головній базовій станції звіт про зміненя кількості працюючих телефонів і свої координати (номер рядка і номер стовпчика відповідно).

Напишіть програму, яка отримує ці звіти та відповідає на запити про поточну загальну кількість мобільних телефонів, що працюють у деякій прямокутній області.

Вхідні та вихідні дані Цілочисельні вхідні дані повинні бути зчитані з стандартного вхідного потоку. Вхідні дані закодовано наступним чином. Кожен рядок містить одну команду. Команда складається з коду, та набору цілочисельних параметрів у відповідності до наступної таблиці:

Завершает программу. Эта команда выдается только один раз и должна быть последней.

0	S	Ініціалізує матрицю розміром $S \times S$ нулями. Ця команда видається тільки один раз і повинна бути першою.
1	$X\ Y\ A$	Добавляє A до кількості працюючих телефонів у комірці таблиці (X, Y). A може бути як додатнім, так і від'ємним
2	$L\ B\ R\ T$	Запитує поточну сумарну кількість працюючих телефонів у комірках (X, Y), де $L \le X \le R$, $B \le Y \le T$
3		Завершую програму. Ця команда видається тільки один раз і повинна бути останньою.

Значення завжди будуть у наведеному діапазоні, їх перевіряти не потрібно. Зокрема, якщо A — від'ємне, можна вважати що значення у комірці не стане менше за нуль. Індексація починається з 0, тобто для таблиці з розмірами 4×4, маємо $0 \le X \le 3$ та $0 \le Y \le 3$.

Ваша програма нічого не повинна відповідати у відповідь на команди, що відрізняються від команд з кодом 2. Якщо надійшла команда 2, то ваша програма повинна відповісти на запит, видавши у стандартний вихідний потік рядок, що містить єдине ціле число.

Приклад

stdin	stdout	Пояснення
0 4		Ініціалізується матриця розмірами 4×4.
1 1 2 3		Значення в квадраті $(1, 3)$ збільшується на 3.
2 0 0 2 2		Запитується поточна сума значень з прямокутника $0 \le X \le 2$, $0 \le Y \le 2$.
	3	Відповідь на запит.
1 1 1 2		Значення у квадраті $(1, 1)$ збільшується на 2.
1 1 2 -1		Значення у квадраті $(1, 2)$ зменьшується на 1.
2 1 1 2 3		Запитується поточна сума значень з прямокутника $1 \le X \le 2$, $1 \le Y \le 3$.
	4	Відповідь на запит.
3		Програма завершується.

Обмеження

Розмір матриці	$S \times S$	$1 \times 1 \le S \times S \le 1\,024 \times 1\,024$
В будь-якому квадраті матриці у будь-який час може знаходитися V працюючих телефонів	V	$0 \le V \le 2^{15} - 1$ (=32 767)
Зміна кількості телефонів у квадраті матриці	A	$-2^{15} \le A \le 2^{15} - 1 (= 32\,767)$
Кількість команд у вхідному потоці	U	$3 \le U \le 60\,002$
Максимальна сумарна кількість телефонів у всій матриці	M	$M = 2^{30}$

У 16-ти з 20-ти тестів розмір матриці буде не більше 512×512.

13.1.2 «Гра ioiwari»

Гра с намистинками та ямками є однією з найстаріших форм розваг. В цій задачі подається варіант гри ioiwari подібної гри, спеціально розроблений для міжнародної олімпіади з інформатики.

У гру грають два гравця. Для гры використовується дошка, що має сім ямок, розташованих по колу. Крім того, кожний гравець має свій банк, спочатку пустий. На початку гри 20 намистинок випадково розподіляються по усіх сіми ямках так, що в кожну ямку потрапляє від 2-х до 4-х намистинок. Два гравця ходять по черзі. Під час свого ходу гравець оберає довільну непусту ямку, забирає з неї всі намистинки в руку і виконує ряд дій, доки в його руці залишається хоча б одна намистинка. Починаючи з ямки, розташованої одразу за тою, з якої він брав намистинки на початку ходу, він послідовно переходить за годинниковою стрелкою від однієї поточної лунки к другої, та виконує наступні дії:

- Якщо у руці гравця більше однієї намистинки, тоді якщо поточна ямка вже містить 5 намистинок, то гравець забирає з неї одну намистинку та переносить її у свій банк; якщо ж у поточній ямці менше ніж п'ять намистинок, то гравець кладе в неї одну намистинку зі своєї руки.

- Якщо ж у руці гравця рівно одна намистинка, тоді якщо поточна ямка містить від однієї до чотирьох намистинок, то всі намистинки з цієї лунки та намистинка з руки гравця переносяться у його банк; у протилежному випадку (тобто поточна ямка містить 0 або 5 намистинок), намистинка з руки гравця переноситься в банк противника.

Хід вважається виконаним, коли у гравця у руці не залишиться жодної намистинки.

Гра закінчується, коли по закінченні чергового хода всі ямки будуть пустими. Переможцем вважається той з гравців, у банку якого буде більше намистинок.

Перший гравець завжди має виграшну стратегію. Ви маєте написати програму, яка грає в ігру ioiwari за першого гравця і виграє. Майте на увазі, що другий гравець буде грати оптимально, тобто при першому неправільному ході з боку вашої програми воне не виграє.

Вхід та вихід Ваша програма зчитує вхідні дані з стандартного потока ввода і записує вихідні дані в стандартний потік виводу. Ваша програма грає за першого гравця, а суперник — за другого. На початку роботи ваша програма повинна зчитати рядок з 7-ми цілих чисел $p_1, \ldots, p_7$: початкова кількість намистинок з номерами $1, \ldots, 7$ відповідно (ямки пронумеровані від 1 до 7 за годинниковою стрілкою).

Після цього починається гра. Ваша програма повинна грати наступним чином:

- Якщо хід робить ваша програма, то вона повинна вивести у стандартний потік виводу номер ямки, з якої забираються усі намистенки на початку ходу.

- Якщо хід робить суперник вашої програми, то ваша програма повинна зчитати з стандартного потоку вводу номер ямки, з якої усі намистенки на початку свогоходу забрав суперник.

Допоміжні програмні засоби Вам надано програму (`ioiwari2` для `Linux`, `ioiwari2.exe` для `Windows`), яка для однієї конкретної початкової ситуації грає за другого гравця оптимальним чином. В перший рядок стандартного потоку виводу ця програма виводить дані, що описують початкове

розподілення намистенок по ямкам, яке повинна зчитати ваша програма на початку своєї работи:

```
4 3 2 4 2 3 2
```

Після цього надана вам програма буде грати у гру, зчитуя ходи першого гравця з стандартного потоку вводу і записувати свої ходу у стандартний потік виводу. Ви можете запустити вашу програму, і програму `ioiwari2` у окремих окнах і вручну організувати гру між ними. Програма `ioiwari2` записує весь діалог в файл `ioiwari.out`

Приклад Нижче наведено коректний опис 6-ти ходів і вміст ямок і банків після кожного ходу.

Операція/мітки ямок	1.	2.	3.	4.	5.	6.	7.	Банк1	Банк2
Початкова ситуація	4	3	2	4	2	3	2	0	0
Хід першого гравця:2	4	0	3	5	0	3	2	3	0
Хід другого гравця:3	4	0	0	4	1	4	0	3	4
Хід першого гравця:5	4	0	0	4	0	0	0	8	4
Хід другого гравця:4	0	0	0	0	1	1	1	8	9
Хід першого гравця:5	0	0	0	0	0	0	1	10	9
Хід другого гравця:7	0	0	0	0	0	0	0	11	9

Оцінки Якщо ваша програма при тестуванні виграє, то ви отримуєте 4 бали за тест, у випадку поразки ви отримуєте 2 бали, і 0 балів — у випадку поразки.

13.1.3 «Twofive»

Таємні повідомлення між Санта-Клаусом і його маленькими помічниками звичайно кодуються так званою 25-мовою. В цій мові використовується 25-алфавіт, який співпадає з латинським алфавітом з одним виключенням —- в ньому відсутня буква `Z`, тобто 25-алфавіт містить 25 латинських букв від `A` до `Y` у тому ж порядку, що і латинський алфавіт. Кожне слово в 25-мові складається рівно з 25 різних букв. Слово записується у таблиці разміра 5×5, рядок за рядком. Наприклад, слово `ADJPTBEKQUCGLRVFINSWHMOXY` буде записано так:

A	D	J	P	T
B	E	K	Q	U
C	G	L	R	V
F	I	N	S	W
H	M	O	X	Y

Правильним словом 25-мови вважається слово, букви якого у кожному рядку та кожному стовпчику розташовані за зростанням їх номера в алфавіте. Тому слово `ADJPTBEKQUCGLRVFINSWHMOXY` є правильним словом, а слово `ADJPTBEGQUCKLRVFINSWHMOXY` — ні (зростаючий порядок порушується у другому та третьому стовпчиках).

Лексикон Санта-Клауса складається із усіх правильних слів 25-мови. Слова розташовані за зростанням у лексикографічному порядку і занумеровані, починаючи з одиниці. Наприклад, у лексиконі Санта-Клауса слово `ABCDEFGHIJKLMNOPQRSTUVWXY` має номер 1, а слово `ABCDEFGHIJKLMNOPQRSUTVWXY` має номер 2. У другому слові, у порівнянні з першим, букви `U` і `T` поміняні місцями.

Нажаль, лексикон Санта-Клауса величезний. Напишіть програму, яка визначить порядковий номер по заданому слову з лексикона Санта-Клауса, а по заданому порядковому номеру визначить відповідне слово. У лексиконі Санта-Клауса не більше 2^{31} слів.

Вхідні дані Вхідний файл `twofive.in` містить два рядка. Якщо перший рядок містить `W`, тоді другий рядок містить правильне слово 25-мови, тобто є рядком з 25 символів. Якщо перший рядок — містить `N`, то другий містить порядковий номер існуючого слова 25-мови.

Вихідні дані Вихідний файл має назву `twofive.out`. Якщо другий рядок вхідного файлу містить слово 25-мови, то єдиний рядок вихідного файлу повинен містити його порядковий номер у лексиконі Санта-Клауса. Якщо другий рядок вхідного файлу містить число, то єдиний рядок вихідного файлу повинен містити слово 25-мови з цим номером.

Приклади вхідних та вихідних даних

twofive.in	twofive.out
W ABCDEFGHIJKLMNOPQRSUTVWXY	2

twofive.in	twofive.out
N 2	ABCDEFGHIJKLMNOPQRSUTVWXY

13.2 Завдання другого туру

13.2.1 «Score»

Score — це гра для двох гравців. Ігровий майданчик складається з N точок, занумерованих від 1 до N, частину з яких поєднано стрілками. Кожна стрілка йде від однієї точки до деякої іншої точки. Кожна тоска належить одному з двох гравців. Гравець, якому належить точка, зветься її власником. Крім того, кожній точці приписане деяке додатне число. Всі приписані числа різні. Точка з номером 1 є початковою. На початку гри кожен з гравців має 0 очок.

Гра полягає у наступному. Будемо позначати буквою C поточну точку, тобто точку у якій ми знаходимось на початку хода. На початку гри C є точкою з номером 1. Хід у грі робить власник C, при цьому виконуються такі операції:

1. Якщо значення, приписане C, більше, ніж поточна кількість очок у власника C, то значення числа, приписанного C, стає новою кількістю очок для власника C. У протилежному випадку, число очок у власника C не змінюється. Кількість очок у другого гравця у обох випадках не змінюється.

2. Далі власник C обирає за своїм бажання одну із стрілок, що виходять з поточної точки, і точка, на яку ця стрілка вказує, стає новою поточною. Зазначимо, що один гравець може зробити поспіль декілька ходів.

Гра закінчується, коли після деякого хода поточною стає початкова точка. Переможцем вважається гравець, який має більшу кількість очок у кінці гри.

Набір стрілок між точками завжди такий, що:

- з довільної точки виходить не менше однієї стрілки;
- довільна точка P досяжна із початкової точки, тобто існує така послідовність стрілок, по якій можна дійти від початкової точки до P;
- гарантується, що гру завжди можна закінчити за скінчену кількість кроків.

Напишіть програму, яка грає в цю гру та виграє. Набори точок та стрілок у всіх іграх, у які буде запропоновано грати вашій програмі для її оцінки, організовані так, що ваша програма завжди зможе виграти незалежно від того чи належить їй перший хід чи ні.

Введення та виведення Ваша програма зчитує вхідні дані з стандартного вхідного потоку, та записує вихідні дані у стандартний вихідний потік. Ваша програма — це Гравець 1, а суперник — Гравець 2. Ваша програма посинаэ свою работу з считування наступних вхыдних даних з стандартного потоку вводу.

Перший рядок вхідних даних містить одне ціле число — кількість точок N, $1 \leq N \leq 1\,000$. Кожний з наступних N рядків містить по N цілих чисел (нулей або одиниць), що описують стрілки. Якщо на гральному полі існує стрілка, що направлена від точки i до точки j, то j-е число у i-му рядку (з N цих рядків) рівно 1, у противному випадку — 0.

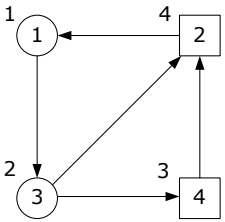
Наступний рядок містить N цілих чисел, що позначать володарів кожної з N точок. Якщо точкою володіє Гравець 1, то i-те число — одиниця, а якщо Гравець 2, то двійка.

Наступний рядок містить N цілих чисел, що позначають значення, приписані точкам. Якщо i-те число у цьому рядку рівне j, то i-ій точці приписане число j. Всі значення, приписані точкам, різні та знаходяться у діапазоні від 1 до N ($1 \leq j \leq N$).

Після зчитування даних починається гра, стартовою точкою при цьому завжди буде точка з номером 1. Ваша програма повинна грати наступним чином:

- якщо хід робить ваша програма (вона є володарем поточної точки), то вона повинна вивести у стандартний потік виводу номер точки, в яку потрібно перейти по вибраній вашей програмой стрілці;
- якщо ж хід робить ваш суперник, то ваша програма повинна зчитати з стандартного потоку вводу номер точки, в яку потрібно перейти по стрілці вибраною суперником;
- коли поточною знову стане точка з номером 1, програма повинна завершувати свою роботу.

Розглянемо наступний приклад для грального поля, зображеного на мал. 1. Точки, зображені у вигляді кіл, належать Гравцю 1, а у вигляді квадратів — Гравцю 2. Значення, приписані точкам, поміщенні всередені кіл та квадратів. Номера точок розташовані зовні геометричних фігур.



Наступні вхідні та вихідні дані описують гру, зіграну для зображеного на мал. 1 грального поля.

stdin	stdout	Пояснення
4		N
0 1 0 0		Інформація о стрілках, що виходять з точки 1
0 0 1 1		Інформація о стрілках, що виходять з точки 2
0 0 0 1		Інформація о стрілках, що виходять з точки 3
1 0 0 0		Інформація о стрілках, що виходять з точки 4
1 1 2 2		Володарі точок
1 3 4 2		Числа, приписані точкам
	2	Хід гравця 1
	4	Хід гравця 1
1		Гравець 2 ходить у початкову точку — гра закінчується.

У резільтаті цієї гри Гравець 1 набирає 3 очки, а Гравець 2 — 2. Гравець 1 виграв.

13.2.2 «Подвійне кодування»

Удосконалений Стандарт Кодування (УСК) використовує новий потужний алгоритм кодування. Він працює з трьома блоками по 128 біт. За блоком повідомлення p та блоком ключа k, функція кодування E УСК повертає закодований блок c:

$$c = E(p, k)$$

Оберненою до фукції кодування є функція розкодування D, така що:

$$D(E(p, k), k) = p$$

$$E(D(c, k), k) = c$$

У подвійному УСК послідовно застосовуються два незалежних ключа k_1 та k_2, спочатку k_1, потім k_2:

$$c_2 = E(E(p, k_1), k_2)$$

В цій задачі також задане ціле число s. Тільки ліві $4 \times s$ біта всіх ключів є суттевими, інші біти (праві 128 мінус $4 \times s$ бітів) є нульовими.

Вам необхідно відновити пари ключів кодування для повідомлень закодованих за допомогою подвійного УСК. Вам задано як вихідний текст p так і закодований c_2, а також структура ключа кодування за допомогою значення s.

Ви повинні здати відновлені ключі, а не програму, що їх відновлює!

Вхідні дані Вам надається десять прикладів задачі в текстових файлах з назвами від `double1.in` до `double10.in`. Кожен файл скдадається з трьох рядків. Перший рядок містить ціле число s, другий рядок блок p, а третій блок c_2, що був отриманий з p за допомогою подвійного УСК. Другий та третій рядки вхідного файлу — рядки, що складаються з 32 шеснадцядкових цифр (`0..9`, `A..Z`). Бібліотека надає процедуру, що конвертує рядки у блоки. Для всіх файлів розв'язок існує.

Вихідні дані Ви повинні здати десять вихідних файлів, що відповідають десяти вхідним файлам. Кожен вихідний файл складається з трьох рядків. Перший рядок містить текст:

```
#FILE double I
```

де I — номер відповідного вхідного файлу. Другий рядок містить знайдений блок k_1, а третій рядок блок k_2, так що:

$$c_2 = E(E(p, k_1), k_2)$$

Обидва блоки повинні бути записані як рядки, що складаються з шіснадцяткових цифр (`0..9`, `A..Z`). Бібліотека надає процедуру, що конвертує блоки у рядки. Якщо існує декілька розв'язків, ви повинні здати лише один з них.

Приклад Цей приклад наведено для вхідного файлу з номером 0.

double0.in
1 0011223344556677↓ 8899AABBCCDDEEFF 6323B4A5BC16C479↓ ED6D94F5B58FF0C2

Можлива відповідь
#FILE double 0 A000000000000000↓ 0000000000000000 7000000000000000↓ 0000000000000000

Обмеження Для числа s кількості суттєвих шіснадцяткових цифр ключа, вважається що $1 \leq s \leq 5$.

Підказка Добре написана програма може відновляти ключи менше ніж за 10 секунд для будь-якого допустимого вхідного файлу.

13.2.3 «Склад»

У фінській компанії, що випускає високотехнологічне обладнання, є великий прямокутний склад. На складі працюють менеджер і робітник. Сторони прямокутника, що зображають склад на плані, у порядку їх обхода називають лівою, верхньою, правою і нижньою. Підлога склада розділена на одиничні квадрати. На плані вони утворюють ряди і колонкі. Ряди занумеровані цілими числами $1, 2, \ldots$, починаючи з верхньої сторони, а колонки занумеровані цілими числами $1, 2, \ldots$, начинаючи з лівої сторони.

Для зберігання технологічних пристроїв на складі є контейнери. Контейнери мають індивідуальні идентифікаційні номери. Кожен контейнер займає один квадрат. Склад настільки великий, що кількість контейнеров, що зберігаються, завжди менше, ніж кількість рядів, і менше, ніж кількість колонок. Контейнери зі складу не вивозяться, але інколи на склад надходять нові контейнери. Вхід до складу розташований у лівому верхньому куті.

Робітник розташував контейнери поблизу лівого верхнього кута складу деяким чином, щоб можна було простіше знайти їх по ідентифікаційним номерам. Він використовує наступний метод.

Нехай необхідно розташувати на складі контейнер з ідентифікаційним номером k (будемо називати його контейнер k). Робітник проходить по першому ряду зліва направо і шукає перший контейнер з ідентифікаційним номером, більшим ніж k. Якщо він не знаходить такого контейнера, то контейнер k встановлюється безпосередньо після самого правого контейнера у цьому ряду. Якщо ж такий контейнер l знайдено, то контейнер l замінюється на контейнер k, а контейнер l розміщується у наступному ряді за допомогою того ж метода. Якщо робітник досягає ряда, у якому немає контейнерів, то контейнер встановлюється у самому лівому квадраті цього ряда.

Припустимо, що контейнери `3`, `4`, `9`, `2`, `5`, `1` прибули на склад саме в такому порядку. Тоді розташування контейнерів на складі буде таким:

```
1 4 5 2 9 3
```

Напишіть программу, яка за заданим розташуванням контейнерів обчислює усі можливі послідовності надходження контейнерів.

Вхідні дані Перший рядок вхідного файлу `depot.in` містить єдине ціле число R: кількість рядів, у яких є контейнери. Наступні R рядків містять інформацію о рядах $1, \ldots, R$. Першим у кожному з цих рядків знаходиться число M: кількість контейнерів у ряду. Після цього у рядку слідують M цілих чисел: ідентифікаційні номери контейнерів у ряду зліва направо. Всі ідентифікаційні номери контейнерів I задовільняють обмеженню: $1 \le I \le 50$.

Обозначимо загальну кількість контейнерів на складі через N, $1 \leq N \leq 13$

Вихідні дані Вихідний файл `depot.out` містить стільки рядків, скільки існує можливих різних послідовностей надходження контейнерів. Кожен з цих рядків містить N цілих чисел — ідентифікаційні номери контейнерів у можливій послідовності прибуття. Кожен рядок описує послідовність, яка не записана ні в одному іншому рядку.

Приклади вхідних та вихідних даних

depot.in
3 3 1 4 5 2 2 9 1 3

depot.out
3 2 1 4 9 5 3 2 1 9 4 5 3 4 2 1 9 5 3 2 4 1 9 5 3 2 9 1 4 5 3 9 2 1 4 5 3 4 2 9 1 5 3 4 9 2 1 5 3 2 4 9 1 5 3 2 9 4 1 5 3 9 2 4 1 5 3 4 2 9 5 1 3 4 9 2 5 1 3 2 4 9 5 1 3 2 9 4 5 1 3 9 2 4 5 1

depot.in
2 2 1 2 1 3

depot.out
3 1 2 1 3 2

Розділ 14. Південна Корея'2002

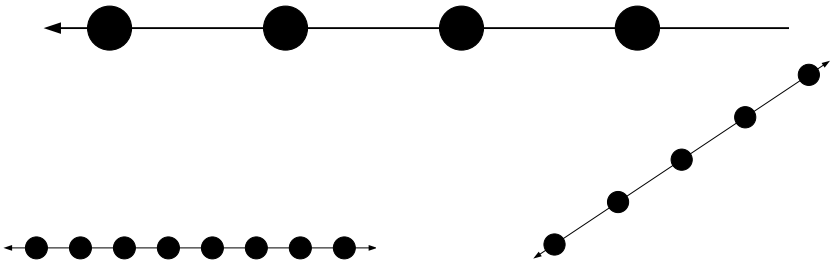
14.1 Завдання першого туру

14.1.1 «Шкідлива жаба»

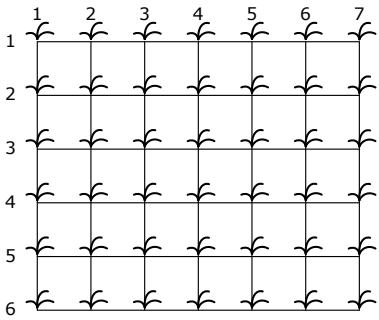
Завдання Маленькі шкідливі жаби «ченгайтурі» стали легендарними у Кореї. Ночами вони стрибають по рисовим полям, втоптуючи у землю кущики рису, завдаючи втрати посівам. Вранці, помітивши ушкоджені кущики, вам захотілося визначити шлях тієї жаби, яка завдала найбільшої

шкоди. Жаба завжди стрибає по полю по прямій лінії, при чому довжина кожного стрибка однакова.

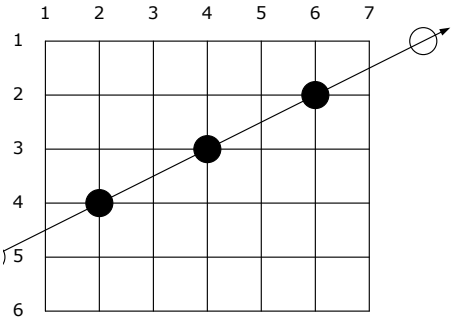
Різні жаби можуть мати різну довжину стрибка і різні напрямки



Кущики рису на вашому полі посаджені строго вздовж перпендикулярних ліній на однаковій відстані один від одного, як це зображено на мал. 14.1. Шкідливі жаби прострибали через усе ваше поле, при чому шлях кожної жаби починався за межами поля з одного його боку, а закінчувався за його межами з іншого боку, як це зображено на мал. 14.2.

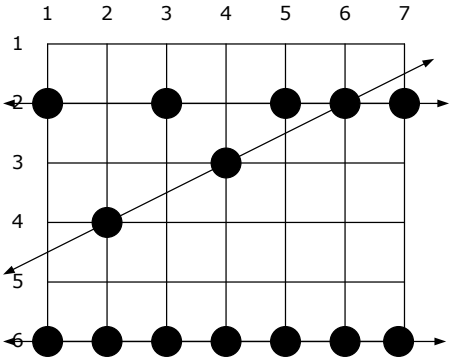


Мал. 14.1: Рисове поле

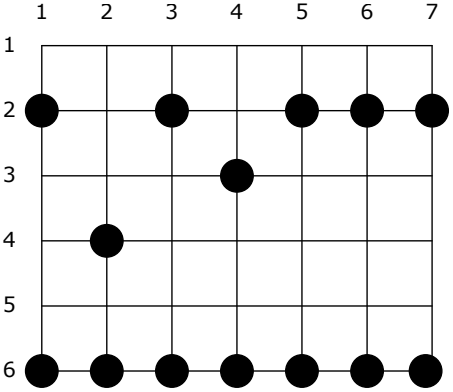


Мал. 14.2: Шлях жаби

По полю може стрибати багато жаб, перестрибуючи від одного рисового кущика до іншого. При кожному стрибку жаба втоптує у землю черговий кущик рису, як на мал. 14.3. Відзначимо, що за нічь одна і таж сама рослина може бути затоптана більш, ніж однією жабою. Звичайно, ви не зможете побачити прямих ліній, вздовж яких стрибали жаби, і будь-які сліди їх переміщення за межами поля. Для прикладу, зображеного на мал. 14.3. ви побачите картину ушкодженого поля, зображену на мал. 14.4.



Мал. 14.3: Рисове поле



Мал. 14.4: Шлях жаби

Виходячи з мал. 14.4, ви зможете відновити усі можливі шляхи, по яким жаби могли б переміщуватися через ваше поле. Нас цікавять тільки ті жаби, які ушкодили не менше трьох рисових кущиків продовж свого переміщення по полю. Таким чином, ми визначили поняття «шлях жаби».

Можна помітити, що три шляхи, зображені на мал. 14.3, є шляхами жаб (існують також і інші можливі шляхи жаб). Вертикальний шлях, що проходить вниз уздовж стовпця 1 міг би бути шляхом жаби, що має довжину стрибка 4, однак при цьому могло бути ушкоджено тільки 2 кущики рису, і такий шлях нас не цікавить. Розглянемо діагональний шлях, що проходить через кущики, які знаходяться на перетині рядка 2 і стовпця 3, рядка 3 і стовпця 4, рядка 6 і стовпця 7. На цьому шляху знаходиться три

ушкоджених кущика, але шлях нерівномірний і містить інтервали нерівної довжини; таким чином, цей шлях також не є шляхом жаби. Відмітимо також, що вздовж лінії руху жаби можуть зустрічатися ушкоджені кущики, які не були ушкоджені саме цією жабою (наприклад, кущик в точці (2,6) на горизонтальному шляху вздовж рядка 2 на мал. 14.4), і на полі можуть знаходитися ушкоджені кущики, причину ушкодження яких неможна пояснити ніяким шляхом жаби.

Розглянемо усі можливі шляхи жаб. Ваша програма повинна визначити кількість ушкоджених кущиків рису на шляху тієї жаби, яка ушкодила більше за всіх кущиків (тобто задала найбільшої шкоди рисовій плантації). На мал. 14.4 таким міг бути шлях вздовж рядка 6, а відповіддю число 7.

Вхідні дані Ваша програма повинна зчитати дані з стандартного входу. Перший рядок містить два цілих числа R і C, що задають відповідно кількість рядків і стовпців на вашому рисовому полі, $1 \le R, C \le 5\ 000$. Другий рядок містить одне ціле число N — кількість ушкоджених рисових кущиків, $3 \le N \le 5\ 000$. Кожний з наступних N рядків містить два цілих числа, номер рядка ($1 \le$ *номер рядка* $\le R$) і номер стовпця ($1 \le$ *номер стовпця* $\le C$), що визначають позицію ушкодженого кущика. Ці числа відокремлено пропуском. Кожен ушкоджений кущик описаний лише один раз.

Вихідні дані Ваша програма повинна виводити відповідь у стандартний вивід. Вивід програми повинен містити один рядок з найбільшим числом, рівним кількості кущиків, що лежать на шляху жаби, яка завдала найбільшої втрати, якщо такий шлях існує. В іншому випадку необхідно вивести 0.

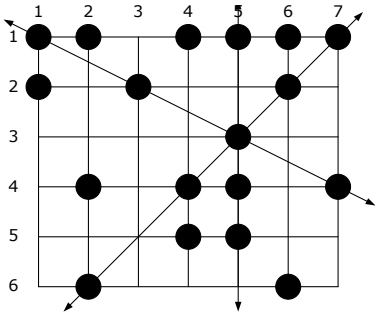
Приклад вхідних та вихідних даних

input	output
6 7 14 2 1 6 6 4 2 2 5 2 6 2 7 3 4 6 1 6 2 2 3 6 3 6 4 6 5 6 7	7

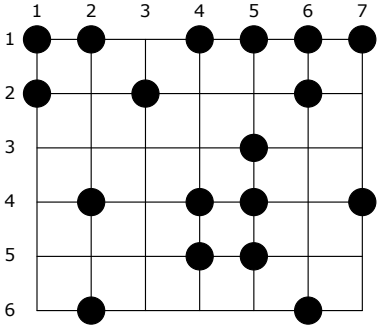
Приклад на мал. 14.4

input	output
6 7 18 1 1 6 2 3 5 1 5 4 7 1 2 1 4 1 6 1 7 2 1 2 3 2 6 4 2 4 4 4 5 5 4 5 5 6 6	4

Приклад на мал. 14.5



Мал. 14.6:

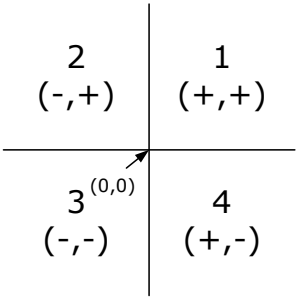


Мал. 14.5:

14.1.2 «Утопія»

Якось прекрасна країна Утопія була разорена війною. Коли ворожість затихнула, країна була розділена на чотири області по долготі (лінія північ-південь) і широті (лінія схід-захід). Точку перетину цих ліній позначили $(0, 0)$. Усі чотири частини претендували на назву Утопія, но з часом їх стали називати Утопія 1 (північно-східна), 2 (північно-західна), 3 (південно-західна) і 4 (південно-східна). Кожна точка в будь-якому регіоні визначалася відстанню на схід і відстанню на північ від (0,0). Ці відстані могли

бути від'ємними, тобто точка в Утопії 2 позначалася парою (від'ємне, додатнє), в Утопії 3 — парою (від'ємне, від'ємне), в Утопії 4 — парою (додатнє, від'ємне), і Утопії 1 — парою додатніх чисел.



Мал. 14.7:

Найбільшою проблемою було те, що громадянам заборонялося перетинати границі. На щастя, винахідливий учасник IOI з Утопії розробив простий спосіб телепортації. Телепортер потребує кодові числа, кожне з яких може бути використане лише один раз. Вам необхідно провести телепортер з його початкової позиції $(0, 0)$ через області Утопії в необхідному порядку. У вас буде послідовність з N номерів областей, в кожній з яких ви повинні приземлитися, при цьому не має значення, де саме в області ви приземлилися. Після того, як ви залишите точку $(0, 0)$, вам неможна потрапляти за границі областей.

Вихідні $2N$ кодових числа ви повинні записати як послідовність N кодових пар, додаючи знак плюс або мінус перед кожним числом. Якщо ви знаходитесь у точці (x, y) і використовуєте кодову пару $(+u, v)$, вас буде телепортовано у точку $(x + u, y - v)$. У вас є $2N$ чисел, і ви можете використовувати їх у будь-якому вибраному вами порядку, кожне із знаком полюс або мінус.

Припустимо, у вас є кодові числа 7, 5, 6, 1, 3, 2, 4, 8, і ви повинні провести телепортер по послідовності областей з номерами 4, 1, 2, 1. Послідовність кодових пар $(+7, -1)$, $(-5, +2)$, $(-4, +3)$, $(+8, +6)$ дозволяє це зробити, телепортуя вас з точки $(0, 0)$ послідовно у точки $(7, -1)$, $(2, 1)$, $(-2, 4)$ і $(6, 10)$. Ці точки розташовані в Утопії 4, Утопії 1, Утопії 2 і Утопії 1 відповідно.

Завдання Вам надано $2N$ різних кодових чисел і послідовність з N номерів областей, що вказують де повинен приземлитися телепортер. Побудуйте послідовність кодових пар із заданих кодових чисел, яка проведе телепортер через задану послідовність областей.

Вхідні дані Ваша програма повинна читати стандартний вхід. Перший рядок містить ціле N ($1 \le N \le 10\ 000$). Другий рядок містить $2N$ різних цілих кодових чисел ($1 \le$ *кодове число* $\le 100\ 000$), розділених одиночними пропусками. Останній рядок містить послідовність N номерів областей, кожний з яких є 1,2,3 або 4.

Вихідні дані Ваша програма повинна писати в стандартний вихід. Вивід повенен складатися з N рядків. Кожний рядок повинен містити пару кодових чисел, перед кожним з яких вказаний знак (плюс або мінус). Ці кодові пари будуть направляти телепортер по заданій послідовності областей. Зверніть увагу, що неповинно бути пропусків між знаком і числом, но повинен бути один пропуск після першого кодового числа.

Якщо існує декілька розв'язків, ваша програма повинна вивести будь-який з них. Якщо розв'язків немає, ваша програма повинна вивести єдине ціле число 0.

Приклади вхідних та вихідних даних

Приклад 1

input
4 7 5 6 1 3 2 4 8 4 1 2 1

output
+7 -1 -5 +2 -4 +3 +8 +6

Приклад 2

input
4 2 5 4 1 7 8 6 3 4 2 2 1

output
+3 -2 -4 +5 -6 +1 +8 +7

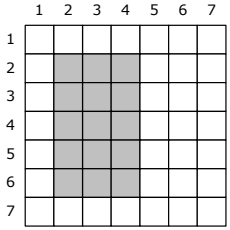
Оцінки Якщо ваша програма виводить правильну відповідь за відведений для тесту час, ви отримуєте усі бали за цей тест, інакше ви отримуєте за цей тест 0 балів.

14.1.3 «XOR»

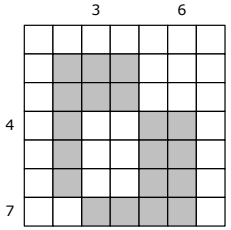
Ви розробляєте програму для мобільного телефона, оснащеного чорно-білим екраном, на якому x-координати відраховуються зліва направо, а y-координати — зверху вниз, як зображено на малюнках. Ваша програма використовує різноманітні картинки, які можуть бути різного розміру. Замість того, щоб зберігати ці картинки, ви хочете створювати їх, використовуючи графічну бібліотеку телефона. На початку малювання усі пікселі

екрана пофарбовані у білий колір. Єдина графічна операція, що підтримується бібліотекою, — це XOR(L, R, T, B), яка перефарбовує в протилежний колір піксели, що потрапляють в прямокутник з лівим верхнім кутом в точці (L, T), а правим нижнім — в точці (R, B), де L означає ліву, T — верхню, R — праву, B — нижню координати. Зауважимо, що в деяких інших графічних бібліотеках порядок аргументів може відрізнятися.

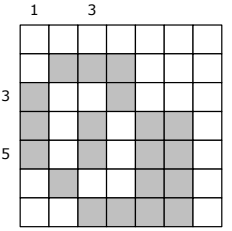
Наприклад, розглянемо отримання картинки на мал. 14.10. Застосовуючи XOR$(2, 4, 2, 6)$ до повністю білої картинки, отримуємо картинку, зображену на мал. 14.8. Застосовуючи XOR$(3, 6, 4, 7)$ до картинки з мал. 14.8, отримуємо картинку, зображену на мал. 14.9, нарешті, застосовуючи XOR$(1, 3, 3, 5)$ до картинки з мал. 14.9, отримуємо картинку, зображену на мал. 14.10.



Мал. 14.8:



Мал. 14.9:



Мал. 14.10:

Ваша задача: для заданої чорно-білої картинки побудувати за можливістю найкоротшу послідовність викликів XOR, що приводить до генерації, починаючи з білого екрану, цієї картинки. Вам дані вхідні файли, що описують картинки. Ви повинні здати файли, що описують параметри викликів XOR, а не вихідний текст програми для створення цих файлів.

Вхідні дані Вам дано 10 текстових файлів з назвами від `xor1.in` до `xor10.in`. Кожен вхідний файл улаштований наступним чином. Перший його рядок містить одне ціле число N, $5 \le N \le 2\,000$, яке позначає, що картинка має N рядків і N стовпчиків. Рядки вхідного файлу, що залишилися описують рядки картинки зверху вниз. Кожний рядок містить N цілих чисел: кольори пікселів зліва направо. Значення 0 позначає білий піксель, а значення 1 — чорний.

Вихідні дані Ви повинні здати 10 вихідних файлів, що відповідають вхідним даним.

Перший рядок файла повинен містити текст:

```
#FILE xor I
```

де ціле число I — номер відповідного вхідного файла. Другий рядок повинен містити ціле число K — кількість викликів XOR. В наступних K

рядках повинні знаходитися описи цих викликів в порядку від першого до останнього. Кожний з цих K рядків повинен містити чотири цілих числа: параметри виклику XOR в порядку L, R, T, B.

Приклад вхідних та вихідних двних

xor0.in
7 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0

xor0.out
#FILE xor 0 3 2 4 2 6 3 6 4 7 1 3 3 5

Система оцінки Якщо

- послідовність викликів XOR, що вказана в вашому файлі, не приводить к генерації цільової картинки, або
- кількість викликів XOR, вказаних у вашому файлі, не рівне K, або
- в вашому вихідному файлі $K > 40\,000$, або
- ваш вихідний файл містить виклик XOR з параметрами $L > R$ або $T > B$, або
- ваш вихідний файл містить виклик XOR з недодатніми координатами, або
- ваш вихідний файл містить виклик XOR із значенням координати, що перевищує N, то ви отримаєте за даний тест 0 балів. В протилежному випадку ваш бал обчислюється за формулою:

$$1 + 9 \times \frac{\text{ЧислоВикликівУКращомуРозв'язкуУчасників}}{\text{ЧислоВикликівУВашомуРозв'язку}}$$

Бали округлюються до першого знаку після десяткової коми по кожному тесту. Загальний бал округлюється до найближчого цілого.

Припустимо, що ви здали розв'язок з 121 викликом XOR. Якщой цей розв'язок — кращий серед усіх учасників, ви отримаєте 10 балів. Якщо краще серед зданих учасниками розв'зків використовує 98 викликів XOR, то ваш бал складе $1 + 9 \times 98/121 (= 8.289\ldots)$ і буде округлений до 8.3.

14.2 Завдання другого туру

14.2.1 «Пакетна обробка завдань»

Існує послідовність N завдань, призначених для виконання на одній машині. Завдання пронумеровані від 1 до N наступним чином $1, 2, \ldots, N$. Завдання повинні бути згруповані в один або декілька пакетів, щоб кожен пакет складався з ідучих один за одним завдань вихідної послідовності. Виконання першого завдання починається в момент часу 0. Пакети обробляється один за одним, починаючи з першого, наступним чином. Якщо пакет b містить завдання з меншими номерами, ніж пакет c, то пакет b потрапляє на обробку раніше пакета c. Завдання кожного пакету виконуються на машині послідовно. Відразу ж після того, як всі завдання пакету виконано, машина виводить результати виконання усіх завданнь цього пакета. Час завершення виконання j-го завдання визначається часом завершення обробки всього пакету, що містить j-те завдання.

Щоб підготувати машину до обробки кожного пакету, необхідний час S, який назвемо підготовчим. Для кожного i-го завдання відомий вартісний коеіфцієнт F_i і час T_i, необхідний для виконання цього завдання. Якщо пакет складається з завдань $x, x+1, \ldots, x+k$ і потрапляє на обробку в момент часу t, то час завершення виконання кожного завдання пакету обраховується за формулою $t + S + (T_x + T_{x+1} + \ldots + T_{x+k})$. Зауважимо, що машина виводить результати всіх завдань пакета в один і той же момент часу. Якщо час завершення виконання i-го завдання — O_i, то вартість виконання цього завдання складе $O_i \times F_i$. Нехай є 5 завдань і відомо: $S = 1$; $(T_1, T_2, T_3, T_4, T_5) = (1, 3, 4, 2, 1)$; $(F_1, F_2, F_3, F_4, F_5) = (3, 2, 3, 3, 4)$.

Якщо завдання згруповані в три пакета $\{1, 2\}$,$\{3\}$,$\{4, 5\}$, то можна обчислити час завершення виконання для них $(O_1, O_2, O_3, O_4, O_5) = (5, 5, 10, 14, 14)$ і вартості виконання завдань $(15, 10, 30, 42, 56)$, відповідно. Загальна вартість виконання згрупованих упакети завдання визначається як сума вартостей виконання кожного завдання. Таким чином, загальна вартість виконання всіх завдань для наведеного прикладу буде 153.

Вам задана величина підготовчого часу S і послідовність N завдань, для кожного з яких визначений час виконання і вартісний коефіцієнт. Потрібно написати програму, яка обчислює мінімальну можливу вартість виконання послідовності N завдань.

Вхідні дані Ваша програма повинна читати дані з стандартного входу. Перший рядок містить кількість завдань N ($1 \le N \le 10\,000$). Другий рядок містить підготовчій час S, який є цілим числом ($0 \le S \le 50$). Наступні N рядків містять інформацію о завданнях $1, 2, \ldots, N$ у вказаному порядку. В кожному з цих рядків першим задається число T_i ($1 \le T_i \le 100$) — час

виконання завдання. За ним слідує ціле число F_i ($1 \le F_i \le 100$) — вартісний коефіцієнт завдання.

Вихідні дані Ваша програма повинна записати у стандартний вихід один рядок, що містить одне ціле число — мінімальну загальну вартість виконання послідовності з N завдань

Приклади вхідних та вихідних даних

Приклад 1

input	output
2 50 100 100 100 100	45000

Приклад 2

input	output
5 1 1 3 3 2 4 3 2 3 1 4	153

Приклад 2 відповідає прикладу з текста задачі.

Примітка Для кожного тесту загальна вартість будь-якого групування не перевищує $2^{31} - 1$.

Оцінки Якщо ваша програма виводить правильну відповідь на тест за відведений час, ви отримуєте повну кількість балів за цей тест, інакше — ви отримуєте 0 балів.

14.2.2 «Автобусні зупинки»

В місті Yong-In планується створити мережу автобусних маршрутів з N автобусними зупинками. Кожна зупинка знаходиться на деякому перехресті. Оскільки Yong-In — сучасне місто, на карті він представлений вулицями з квадратними кварталами однакового розміру. Дві з N зупинок обираються пересадними станціями H_1 і H_2, які будуть з'єднані одна з одною автобусним маршрутом, а кожна з $N - 2$ зупинок, що залишилися, буде беспосередньо з'єднана маршрутом з однією з пересадних станцій H_1 або H_2 (але не з двома), і не з'єднана з жодною з залишившихся зупинок.

Відстань між будь-якими двома зупинками визначається як довжина найкоротшего шляху по вулицям міста. Це означає, що якщо зупинка пред-

ставлена парою координат (x, y), то відстань між двома зупинками (x_1, y_1) і (x_2, y_2) буде рівною $|x_1 - x_2| + |y_1 - y_2|$. Якщо зупинки A і B з'єднані з однією і тою ж пересадною станцією, наприклад H_1, то довжина шляху з A у B є сумою відстаней від A до H_1 і від H_1 до B. Якщо автобусні зупинки A і B з'єднані з різними пересадними станціями, наприклад A с H_1 і B с H_2, то довжина шляху з A у B є сумою відстаней від A до H_1, від H_1 до H_2 і від H_2 до B.

Проектувальники міста Yong-In хочуть бути впевненими, що будь-який мешканець міста зможе досягти будь-якої точки міста достатньо швидко. Тому проектувальники хочуть зробити пересадними станціями такі дві зупинки, щоб у отриманій мережі автобусних маршрутів максимальна довжина шляху між будь-якими двома зупинками була мінімальною.

Варіант P вибору пересадних станцій і з'єднання зупинок з ними буде краще варіанта Q, якщо максимальна довжина шляху між будь-якими двома зупинками в варіанті P буде меньше, ніж у варіанті Q. Ваша задача — написати програму обчислення максимальної довжини шляху між будь-якими двома зупинками для найкращего варіанта P вибору пересадних станцій між ними і з'єднань зупинок з ними.

Вхідні дані Ваша програма повинна читати стандартний вхід. Перший рядок містить додатнє ціле число N $(2 \leq N \leq 500)$ — кількість зупинок. Кожен з N рядків, що залишилися, містить пару чисел x і y — координати автобусної зупинки. Координати x і y — додатні цілі числа $\leq 5\ 000$. Ніякі дві зупинки не можуть бути представлені однією і тою ж парою координат.

Вихідні дані Ваша програма повинна записувати у стандартний вихід один рядок з єдиним додатнім числом — мінімальною можливою довжиною максимального шляху між зупинками.

Приклади вхідних та вихідних даних

Приклад 1

input	output
6 1 7 16 6 12 4 4 4 1 1 11 1	20

Приклад 2

input	output
7 7 9 10 9 5 3 1 1 7 2 15 6 17 7	25

Малюнки нижче показують мережі автобусних маршрутів для наведених прикладів вхідних даних. Якщо в Прикладі 1 вибрати зупинки 3 і 4 як пересадні станції, то найдовший шлях буде між зупинками 2 і 5, або між зупинками 2 і 1. Оскільки кращого варіанту вибору пересадних станцій немає, відповіддю буде 20.

Для Приклада 2, якщо у якості пересадних станцій обрані зупинки 5 і 6, то найдовший шлях буде між зупинками 2 і 7. Оскількі кращих варіантов немає, то відповіддю буде 25.

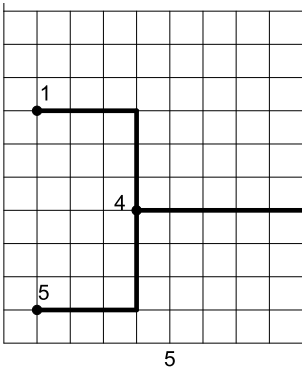
Оцінки Якщо ваша програма виводить правильну відповідь у відведений для теста час, ви отримуєте усі бали за цей тест, інакше ви отримуєте за цей тест 0 балів.

14.2.3 «Дві стежки»

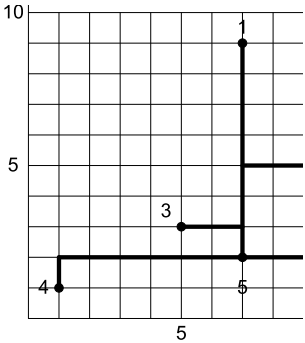
Стежка являє собою горизонтальну або вертикальну послідовність клітин сітки, та складається не меньше ніж з двох послідовних клітин. Дві стежки розташовані на сітці розміром $N \times N$. Одна стежка — горизонтальна, а друга — вертикальна. На Мал. 14.13 ці дві стежки відмічені символом `X`. Стежки можуть бути однакової або різної довжини. Стежки можуть мати загальну клітину, а можуть не мати такої. Якщо на діаграмі, такой як на Мал. 14.13, можна інтерпретувати клітину, наприклад $(4, 4)$, як таку, що належить одній стежці або двом стежкам, то потрібно вважати, що дана клітина належить і той і іншій стежці відразу. Таким чином, на Мал. 14.13 верхня клітина вертикальної стежки буде $(4, 4)$, але не $(5, 4)$.

Спочатку невідомо, де розташовані стежки. Ваша задача — написати програму, що визначає їх положення. Назвемо горизонтальну стежку СТЕЖКА1, а вертикальну — СТЕЖКА2. Кожна клітина сітки представлена парою чисел (r, c), що задають номер рядка і стовпця відповідно. Ліва верхня клітина сітки має координати $(1, 1)$. Кожна стежка задається парою клітин: $< (r_1, c_1), (r_2, c_2) >$. На Мал. 14.13 СТЕЖКА1 задається як $< (4, 3), (4, 8) >$, а СТЕЖКА2 — як $< (4, 4), (9, 4) >$.

При розв'язку даної задачі ви повинні використовувати бібліотеку для

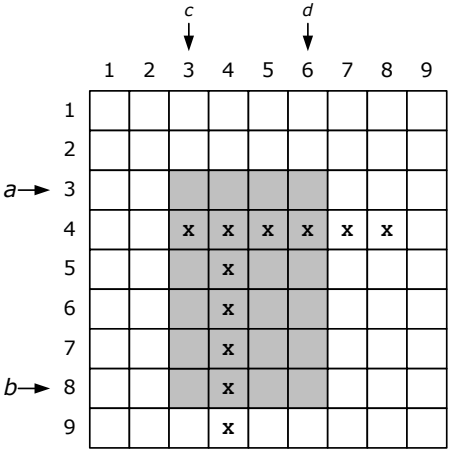


Мал. 14.11: Мережа для Прикладу 1



Мал. 14.12: Мережа для Прикладу 2

отримання вхідних даних, для пошука розв'язка і для виводу відповіді. Розмір N квадратної сітки повертається бібліотечною функцією gridsize, яку ви повинні викликати на початку роботи. Для визначення положень стежок ви можете використовувати тільки бібліотечну функцію rect(a, b, c, d), яка отримує на вході координати прямокутної області і повертає число 1, якщо у вказаній області зустрічається хоча б одна клітина, що належить хоча б одній стежці, і число 0 — у протилежному випадку. Область задається у вигляді $[a, b] \times [c, d]$, де $a \le b$ і $c \le d$, при цьому необхідно звернути увагу на порядок аргументів. Наприклад, на Мал. 14.13 область, що перевіряється зафарбована сірим кольором. Від вас вимагається написати програму, що визначає точне положення стежок, використовуючи обмежану кількість викликів функції rect. Виклик функції report призведе до завершення роботи



Мал. 14.13:

вашої програми. Зверніть увагу, що СТЕЖКА1 горизонтальна, а СТЕЖКА2 — вертикальна, і (r_1, c_1) — координати лівого кінця горизонтальної СТЕЖКИ1, а (p_1, q_1) — координати верхнього кінця вертикальної СТЕЖКИ2. Таким чином, $r_1 = r_2$, $c_1 < c_2$, $p_1 < p_2$ і $q_1 = q_2$. Якщо параметри функції report не задовільняють цим обмеженням, то стандартний вихід буде містити повідомлення про помилку.

Обмеження

- Вхідні дані достіпні тільки через виклики бібліотечних функцій gridsize і rect.
- Розмір сітки N задовільняє обмеженню $5 \leq N \leq 10\,000$.
- Кількість викликів функції rect не повинно перевищувати 400 для будь-якого тесту. Якщо ваша програма здійснить більше 400 викликів функції rect, то це призведе до її дострокового завершення.
- Ваша програма повинна виконувати більше одного виклику функції rect, і рівно один виклик функції report.
- Якщо виклик функції rect буде здійснено з некоректними параметрами, то це призвиде до дострокового завершення вашої програми.
- Ваша програма не повинна зчитувати або записувати будь-які файли, а також використовувати стандартні потоки вводу та виводу.

Бібліотека У вашому розпорядження знаходиться бібліотека, що містить:

FreePascal бібліотеку (`prectlib.ppu`, `prectlib.o`)

```
function gridsize: LongInt;
function rect(a,b,c,d : LongInt) : LongInt;
procedure report(r1, c1, r2, c2, p1, q1, p2, q2 : LongInt);
```

Інструкція: при компіляції вашого файлу `rods.pas` використовуйте оператор

```
uses prectlib;
```

і командний рядок

```
fpc -So -O2 -XS rods.pas
```

Програма `prodstool.pas` являє собою приклад використання даної FreePascal бібліотеки.

GNU C/C++ бібліотеку (`crectlib.h`, `crectlib.o`)

```
int gridsize();
int rect(int a, int b, int c, int d);
void report(int r1, int c1, int r2, int c2, int p1, int q1,
            int p2, int q2);
```

Інструкції: при компіляції вашего файла `rods.c` використовуйте директиву

```
#include "crectlib.h"
```

і командний рядок

```
gcc -O2 -static rods.c crectlib.o -lm
g++ -O2 -static rods.cpp crectlib.o -lm
```

Програма `prodstool.c` являє собою приклад використання даної GNU C/C++ бібліотеки.

Для C/C++ в середовищі RHIDE

Переконайтеся, що в опціях Option→Linker вказано `crectlib.o`.

Експерементування Для проведення експериментів з бібліотекою ви повинні створити текстовий файл `rods.in`. Файл повинен складатися з трьох рядків. В першому рядку файлу повинно бути записане число N, розмір сітки. Другий рядок повинен містити координати СТЕЖКИ1 $r_1\ c_1\ r_2\ c_2$, де (r_1, c_1) — координати лівого кінця стежки. Третій рядок повинен містити координати СТЕЖКИ2 $p_1\ q_1\ p_2\ q_2$, де (p_1, q_1) — координати верхнього кінця стежки.

Після запуску вашої програми з викликом функції report, буде створений вихідний файл `rods.out`. Цей файл буде містити кількість викликів функції rect і координати кінців стежок, які ви передасте при виклику функції report. Якщо в процесі роботи програми виникли будь-які помилки або порушення вимог умови при звертанні до біліотеки, то файл `rods.out` буде містити відповідне повідомлення про помилку.

Діалог між вашою програмою і бібліотекою записується в файл `rods.log`. Цей файл `rods.log` буде містити послідовність викликів функції rect у формі «k : rect(a, b, c, d) = ans». Це означає, що k-ий виклик функції rect(a, b, c, d) повернув число ans.

Приклади вхідних та вихідних даних

rods.in
9 4 3 4 8 4 4 9 4

rods.out
20 4 3 4 8 4 4 9 4

Оцінки Якщо ваша програма порушує будь-які з вищезгаданих обмежень (наприклад, число викликів функції rect більше 400), або якщо відповідь вашої програми (розташування стежок) некоректна, то ви отримуєте 0 балів.

Якщо відповідь вашої програми правильна, то ваши бали будуть залежати від кількості викликів функції rect на кожному тесті. Якщо це число не перевищує 100, то ви отримуєте 5 балів, якщо воно від 101 до 200, то ви отримуєте 3 бали, а якщо від 201 до 400, то ви отримуєте 1 бал.

Розділ 15. США'2003

15.1 Завдання першого туру

15.1.1 «Вибір доріг»

Завдання Корови фермера Джона бажають вільно ходити по N ($1 \le N \le 200$) послідовно пронумерованим від 1 до N полям ферми, навіть

якщо поля розділені лісом. Корови обирають систему доріг між парами полів таким чином, щоб можна було пройти від любого поля до любого іншого поля, рухаючись по дорогам. Корови можуть ходити по дорогам у любому напрямку.

Для доріг корови можуть використовувати тільки тропи диких звірів. Кожен тиждень вони можуть обирати для використання всі або деякі тропи диких звірів, про яких їм відомо.

Цікаво, що корови завжди знаходять рівно одну нову тропу диких звірів на початку кожного тижня. Вони мають прийняти рішення про множину троп, які будуть використовуватися у якості системи доріг на протязі поточного тижня. Корови можуть обирати довільну підмножину троп диких звірів незалежно від того, які тропи обирались на попередньому тижні.

Корови завжди хочуть мінімізувати сумарну довжину доріг, яку вони збираються обрати.

Тропи диких звірів не є прямими. Дві тропи, які з'єднують одні й ті ж два поля, можуть мати різну довжину. Більш того, у випадку перетину двох троп (що можливо тільки поза полем), корови не можуть переходити с однієї тропи на другу у точці їх перетину.

На початку кожного тижня є інформація про тропу диких звірів, яку знайшли корови. На основі цього ваша програма має виводити мінімальну загальну довжину доріг, обраних коровами на цьому тижні, або вказувати, що шукана система доріг не існує.

Вхідні дані

- Перший рядок вхідного файлу містить два цілих числа, розділених пропуском, N та W. W — кількість тижнів, які має обробити програма ($1 \leq W \leq 6000$).

- Для кожного тижня читайте один рядок, який містить інформацію про нещодавно знайдену тропу диких звірів. Цей рядок містить три цілих числа, розділених пропусками: кінцеві точки тропи (номери полів) і цілочисельну довжину тропи ($1 \leq 10\ 000$). Не буває тропи, у якій кінцеві точки знаходяться на одному полі.

Вихідні дані Зразу після того, як ваша програма узнає про нещодавно знайдену тропу диких звірів, вона має вивести один рядок, який містить мінімальну загальну довжину доріг, які треба вибрати, щоб зв'язати всі поля. Якщо неможливо зв'язати всі поля системою доріг, використовуючи тільки тропи диких звірів, то треба вивести `-1`.

Ваша програма має закінчувати роботу після вивода відповіді для останнього тижня.

Приклад діалога

Вхід	Вихід	Пояснення
4 6		
1 2 10	-1	Поле 4 не з'єднано с другими полями
1 3 8	-1	Поле 4 не з'єднано с другими полями
3 2 3	-1	Поле 4 не з'єднано с другими полями
1 4 3	14	Вибрати 1 4 3, 1 3 8 і 3 2 3.
1 3 6	12	Вибрати 1 4 3, 1 3 6 і 3 2 3.
2 1 2	8	Вибрати 1 4 3, 2 1 2 і 3 2 3.
		Завершення роботи програми

15.1.2 «Reverse»

Завдання Розглянемо обчислювальну машину, що виконує дві операції (назвемо її МДО). Вона має дев'ять регістрів, пронумерованих числами від 1 до 9. Кожен регістр зберігає невід'ємне ціле число в диапазоні від 0 до 1000 включно. МДО здатна виконувати дві операції:

`S` i j	Записати значення регістра i, збільшене на 1, в регістр j (i может бути рівним j).
`P` i	Надрукувати значення регістра i.

Програма для МДО складається з початкових значень кожного регістра і послідовності операцій. Для заданого цілого числа $N (0 \le N \le 255)$ необхідно написати для МДО програму, яка друкує спадаючу послідовність цілих чисел $N, N-1, N-2, \ldots, 0$. Максимальна кількість підряд ідучих `S`-операцій має бути як можна менше.

Приклад програми для МДО і її виконання при $N = 2$:

Операція	Нові значення регістрів	Надруковане значення
	1 2 3 4 5 6 7 8 9	
Початкові значення	0 2 0 0 0 0 0 0 0	
`P` 2	0 2 0 0 0 0 0 0 0	2
`S` 1 3	0 2 1 0 0 0 0 0 0	
`P` 3	0 2 1 0 0 0 0 0 0	1
`P` 1	0 2 1 0 0 0 0 0 0	0

Варіанти вхідних даних, пронумеровані від 1 до 16, розташовані на сервері змагання.

Вхідні дані Перший рядок вхідного файлу містить одне ціле число K, що визначає номер теста. Другий рядок містить число N.

Вихідні дані Перший рядок вихідного файла має містити текст

`FILE reverse` K

де K визначає номер теста.

Другий рядок вихідного файла має містити дев'ять розділених пропусками чисел, що є начальними значеннями регістров и розташовані у наступному порядку: регістр 1, регістр 2, ... , регістр 9.

Решта рядків вихідного файла мають містити послідовність операцій у порядку їх виконання, по одній операції у рядку. Таким чином, третій рядок має містити першу операцію, четвертий — другу, і т.д. Останній рядок має містити операцію, що друкує 0. У кожному рядку має міститись тільки припустима операція. Операції мають бути відформатовані як вказано у прикладі.

Приклад вхідних та вихідних даних

Приклад вхідних даних
1 2

Неповний бал
FILE reverse 1 0 2 0 0 0 0 0 0 0 P 2 S 1 3 P 3 P 1

Повний бал
FILE reverse 1 0 2 1 0 0 0 0 0 0 P 2 P 3 P 1

Система оцінки Оцінка кожного теста виконується з урахуванням коректності і оптимальності отриманої програми для МДО.

Коректність: 20%

Програма для МДО є правильною, якщо вона виконує не більше ніж 131 `S`-операції поспіль, і послідовність надрукованих значень є правильною (містить рівно $N+1$ ціле число, починаючи з числа N та закінчуючи 0). Якщо якась `S`-операція спричинює переповнення регістра, програма для МДО вважається невірною.

Оптимальність: 80%

Оптимальність коректної програми оцінюється по максимальній кількості `S`-операций поспіль в програмі, яка має бути якомога менше. Кількість балів буде визначатись різницею між вашою програмою та найкращою відомою програмою.

15.1.3 «Порівняння кодів»

Завдання Компанія Racine Business Networks (RBN) подала до суду на компанію Heuristic Algorithm Languages (HAL), стверджуючи що HAL використовувала вихідний код з RBN UNIX і внесла його в відкритий код операційної системи HALinux.

Як RBN, так і HAL використовують мову програмування, в якій кожен оператор розташований у окремому рядку та має вигляд:

```
STOREA = STOREB + STOREC
```

(`STOREA`, `STOREB` та `STOREC` — імена змінних).

Кожен оператор записується наступним чином: ім'я першої змінної починається з першої позиції рядка, потім — пропуск, знак рівності, пропуск, ім'я другої змінної, пропуск, знак додавання, пропуск та ім'я третьої змінної. Одне і те ж ім'я змінної може зустрічатися в рядку більше одного разу. Імена змінних мають довжину від 1 до 8 символів і складаються з великих латинських ASCII літер (`A...Z`).

Стверджується, що HAL зкопіювала послідовність рядків програми прямо із вихідного кода RBN с мінімальними змінами. Щоб приховати свій злочін, HAL змінила імена деяких змінних. Точніше, HAL взяла послідовність рядків із програми RBN і для кожної змінної в ній змінила її ім'я. При цьому нове ім'я змінної може співпасти зі старим. Після зміни імен ніякі дві різні змінні не можуть називатися однаково. HAL могла змінити в деяких рядках порядок імен змінних в правій частині оператора присвоювання. Напрклад, оператор вигляду

```
STOREA = STOREB + STOREC
```

міг бути зміненим так:

```
STOREA = STOREC + STOREB
```

Стверджується також, що HAL не змінила порядок, в якому рядки кода слідують у вихідному тексті програми.

Необхідно по заданим вихідним кодам програм RBN і HAL знайти саму довгу послідовність рядків програми HAL, що йдуть підряд, котру можна отримати з послідовності рядків програми RBN, що йдуть підряд, за допомогою вказаних вище перетворень. Зауважте, що знайдена послідовність в коді HAL і відповідна послідовність в коді RBN не обов'язково починаються в рядках з однаковими номерами.

Вхідні дані Перший рядок вхідного файлу `code.in` містить два розділених пропусками цілих числа R і H ($1 \le R \le 1000; 1 \le H \le 1000$). R задає число рядків в коді програми RBN, а H — число рядків в коді програми HAL. Наступні R рядків містять програму RBN. Наступні H рядків містять програму HAL.

Вхідні дані Вихідний файл `code.out` має містити один рядок з єдиним цілим числом — довжиною самої довгої послідовності рядків, що йдуть підряд, яку HAL могла скопіювати із змінами у RBN.

Приклад вхідних та вихідних двних

code.in
4 3 RA = RB + RC RC = D + RE RF = RF + RJ RE = RF + RF HD = HE + HF HM = HN + D HN = HA + HB

code.out
2

Рядки з 1-го по 2-й з програми RBN співпадають з рядками з 2-го по 3-й із програми HAL при умові, що в програмі RBN виконано наступні зміни імен змінних: `RA`→`HM`, `RB`→`D`, `RC`→`HN`, `D`→`HA`, `RE`→`HB`. Розв'язку з трьома та більше відповідними рядками не існує.

15.2 Завдання другого туру

15.2.1 «Вгадайте корову»

Завдання В стаді фермера Джона знаходиться N пронумерованих від 1 до N ($1 \le N \le 50$) та схожих корів. Коли фермер Джон ставить корову до стойла, він має знати, яку корову він туди ставить.

Корови відрізняються по P ознакам, пронумерованим від 1 до P ($1 \le P \le 8$), кожен з яких має по три можливі значення. Наприклад, колір мітки на вусі корови може бути жовтим, зеленим або червоним. Значення кожної ознаки визначається однією з букв `X`, `Y` або `Z`. Довільна пара корів фермера Джона відрізняється принаймні за однією з ознак.

Напишіть програму, яка по заданим ознакам корів допоможе фермеру Джону визначити номер корови, яку він ставить до стойла. Ваша програма може задати Джону не більше 100 запитань вигляду: «Чи належить значення деякої ознаки T у корови деякій множині значень S?»

Задайте як можна менше питань для визначення номера корови.

Вхідний файл ознак Перший рядок вхідного файла `guess.in` містить два цілих числа N і P, розділених пропуском. Кожен з наступних N рядків

описує ознаки корови, використовуючи P літер, розділених пропусками. Перша літера кожного рядка — значення ознаки 1, і так далі. Другій рядок у вхідному файлі описує корову з номером 1, третій рядок — корову з номером 2 і т.д.

Приклад вхідного файлу ознак

guess.in

```
4 2
X Z
X Y
Y X
Y Y
```

Інтерактивність Крок «питання / відповідь» виконується через стандартний вхід і стандартний вихід.

Ваша програма задає питання про корову записуючи у стандартний вихід рядки наступного вигляду: літера `Q`, номер ознаки, значення ознаки (одне або більше), розділені пропусками. Наприклад, рядок `Q 1 Z Y` відповідає питанню: «Чи має перша ознака корови значення `Z` або `Y`?». Ознака може бути цілим числом в межах від 1 до P. Значення ознаки має бути тільки `X`, `Y` або `Z` і не повинно повторюватись в одному рядку.

Після запитання ваша програма повинна читати один рядок, який містить одне з цілих чисел — 0 або 1. Число 1 визначає, що корова володіє одним з вказаних значень ознаки. Число 0 визначає, що жодним з вказаних значень ознаки корова не володіє.

Останній рядок виводу програми має містити літеру `C`, пропуск і одне число (номер корови).

Приклад діалога: (для наведеного прикладу)

Вхід	Вихід	Коментар
	`Q 1 X Z`	
0		Может бути корова 3 або корова 4
	`Q 2 Y`	
1		Має бути корова 4!
	`C 4`	
		Завершення роботи програми

Система оцінки Коректність: 30% балів

Програма, яка виводить вірний номер корови, отримає повний бал за коректність, якщо існує єдина корова, що відповідає отриманим відповідям. Програма, яка задає більше 100 запитань, балів за тест не отримає.

Залежність від кількості запитань: 70% балів

Решта балів будуть визначені в залежності від кількості запитань, які задано для коректного визначення корови. Система оценки влаштована так,

щоб оцінити мінімальну кількість запитань, яку задасть ваша програма у найгіршому випадку. Близькі до оптимального розв'язки отримають часткові бали.

15.2.2 «Роботи в лабіринті»

Завдання Ви є щасливим володарем двох роботів, які розташовані у двох різних прямокутних лабіринтах. Квадрати з координатами (1, 1) розташовані у верхніх лівих (північно-західних) кутах лабіринтів. У лабіринті з номером i ($i = 1, 2$) знаходиться G_i охоронців ($0 \le G_i \le 10$), що намагаються спіймати робота. Для цього вони патрулюють лабіринти вперед і назад по своїй ланці патрулювання, що являє собою декілька послідовних клітин лабіринту на одній прямій. Ваша задача — визначити послідовність команд, що призводить до виходу обох роботів з лабіринтів так, щоб їх не спіймали охоронці.

На початку кожної хвилини обом роботам надсилається одна і та же команда, яка задає одне з чотирьох напрямків (північ, південь, схід або захід). Отримавши команду, робот переміщується на одну клітину у вказаному напрямку. Якщо у клітині, куди рухається робот, стоїть стіна, то замість виконання цієї команди робот на протязі цієї хвилини залишається на місці. Вважається, що робот виходить з лабіринту, коли опиняється за його межами. Покинувши лабіринт, робот ігнорує все наступні команди.

Для кожного охоронця у початковий момент відома його позиція в лабіринті і напрямок, у якому він починає рухатись. На початку кожної хвилини одночасно з роботами охоронці пересуваються на одну клітину. Охоронець рухається вперед зі швидкістю одна клітина за хвилину до тих пір, поки він не досягне кінця своєї ланки патрулювання (тобто поки він не зробить на одне пересування менше, ніж число клітин в його ланці патрулювання). По досягненні цього моменту, охоронець миттєво розвертається і в подальшому продовжує рух в зворотному напрямку до своєї початкової позиції. Досягнувши початкової позиції, він знову розвертається і повторює свій рух по ланці патрулювання до тих пір, поки обидва робота не вийдуть з лабіринтів.

Ланки патрулювання охоронців такі, що вони не проходять крізь стіни і не виходять за межі лабіринту. Ланки патрулювання охоронців можуть перетинатися, але ніякі два охоронця ніколи не зіткнуться, тобто вони ніколи не будуть займати одну і ту ж позицію то закінченні хвилини. Охоронці також не будуть одночасно знаходитись у сусідніх клітинах та рухатись назустріч один одному на протязі цієї хвилини. Початкове положення охоронця не буде співпадати з початковим положенням робота.

Вважається, що охоронець піймав робота, якщо охоронець і робот опи-

нилися в одній позиції наприкінці хвилини, або якщо охоронець і робот знаходяться у сусідніх клітинах та рухаються назустріч один одному.

За заданим описом двох лабіринтів (розмір кожного з лабіринтів не більше 20x20), відомим початковим позиціям кожного з роботів і ланками патрулювання охоронців визначте послідовність команд, яка приводить до виходу з лабіринтів обох роботів, не зловленими охоронцями. При цьому треба мінімізувати час, який потрібен для того, щоб обидва робота вийшли з лабіринтів. Якщо роботи виходять з лабіринтів у різні моменти часу, то час того робота, який вийшов раніше, не суттєвий (враховується лише час робота, який вийшов пізніше).

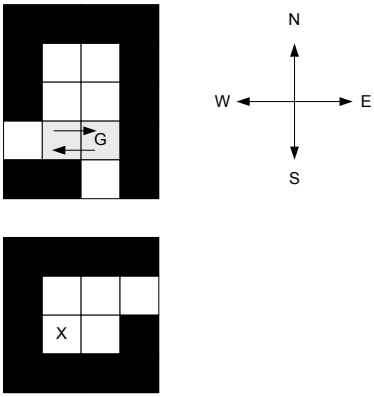
Вхідні дані Перша частина рядків у вхідному файлі `robots.in` описує перший лабіринт і тих, хто у ньому знаходиться. Відповідно, друга частина описує другий лабіринт і тих, хто в ньому знаходиться.

Перший рядок вхідного файлу містить два розділених пропуском цілих числа R_1 і C_1 — кількості рядків і стовпчиків у першому лабіринті відповідно. У кожному наступному з R_1 рядків міститься по C_1 символів, що описують конфігурацію лабіринту. Початкове положення робота позначається символом `X`. Символ `.` позначає пусту клітину, а символ `#` — стіну. У лабіринті завжди рівно один робот. Після описання лабіринту йде рядок, що містить одне ціле число G_1 — кількість охоронців у першому лабіринті ($0 \le G_1 \le 10$). Кожен з наступних G_1 рядків описує ланку патрулювання охоронця у вигляді трьох цілих чисел і символу, розділених пропусками. Перше і друге числа вказують рядок та стовпчик початкового розташування охоронця. Третє число ($2 \ldots 4$) задає кількість клітин у ланці патрулювання цього охоронця. Символ `N`, `S`, `E` або `W` (північ, південь, схід и захід відповідно) задає початковий напрямок руху охоронця.

Опис другого лабіринту і тих, хто в ньому знаходяться, слідує за описом першого лабіринту у тому ж форматі.

Вихідні дані Якщо шуканий розв'язок існує, то перший рядок вихідного файлу `robots.out` має містити єдине ціле число K ($K \le 10\,000$) — кількість команд для обох роботів, що приводять їх до виходу з лабіринту не зловленими. Гарантується, що в цьому випадку найкоротша послідовність команд має довжину не більше 10 000. У наступних K рядках мають бут розташовані команди, кожна з яких є один з символів `N`, `S`, `E` або `W`. Якщо розв'язка не існує, то необхідно вивести єдиний рядок, що містить `-1`.

Після виконання наданої послідовності команд обидва робота мають знаходитись за межами лабіринтів. Остання команда має призводити до виходу з лабіринта хоча б одного з роботів. Якщо існує декілька розв'язків з мінімальним часом, то довільний з них вважається правильним.



Мал. 15.1:

Приклад вхідних та вихідних даних

robots.in
5 4 #### #X.# #..# ...# ##.# 1 4 3 2 W 4 4 #### #... #X.# #### 0

robots.out
8 E N E S S S E S

Система оцінки У тих тестах, де розв'язку не існує, буде оцінюватись лише правильна відповідь. Часткові бали за інші тести будуть нараховуватись наступним чином:

Коректність: 20% балів

Вихідний файл вважається коректним, коли він оформлений у відповідності з вимогами умови задачі, та містить більше 10 000 команд, а їх послідовність приводить до виходу обох роботів з лабіринтів. При цьому остання команда має приводити до виходу хоча б одного з роботів.

Мінімальність: 80% балів

Вважається, що розв'язок, представлений у вихідному файлі мінімальний, якщо він коректний і не існує більш короткої послідовності команд, яка також коректна. Розв'язок, в якому послідовність команд не мінімальна, не

отримує балів за мінімальність.

15.2.3 «Паркан»

Завдання Фермер Дон оглядає паркан, яким оточено його квадратне плоске поле розміром N на N метрів ($2 \le N \le 500\,000$). Один кут паркана розташовано в точці (0, 0), протилежний йому кут знаходиться в точці (N, N). Боки поля паралельні осям X та Y.

Стовпи, до яких кріпиться паркан, стоять не тільки у кутах, а і через метр вздовж кожного боку поля, всього у паркані $4N$ стовпів. Стовпи стоять вертикально і не мають товщини (радіус дорівнює нулю). Фермер Дон хоче визначити, скільки стовпів він побаче, якщо стане в деякій точці поля.

Проблема в тому, що на полі фермера Дона розташовано R величезних каменів ($1 \le R \le 30\,000$), за якими не видно деякі стовпи, оскільки Дон недостатньо високий, щоб дивитися над цими каменями. Основою кожного каменя є опуклий багатокутник ненульової площі, вершини якого мають цілочисельні координати. Камені стоять на полі вертикально. Камені не мають спільних точок між собою, а також з парканом. Точка, де розташувався фермер Дон, лежить всередині, але не на границі поля, а також не лежить середині та на границі каменів.

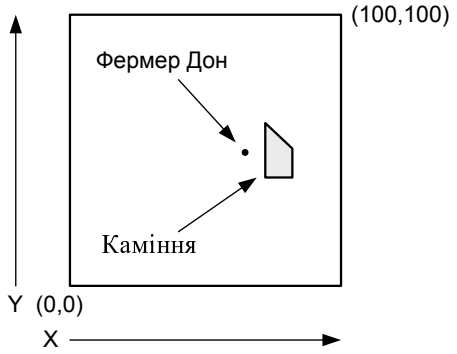
По заданим розміру поля фермера Дона, розташуванню та формі каменів на полі, місцю, де розташувався фермер Дон, обчисліть кількість стовпів, які може бачити фермер Дон. Якщо вершина основи каменя знаходиться на одній лінії з місцем де розташувався Дон і деяким стовпом, то Дон не бачить цей стовп.

`boundary.in`

Вхідні дані Перший рядок вхідного файлу містить два цілих числа N і R, розділених пропуском. Другий рядок вхідного файлу містить два цілих числа — координати X та Y місця, де стоїть фермер Дон на полі. Наступні рядки вхідного файлу описують розташування R каменів: опис i-го каменя починається з рядка, який містить ціле число $p_i (3 \le p_i \le 20)$, що визначає кількість вершин в основі каменя. Кожен з наступних p_i рядків містить пару розділених пропуском цілих чисел X і Y, які є координатами вершини. Вершини основи каменя різні між собою і перераховані в напрямку проти годинникової стрілки.

`boundary.out`

Вихідні дані Вихідний файл має містити один рядок з одним цілим числом — кількістю стовбів, які буде бачити фермер Дон.



Мал. 15.2:

Приклад вхідних та вихідних даних

boundary.in	boundary.out
100 1 60 50 5 70 40 75 40 80 40 80 50 70 60	319

Зверніть увагу на те, що основа каменя має три вершини на одній прямій: (70,40), (75,40) і (80,40).

Розділ 16. Греція'2004

16.1 Завдання першого туру

16.1.1 «Гермес»

Завдання У сучасному місті грецьких богів вулиці розміщені у вигляді цілочисельної решітки і паралельні осям x та y. Для кожного цілого числа Z існує горизонтальна вулиця з $y = Z$ і вертикальна вулиця з $x = Z$. В кожній точці з цілочисельними координатами знаходиться вуличне перехрестя. Таким чином, кожна пара цілих чисел задає певне перехрестя. У спекотні дні боги відпочивають в кафетеріях, що знаходяться на вуличних перехрестях. Посильний Гермес повинен надіслати світові повідомлення бо-

гам, що відпочивають у кафетеріях, рухаючись тільки по вулицях міста. Кожне повідомлення призначається тільки для одного бога, але нічого не трапиться, якщо його побачать інші боги.

Повідомлення повинні бути відіслані богам строго в заданому порядку, тому Гермесу дані координати кафетеріїв саме у цьому порядку. Гермес стартує з точки із координатами (0,0). Для того, щоб надіслати повідомлення в кафетерій з координатами (X_i, Y_i), Гермесу достатньо відвідати певну точку на цій горизонтальній вулиці (з y-координатою Y_i) або на цій вертикальній вулиці (з x-координатою X_i). Після відправки всіх повідомлень Гермес зупиняється.

Ви повинні написати програму, яка за заданою послідовністю кафетеріїв знаходить мінімальну сумарну довжину путі, який повинен пройти Гермес для відсилки усіх повідомлень.

Вхідні дані Перший рядок вхідного файлу з ім'ям `hermes.in` містить одне ціле число N - кількість повідомлень, які повинні бути відправлені. Наступні N рядків містять координати N вуличних перехресть, куди повинні бути надіслані повідомлення. Ці N рядків визначають порядок, згідно з яким повинні бути надіслані повідомлення. Кожний з цих N рядків містить два цілих числа: перше - x-координата та друге — y-координата вуличного перехрестя.

Вихідні дані Вихідний файл с ім'ям `hermes.out` повинен містити тільки єдиний рядок з єдиним цілим числом - мінімальною сумарною довжиною шляху, необхідного Гермесу для відсилки всіх повідомлень.

Приклад вхідних та вихідних даних

hermes.in
5 8 3 7 -7 8 1 -2 1 6 -5

hermes.out
11

Обмеження Для всіх тестів $1 \le N \le 20000$, $-1000 \le X_i, Y_i \le 1000$. В 50% тестів: $1 \le N \le 80$.

16.1.2 «Артеміда»

Завдання Зевс надав Артеміді, богині дикої природи, прямокутну область для вирощування лісу. Ліва сторона області відповідає відрізку додатної частини вісі OY, нижня сторона відповідає відрізку додатної частини вісі OX, а точка (0,0) — лівому нижньому куту області. Зевс наполягав, щоб Артеміда саджала дерева тільки в тих точках області, які мають цілі координати. Артеміді подобалось, коли ліс виглядає природно, тому вона саджала дере-

ва таким чином, щоб відрізок, який сполучає будь-яку пару дерев, не був паралельним вісім OX та OY.

Одного разу Зевс захотів, щоб Артеміда зрубила для нього дерева, слідуючи правилам:

1. Зевс вимагає зрубити не менше T дерев.

2. Щоб отримати прямокутну футбольну площадку для майбутніх футбольних перемог, Артеміда повинна зрубити всі дерева всередині деякої прямокутної області та жодного дерева зовні області.

3. Сторони цієї прямокутної області повинні бути паралельними вісям OX та OY.

4. Два протилежних кута області повинні находитися в місцях знаходження дерев, відповідно ці дерева також повинні бути зрублені.

Так як Артеміда любить дерева, вона хоче виконати умови, зрубив при цьому якомога менше дерев. Ви повинні написати програму, яка за інформацією про місцезнаходження дерев в лісі та про мінімальну кількість дерев T, яку необхідно зрубити, обирає область вирубки дерев для Артеміди.

Вхідні дані У вхідному файлі з ім'ям `artemis.in` перший рядок містить єдине ціле число N — кількість дерев у лісі. Другий рядок містить єдине ціле число T — мінімальна кількість дерев для вирубки. Наступні N рядків описують положення N дерев. Кожний з цих рядків містить два цілих числа: X та Y, x-координату і потім y-координату дерева.

Вихідні дані Вихідний файл з ім'ям `artemis.out` повинен містити один рядок з двома цілими числами I та J, розділеними одним пропуском. Артеміда повинна розглядати I-те дерево з координатами, заданими в рядку $I+2$ вхідного файлу, і J-те дерево з координатами, заданими в рядку $J+2$ вхідного файлу, як кути області для вирубки дерев. Порядок виводу цих двох чисел не суттєвий. Якщо існує декілька варіантів вибору такої пари дерев, ви повинні знайти і видати тільки один з них. Для всіх тестів існує хоча б один розв'язок.

Приклад вхідних та вихідних даних

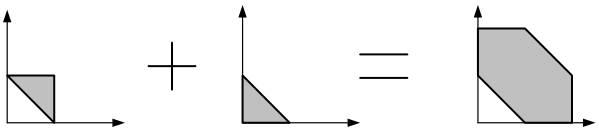
artemis.in	artemis.out
3 2 1 1 2 3	1 2

Обмеження У всіх тестах: $1 < N \leq 20000$, $0 \leq X, Y \leq 64000$ і $1 < T \leq N$. У 50 % тестів: $1 < N < 5000$.

16.1.3 «Багатокутник»

Завдання Точками багатокутника є як його внутрішні точки, так і точки на границі. Опуклим багатокутником називається такий багатокутник, в якому для будь-яких двох його точок X та Y відрізок, що сполучає X и Y, повністю належить багатокутнику. Всі багатокутники і цій задачі опуклі і містять не менше 2-ох вершин. Всі вершини будь-якого багатокутника в цій задачі різні і мають цілочисельні координати. Ніякі три вершини не лежать на одній прямій. Слово "багатокутник"надалі означатиме саме такі багатокутники.

Для двох заданих багатокутників A та B їх сумою Мінковського називається фігура, яка складається з всіх точок виду $(x_1 + x_2, y_1 + y_2)$, де (x_1, y_1) — усі можливі точки багатокутника A, і (x_2, y_2) — усі можливі точки багатокутника B. Сума Мінковського двох багатокутників також є багатокутником. Малюнок ілюструє суму Мінковського для двох трикутників.



Мал. 16.1:

Будемо розглядати операцію, обернену сумі Мінковського. Для даного багатокутника P необхідно знайти такі два багатокутника A та B, що:

- P є сумою Мінковського A та B;
- A має від 2 до 4 різних вершин, тобто це відрізок (2 вершини), трикутник (3 вершини) або чотирикутник (4 вершини);
- A повинен мати якомога більше вершин, тобто
  - A повинен бути чотирикутником, якщо це можливо;
  - якщо A не може бути чотирикутником, він повинен бути трикутником, якщо це можливо;
  - інакше він повинен бути відрізком.

Очевидно, що жоден з багатокутників A та B не може бути рівним P, оскільки в цьому випадку один з багатокутників A або B повинен бути точкою, що не є багатокутником за означенням.

Вам даний набір вхідних файлів, кожен з яких містить опис багатокутника P. Для кожного вхідного файлу ви повинні знайти багатокутникі A і

B такі, як описано вище, і створити вихідний файл, що містить опис A і B. Гарантується, що для кожного з даних вхідних файлів такі багатокутники існують. Якщо існує декілька варіантів відповіді, запишіть будь-який з них. Ви не повинні здавати будь-які програми, ви повинні здати на перевірку лише вихідні файли.

Вхідні дані Вам дано набір з 10 файлів з назвами `polygon1.in`, `polygon2.in`, ..., `polygon10.in`, де число після слова polygon є номером тесту. Кожний вхідний файл має такий формат. Перший рядок містить одне ціле число N — кількість вершин багатокутника P. Наступні N рядків описують вершини багатокутника в порядку їх обходу проти часової стрілки, по одній вершині в рядку. Рядок $i+1$ $(i = 1, 2, \ldots, N)$ містить два цілих числа X_i та Y_i, розділених пропуском: координати i-ої вершини багатокутника. Всі координати у вхідних файлах — невід'ємні цілі числа.

Вихідні дані Ви повинні здати 10 вихідних файлів, відповідних даним вхідним файлам з описами шуканих багатокутників A та B. Перший рядок кожного вихідного файлу повинен містити текст:

```
#FILE polygon I
```

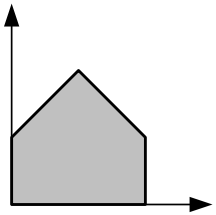
де ціле число I $(1 \le I \le 10)$ є номером відповідного вхідного файлу. Далі формат вихідного файлу подібний до формату вхідного файлу. Другий рядок повинен містити одне ціле число N_A — кількість вершин багатокутника A $(2 \le N_A \le 4)$. Наступні N_A рядків повинні описувати вершини багатокутника A в порядку обходу проти часової стрілки, по одній вершині в рядку. Рядки з номерами $i+2$ $(i = 1, 2, \le, N_A)$ повинні містити по два цілих числа X та Y — координати i-ої вершини багатокутника A. Числа в рядку повинні розділятися єдиним пропуском.

Рядок $N_A + 3$ повинен містити одне ціле число N_B - кількість вершин багатокутника B, $(2 \le N_B)$. Наступні N_B рядків повинні описувати вершини багатокутника B в порядку обходу проти часової стрілки, по одній вершині в рядку. Рядки з номерами $N_A + i + 3$ $(j = 1, 2, \le, N_B)$ повинні містити по два цілих числа X і Y - координати j-ой вершини багатокутника B. Числа в рядку повинні розділятися єдиним пропуском.

Приклади вхідних та вихідних даних

polygon0.in
5 0 1 0 0 2 0 2 1 1 2

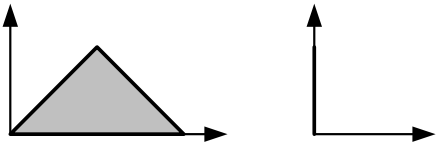
Для наведеного вхідного файлу будь-який з двох наведених нижче вихідних файлів коректний, так як в обох випадках A є трикутником і не



Мал. 16.2:

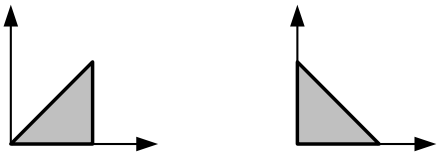
може бути чотирикутником.

```
#FILE polygon 0
3
0 0
2 0
1 1
2
0 1
0 0
```



Мал. 16.3:

```
#FILE polygon 0
3
0 0
1 0
1 1
3
0 1
0 0
1 0
```



Мал. 16.4:

16.2 Завдання другого туру

16.2.1 «Фідій»

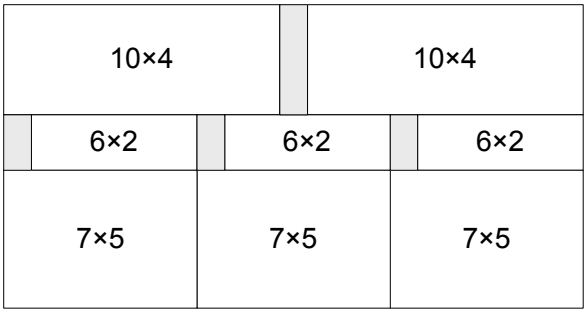
Завдання Відомий давньогрецький скульптор Фідій готується до побудови чергового видатного монумента. Для цього йому потрібні мармурові прямокутні плитки заданих розмірів $W_1 \times H_1, W_2 \times H_2, \ldots, W_N \times H_N$.

Нещодавно Фідій отримав велику прямокутну мармурову плиту. Він хоче розрізати плиту так, щоб отримати плитки заданих розмірів. Мармурова плита або плитка може бути розрізана тільки вертикальним або горизонтальним розрізом на дві прямокутні плитки. Розріз здійснюється від краю до краю. Це єдиний спосіб розрізання плити та плиток. Вирізанні плитки неможна сполучати ніяким чином. Так як мармур має рисунок, плитки неможна повертати. Тобто, якщо Фідій вирізав плитку розміром $A \times B$, вона не може бути використана в якості плитки $B \times A$, окрім випадків, коли $A = B$. Він може вирізати 0 або більше плиток кожного з потрібних розмірів, і при цьому можуть вийти плитки інших розмірів, які не будуть використовуватися. Фідій хоче знати, як потрібно розрізати вихідну плиту, щоб сумарна площа плиток, що не використовуються, була мінімальна.

Наприклад, нехай ширина вихідної плити рівна 21 і висота рівна 11, а задані розміри плиток: $10 \times 4, 6 \times 2, 7 \times 5$ і 15×10. Тоді мінімальна можлива площа плиток, що не використовуються, рівна 10. Малюнок ілюструє один з способів розрізання вихідної плити.

Треба написати програму, яка за заданими розмірами вихідної плити і розмірами потрібних плиток обчислює мінімальну сумарну площу плиток, що не використовуються.

Вхідні дані В першому рядку вхідного файлу `phidias.in` містяться два цілих числа. Перше число W — ширина вихідної плити, друге H — висота вихідної плити. Другий рядок містить єдине ціле число N — кількість потрібних розмірів плиток. Наступні N рядків містять потрібні розміри плиток. Кожний з цих рядків містить по два цілих числа: W_i — ширину і H_i — висоту ($1 \le i \le N$).



Мал. 16.5:

Вихідні дані Вихідний файл `phidias.out` повинен містити одне ціле число - мінімальну сумарну площу плиток, що не використовуються.

Приклад вхідних та вихідних даних

phidias.in	phidias.out
21 11 4 10 4 6 2 7 5 15 10	10

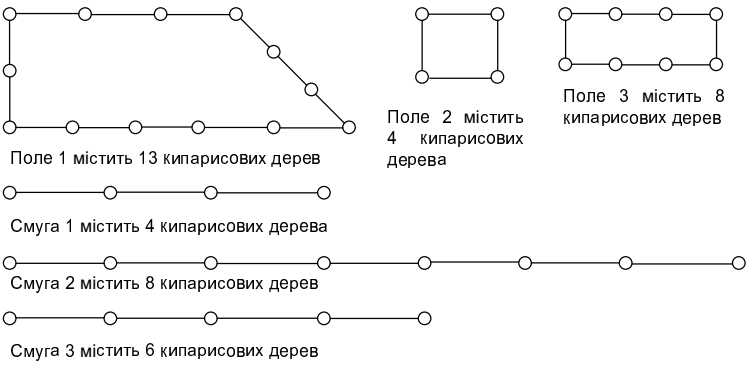
Обмеження У всіх вхідних файлах $1 \leq W \leq 600, 1 \leq H \leq 600, 0 < N \leq 200, 1 \leq W_i \leq W$ та $1 \leq H_i \leq H$. У 50% тестів: $W \leq 20, H \leq 20$ та $N \leq 5$.

16.2.2 «Фермер»

Завдання Фермер володіє декількома полями, на границі кожного з яких ростуть кипарисові дерева. У фермера також є смуги землі, на кожній з яких росте ряд кипарисових дерев. На границях полів і на смугах між кожною парою підряд стоячих кипарисових дерев росте одно оливкове дерево. Всі кипарисові дерева фермера ростуть або по границі поля, або на смузі, і всі оливкові дерева фермера знаходяться між двома сусідніми кипарисовими деревами на границі поля або на смузі.

Одного разу фермер сильно захворів і відчув, що скоро помре. За декілька днів до смерті він позвав свого старшого сина і сказав йому: "Я заповідаю тобі будь-які Q кипарисових дерев на твій вибір, а також всі оливкові дерева, які ростуть між будь-якими двома підряд стоячими кипарисовими деревами, обраними тобою". З кожного поля і з кожної смуги син може обрати будь-яку комбінацію дерев. Так як старший син любить оливки, він

хоче обрати такі Q кипарисових дерев, які дозволять йому успадкувати якомога більшу кількість оливкових дерев.



Мал. 16.6:

Розглянемо приклад, коли син отримує для полів і смуг, зображених на мал. 16.6, $Q = 17$ кипарисових дерев. Для того, щоб максимізувати кількість успадкованих оливкових дерев, він повинен обрати всі кипарисові дерева на полі 1 і полі 2, отримав, таким чином, 17 оливкових дерев. Необхідно написати програму, яка за інформацією про поля, смуги та кількість обраних сином кипарисових дерев визначає найбільшу можливу кількість оливкових дерев, яке син може успадкувати.

Вхідні дані Перший рядок вхідного файлу `farmer.in` мітить три цілих числа Q — кількість кипарисових дерев, яку повинен обрати син, потім M — кількість полів і потім K — кількість смуг. Другий рядок містить M цілих чисел $N_1, N_2, \ldots, N_M$ — кількості кипарисових дерев на полях. Третій рядок містить K цілих чисел $R_1, R_2, \ldots, R_K$ — кількості кипарисових дерев у смугах.

Вихідні дані В єдиному рядку вихідного файлу `farmer.out` повинно знаходитися одне ціле число - найбільша можлива кількість оливкових дерев, яку може успадкувати син.

Приклад вхідних та вихідних даних

farmer.in	farmer.out
17 3 3 13 4 8 4 8 6	17

Обмеження У всіх тестах $0 \le Q \le 150000, 0 \le M \le 2000, 0 \le K \le 2000, 3 \le N_1 \le 150, 3 \le N_2 \le 150, \ldots, 3 \le N_M \le 150, 2 \le R_1 \le 150, 2 \le R_2 \le$

$150, \ldots, 2 \leq R_K \leq 150$. Загальна кількість кипарисових дерев на полях и в смугах більша або рівна Q. У 50% тестів $Q \leq 1500$.

16.2.3 «Емподіо»

Завдання Відомий математик та філософ Піфагор вважав, що реальність має математичний характер. Сучасні біологи вивчають властивості біопослідовностей. Біопослідовність - це послідовність M цілих чисел, яка:

- містить всі цілі числа $0, 1, \ldots, M-1$;
- починається з 0 і закінчується числом $M-1$;
- не має двох підряд елементів E, $E+1$ саме в цьому порядку.

Підпослідовність елементів біопослідовності, що слідують підряд, називається сегментом.

Сегмент біопослідовності називається фрейм-інтервалом, якщо він містить всі цілі числа, значення яких знаходяться між значеннями першого і останнього елементів сегмента. При цьому перший елемент повинен бути найменшим в сегменті, а останній елемент - найбільшим в сегменті і відрізнятися від першого. Фрейм-інтервал називається емподіо, якщо він не містить всередині себе більш короткий фрейм-інтервал.

В якості приклада розглянемо біопослідовність $(0, 3, 5, 4, 6, 2, 1, 7)$. Вся біопослідовність є фрейм-інтервалом. Ця біопослідовність, в свою чергу, містить інший фрейм-інтервал $(3, 5, 4, 6)$, і, відповідно, вона не є емподіо. Фрейм-інтервал $(3, 5, 4, 6)$ не містить більш короткий фрейм-інтервал, і тому, він є емподіо. Цей фрейм-інтервал є єдиним емподіо в даній біопослідовності. Треба написати програму, яка по задані біопослідовності знаходить всі емподіо, які в ній містяться.

Вхідні дані Перший рядок вхідного файлу з ім'ям `empodia.in` містить єдине ціле число M — кількість цілих чисел у вхідній біопослідовності. Наступні M рядків містять цілі числа — елементи біопослідовності. Кожний з цих M рядків містить єдине ціле число.

Вихідні дані Перший рядок вихідного файлу з ім'ям empodia.out містить єдине ціле число H — кількість емподіо, які містяться у вхідній біопослідовності. Наступні H рядків описують всі наявні емподіо у вхідній біопослідовності. Кожен з цих рядків містить два цілих числа A та B (саме в такому порядку), розділених пропуском, де A — порядковий номер елемента вхідної біопослідовності, який є першим елементом емподіо; B — порядковий номер елемента вхідної біопослідовності, який є останнім елементом емподіо. Емподіо необхідно виводити в порядку збільшення числа A.

Приклад вхідних та вихідних даних

empodia.in
8 0 3 5 4 6 2 1 7

empodia.out
1 2 5

Обмеження Для одного тесту $1000000 \leq M \leq 1100000$. Для всіх інших тестів $1 \leq M \leq 60000$. У 50% тестів $M \leq 2600$.

Розділ 17. Польща'2005

17.1 Завдання першого туру

17.1.1 «Садок»

Умова Байтмен володіє найкрасивішим садком в Байттауні, в якому він посадив n троянд. Прийшло літо и квіти виросли великими и красивими. Байтмен зрозумів, що він не в змозі самостійно піклуватися за усіма трояндами, і вирішив найняти двох садівників на допомогу. В такому випадку йому треба вибрати дві прямокутні області, щоб кожний з садівників піклувався за трояндами в одній з них. Області не повинні перетинатися, і в кожній має бути рівно k троянд.

Байтмен хоче встановити паркан, що огороджує прямокутні області. Для економії грошей паркан має бути якомога коротший. Ваша задача — допомогти Байтмену обрати дві прямокутні області.

Сад є прямокутником довжиною l метрів і шириною w метрів, який розділений на $l \cdot w$ однакових одиничних квадратів розміром 1×1 метрів кожний. Зафіксуємо координатну систему так, щоб осі координат були паралельні сторонам саду. Всі квадрати мають цілі координати (x, y), що задовольняють обмеженням $1 \leq x \leq l$, $1 \leq y \leq w$. В кожному одиничному квадраті може міститися довільна кількість троянд.

Сторони прямокутних областей, які обираються, мають бути паралельні сторонам саду, а їх кутові одиничні квадрати - мати цілі координати. Прямокутна область з кутовими одиничними квадратами (l_1, w_1), (l_1, w_2), (l_2, w_1) и (l_2, w_2) (для $1 \leq l_1 \leq l_2 \leq l$ и $1 \leq w_1 \leq w_2 \leq w$):

- містить всі одиничні квадрати з координатами (x, y), які задовольня-

ють умові $l_1 \le x \le l_2$ и $w_1 \le y \le w_2$, и

- має периметр $2 \cdot (l_2 - l_1 + 1) + 2 \cdot (w_2 - w_1 + 1)$.

Дві прямокутні області не повинні перетинатися, тобто, вони не повинні мати ні одного спільного квадрата. Навіть якщо вони мають спільну сторону або її частину, вони огороджуються різними парканами.

Завдання Напишіть програму, яка:

- читає зі стандартного вводу розміри садка, загальну кількість троянд в садку, кількість троянд, що має знаходитись у кожній прямокутній області, і позицію кожної троянди в садку, що визначається координатами одиничного квадрата, в якому вона знаходиться,
- знаходить кутові одиничні квадрати двох таких прямокутних областей з мінімальною сумою периметрів, які задовольняють заданим умовам,
- виводить у стандартний вивід мінімальне значення суми периметрів двох прямокутних областей, що не перетинаються, кожна з яких містить точно задану кількість троянд (або єдине слово `NO`, якщо такої пари прямокутних областей не існує).

Вхідні дані Перший рядок стандартного вводу містить два числа: l та w ($1 \le l, w \le 250$), що розділені одним пропуском — довжину та ширину садка. Другий рядок містить два числа: n та k ($2 \le n \le 5000$, $1 \le k \le n/2$), що розділені одним пропуском і позначають загальну кількість троянд в садку та кількість троянд, що має бути у кожній з прямокутних областей. Наступні n рядків містять позиції троянд, по одній троянді у кожному рядку. Кожний $(i+2)$-й рядок містить два числа l_i, w_i ($1 \le l_i \le l$, $1 \le w_i \le w$), що розділені одним пропуском — координати квадрата, що містить i-у троянду.

В одному квадраті може міститися дві або більша кількість троянд.

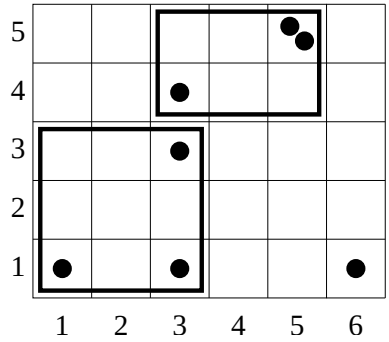
В 50% тестів розміру садка будуть задовольняти обмеженням $l, w \le 40$.

Вихідні дані В перший і єдиний рядок стандартного виводу ваша програма має вивести одно число — мінімальну суму периметрів двох прямокутних областей, кожна з яких містить рівно k троянд, або єдине слово `NO`, якщо таких прямокутників немає.

Приклад вхідних та вихідних даних

Стандартний вхід
6 5 7 3 3 4 3 3 6 1 1 1 5 5 5 5 3 1

Стандартний вихід
22



Мал. 17.1:

17.1.2 «Посідовність середніх»

Умова Розглянемо неспадну послідовність $s_1, \ldots, s_{n+1}$ ($s_i \leq s_{i+1}$ для $1 \leq i \leq n$). Послідовність $m_1, \ldots, m_n$ в якій кожний член визначений як $m_i = \frac{1}{2}(s_i + s_{i+1})$ для $1 \leq i \leq n$, назвемо "середньою послідовністю" для послідовності $s_1, \ldots s_{n+1}$. Наприклад, середня послідовність для послідовності 1, 2, 2, 4 є 1.5, 2, 3. Зауважимо, що елементи середньої послідовності можуть бути дробовими числами. Тим не менш, в даній задачі використовуються тільки ті середні послідовності, в яких всі числа цілі.

Для заданої неспадної послідовності з n цілих чисел $m_1, \ldots, m_n$, необхідно обчислити кількість всіх неспадних послідовностей з $n + 1$ цілих чисел $s_1, \ldots, s_{n+1}$, для яких подана послідовність $m_1, \ldots, m_n$ є середньою послідовністю.

Завдання Напишіть програму, яка:

- читає зі стандартного вводу неспадну послідовність цілих чисел;

- вираховує кількість всіх неспадних послідовностей цілих чисел, для яких подана послідовність є середньою послідовністю;
- виводить відповідь у стандартний вивід.

Вхідні дані Перший рядок стандартного вводу містить одне ціле число n ($2 \le n \le 5\,000\,000$). n рядків, що залишилось, містять значення послідовності $m_1, \ldots, m_n$. Рядок $i+1$ містить одне ціле число m_i ($0 \le m_i \le 1\,000\,000\,000$). Можете вважати, що для 50% тестів будуть справедливими нерівності $n \le 1\,000$ и $0 \le m_i \le 20\,000$.

Вихідні дані Ваша програма повинна вивести в стандартний вивід рівно одне ціле число — кількість всіх неспадних послідовностей цілих чисел, для яких друга послідовність є середньою послідовністю.

Приклад вхідних та вихідних даних

Стандартний вхід	Стандартний вихід
3 2 5 9	4

Для даного прикладу існує рівно чотири неспадних послідовності цілих чисел, для яких послідовність $2, 5, 9$ є середньою послідовністю:

- $2, 2, 8, 10$,
- $1, 3, 7, 11$,
- $0, 4, 6, 12$,
- $-1, 5, 5, 13$.

17.1.3 «Гори»

Умова В Гірському Парку Разваг відкрився новий атракціон з американськими гірками. Трек в атракціоні складається з n колій, з'єднаних послідовно один з одним, причому перша колія починається на висоті 0. Оператор Байтмен може змінювати конфігурацію треку на власний розсуд, корегуючи нахил деяких послідовно з'єднаних рельсів. Нахил решти колій при цьому не змінюється. Кожного разу, коли нахил деяких колій змінюється, всі наступні колії відповідно підіймаються або опускаються. При цьому висота початку треку завжди залишається рівною 0. Малюнок на наступній сторінці показує приклад зміни конфигурацій треку.

Кожний заїзд починається з того, що кабіна запускається з початку треку з енергією, достатньою для досягнення висоти h. Тобто, кабіна буде продовжувати рух по рельсам, доки її висота не стане більше h або доки вона не досягне кінця треку.

Обчисліть по записам про всі заїзди та змінах конфігурацій треку кількість колій, які були повністю пройдені кабіною в кожному з заїздів до її зупинки.

В атракціоні трек описується послідовністю з n нахилів, по одному для кожної колії. i-е число d_i рівне різниці висот (в сантиметрах) між кінцем i-ї колії та її початком. Тобто, якщо після прохождення по першим $i-1$ коліям кабіна опиняється на висоті h сантиметрів, то після проходження по i коліям вона буде на висоті $h+d_i$ сантиметрів.

На початку всі колії горизонтальні, тобто $d_i = 0$ для всіх i. Заїзди та зміни конфигурації відбуваються протягом дня. Кожна зміна конфігурації описується трьома числами: a, b та D. Така зміна торкається колій з a-го по b-й (включно). Нахил кожної з цих колій встановлюється рівним D. Тобто, $d_i = D$ для всіх $a \le i \le b$.

Кожний заїзд задається одним числом h — максимальною висотою, на яку може поднятися кабіна.

Завдання Напишіть програму, яка:

- читає зі стандартного вводу послідовність описів змін конфігурацій треку та заїздів,
- для кожного заїзду обчислює кількість колій, які були пройдені кабіною,
- виводить результати в стандартний вивід.

Вхідні дані Перший рядок вхідних даних містить одне додатнє ціле число n — кількість колій в треку, $1 \le n \le 1\,000\,000\,000$. Наступні рядки містять опис змін конфігурації та заїздів. Останній рядок вхідних даних містить ознаку закінчення. Кожний рядок, починаючи з другого, може містити:

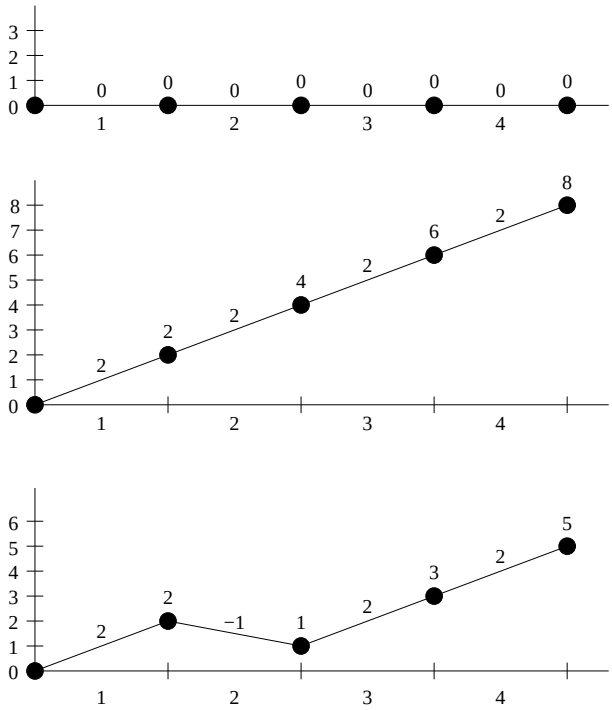
- Опис зміни конфігурації треку — один символ '`I`' та цілі числа a, b и D, розділені одним пропуском ($1 \le a \le b \le n$, $-1\,000\,000\,000 \le D \le 1\,000\,000\,000$).
- Опис заїзду — один символ '`Q`' та ціле число h ($0 \le h \le 1\,000\,000\,000$), розділені одним пропуском.
- Один символ "`E`" — ознака кінця вхідних даних.

Ви можете вважати, що в кожний момент висота довільної точки треку міститься на проміжку $[0, 1\,000\,000\,000]$. Вхідні дані містять не більше 100 000 рядків. В 50% тестів число n буде задовольняти обмеженню $1 \le n \le 20\,000$, та вхідні дані будуть містити не більше 1 000 рядків.

Вихідні дані *i*-й рядок вихідних даних повинен містити єдине ціле число — кількість колій, які пройдені кабіною в *i*-му заїзді.

Приклад вхідних та вихідних даних

Стандартний вхід	Стандартний вихід
4 Q 1 I 1 4 2 Q 3 Q 1 I 2 2 -1 Q 3 E	4 1 0 3



Мал. 17.2:

На малюнку представлені зображення треку до та після кожної зміни конфігурації. По віси x відкладені номери рельсів. По віси y відкладена висота. Числа поряд з виділеними точками означають їхню висоту. Числа поряд з відрізками означають нахили.

17.2 Завдання другого туру

17.2.1 «День народження»

Умова Сьогодні день народження Байтмена. На святкування дня народження запрошені n дітей (включаючи самого Байтмена). Всі діти пронумеровані числами від 1 до n. Батьки Байтмена приготували великий круглий стіл та поставили біля нього n стільців.

Як тільки діти приходять на день народження, вони разсаджуються за столом. Дитина з номером 1 займає одне з місць. Дитина з номером 2 займає місце зліва від дитини з номером 1. Дитина з номером 3 займає наступне за ним місце зліва і так далі. Наприкінці, дитина з номером n займає вільне місце, що залишилось, між дітьми с номерами 1 та $n-1$.

Батьки Байтмена добре знають, що деякі з запрошених дітей поводять себе доволі шумно за столом, якщо сидять поряд. Тому батьки збираються пересадити дітей в деякому порядку. Цей порядок описується перестановкою $p_1, p_2, \ldots, p_n$ ($p_1, p_2, \ldots, p_n$ різні цілі числа від 1 до n). Тобто, дитина p_1 повинна сидіти між p_n та p_2, дитина p_i (для $i = 2, 3, \ldots, n-1$) повинна сидіти між p_{i-1} та p_{i+1}, та дитина p_n повинна сидіти між p_{n-1} та p_1.

Нажаль, всі діти прийшли й розсілися так, як вказано у першому абзаці. Тому для того, щоб розсадити дітей в потрібному порядку, батькам прийдеться пересадити когось з дітей на деяку кількість місць вліво або вправо. Для кожної дитини батькам необхідно вирішити, як вона буде пересаджуватися: вони повинні обрати напрямок (вліво чи вправо) та відстань (на скільки місць треба переміститися). Потім, по заданому сигналу всі діти повинні одночасно встати та переміститися на місця, визначені батьками Байтмена.

Пересаджування дітей вносить гармидер у святкування дня народження. Розміри гармидеру пропорційні максимальній з відстаней, на яку діти будуть пересаджені. Пересаджування дітей можна організувати багатьма способами. Батьки вирішили вибрати той з них, при якому розмір гармидеру буде мінімальним, тобто той спосіб, при якому максимальна з відстаней пересаджування буде мінімальною. Допоможіть їм знайти такий спосіб.

Завдання Ваша задача - написати програму, яка:

- читає з стандартного вводу кількість дітей та перестановку, що задає бажаний порядок разсадки дітей,
- вираховує мінімально можливий розмір безпорядку,
- виводить результат у стандартний вивід.

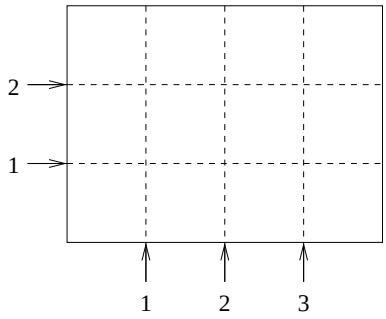
Мал. 17.3:

На малюнку зліва зображена початкова розсадка дітей. На малюнку в середині показано результат пересаджування, при якому діти з номерами 1 та 2 переміщуються на одне місце, діти з номерами 3 та 5 переміщуються на два місця та діти з номерами 4 та 6 не міняють свого положення. Умови розсадження, що вимагаються - виконані, оскільки 3-а сидить між 6-ю та 4-ю, 4-а сидить між 3-ю та 5-ю, 5-а сидить між 4-ю и 1-ю, 1-а сидить між 5-ю та 2-ю, 2-а сидить між 1-ю та 6-ю та 6-а сидить між 2-ю та 3-ю. Також існує інший варіант кінцевої разсадки дітей, зображених на малюнку справа. В обох варіантах розмір гармидеру дорівнює 2.

17.2.2 «Гра в прямокутник»

Умова Розглянемо гру для двох гравців. Гравці мають прямокутник розміром $x \times y$ (де x та y — додатні цілі числа).Гравці ходять по черзі. Хід складається з розділення прямокутника на два прямокутика одним горизонтальним або вертикальним розрізом. Отриманні прямокутники повинні мати додатні цілочисельні розміри.

Після кожного розрізу прямокутник з меншою площею прибирається, а той що залишився передається іншому гравцю. Якщо прямокутник розділено на дві рівні частини, то прибирається одна з них. Гравець, який отримує



Мал. 17.4: Можливі розрізи прямокутника розміром 4×3

прямокутник розміром 1×1, програє, так як не може зробити наступний хід.

Ваша задача - написати програму, яка б грала в гру з прямокутниками й вигравала. Для того щоб грати, програма повинна використовувати спеціальну бібліотеку. В бібліотеці є функції `dimension_x()` та `dimension_y()` (), що повертають розміри прямокутника. Початкові розміри прямокутника - цілі числа від 1 до 100 000 000. Як мінімум один з розмірів більше 1. До того ж, у 50% розміри прямокутника не будуть перевищувати 25.

В бібліотеці є також процедура `cut(dir, position)`), яка повинна визиватися вашою програмою, щоб зробити хід. Параметри `dir` та `position` описують напрямок та позицію розрізу відповідно. Параметр `dir` може приймати одне з двох значень: `vertical` або `horizontal`. Якщо `dir` = `vertical`, то проводиться вертикальний розріз, а параметр `position` вказує x-розрізу, як показано на малюнку вище. При цьому ви повинні гарантувати виконання нерівності $1 \leq$ `position` $\leq$ `dimension_x`$() - 1$. Якщо `dir` = `horizontal`, то проводиться горизонтальний розріз, а параметр `position` вказує y-координату розрізу. При цьому, ви повинні гарантувати виконання нерівності $1 \leq$ `position` $\leq$ `dimension_y`$() - 1$.

Після запуску вашої програми вона буде грати за одного з гравців. Ваша програма ходить першою, вона повинна розрізати початковий прямокутник. Коли ваша програма визиває процедуру `cut` procedure, ваш хід записується та керування передається програмі суперника. Після ходу суперника керування повертається вашій програмі. Значення, які повертаються функціями `dimension_x()` та `dimension_y()` будуть відображувати результат вашого ходу та ходу суперника. Як тільки ваша програма виграє, програє або робить невірний хід, її виконання буде перервано. Переривання вашої програми - це автоматичний процес, так що ваша програма повин-

на продовжувати робити стільки ходів, скільки можливо до автоматичного переривання її виконання. Ви можете припустити, що для запропонованих вхідних даних завжди існує виграшна стратегія для вашої програми. Ваша програма не повинна читати які-небудь файли або писати в них, не повинна використовувати стандартний ввід та вивід та не повинна пробувати міняти вміст якої-небудь пам'яті поза вашою програмою. Порушення довільного з цих правил може призвести до дискваліфікації.

Експериментування Для того, щоб дати вам можливість поекспериментувати з бібліотекою, у вашому розпорядженні є приклади бібліотек: їхні програмні тексти знаходяться у файлах `preclib.pas`, `creclib.c` та `creclib.h` Вони реалізують дуже просту стратегію. Коли ви запустите вашу програму, вона буде грати проти цих простих суперників. Ви можете змінювати їх, щоб протестувати вашу програму з кращими суперниками. Майте на увазі, що під час тестування після закінчення туру, ваша програма буде грати проти іншого суперника.

Під час посилки вашої програми на перевірку з використанням інтерфейсу TEST , вона буде відкомпільована з немодифікованою бібліотекою суперника. При цьому файл що відсилається вами буде переданий як стандартний ввід вашої програми. Вхідний файл повинен складатися з двох рядків, кожна з яких повинна містити по одному цілому числу. Перший рядок має містити початкову ширину, друга - початкову висоту прямокутника. Ці розміри будуть прочитані бібліотекою, що пропонується як приклад.

Бібліотеки В вашому розпорядженні є бібліотеки, які забезпечують наступну функціональність:

- **Бібліотека для FreePascal** (`preclib.ppu`, `preclib.o`)

```
type direction = (vertical, horizontal);
function dimension_x():longint;
function dimension_y():longint;
procedure cut(dir:direction; position:longint);
```

- **Бібліотека для GNU C/C++** (`creclib.h`, `creclib.c`)

```
typedef enum __direction {vertical, horizontal} direction;
int dimension_x();
int dimension_y();
void cut(direction dir, int position);
```

Приклад взаємодії Нижче приведено приклад взаємодії вашої програми з бібліотекою. Він показує, як може проходити гра. Гра починається з прямокутника розміром 4×3. Існує виграшна стратегія для цього випадку.

Ваша програма визиває	Що робиться
`dimension_x()`	повертає 4
`dimension_y()`	повертає 3
`cut(vertical, 1)`	ваш розріз записується, та прямокутник розміром 3×3 передається вашому супернику, який розрізає його та отримує прямокутник розміру 3×2; після цього керування передається вашій програмі
`dimension_x()`	повертає 3
`dimension_y()`	повертає 2
`cut(horizontal, 1)`	ваш розріз записується, та прямокутник розміром 3×1 передається вашому супернику, який розрізає його та отримує прямокутник розміру 2×1; після цього керування передається вашій програмі
`dimension_x()`	повертає 2
`dimension_y()`	повертає 1
`cut(vertical, 1)`	в результаті вашого розрізу виходить прямокутник розміром 1×1 , так що ви виграли; після цього робота вашої програми автоматично перестає працювати

17.2.3 «Річки»

Умова Майже все Королівство Байтленд вкрито лісами та ріками. Малі ріки зливаються в крупніші ріки, які, в свою чергу, зливаються одна з одною; наприкінці, всі річки зливаються разом в одну велику ріку. Велика ріка впадає в море поблизу міста Байттаун.

В Байтленді є n лісозаготівельних селищ, кожне з яких розташоване поблизу якої-небудь ріки. На сьогодні в Байттауні знаходиться велика пилорама, яка опрацьовує всі дерева, зрублені в Королівстві. Дерева сплавляються вниз по рікам від селищ, де вони зрублені, до пилорами в Байттауні. Король Байтленду вирішив поставити k додаткових пилорам в селищах, щоб зменшити вартість сплаву дерев. Після установки пилорам дерева не обов'язково повинні сплавлятися в Байттаун, тепер їх можна обробити на найближчій пилорамі, що знаходиться нижче за течією рік. Очевидно, що дерева, зрублені в околі селища з пилорамою, взагалі не сплавляються по рікам.

Необхідно відмітити, що ріки в Байтленді не розгалужуються. З цього слідує, що для кожного селища існує єдиний шлях сплаву дерев вниз за течією рік від нього в Байттаун.

Королівські рахувальники підрахували кількість дерев, що зрубуються в кожному селищі за рік. Вам необхідно визначити, в яких селищах треба встановити пилорами, щоб мінімізувати сумарну вартість сплаву дерев за рік. Вартість сплаву одного дерева коштує один цент за кожний кілометр шляху.

Завдання Напишить програму, що:

- читає зі стандартного вводу кількість селищ, кількість додаткових пилорам, які будуть встановлені, кількість зрублених в кожному селищі дерев та опис рік,
- вираховує мінімальну вартість сплаву дерев після встановлення додаткових пилорам,
- виводить результат у стандартний вивід.

Вхідні дані Перший рядок вхідних даних містить два цілих числа: n — кількість селищ, не рахуючи Байттауна ($2 \leq n \leq 100$), та k — кількість додаткових пилорам, які будуть встановлені ($1 \leq k \leq 50$ та $k \leq n$).Селища нумеруються числами $1, 2, \ldots, n$, а Байттаун має номер 0.

Кожен з наступних n рядків містить три цілих числа, розділених одним пропуском. Рядок $i + 1$ містить:

- w_i — кількість дерев, що зрубуються в селищі i за рік ($0 \leq w_i \leq 10\,000$),
- v_i — найближче селище(або Байттаун) вниз по річці від селища i ($0 \leq v_i \leq n$),
- d_i — відстані (в кілометрах) по річці від селища i до v_i ($1 \leq d_i \leq 10\,000$).

Гарантується, що сумарна вартість сплаву всіх дерев до пилорами в Байттауні не перевищує 2 000 000 000 центів в рік.

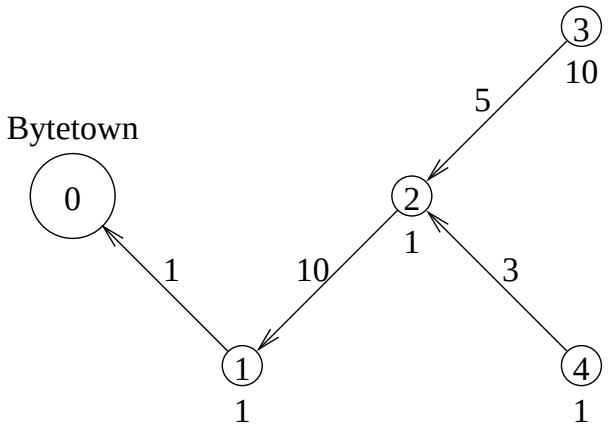
В 50 % тестів число n не перевищує 20.

Вихідні дані Перший і єдиний рядок вихідних даних повинен містити одне ціле число: мінімальну вартість сплаву (в центах).

Приклад вхідних та вихідних даних

Стандартний вхід
4 2 1 0 1 1 1 10 10 2 5 1 2 3

Стандартний вихід
4



Мал. 17.5:

Малюнок згори ілюструє вхідні дані прикладу. Номери селищ вказані всередині кругів. Числа під кругами позначають кількість дерев, що зрубуються поблизу цього селища. Числа над стрілками вказують довжини рік.

Пилорами повинні бути встановлені в селищах 2 та 3.

Частина II

Рекомендації до розв'язання

Розділ 18. Болгарія'1989

18.1 Завдання першого туру

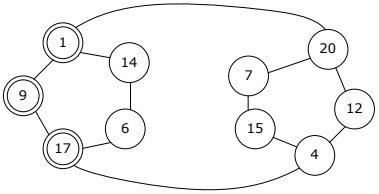
18.1.1 «AB»

Оскільки перший пункт завдання очевидний, то перейдемо відразу до другого й третього пунктів. Дамо відповідь на питання, скільки різних послідовностей існує для заданого значення N. Кількість способів розміщення `OO` у послідовності довжини $2N$ дорівнює $2N-1$. Для кожного способу розташування `OO` у $2N-2$ позиціях, що залишилися, $N-1$ букву `A` можна розмістити C_{2N-2}^{N-1} способами. Разом одержуємо $(2N-1)\cdot C_{2N-2}^{N-1}$ варіантів.

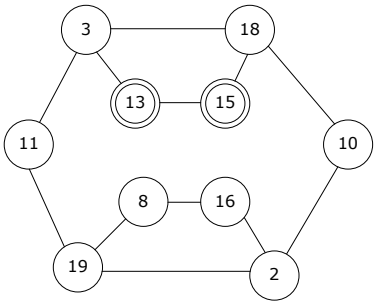
Зауважимо, що при $N \le 5$ кількість можливих станів нашої смужки не перевищує $9C_8^4 = 630$. Ми можемо побудувати граф, вершини якого будуть відповідати можливим конфігураціям смужки, а ребра — можливим переходам між ними. Тоді наша вихідна задача про пошук найкоротшої послідовності ходів зведеться до задачі про знаходження найкоротшого шляху в графі від заданої вершини до однієї з «кінцевих» вершин. Для розв'язання цієї задачі можна застосувати відомий алгоритм пошуку в ширину. Розглянемо, наприклад, випадок $N = 3$. Випишемо всі можливі послідовності довжини 6 (таких 30) і пронумеруємо їх.

№	Послідовність	№	Послідовність	№	Послідовність
1	`OOAABB`	11	`BAOOAB`	21	`OOABAB`
2	`OOABBA`	12	`BBOOAA`	22	`ABOOAB`
3	`OOBAAB`	13	`AABOOB`	23	`ABABOO`
4	`OOBBAA`	14	`ABAOOB`	24	`AOOBBA`
5	`AOOABB`	15	`BABOOA`	25	`ABBOOA`
6	`AOOBAB`	16	`BBAOOA`	26	`OOBABA`
7	`BOOABA`	17	`AABBOO`	27	`BAOOBA`
8	`BOOBAA`	18	`ABBAOO`	28	`BABAOO`
9	`AAOOBB`	19	`BAABOO`	29	`BOOAAB`
10	`ABOOBA`	20	`BBAAOO`	30	`BAAOOB`

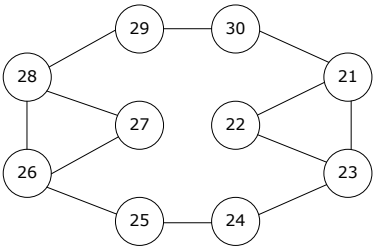
Почнемо з послідовності `OOAABB` (першої в таблиці 1) і розглянемо всі можливі послідовності, які можна отримати з неї за допомогою правила переміщення за один або більшу кількість кроків. Отримаємо множину з



Мал. 18.1: `OOAABB`



Мал. 18.2: `OOABBA`



Мал. 18.3: `OOABAB`

10 послідовностей, пов'язаних між собою так, як це показано на рис. 18.1. Якщо почати з послідовності `OOABBA` (другої в таблиці 1), то результат — множина послідовностей на рис. 18.2. І нарешті, для початкової послідовності `OOABAB` результат представлений на рис. 18.3. Таким чином, множину послідовностей довжини 6 ми розбили на три не зв'язаних між собою підмножини. Якщо введена послідовність належить третій підмножині, то мета недосяжна, а якщо одній з перших двох, то досяжна, при цьому наочно видно шлях її досягнення за мінімальну кількість переміщень. На рис. 18.1, 18.2 подвійною лінією виділені номери послідовностей, що задовольняють умові задачі.

Для $N > 3$ отриманий граф завжди буде зв'язний, тобто розв'язок завжди існує.

Залишається незрозумілим тільки одне питання: необхідно якимось чином кодувати позиції числами. Зауважимо, що, вилучивши з рядка символів, що описує позицію, нулі, ми одержимо рядок з $N-1$ символу `A` і $N-1$ символу `B`. Позиції символів `A` в отриманому рядку утворюють $(N-1)$-елементну підмножину множини $[1..2N-2]$. Всі такі сполучення з $2N-2$ елементів по $N-1$ ми можемо занумерувати натуральними числами в ле-

ксикографічному порядку. Нехай K — номер сполучення, що відповідає отриманому рядку, а нулі у вихідному рядку займали позиції з номерами P і $P+1$. Тоді вихідному рядку можна поставити у відповідність пару чисел (P, K). При такій відповідності кожна позиція кодується парою чисел (P, K), де $1 \le P \le 2-1$ і $1 \le K \le C_{2N-2}^{N-1}$, причому кожній такій парі відповідає якась позиція, а різні позиції кодуються різними парами (при заданому N).

У таблиці 2 наведені тести, що використовувалися на міжнародній олімпіаді.

№	Конфігурація	Очікувана відповідь
1	OOABAB	Немає розв'язку
2	OABAOBAB	Некоректне
3	OOABABABAB	Розв'язок з 4 ходів
4	ABBAOOABAB	Розв'язок з 3 ходів

Розділ 19. CPCP'1990

19.1 Завдання першого туру

19.1.1 «Гра 14»

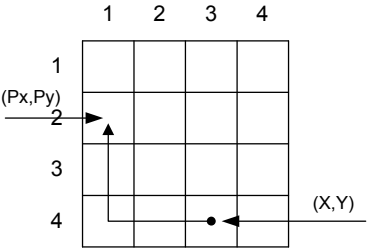
Надалі, за аналогією з відомою грою «П'ятнадцять», будемо називати числа, що містяться у клітинках, фішками. Про порожню клітинку будемо говорити, що в ній міститься «порожня» фішка.

Для початку розв'яжемо задачу зі спрощеним правилом переміщення. Нехай дозволяється обмін між сусідніми клітинками незалежно від того, зайняті вони чи ні. Реалізуємо пункти завдання при цьому правилі. Якщо ми це зробимо, то потім залишиться тільки додати логіку врахування переміщення фішок через порожні клітинки. У масиві A фіксуємо місця розташування всіх фішок, так, якщо $A[2,3] = 10$, то фішка з номером 10 міститься у другому рядку й у третьому стовпці. Стовпці нумеруються зліва праворуч, рядки — згори донизу.

Загальна логіка зводиться в цьому випадку до одного рядка: кожну з 14 фішок встановити на своє місце.

Позначимо через (X, Y) координати клітинки, у якій міститься фішка з номером N, а через (P_x, P_y) — місце, де вона повинна міститися (рис. 2).

На рис. 2 виділено шлях переміщення фішки на своє місце. Він складається з окремих кроків: перемістити фішку N із клітки (X, Y) у клітку $(X + D_x, Y + D_y)$; вивести таблицю до переміщення й після переміщення.



Мал. 19.1:

І в цілому повне переміщення реалізується наступним чином:

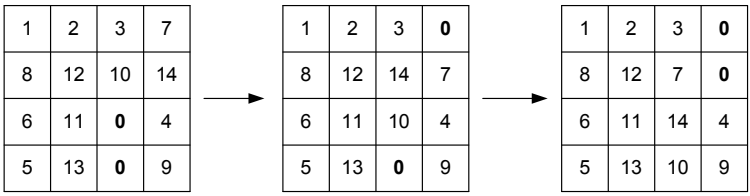
Обчислюємо P_x і P_y: $P_x := (N-l)mod4+1; P_y := (N-l)mod4+1;$

Знаходимо в таблиці клітинку (X,Y) з фішкою $ND_x := Sgn(P_x - X); D_y := Sgn(P_y - Y);$

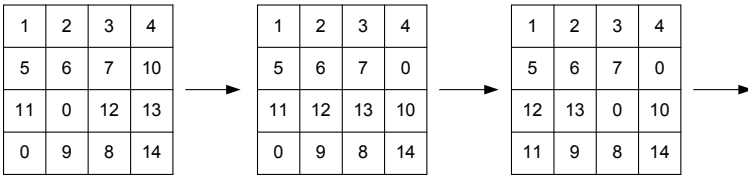
Доки $X <> P_x$, переміщати фішку N із клітинки (X,Y) у клітинку $(X+D_x,Y)$; Доки $Y <> P_y$, переміщати фішку N із клітинки (X,Y) у клітинку $(X,Y+D_y)$.

Таким чином, якби не було порожніх клітинок, тобто правило переміщення зводилося до обміну значень у сусідніх клітинках, то завдання було б виконано. Перейдемо до подальшого розгляду завдання. Отже, є дві порожні клітинки, і переміщення фішки на своє місце можливо тільки через порожню клітинку. Ідея розв'язку в цьому випадку досить проста. Перед кожним ходом на шляху із клітинки (X,Y) у клітинку (P_x,P_y) необхідно ставити порожню фішку. Координати порожньої фішки позначимо через (S_x,S_y). При цьому вибираємо найближчу до клітинки (X,Y) порожню фішку, крім особливих випадків. На яке місце ставити порожню фішку? Це клітинка з координатами $(X+D_x,Y)$ при переміщенні по осі X і $(X,Y+D_y)$ при переміщенні по осі Y. Процедура переміщення при цьому незначно зміниться. Необхідно поміняти місцями порожню фішку й фішку з номером N, а також запам'ятати нове місце порожньої фішки. Крім того, будемо дотримуватися наступних обмежень, що спрощують логіку розміщення фішок: по-перше, фішку з номером N, яку ми встановлюємо на своє місце, при переміщенні порожньої фішки переміщати не дозволяється; по-друге, раніше встановлені на свої місця фішки більше «торкати» не можна.

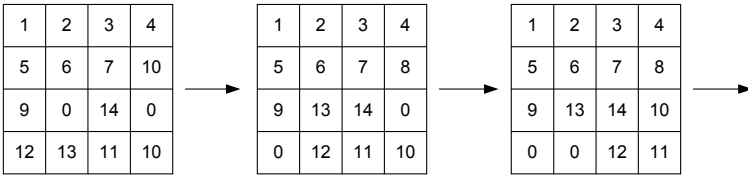
Одне з основних питань, що виникає при розв'язуванні задачі: навіщо дві порожні фішки? Відповідь на нього дають особливі випадки у розміщенні фішок. Нехай на своєму місці містяться фішки з номерами 1, 2, 3 рис. 19.2-19.4. Черга за фішкою з номером 4. При переміщенні фішки 4 ми домовилися «не торкати» фішки з номерами 1, 2, 3. Наші дії. Поміщаємо в



Мал. 19.2:)



Мал. 19.3:)



Мал. 19.4:

клітинку (1, 4) порожню фішку (рис. 19.2), а потім звичайним чином «женемо» фішку з номером 4 на своє місце, установлюючи кожного разу перед нею іншу порожню фішку (рис. 19.2). Колізій не виникає, тому що є дві порожні фішки й одна з них знаходиться на тому місці, куди ми повинні поставити фішку з номером 4. Аналогічна ситуація з фішкою 8: фішки 1, 2, 3, 4, 5, 6, 7 вже містяться на своїх місцях (рис. 19.3). Щоб реалізувати цю ідею для фішок з номерами 13 і 14, фішку 13 необхідно встановлювати на своє місце після фішки 9 (рис. 19.4), а фішку 14 — після фішки 10. Тому фішки будемо встановлювати в наступному порядку: 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 10, 14, 11, 12.

В особливому випадку — для фішок 4, 8, 13, 14 — необхідно спочатку встановити одну з порожніх фішок в клітинку (P_x, P_y). Після цього подальша робота з цією фішкою нічим не відрізняється від встановлення фішок з

іншими номерами. Для фішок 4, 8 спочатку перемішуємо порожню фішку по осі X, потім по осі Y; для фішок 13, 14 — по осі Y, потім по осі X, тим самим раніше встановлені на свої місця фішки не зачіпаються.

У загальному випадку при кожному переміщенні фішки, що встановлюється, нам необхідно вибирати одну з двох порожніх фішок. Будемо вибирати ту з них, для переміщення якої на потрібне місце необхідна найменша кількість ходів. Число ходів для переміщення порожньої фішки із клітинки (x_l, y_l) в клітинку (x_2, y_2) можна оцінити знизу виразом $Abs(x_l - x_2) + Abs(y_l - y_2)$. В «особливих» випадках необхідна кількість ходів не дорівнює цій оцінці, а більше за неї на 2 або 4. Ці випадки виникають через те, що при переміщенні порожньої фішки ми не маємо права «торкати» фішки, що встановлені раніше і встановлюються зараз. На прикладах вони показані на рис. 19.5 (переміщення по осі X) і на рис. 19.6-19.8 (переміщення по осі Y). Зверніть увагу на рух порожніх фішок у цих прикладах.

1	2	3	4
7	14	12	6
10	5	0	13
11	9	8	0

1	2	3	4
7	14	12	6
10	5	8	13
11	0	9	0

Мал. 19.5:

1	2	3	4
5	14	10	7
0	6	9	11
8	14	13	0

1	2	3	4
5	14	10	7
8	6	0	11
14	13	9	0

Мал. 19.6:

1	2	3	4
5	14	10	7
8	6	9	11
0	14	13	0

1	2	3	4
5	14	10	7
8	6	9	11
14	13	0	0

Мал. 19.7:

1	5	7	3
4	2	6	14
13	0	11	9
12	10	8	0

1	5	7	3
4	2	0	14
13	11	6	9
12	10	8	0

Мал. 19.8:

19.2 Завдання другого туру

19.2.1 «Картинна галерея»

Перші два пункти завдання. Підрахуємо для кожного J від 0 до $EndTime-1$ в елементі масиву $time[J]$ кількість сторожів, які знаходяться у галереї в проміжок часу $[J, J+1]$. Якщо для якогось значення J їх більше двох, то записувати в $time[J]$ будемо тільки двійку. Після заповнення масиву $time$ маємо:

$$time[j] = \begin{cases} 0, & \text{якщо в момент часу } j \text{ в галереї немає жодного сторожа;} \\ 1, & \text{якщо в момент часу } j \text{ в галереї чергує один сторож;} \\ 2, & \text{якщо в момент часу } j \text{ в галереї чергує два і більше сторожів.} \end{cases}$$

Дані в масиві time є вихідними для одержання відповіді на перше завдання і формування інтервалів часу з недостатньою охороною. Ці інтервали фіксуються в масиві $NoGuard : Array[1..N, 1..3]OfInteger$. У кожному рядку масиву записується початковий час, кінцевий час і кількість сторожів, що чергують (0 або 1). Приклад: $EndTime = 5$, кількість сторожів $N = 6$; інтервали чергувань:

$$(0,1), (0,1), (3,4), (3,4), (3,5), (0,2)$$

Масив *time* буде мати вигляд:

J	0	1	2	3	4
Time[J]	2	1	0	2	1

а масив *NoGuard* буде виглядати так (таблиця 3, ітерація 1):

Номер ітерації	N	Початковий час	Кінцевий час	Кількість сторожів, що чергують
1	1	1	2	1
1	2	2	3	0
1	3	4	5	1
2	1	2	3	1
2	2	4	5	1
3	1	4	5	1

Для завершення виконання другого пункту завдання залишається вивести значення елементів масиву *NoGuard*.

Третій пункт завдання. Нехай введено тривалість чергування, рівну 2. Розглянемо наш приклад. При першій ітерації масив *NoGuard* має наведений вище вид. Додамо одного сторожа з інтервалом чергування (1, 3) і після цього знову сформуємо масив *NoGuard* (таблиця 3, ітерація 2). Приймаємо на роботу ще одного сторожа з інтервалом чергування (2, 4) і знову перевіряємо достатність охорони (таблиця 3, ітерація 3). Так як і раніше є інтервал з недостатньою охороною, то додаємо наступного сторожа, його інтервал чергування — (3, 5). Після цього в будь-який момент часу в галереї знаходиться не менше двох сторожів.

Четвертий пункт завдання. Ідея розв'язку полягає в розбивці множини чергувань сторожів на дві підмножини так, щоб сума часів чергувань сторожів у кожній із підмножин була більше або дорівнювала $EndTime$. Очевидно, що досить знайти одну підмножину, другу отримуємо доповненням до всієї множини чергувань сторожів. Знаходження розбивки можливо тільки в тому випадку, якщо сума часів чергувань всіх сторожів (Sum) більше або дорівнює $2 * EndTime$. При цьому сума часів чергувань сторожів з першої підмножини (cnt) повинна задовольняти умові $EndTime \le cnt \le Sum - EndTime$. Інакше сторожа з другої підмножини не зможуть забезпечити необхідної охорони галереї. При знаходженні розбивки графік чергування сторожів з кожної підмножини складається за принципом: «перший закінчує чергування, другий починає чергувати».

П'ятий пункт завдання (розв'язок для невеликих значень N). Розглядаємо всілякі переміщення часу чергування одного сторожа (спочатку першого, потім другого й т.д.) і намагаємося «залатати діри» у розкладі. Якщо

це не вдається, то переходимо до «двоелементних» переміщень, тобто переміщень часів чергувань якихось двох з наших N сторожів, а потім до «триелементних», і так далі до «N-елементних», поки не буде знайдено правильний розклад.

Отже, одна з підзадач — це генерація k-елементних підмножин N-елементної множини. Вона добре відома і не вимагає роз'яснень. Друга — перебір всіх можливих способів «затикання дір» у розкладі за допомогою сторожів з обраної підмножини. Її розв'язок — модифікація пункту в) задачі.

Розділ 20. Греція'1991

20.1 Завдання першого туру

20.1.1 «Матриця»

В основі розв'язку задачі лежить перебір з поверненням. На кожному кроці шляхом послідовного використання представлених правил визначається можливість розміщення наступного номера у незайнятих позиціях матриці. Процес закінчується, якщо вдається матрицю заповнити повністю або жодне з правил не може бути використано для заповнення таблиці. В останньому випадку змінюється послідовність вибору правил і процес повторюється. Якщо при виконанні завдання А) достатньо визначити тільки один спосіб нумерації таблиці для заданої стартової позиції, то в завданні В) необхідно визначити всі можливі варіанти.

Перевірка завдання А) здійснювалась шляхом візуального контролю результатів заповнення матриці.

Правильні відповіді для завдання В) наступні (наведено стартова точка і кількість варіантів):

(1, 1) — 552;	(1, 2) — 548;	(1, 3) — 552;
(1, 4) — 548;	(1, 5) — 552;	(2, 2) — 412;
(2, 3) — 400;	(2, 4) — 412;	(2, 5) — 548;
(3, 3) — 352;	(3, 4) — 400;	(3, 5) — 552;
(4, 4) — 412;	(4, 5) — 548;	(5, 5) — 552.

20.2 Завдання другого туру

20.2.1 «S-терми»

При виконанні завдання 1 необхідно звернути увагу на правило формування S-термів. Зокрема, у відповідності з визначенням послідовність `SSS` не є S-термом. Для $n = 1$ кількість можливих термів дорівнює 1, $n = 2$ — 1, для $n = 3$ — 2, для $n = 4$ — 5, для $n = 5$ — 14, для $n = 8$ — 429, для $n = 10$ — 4862.

У якості структури даних для внутрішнього представлення S-термів (див. завдання 2) може бути використано або зв'язаний список або двійкове дерево. Наприклад, для S-терма

```
(((S(SS))((SS)S))((SS)(SS)))
```

лінійний список може мати вид $\rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 0 \rightarrow 0 \rightarrow 2 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow nil$.

При виконанні завдання 4 необхідно чітко зрозуміти поняття «ненормалізований» S-терм. Зокрема, якщо до вихідного S-терма неможна застосувати процедуру редукції, то такий S-терм не вважається «ненормалізованим». Якщо задано S-терм виду

```
((((SS)(SS))S)(SS))
```

то в результаті використання процедури нормалізації отримаємо послідовність S-термів:

```
(((SS)((SS)S))(SS))→((S(SS))(((SS)S))SS)))
                   →((S(SS))((S(SS))(S(SS))))
```

У той же час нормалізація S-терма `((((SS)S)((SS)S))((SS)S))` після 30 кроків не закінчиться, тому такий S-терм будемо вважати «ненормалізованим».

Для перевірки правильності виконання завдання 5 можна скористатися наступною таблицею:

n	Кількість S-термів	Кількість «ненормалізованих» S-термів
1	1	0
2	1	0
3	2	0
4	5	0
5	14	0
6	42	0
7	132	2
8	429	41
9	1430	276

Розділ 21. Німеччина'1992

21.1 Завдання першого туру

21.1.1 «Острови у морі»

Запропонована задача є комбінаторною. Основна ідея алгоритму розв'язання задачі полягає у послідовному заповненні карт, представлених у вигляді матриць $N \times N$, за кодовою інформацією про розташування островів у горизонталях з подальшою перевіркою на відповідність інформації про розташування островів по вертикалях. У загальному випадку робота алгоритму повинна закінчуватися, коли будуть побудовані всі допустимі карти.

Найкращим способом реалізації цього алгоритму є перебір з поверненням, який можна представити у вигляді дерева пошуку. На першому рівні цього дерева формуються вищеназвані матриці, перші рядки яких відповідають кодовій інформації про розташування островів у горизонталях і не суперечать кодовій інформації про розташування островів у вертикалях. На другому рівні отримані вище матриці довизначаються другим рядком на основі кодової інформації про другу горизонталь, причому вони також не повинні суперечити інформації про розташування островів у вертикалях. Продовжуючи цей процес далі, на нижньому рівні отримаємо повністю заповнені матриці, що відповідають шуканим картам.

При реалізації перебору з поверненням здійснюється обхід дерева розв'язків у глибину, відсікаючи ті гілки, які суперечать вихідній кодовій інформації. Це дозволяє значно скоротити об'єм перебору, так як безглуздо на якомусь етапі достроювати карту, якщо інформація в початкових рядках матриці, яку заповнюємо, протирічить інформації про острови у вертикалях.

Важливо відзначити, що при коректних вхідних даних існує не єдиний розв'язок. Крім того, необхідно виділити випадки, коли відсутність розв'язку може бути визначена не в результаті роботи описаного вище алгоритму, а в результаті аналізу вхідних даних. Зокрема, такими випадками є:

- перевищення в рядку суми символів * розміру сітки N;
- рівність суми кодів трьох груп островів у рядку значенню $(N-1)$, що визначає відсутність місця у рядку для розділових пробілів;
- відмінність суми всіх кодів по рядкам від суми всіх кодів по стовпчикам.

Таким чином задача відноситься до класу перечислювальних комбінаторних задач.

21.2 Завдання другого туру

21.2.1 «Альпіністи»

Задача за своїм характером є оптимізаційною. Зрозуміло, що оптимальний розв'язок можна отримати у цьому випадку тільки шляхом повного перебору всіх допустимих розкладів сходження альпіністів. Однак спроби розв'язати задачу таким способом навіть для вказаних в умові інтервалів зміни величин N і P не можуть привести до бажаного результату, у чому читач може сам впевнитися шляхом нескладних оцінок, так як задача є $\mathcal{NP}$-повною. Єдиним шляхом є скорочення повного перебору на основі використання сприйнятних евристик. Зокрема, такі евристики можуть будуватися, виходячи з наступних тверджень, що випливають з умови задачі:

- вершини повинен досягти лише один альпініст;
- кожен альпініст повинен на старті взяти ресурсів якомога більше і передати те, що йому не потрібно, якомога раніше;
- найбільш ймовірними для включення в групу сходження є альпіністи, здатні опуститися і піднятися якомога вище.

Важливо чітко сформулювати твердження, на основі яких можна обмежити повний перебір. До того ж необхідно відзначити, що відсутність можливості сходження у деяких випадках можна визначити на основі логічного аналізу вхідних даних. Наприклад, це справедливо для випадку, коли для жодного альпініста не виконується умова $S(j)/C(j) \geq N$. Що стосується умови $S(j) > C(j)$, то справедливість її повинна встановлюватися при перевірці даних на коректність.

Розділ 22. Аргентина'1993

22.1 Завдання першого туру

22.1.1 «Буси»

Ідея розв'язку досить проста. Необхідно перевірити всі можливі точки розриву і для кожної точки розриву визначити кількість бусинок, що знімають.

У символьному масиві $A : Array[1..3 * N]OfChar$ будемо зберігати потроєну конфігурацію намиста. Це допоможе уникнути труднощів при переході через точку «склейки» бус, тобто від бусинки N до бусинки 1 і навпаки.

Розглянемо точку розриву, розташовану між бусинками з номерами k і $k + 1$ ($l \le k < N$) або між бусинками N і 1 ($k = N$), і будемо вважати $i = k + N$. Візьмемо i-ий елемент масиву A і будемо рухатися від нього вліво, «знімаючи» бусинки, доти, поки це можливо. Потім, аналогічним чином, будемо рухатися від елемента $(i + 1)$ вправо. Склавши кількість бусинок, «знятих» при русі вліво, і кількість бусинок, «знятих» при русі вправо, ми одержимо загальну кількість бусинок, що знімають, для даної точки розриву.

Розглянемо, однак, наступний вироджений випадок бус: коли в них зовсім немає або червоних, або блакитних бусинок. У цьому випадку з намиста можна зняти всі бусинки й необхідно передбачити який-небудь «бар'єр» для нашого просування. Цим «бар'єром», наприклад, може служити перевірка того, що номер елемента масиву A, який перевіряється, належить діапазону $[i - N + l..i + N - l]$, тому що вихід із цього діапазону саме й означає «виродженість» бус.

Наступний виклик рекурсивної функції вирішує задачу підрахунку кількості бусинок, що знімають, для точки розриву, що відповідає числу k : Кількість-бусинок := $Number(k+N, -1,' w') + Number(k+N+1, +1,' w')$.

У випадку «вироджений» бус кількість-бусинок буде рівною $2 * N - 1$, і цей випадок варто розглянути окремо.

В основній програмі залається лише знайти максимальну кількість бусинок, що знімають, при різних значеннях i.

22.1.2 «Компанії»

Введемо матрицю A, елемент $A[i, j]$ якої означає, скількома відсотками акцій компанії j володіє (безпосередньо) компанія i. Алгоритм розв'язання задачі очевидний: для всіх компаній i шукаємо компанії, контрольовані компанією i.

Для розглянутої компанії i елемент масиву $B[j]$ означає, скількома відсотками акцій компанії j володіє компанія i безпосередньо, або через контрольовані нею компанії. Як тільки цей відсоток перевищить 50, компанія j також виявляється контрольованою, і її акції додаються в масив B.

Зауважимо, що відношення контролювання рефлексивно (тобто кожна компанія контролює сама себе) і транзитивно (тобто з того, що A контролює B і B контролює C, випливає, що A контролює C).

22.1.3 «Прямокутники»

Задача вирішується, якщо буде знайдено рішення задачі обчислення площі однокольорової фігури, складеної із прямокутників, що мають хоча б одну спільну точку.

Скористаємося ідеєю розв'язання задачі про проходження лабіринту. Обчислення площі ґрунтується на обході в глибину:

- Почнемо обхід з довільної клітинки нашої фігури, у лічильник площі занесемо 0
- Потрапивши в будь-яку клітинку, збільшимо лічильник площі на 1, позначимо цю клітинку, і проаналізуємо по черзі всі сусідні клітинки, які ще не позначені й також належать фігурі.

22.2 Завдання другого туру

22.2.1 «Канадські авіалінії»

При розв'язуванні цієї задачі нам буде зручніше вважати, що міста пронумеровані у зворотному порядку: зі сходу на захід. Нехай $C_1, \ldots, C_N$ — задані міста. Розглянемо спочатку більш просте завдання. Нехай нам потрібно потрапити з міста C_1 у місто C_N, рухаючись тільки на захід і відвідавши при цьому максимально можливе число міст.

Ця задача може бути вирішена методом динамічного програмування: послідовно для $i = l, 2, \ldots, N$ будемо знаходити числа $D[i]$ — кількість міст у найкращому маршруті із C_1 в C_i або $-\infty$, якщо немає жодного такого маршруту. При обчисленні $D[i](i > l)$ ми користуємося тим, що числа $D[l], \ldots, D[i-l]$ вже відомі.

Нехай Q_i - множина номерів міст, розташованих на схід від міста C_i, таких, що мають з ним пряме повітряне сполучення. Очевидно, що в будь-якому маршруті, що зв'язує C_1 і C_i, номер передостаннього міста зобов'язаний належати множині Q_i. У силу цього виконуються наступні рівності (максимум з нуля чисел вважаємо рівним $-\infty$):

$$D[1] = 1; D[i] = \max_{j \in Q_i} D[j] + 1, i > 1. \tag{22.1}$$

Використовуючи їх, ми без ускладнень знайдемо $D[N]$ — кількість міст у потрібному нам маршруті. Але як же знайти сам маршрут? Нехай $How[i](i > l, D[i] \neq -\infty)$ — номер передостаннього міста в найкращому маршруті до міста C_i (це те j, на якому досягається максимум у другій

рівності). Тепер виведення маршруту можна здійснити за допомогою рекурсивного спуска.

Повернемося, однак, до вихідної постановки задачі. Будемо тепер під словами «шлях з міста C_i у місто C_j» розуміти маршрут, що складається із двох частин: від C_i до C_1 (на схід) і від C_1 до C_j (на захід), будь-які два міста в якому (крім, можливо, початкового й кінцевого) різні. За умовою нам потрібно знайти найкращий (за кількістю міст) шлях з C_N у C_N. Будемо розв'язувати цю задачу аналогічно спрощеній. Позначимо через $D[i,j]$ кількість різних міст у найкращому шляху із C_i, в C_j. Вірні наступні співвідношення:

$$\begin{aligned} D[1,1] &= 1 \\ D[j,i] &= D[i,j] \\ D[i,j] &= \max_{k \in Q_i, k \neq j, \text{ЯКЩО} j \neq 1} D[k,j] + 1, (i > j) \\ D[i,i] &= \max_{k \in Q_i} D[k,i] + 1, (i > 1) \end{aligned}$$

Перші дві рівності очевидні. Перевіримо третю рівність. Нехай $i > j$. Візьмемо найкращий шлях із C_i, в C_j. Нехай другим містом у цьому шляху є місто C_k, $k \in Q_i$ (причому $k \neq j$, якщо $j \neq 1$). Тоді, викинувши місто C_i зі шляху, одержимо найкращий шлях з C_k в C_j, що містить $D[k,j]$ міст. Навпаки, взявши будь-який шлях із C_k ($k \in Q_i$; $k \neq j$, якщо $j \neq 1$) в C_j і додавши до нього у якості першого міста місто C_i, ми одержимо шлях із C_i в C_j. (Треба зауважити, що саме тут ми скористалися умовою $i > j$ для того, щоб забезпечити неповторюваність міст у шляху!) Із цих двох тверджень і випливає третя рівність. Четверте співвідношення встановлюється зовсім аналогічно.

Нехай $How[i,j]$ при $i \geq j$ означає номер міста, наступного за C_i у найкращому шляху із C_i в C_j, — це те k, на якому досягається максимум у третій (при $i > j$) або четвертій (при $i = j$) рівності. При $i < j$ число $How[i,j]$ вважаємо рівним $How[j,i]$. Елемент $How[l,l]$ не визначений.

Знаючи числа $How[i,j]$, ми зможемо визначити найкращий шлях, знов-таки використовуючи рекурсивний спуск.

На закінчення відзначимо, що найкращий шлях можна шукати і повним перебором варіантів, але при великій кількості міст такий алгоритм працював би надзвичайно повільно. Повний бал за завдання одержали учасники олімпіади, які реалізували розв'язок, що базується на принципі «динамі-

чного програмування» .

Розділ 23. Швеція'1994

23.1 Завдання першого туру

23.1.1 «Трикутник»

Це типове завдання на метод динамічного програмування. Нехай $A[i, j]$ означає j-е число в i ому рядку трикутника, а $S[i, j]$- найбільшу суму чисел на шляху від вершини трикутника до цього числа. Очевидні співвідношення:

$$S[1, 1] = A[1, 1];$$

$$S[i, j] = \max(S[i - 1, j], S[i - 1, j - 1]) + A[i, j](1 \le j \le i \le N, i > 1)$$

Тут ми покладемо числа $S[i, 0]$ й $S[i, i + l]$ $(1 \le i \le N)$ рівними нулю. Відповідь на задачу можна отримати, обчисливши $\max_{1 \le j \le N} S[N, j]$.

Для наведеного в умові задачі приклада числа $S[i, j]$ будуть такими (максимальне число в останньому рядку є відповіддю):

7				
10	15			
18	16	15		
20	25	20	19	
24	30	27	26	24

При розв'язуванні цієї задачі повним перебором варіантів шляхів необхідне число дій росте приблизно як 2^N, що неприйнятно при заданих в умові задачі обмеженнях.

23.1.2 «Замок»

Перші два пункти задачі вирішуються зовсім аналогічно задачі про прямокутники з п'ятої міжнародної олімпіади з тією лише різницею, що замість восьми сусідів для кожної клітинки ми повинні розглядати лише чотирьох. Зміни, які необхідно внести в алгоритм, досить очевидні й можуть бути здійснені читачем.

Якщо для кожної клітинки ми будемо знати номер кімнати, що містить її, і площі всіх кімнат, то відповідь на пункт 3 завдання можна одержати, по черзі спробувавши видалити кожну зі стінок.

23.1.3 «Прості»

Це завдання на вміння правильно організувати перебір. По-перше, знайдемо всі п'ятизначні прості числа із заданою сумою цифр S (назвемо такі числа правильними). Результати пошуку занесемо в масив: $VarR : Array[1..9, 0..9, 0..9, 0..9]OfBoolean$; елемент $R[a, b, c, d]$ якого буде означати, чи існує правильне число, перші чотири цифри якого суть a, b, c, d (тоді остання цифра цього числа може бути знайдена як $S - a - b - c - d$).

Назвемо рядки, стовпці й діагоналі матриці лініями. Всього існує 12 ліній. Заповнювати матрицю цифрами можна зліва праворуч і зверху вниз, перевіряючи числа, що утворюються на лініях, на правильність. Однак цей алгоритм надзвичайно повільний. Нам хотілося б якомога раніше відтинати «гілки» перебору, що не приводять до розв'язку. Для цієї мети корисно зменшити кількість розгалужень на верхніх рівнях дерева варіантів.

Процес знаходження матриці буде складатися з 12 кроків, на кожному з яких заповнюються порожні клітинки, що залишилися, на одній з ліній. Ми перебираємо всі різні варіанти заповнення, що приводять до правильного числа (правильність числа перевіряється з використанням масиву R). При цьому корисно відмітити, що цифри в останньому рядку й останньому стовпці повинні бути непарними, а цифри в першому рядку й першому стовпці не можуть бути нулями — це дозволяє істотно скоротити кількість варіантів, що аналізуються.

У таблиці 4 для кожної клітинки зазначений номер кроку, на якому в неї записується цифра, а в таблиці 5 для кожного кроку зазначена лінія, що заповнюється. Ліва верхня клітинка матриці позначена нулем, тому що цифра в ній відома заздалегідь.

0	1	1	1	1
9	3	9	4	2
8	5	3	7	2
11	4	10	3	2
4	5	6	6	2

№	Лінія	№	Лінія
1	Рядок 1	7	Стовпець 4 (Таблиця 5)
2	Стовпець 5	8	Рядок 3
3	Діагональ 1	9	Рядок 2
4	Діагональ 2	10	Стовпець 3
5	Стовпець 2	11	Рядок 4
6	Рядок 5	12	Стовпець 1

На кроках 7, 8, 10 і 11 ми обчислюємо цифру, що залишилася, виходячи із суми S і вже відомих цифр, і перевіряємо правильність числа, що утворилося. Заповнювати на кроці 12, загалом кажучи, уже нема чого — потрібно лише перевірити, що число в першому стовпці є правильним. Якщо це так, то ми одержали черговий розв'язок задачі й повинні вивести його у вихідний файл.

Хоча на міжнародній олімпіаді на кожен тест виділялося по 90 секунд, програма, складена за описаним алгоритмом, на найскладнішому тесті працює на комп'ютері 486DX2 лише близько 0.6 секунди. Набір тестів, що використовувалися на олімпіаді, наведено в таблиці 6.

№	Вхідні дані	Кількість розв'язків
1	11 5	1
2	37 9	3
3	35 5	1
4	13 4	1

23.2 Завдання другого туру

23.2.1 «Годинник»

Пронумеруємо циферблати в матриці зліва праворуч і зверху вниз, починаючи з 1. Нехай число a_i означає, скільки разів потрібно повернути стрілку i-го циферблата на 90 градусів, щоб на ньому встановився час 12 годин ($0 \le a_i \le 3$). Легко помітити, що порядок кроків у розв'язку не має ніякого значення. Отже, розв'язок повністю описується набором чисел $(S_1, S_2, \ldots, S_9)$, де S_i, дорівнює числу застосувань способу $i (0 \le S_i \le 3)$. Для знаходження цього набору потрібно розв'язати наступну систему лінійних рівнянь по модулю 4:

$$\begin{cases} x_1 + x_2 + x_4 = a_1 \\ x_1 + x_2 + x_3 + x_5 = a_2 \\ x_2 + x_3 + x_6 = a_3 \\ x_1 + x_4 + x_5 + x_7 = a_4 \\ x_1 + x_3 + x_5 + x_7 + x_9 = a_5 \\ x_3 + x_5 + x_6 + x_9 = a_6 \\ x_4 + x_7 + x_8 = a_7 \\ x_5 + x_7 + x_8 + x_9 = a_8 \\ x_6 + x_8 + x_9 = a_9 \end{cases} \tag{23.1}$$

Розглянемо цю систему над полем раціональних чисел. Для будь-яких чисел a_i вона має єдиний розв'язок, тому що визначник матриці коефіцієнтів дорівнює 5, тобто відмінний від нуля. Помножимо праві частини цих

рівнянь на 5, і нехай $(x_1, x_2, \ldots, x_9)$ — розв'язок модифікованої в такий спосіб системи. Цей розв'язок цілочисельний (це випливає, наприклад, з формул Крамера), причому, вважаючи S_i для всіх $i = 1, \ldots, 9$ рівним $x_i \bmod 4$, одержимо розв'язок вихідної системи по модулю 4.

Чи буде цей розв'зок єдиним, задовольняючим нерівностям $0 \le S_i \le 3 (1 \le i \le 9)$? Ми показали, що для кожного вхідного набору $(a_1, \ldots, a_9)$ існує хоча б один такий розв'язок системи $(S_1, \ldots, S_9)$, причому різним наборам відповідають, очевидно, різні розв'язки. Кількість різних вхідних наборів дорівнює 49. Але кількість різних розв'язків, що задовольняють цим нерівностям, також дорівнює 49! У силу цього для будь-якого набору вхідних даних існує єдиний (з точністю до порядку кроків) оптимальний розв'язок, який можна знайти за допомогою описаної процедури.

Простіше, однак, перебрати всі можливі значення для S_1, S_2 і S_3, виражаючи через них числа $S_4, S_5, \ldots, S_9$ за допомогою деяких з рівнянь системи. Якщо отриманий набір чисел $(S_1, \ldots, S_9)$ задовольняє й рівнянням, що залишилися, значить нами знайдено розв'язок задачі.

23.2.2 «Автобуси»

Записані часи прибуття автобусів назвемо відліками. Для прискорення перебору необхідно за заданими відліками заздалегідь порахувати всі можливі маршрути, тобто пари (m, d), де m — час прибуття першого автобуса, а d — інтервал руху $(m > d, m + d \le 59)$. Наше завдання - розбити всю множину відліків на найменшу кількість маршрутів.

Спочатку всі відліки «вільні», тобто жоден з них ще не віднесений ні до якого маршруту. На кожному кроці пошуку розбивки ми знаходимо перший вільний відлік m і перебираємо всі можливі маршрути (m, d) з таким часом прибуття першого автобуса (якщо $m > 29$, то немає жодного потрібного маршруту, і варто повернутися до попереднього кроку). Для кожного маршруту перевіряємо, чи можна його виділити з відліків, що залишилися вільними. Якщо можна, то виділяємо (тобто відносимо відповідні відліки до цього маршруту), збільшуємо лічильник маршрутів Cnt на одиницю й рекурсивно переходимо до наступного кроку. Якщо на черговому кроці всі відліки виявилися задіяними, то ми одержали розбивку, і необхідно порівняти її з найкращою з раніше знайдених.

На жаль, цей простий алгоритм «не укладається» у часові рамки, зазначені в умові задачі. Тому треба думати про скорочення перебору.

Перше спостереження. Більшість маршрутів мають довжину 2, тобто містять два відліки. Назвемо такі маршрути великими парами. Виявляється, що для розбивки якої-небудь множини відліків на великі пари можна застосовувати «жадібний» алгоритм: взяти перший вільний відлік і підібра-

ти до нього перший, що підходить для утворення великої пари, потім знову взяти перший вільний відлік, виконати з ним те ж саме й т.д. Якщо на якомусь кроці не вдається знайти підходящий вільний відлік, то розбивка неможлива.

Тим самим, наведений вище алгоритм можна модифікувати в такий спосіб. Взявши перший вільний відлік m ($m < 29$), ми намагаємося або віднести його до одного з маршрутів (m, d) довжини не менше $3(m+2*d \leq 59)$, або «залишити на потім» (для великих пар), збільшуючи лічильник великих пар $PCnt$ на одиницю. Якщо на якомусь кроці кількість вільних відліків, що залишилися Rem, дорівнює $2 * PCnt$, то намагаємося розбити їх на великі пари за допомогою «жадібного» алгоритму.

Друге спостереження. Нехай до чергового кроку нами вже побудовано Cnt маршрутів, а перший вільний відлік — $m(m \leq 29)$. Всі маршрути, які ми побудуємо надалі, будуть мати довжину не більше $L = [(60-m)/(m+1)]$ (тут $[x]$ означає найменше ціле число, більше або рівне x). Отже, якщо число $Cnt + PCnt + [(Rem - 2 * PCnt)/L]$ більше або дорівнює кількості маршрутів Min у найкращій зі знайдених розбивок, то цю «гілку» перебору можна не розглядати. На початку роботи програми потрібно встановити Min рівним 18, тому що відомо, що число маршрутів у найкращій розбивці не перевершує 17.

Так як кілька відліків можуть відповідати одній і тій же хвилині, то необхідно завести масив, елементи якого б означали, скільки залишилося вільних відліків для кожної із хвилин. Нарешті, можна скоротити перебір, помітивши, що порядок віднесення відліків, відповідних одній хвилині, до маршрутів не має значення. Так, якщо один із цих відліків був віднесений до маршруту (m, d_1), то для наступного будемо перебирати маршрути (m, d_2) з $d_2 \leq d_1$.

23.2.3 «Круги»

Нехай, для визначеності, мінімальне із чисел знаходиться у першому секторі. Очевидно, що це число повинно належати інтервалу від K до M (якщо $K > M$, то задача не має розв'язків). Заведемо масив: $VarS : Array[1..N]OfInteger$; для зберігання поточного стану кола ($S[i]$ дорівнює числу в i-ому секторі). Задачу можна вирішувати за допомогою рекурсивного алгоритму розміщення чисел у секторах: на кожному кроці заповнюється наступний сектор, коректується множина нових чисел, які можна отримати, і здійснюється перехід до наступного сектора. Які числа ми можемо ставити в сектори? В умові завдання сказано, що вони повинні бути не менше K, але нічого не говориться про верхню границю.

Нехай $N-1$ секторів кола заповнено числами, а якийсь один порожній.

Легко помітити, що ми можемо одержати не більш, ніж $N*(N-l)/2$ різних нових чисел. Позначимо через X перше число з послідовності $M, M+1, \ldots$, що ми не зможемо одержати. Зрозуміло, що в сектор, що залишився порожнім, не має смислу ставити число, більше X. Оскільки X не перевищує $N*(N-l)/2+M$, то цей вираз є верхньою границею на числа в секторах. Якщо $S[1] < M$, то цю границю можна зменшити ще на 1. У сектори з номерами $2, 3, \ldots, N$ потрібно поміщати числа, не менші $S[l]$, — це буде нижньою границею.

Для останнього сектора діапазон чисел, що розглядається можна зменшити ще. У якості нижньої границі (якщо $N > 2$) можна взяти число $S[2]$ для того, щоб виключити симетричні варіанти, а в якості верхньої - те число X, про яке ми говорили вище. Якщо при цьому сума чисел, що стоять у секторах $1, \ldots, N-1$, і числа X менше найкращого зі знайдених значень I, то при будь-якому варіанті заповнення останнього сектора новий розв'язок ми не отримаємо.

Ще одне спостереження. Нехай $M < 2*K$. Тоді числа $M, M+1, \ldots, 2*K-1$ не можуть бути отримані як сума чисел з кількох (більше одного) секторів, отже, або (якщо $2*K-M \le N$) всі вони повинні знаходитися в колі, або (якщо $2*K-M > N$) у колі повинні знаходитися числа $M, M+1, \ldots, M+N-1$ і тільки вони.

Для прискорення роботи програми введемо масив: $VarSum : Array[1..N, 1..N]O$

елемент $Sum[i,j]$ якого буде рівним сумі чисел в j послідовних секторах, починаючи із сектора i (якщо, звичайно, всі вони вже заповнені). До моменту заповнення чергового сектора p у нас повинні бути сформовані поточна множина S нових чисел, які можна утворити, i числа $Sum[i,j]$ $(1 \le i < p, i+j \le p)$. Після заповнення сектора p (при переході до наступного сектора) необхідно обчислити числа $Sum[i, p-i+1](l \le i \le p)$ і включити їх у множину S.

Якщо сектор останній ($p = N$), то обчислюємо й елементи матриці Sum, що залишилися, також заносячи їх у множину S. Потім для набору чисел у секторах, що отримали, потрібно знайти число I з умови задачі й порівняти його з максимальним з раніше знайдених ($MaxI$). Якщо $I = MaxI$, то додаємо набір до списку розв'язків (для якого повинен бути виділений масив); якщо ж $I > MaxI$, то очищаємо список розв'язків, вносимо в нього знайдений набір й обновляємо значення $MaxI$.

При виведенні результату для кожного зі знайдених розв'язків із $S[2] < S[N]$ потрібно видати також симетричний йому розв'язок з $S[2] > S[N]$, отриманий шляхом дзеркального відбиття відносно бісектриси першого се-

ктора.

Розділ 24. Нідерланди'1995

24.1 Завдання першого туру

24.1.1 «Упаковка прямокутників»

Пояснимо наведені на рис. 1 (див. умову) шість способів спільного розташування чотирьох прямокутників. Назвемо два прямокутники A і B незалежними по осі Ox (Oy), якщо їх проекції на цю вісь не мають спільних внутрішніх точок.

Можливі 16 варіантів орієнтації 4 прямокутників, так як кожний із прямокутників (якщо він не квадрат) можна орієнтувати двома способами.

Поза залежністю від орієнтації, існують наступні три основних варіанти розташування прямокутників.

- Всі 4 прямокутники попарно незалежні по одній з осей (можна вважати, що по осі Ox, див. фігуру 1 на рис. 1)
- Існують 3 прямокутники, попарно незалежні по одній з осей (можна вважати, що по осі Ox). Позначимо ці три прямокутники буквами A, B, C, а прямокутник, що залишився — буквою D. Можливі наступні три випадки:
  - D не залежить по осі Oy від кожного з A, B, C (фігура 2);
  - D залежить по осі Oy від одного з A, B, C (фігура 3);
  - D залежить по осі Oy від двох з A, B, C. У цьому випадку D розташовується над (або під) прямокутником, що залишився з перших трьох. Тут можливі два випадки: прямокутник, що залишився, може бути крайнім серед A, B, C (фігура 4), або середнім (фігура 5). Однак у нас не буде необхідності розрізняти ці два випадки.
- По кожній з осей максимальна кількість попарно незалежних прямокутників дорівнює двом (фігура 6 на рис. 1).

Розберемо тепер, як у кожному із цих випадків знайти ширину й висоту мінімального прямокутника P, що покриває 4 прямокутники.

У випадку 1 ми можемо вважати, що прямокутники вирівняні по нижньому краю. Тоді ширина P ($w(P)$ повинна бути рівною сумі ширини всіх чотирьох прямокутників, а висота P ($h(P)$) — висоті найвищого з них.

У випадку 2 можна вважати, що прямокутники A, B, C розташовані зліва праворуч у порядку спадання висоти й вирівняні по нижньому краю. У випадку 2а прямокутник D розташовуємо зверху від прямокутників A, B, C. Тоді:

$w(P) = \max(w(A) + w(B) + w(C), w(D))$

$h(P) = h(A) + h(D)$.

У випадку 2б прямокутник D розташовуємо над B і C:

$w(P) = w(A) + \max(w(B), w(C), w(D))$

$h(P) = \max(h(A), h(B) + h(D))$.

У випадку 2в прямокутник D розташовуємо над C:

$w(P) = w(A) + w(B) + \max(w(C), w(D))$

$h(P) = \max(h(A), h(C) + h(D))$.

Нарешті, розглянемо випадок 3. Якщо 4 заданих прямокутники лежать без перекриття усередині більшого, то можна вважати, що вони лежать у його кутах (інакше їх можна пересунути в кути). Можна вважати, що в лівому нижньому куті міститься найвищий із прямокутників. Нехай A — той із прямокутників, який розташований в лівому нижньому, B — у лівому верхньому, C — у правому нижньому і D — у правому верхньому кутах. Зрозуміло, що повинні виконуватися наступні співвідношення:

$w(P) \geq RW = \max(w(A) + w(C), w(B) + w(D))$

$h(P) \geq RH = \max(h(A) + h(B), h(C) + h(D))$.

Припустимо, що прямокутники A і D незалежні по осі Ox. Тоді:

$w(P) = \max(RW, w(A) + w(C))$

$h(P) = RH$.

Якщо ж A і D залежні по осі Ox, то:

$w(P) = RW$

$h(P) = \max(RH, h(A) + h(D))$.

24.1.2 «Торгівельні знижки»

Завдання легко вирішується методом динамічного програмування. Нехай нам необхідно купити k_1 товарів з кодом c_1, k_2 товарів з кодом $c_2, \ldots k_b$ товарів з кодом c_b ($c_l, \ldots, c_b$ — фіксовані числа). Позначимо через $A[k_1, \ldots, k_b]$ найменшу можливу вартість такого набору. Ці числа $A[k_1, \ldots, k_b]$ можна обчислювати динамічно: покладемо $A[0, \ldots, 0]$ рівним нулю, потім послідовно будемо обчислювати числа $A[k_1, \ldots, k_b]$ з $k_l + \ldots + k_b = 1, 2, 3$ і т.д.

Для кожного з наборів, що аналізуємо, намагаємося застосувати всі можливі знижки, при цьому користуємося тим, що для наборів з меншою кількістю товарів ми найменшу вартість уже обчислили.

24.1.3 «Клієнт і сервер»

Підзадача A

Якщо в момент часу 4 об'єкт CLIENT(1) одержить від користувача USER(1) повідомлення `C_Job`, то він відразу (відповідно до процедури прийому повідомлення `C_Job` з документа 2 відправить назад повідомлення

`Try_later`(CLIENT(1),USER(1), `All_Servers_are_busy`),

оскільки буде в цей момент перебувати в стані SC.

Якщо ж у момент часу 4 повідомлення $'C\_Job'$ одержить об'єкт CLIENT(2), то він відправить назад повідомлення

`Try_later`(CLIENT(2), USER(2), `Client_is_busy`)

тому що буде в цей момент перебувати в стані SB.

Підзадача B

Розглянемо розширену мережу з декількома парами семафор-принтер. Розсилка повідомлень `Are_you_open?` одночасно всім семафорам призведе до того, що всі семафори, що перебувають у поточний момент у стані S1 відгукнуться повідомленням `Open` і перейдуть у стан S2, вивести з якого семафор можна тільки друком завдання на відповідному йому принтері. Однак, як легко помітити, будь-яке завдання в розширеній мережі (зі зміненими відповідно до документа 6 процедурами прийому) друкується тільки один раз, тому всі семафори, що перейшли в стан S2, крім одного, просто "застряють"у ньому, виводячи з роботи відповідні принтери.

У нашому прикладі розширеної мережі в силу того, що пріоритет CLIENT(1)'а вище пріоритету CLIENT(2)'а, у момент часу 1 "застряє- SEMAPHORE(2). Тому обидва завдання будуть надруковані на SERVER(1)'е, і правильний варіант відповіді — (c).

Підзадача C.1

Зміні підлягають процедури прийому повідомлень `C_Job`, `Closed` і `C_Ready`.

Підзадача C.2

Уведемо для типу об'єкта CLIENT внутрішню змінну Job_Count, значенням якої буде кількість завдань, які друкують для цього CLIENTа (початкове значення Job_Count дорівнює нулю). У порівнянні з підзадачею C.1

змінити необхідно процедури прийому повідомлень `Open` і `C_Ready`. Як і раніше, стан SA означає, що `CLIENT` готовий до прийому чергового завдання ($Job\_Count < Job\_Maximum$), SB — що `CLIENT` шукає вільний принтер для завдання, що надійшло на друк, а SC — що `CLIENT` зайнятий і не може в даний момент обробляти нове завдання ($Job\_Count = Job\_Maximum$).

24.2 Завдання другого туру

24.2.1 «Словесна гра»

Цю задачу можна розв'язувати різними способами. Наприклад, ми можемо спочатку зчитати весь словник в оперативну пам'ять, а потім намагатися всіма можливими способами з даних нам букв побудувати одне, або два «псевдослова». Під псевдословом тут ми розуміємо довільну послідовність маленьких латинських букв довжиною від трьох до семи символів. Для того, щоб перевірити, чи є дане псевдослово дійсно словом, тобто чи міститься воно в словнику, досить застосувати алгоритм бінарного пошуку, користуючись тим, що словник уже впорядкований. Для кожного з варіантів побудови одного або двох слів обчислюємо сумарну вартість букв і знаходимо той варіант (або варіанти), для якого ця вартість максимальна. Через очевидність алгоритму деталі реалізації надаються читачеві.

24.2.2 «Вулична гонка»

Легко помітити, що точка є неминучою тоді й тільки тоді, коли її вилучення веде до неможливості дістатися від старту до фінішу. Всі точки, у які можливо потрапити після вилучення деякої заданої, ми можемо знайти за допомогою пошуку в глибину. Нехай A — множина точок, у які можна потрапити зі старту, якщо вилучити точку $k (0 < k < N)$, B — множина всіх інших точок (включаючи саму точку k). Точка k є неминучою тоді й тільки тоді, коли фініш (тобто точка N) лежить в B, і точкою розбиття тоді й тільки тоді, коли, по-перше, немає стрілок від точок з B до точок з A, і по-друге, немає стрілок від точок з A до точок з B, за винятком стрілок, що ведуть у точку k. Стрілки від точки k до неї ж самої допускаються, тому що їх можна включити в план для другого дня перегонів. Помітимо, що точка розбиття завжди є неминучою точкою.

24.2.3 «Дроти и вимикачі»

Задача розв'язується методом «розділяй і пануй». Нехай у нас є деяка множина дротів, з'єднаних з m вимикачами з номерами $L, L + 1, \ldots, R$, при-

чому всі вимикачі знаходяться в одному положенні («включений» або «виключений»), і для кожного дроту потрібно визначити той вимикач, з яким він з'єднаний. Якщо $m = 1$, то задача очевидна. Нехай $m > 1$. Змінимо стан перших $[m/2]$ вимикачів на протилежний, потім перевіримо кожний із дротів. За отриманими відповідями ми зможемо розділити нашу множину дротів на дві непересічні підмножини: з тих дротів, які з'єднані з першими $[m/2]$ вимикачами, і тих, які з'єднані з $m - [m/2]$, що залишилися. Тим самим ми звели вихідну задачу до двох аналогічних, але «більш простих» (з меншою кількістю вимикачів). Описаний алгоритм можна записати у вигляді рекурсивної процедури.

Розділ 25. Угорщина'1996

25.1 Завдання першого туру

25.1.1 «Гра»

Наступне просте спостереження веде до розв'язку задачі. Так як ігрова дошка на початку гри містить парну кількість елементів у послідовності, то при ході першого гравця один кінець послідовності непарний, а другий кінець знаходиться у парній позиції. Тому перший гравець може завжди обрати елемент з непарної або з парної позиції.

Алгоритм попередньо аналізує вміст початкового розташування на дошці перед початком гри, обчислюючи значення $OddSum$ та $EvenSum$, сум елементів у непарних позиціях та сум елементів у парних позиціях, відповідно.

Якщо $OddSum \geq EvenSum$, то перший гравець завжди обирає непарні позиції і тим самим вимагає, щоб другий гравець обирав парні позиції. У випадку, коли $OddSum < EvenSum$, перший гравець повинен завжди обирати парні позиції.

25.1.2 «Роботи»

Підзадача A

Очевидно, що оптимальний алгоритм може бути знайдено, якщо кожен верстат починає працювати у момент часу 0, і ніколи не зупиняється, доки не виконано операцію над всіма деталями, що обробляються цим верстатом. Максимальна кількість деталей, що можуть бути оброблені за час t верстатом m типу Op дорівнює $t/PTime[Op, m]$. Тому, мінімальний час, що

необхідний для виконання операції Op для всіх N деталей, найменше з чисел t, таке що сума чисел $(t/PTime[Op, i])$ для $i = 1, \ldots, M[Op]$, більше або дорівнює N. Таким чином обчислюємо загальний час виконання операції Op.

Підзадача B

Очевидно, що алгоритм для верстатів, що виконують операцію A, такий як у підзадачі A. Нехай TAB — мінімальний час, який необхідний для виконання обох операцій для всіх N деталей. Ми можемо вважати, що кожен верстат типу B закінчує обробку точно у час TAB і ніколи не зупиняється між обробкою двох послідовних деталей. Якщо це не початковий випадок, то ми можемо змінити оптимальний алгоритм відповідно, якщо деталь знаходиться у деякий час у проміжному контейнері, то ця деталь буде доступною і пізніше.

Нехай d — час початку обробки першої деталі верстатом типу B, відповідно оптимальному алгоритму. Позначимо TB - мінімальну кількість часу, який необхідний, щоб виконати тільки операцію B для всіх N деталей. Тоді, використовуючи підзадачу A, маємо $TAB = d+TB$. Таким чином залишається обчислити час затримки d.

Нехай $Finish(Op, t)$ — кількість деталей, для яких у час t відповідно оптимальному алгоритму виконано одну операцію Op. Час затримки d — найменше число, яке задовольняє наступній умові: для кожного $t, 0 \leq t < TB$ по меншій мірі $Finish('B', TB - t)$ кількість деталей, доступних у проміжному контейнері у час $d + t$. Для даного значення d можна перевірити вказану умову. Тому значення d обчислюємо, починаючи з 1 і збільшуючи d, доки не виконається умова.

25.1.3 «Мережа шкіл»

Мережа шкіл може бути представлена орієнтованим графом, вершини якого — школи і (A, B) — гілка графа тоді і тільки тоді, коли школа B знаходиться у списку розсилки школи B. Сформулюємо задачу, використовуючи термінологію графів. Будемо використовувати запис $p \to q$, якщо у графі існує спрямований шлях від p до q. Множину вершин D графа G називають домінуючою множиною G, якщо для кожної вершини q існує вершина p в D, така що $p \to q$. У підзадачі A необхідно знайти домінуючу множину графа G з мінімальною кількістю елементів. Множина вершин CD графа G називається ко-домінуючою множиною графа G, якщо для кожної вершини p існує вершина q в CD, така що $p \to q$. Граф G називається строго зв'язаним, якщо для всіх вершин p і q існує шлях $p \to q$ і шлях $q \to p$ в G. Розв'язок підзадачі B — це мінімальна кількість нових гілок, які необхідні, щоб зробити G строго зв'язаним. Позначимо кількість елементів множини

S, як $|S|$. Нехай D мінімальна домінуюча множина, а CD — мінімальна кодомінуюча множина графа G. Можна довести, що розв'язком підзадачі B є 0, якщо G строго зв'язаний, і $\max(|D|, |CD|)$ в іншому випадку. Зрозуміло, що ко-домінуюча множина графа G - це домінуюча множина транспонованого графа GT і навпаки. Тому ми можемо обчислити мінімальну кодомінуючу множину графа G, транспонувавши G і обчисливши домінуючу множину транспонованого графа.

25.2 Завдання другого туру

25.2.1 «Сортування-3»

Основна ідея розв'язку полягає у наступному. Обчислити спочатку кількість появ $Na[x]$ для елементів $x = 1, 2, 3$ у введеній послідовності. Тоді відсортована послідовність складається з $Na[1]$ значень 1, потім $Na[2]$ значень 2 і потім $Na[3]$ значень 3. Будемо використовувати скорочення $x : y$, щоб позначити, що елемент x знаходиться у позиції елемента y у відсортованій послідовності. Обчислимо $NEP[x, y]$, кількість елементів x у позиції елемента y для всіх x та y. Покажемо, що кількість операцій обміну, які виконуються алгоритмом, можна знайти за допомогою виразу:

$$Ch(S) = \min(NEP[1,2], NEP[2,1]) +$$
$$\min(NEP[1,3], NEP[3,1]) +$$
$$\min(NEP[2,3], NEP[3,2]) + 2 * abs(NEP[1,2] - \min[2,1])$$

25.2.2 «Найдовший префікс»

Нехай S послідовність символів і P — множина примітивів. Позначимо $Suff(S, P)$ множину послідовностей v таку, для якої виконуються наступні дві умови:

- v — префікс примитива в P;
- $S = uv$ для деякого u.

Для двох послідовностей символів u і v ми позначаємо конкатенацію u і v, як uv). Очевидно, що S може бути складено з примітивів з P тоді й тільки тоді, коли порожня множина міститься в $Suff(S, P)$. Крім того, S має розширення u праворуч, це робить Su, композицією примітивів з P тоді й тільки тоді, коли $Suff(S, P)$ є непорожньою.

Одна з очевидних проблем — велика довжина файла даних для читання у пам'ять. На щастя, немає необхідності читати всю послідовність у пам'ять. Необхідно зауважити, що $Suff(Sx, P)$ для послідовності S і символа x може бути обчислена з множини $Suff(S, P)$. Позначимо послідовність u, яка є префіксом примітива в P, як пару (i, j), де префікс $P[i]$ складається з перших j символів примітива $P[i]$ рівного u, и i — найменший такий індекс для u. Зверніть увагу, що порожня послідовність представлена парою (1,0). Ми можемо заздалегідь опрацювати множину примітивів, щоб сформувати таблицю переходу T: $T[i, j, x] = 0$, якщо немає примітива в P з префіксом $P[i][1..j]x$, інакше найменший індекс k такий, що $P[i][1..j]x$ є префіксом $P[k]$. ($P[i][1..j]$ означає послідовність символів, що складаються з перших j елементів примітива $P[i]$.) Іншими словами, якщо послідовність u представлена парою (i, j), то послідовність ux - префікс примітива з P тоді і тільки тоді, коли $T[i, j, x] > 0$, і в цьому випадку ux — префікс $P[T[i, j, x]]$ і може бути представлено парою $(T[i, j, x], j + 1)$. Таким чином необхідно знайти матрицю переходу T і заповнити масив $Full$, де $Full[i, j]$ дорівнює $true$ тоді і тільки тоді, коли (i, j) дорівнює примітиву в P.

25.2.3 «Магічні квадратики»

Пошук закінчується, коли досягнута конфігурація Q. Спочатку необхідно дослідити виконання операцій на початковій множині. Кількість всіх конфігурацій $8! = 40320$. Це дуже багато, щоб зберігати конфігурації в масиві. Обминути цю проблему можна, використавши функцію $Rank$, яка відображає конфігурацію в номер в діапазоні $0..8! - 1$. Ми можемо отримати таку функцію, визначивши $Rank(Q)$ як номер конфігурації, що передує Q відповідно до лексикографічного порядку перестановки чисел 1..8. Необхідно відзначити, що кожне базове перетворення C має унікальну інверсію, тобто для кожної конфігурації Q, якщо C перетворює Q в P, то його інверсія перетворює P в Q і навпаки. Інверсії базових перетворень:

- A: A
- B: циклічний зсув ліворуч на один квадрат
- C: поворот проти стрілки годинника чотирьох середніх квадратиків на 90°

Якщо конфігурація Q, отримана застосуванням базового перетворення C до P, то існує послідовність базових перетворень, що переводять початкову конфігурацію в Q, де останній елемент C. Якщо ми знаємо Q і C, то

ми можемо обчислити P, застосувавши інверсію C до Q.

Розділ 26. ПАР'1997

26.1 Завдання першого туру

26.1.1 «Марсохід»

Алгоритм розв'язку задачі: програма читає ландшафт у масив. Потім будуємо бінарне дерево для всіх можливих пересувань від початкової до кінцевої точки. Зрозуміло, що тільки закінчені шляхи можуть переглядатися, так як дерево не продовжується через нерівні ділянки. Шляхи оцінюються, за кількістю зібраних зразків порід. Обираємо найкращий шлях з максимальною кількістю зразків.

26.1.2 «Гра гекс»

Розв'язок цієї задачі заснований на переборі всіх можливих ходів, доки гру не закінчено. Перебираючи рядки row визначаємо координати (spot, spot) клітинки, у яку можливо зробити хід за допомогою виразу.

26.1.3 «Ненажерлива Шонгололо»

Представимо фрукт, який збирається їсти Шонгололо у вигляді 3-вимірного масиву. Цей масив ініціалізовано таким чином, що всі клітинки є їстівними. Шонгололо починає їсти з блоку 1,1,1 і потім переміщується у перший з'їдений блок. Далі вона рухається у додатньому продольному напрямі, поїдаючи всі суміжні блоки (горизонтальні і вертикальні), для яких не порушуються правила. При досягненні сторони, визначаємо, чи може Шонгололо повернутися без порушення правил, і припиняє поїдання суміжних блоків. Потім вона повернеться і продовжить цей процес. Коли ніякі горизонтальні блоки не можуть бути з'їдені, Шонгололо переміститься вниз на 4 блоки і продовжить діяти аналогічно, відмічаючи, що вона їсть настільки багато блоків зверху, знизу та горизонтально, наскільки це можливо, і залишаючи достатньо нез'їдених блоків у кутах для поворотів. Коли не залишиться жодного їстівного блока, якого можна з'їсти, не порушуючи правил, програма закінчує роботу. Щоб оптимізувати розв'язок задачі, розв'язання починаємо з 3 напрямів, почерзі обираючи, вздовж якої грані буде рухатись Шонгололо. Використовуємо найкращий напрямок.

26.2 Завдання другого туру

26.2.1 «Розмітка карт»

Надписи необхідно відсортувати в порядку зростання (визначення ключа сортування залишаємо на роздум читача). Кожний надпис у вказаному порядку вставляємо один за одним в одній з 4 можливих позицій. Як тільки всі надписи, що можуть бути вставлені таким шляхом, отримані, то їх можна повернути навкруги. Це може відкрити проміжки, у які можна розмістити невставлені надписи.

26.2.2 «Розпізнання символів»

Процедура розпізнавання полягає у наступному: порівняти 27 даних коректних символів з позицією екрану і зареєструвати кількість подібних пікселів екрану. Кожен з 27 коректних символів порівнюємо 40 разів для 20 різних випадків по вертикалі, враховуючи пропуски та повторення. Таким чином, 27×41 символів порівнюються з позицією екрану. Шуканий символ — символ з мінімальною кількістю відмінностей.

26.2.3 «Складування контейнерів»

Очевидним є метод, який зберігає у масивах для кожного контейнера його місцерозташування (на складі чи ні), значення очікуваного часу, коли контейнер необхідно відправити, відповідні кількості контейнерів для кожної стопки з координатами (x,y) на складі

Організовуємо перебір варіантів переміщення контейнерів, враховуючи необхідні умови. Новий контейнер поміщаємо на склад у вільну позицію, або, якщо її немає, то на стопку контейнерів з максимальним значенням очікуваного часу, коли контейнер необхідно відправити зі складу.

Розділ 27. Португалія'1998

27.1 Завдання першого туру

27.1.1 «Контакт»

Оскільки послідовність може мати достатньо велику довжину, необхідно побудувати алгоритм, що буде її переглядати мінімальну кількість разів. Покажемо, як за один перегляд вхідного рядка розвзати задачу.

Шукані підпослідовності будемо кодувати цілими числами, інтерпретуючи підпослідовність як двійковий запис цілого числа.

Для кожного символу послідовності (крім кількох останніх) існує $B - A + 1$ підпослідовностей, що ним закінчуються. Для їхнього аналізу достатньо пам'ятати останні B прочитані символи. На кожному кроці, після читання чергового символу, необхідно збільшити лічильники для усіх підпослідовностей, що закінчуються поточним символом. По закінченні перегляду для кожної підпослідовності буде відома частота її входження у вихідну послідовність. Далі, необхідно відсортувати частоти і вивести необхідну їх кількість у файл.

Зауважимо, що обмеження на вхідні дані підібрані таким чином, що достатньо використовувати статичну пам'ять для збереження інформації про частоти входження.

Подробиці кодування підпослідовностей і обробки початку і кінця послідовності залишаємо читачеві.

27.1.2 «Лампи для свята»

Скорочення перебору можна робити двома шляхами: скорочуючи кількість розглянутих лампочок і зменшуючи кількість натискань на кнопки.

Після довільної послідовності натискань на кнопки стан лампочок буде повторюватися з періодом 6. Це випливає з того, що НСК$(1, 2, 2, 3) = 6$. Крім того, будуть збігатися стани лампочок із номерами $6k$ і $6k + 2$, а також із номерами $6k + 3$ і $6k + 5$. Усі лампочки можна розділити на чотири групи за остачею від ділення номера лампочки на 6 так: $(0, 2)$, (1), $(3, 5)$, (4). Лампочки, що належать одній групі, будуть знаходитися в однакових станах при будь-яких натисканнях на кнопки. Номера лампочок у вхідному файлі задають, таким чином, стан не однієї лампочки, а групи, котрій ця лампочка належить.

Результат натискання на кнопки не залежить від послідовності натискань, а залежить тільки від кількості. Дворазове натискання на довільну кнопку повертає лампочки у вихідний стан. З цього випливає, що достатньо розглядати послідовності, одержувані не більш ніж однократним натисканням на кожну кнопку. Якщо задане число натискань парне, то необхідно розглядати натискання на 0 або 2 різноманітні кнопки, якщо непарне — 1 або 3.

Таким чином, необхідно розглянути 16 варіантів натискань на кнопки і вибрати ті, що забезпечують необхідний стан лампочок (4-х груп).

27.1.3 «Зоряна ніч»

Ця задача не потребує спеціальних зусиль по оптимізації, а розрахована на акуратне програмування громіздкої обробки карти зоряного неба.

Кожен крок складається з двох етапів: спочатку виділяється нове сузір'я, а потім воно дорівнюється з раніше знайденими. Нові сузір'я можна шукати, послідовно продивляючись карту до виявлення ще не обробленої зірки. Далі, застосовуючи один з алгоритмів заливання, виділяється усе сузір'я. Перевірка збігу з одним із раніше знайдених сузір'їв проводиться поелементним порівнянням сузір'їв з урахуванням відповідних перетворень координат.

27.2 Завдання другого туру

27.2.1 «Камелот»

Для розв'язання задачі необхідно перебрати усі клітини дошки і вибрати оптимальну для зустрічі. Для кожної клітини необхідно визначити, із яким лицарем необхідно пересуватися королю й у якій клітині повинен починатися їхн спільний рух. Попередньо для всіх пар клітин необхідно обчислити відстань (число ходів) між ними для короля і лицарів

27.2.2 «Полігон»

Цю задачу можна вирішувати динамічним (табличним) методом. Він тут застосовний у силу таких міркувань.

Нехай знайдено розв'язок для деякого відрізка Полігона, тобто відома послідовність видалення ребер, що призводить до оптимального значення. Розглянемо ребро, що видалилося останнім. Це ребро розбиває розглянутий відрізок полігона на дві частини. Очевидно, що можна перевпорядкувати послідовність видалення ребер так, щоб спочатку видалялися ребра з однієї частини, після чого - з іншої. Відповідні підпослідовності будуть забезпечувати оптимальне значення для кожної з частин.

Хоча для розв'язання задачі необхідно знайти максимальне значення Полігона, у процесі розв'язання знадобляться мінімальні значення частин Полігона. Це пов'язано з тим фактом, що видалення ребра з операцією множення і суміжними вершинами з від'ємними значеннями дає в результаті вершину з додатнім значенням.

Алгоритм починає роботу з відрізків Полігона довжини 1 (1 вершина). Для них обчислюється максимальне і мінімальне значення. На кожному

наступному кроці обробюються відрізки на одиницю більшої довжини. Оптимальні значення шукаються перебором ребер відрізка. На останньому кроці обчислюються оптимальні значення для відрізків довжини $N-1$. Серед них необхідно вибрати максимальне - розв'язок задачі.

27.2.3 «Картинки»

Візьмемо пряму, рівнобіжну вертикальної осі і будемо рухати її уздовж горизонтальної осі зліва праворуч, запам'ятовуючи, які прямокутники перетинаються прямою. У вихідному і кінцевому положенні пряма не буде перетинати ніякі прямокутники. Інтерес представляють ситуації, у котрих деякий відрізок прямої входить усередину області, що покривається, або виходить із неї, у цих випадках виявлена вертикальна границя області, і необхідно збільшити довжину виявленого периметра.

Очевидно, що аналогічні міркування вірні і для горизонтальних частин периметра.

Поточний стан прямої можна зберігати за допомогою масиву (оскільки усі координати цілі) або списком відрізків перетину з прямокутниками. Для кожного відрізка необхідно пам'ятати кількість прямокутників, що його покривають. Пряму можна рухати по цілочисельним координатах із кроком 1, але краще відсортувати координати прямокутників і пересувати пряму по ним.

Розділ 28. Турція'1999

28.1 Завдання першого туру

28.1.1 «Квіткова крамничка»

Розглянемо, як додавання нового букету до вже розташованих в вазах впливає на розв'язання задачі. Цей новий букет можна помістити в одну з ваз, але при цьому всі інші букети мають бути розташовані лівіше від нього. Перебравши всі варіанти розташування букетів в попередніх вазах можна знайти оптимальний варіант їхнього розташування, а перебравши всі вази для нового букету — знайти розв'язок задачі.

Ефективно цей процес можна організувати так. Позначимо $b_{i,j}$, $i \geq j$ оптимальне значення естетичної характеристики i розміщення перших j букетів у перших вазах. $b_{i,j}$ можна так визначити через попередні:

$$b_{i,j} = \max(a_{i,j} + b_{i-1,j-1}, b_{i,j-1})$$

Перший варіант відповідає випадку, коли букет i поміщується у вазу j (значення естетичної характеристики цього розміщення додається до значення естетичної характеристики розміщення $i-1$ букету у $j-1$ вазі), другий варіант — букет i було розташовано в одній із $j-1$ попередніх ваз. Значення $b_{F,Y}$ буде розв'язком задачі.

Для того, щоб отримати не тільки оптимальне значення естетичної характеристики, а і, як вимагається в задачі, розташування букетів в вазах, необхідно зберігати додаткову інформацію, а саме для кожного $b_{i,j}$ пам'ятати за якою з альтернатив його було обрано. Просуваючись у зворотному напрямку від $b_{F,Y}$ до $b_{1,1}$ можна отримати оптимальне розташування.

Обчислення $b_{i,j}$ можна проводити одночасно зі зчитуванням значень $A_{i,j}$.

28.1.2 «Сховані коди»

По-перше, звернемо увагу на розмір вхідних даних у цій задачі. Об'єм заданого тесту значно більше, ніж об'єм усіх інших даних, тому розв'язок необхідно оптимізувати відносно саме цього параметру. Скоріш за все потрібно знайти алгоритм, який буде розв'язувати задачу за один прохід тексту, при цьому не буде запам'ятовувати прочитаний текст, а буде працювати з деякою обмеженою послідовністю прочитаних останніми літер, найкраще з одною літерою.

Розв'язання задачі, таким чином, схематично буде виглядати так: починаючи з першого символу вхідного тексту накопичувати деяку інформацію про покриваючі послідовності, змінюючи її після читання наступного символу.

В процесі читання вхідної послідовності необхідно виконувати дві операції:

1. Розпізнавання покриваючих послідовностей.
2. Пошук послідовності покриваючих послідовностей, що призводить до максимальної сумарної довжини кодових слів.

Першу задачу можна розв'язувати, зберігаючи для кожного слова довжину його початкової частини, яка вже зустрілась у тексті. При зчитуванні наступного символу тексту, якщо він співпадає для деякого слова з його першою невикористаною буквою, збільшимо довжину прочитаної частини на одиницю, інакше пропустимо цю букву для даного слова.

Якщо читання наступної літери призвело до знаходження покриваючої послідовності для деякого слова (слів), необхідно врахувати цю послідовність (послідовності) у побудові максимального набору покриваючих послі-

довностей. Можливі два випадки: або знайдену послідовність буде використано у оптимальному покритті, тоді довжину покритого слова необхідно додати до довжини слів оптимальних покриттів частини послідовності що не перекривається зі знайденою, або її використано не буде — ніяких перетворень виконувати не потрібно.

Обмеження, які у великій кількості присутні у задачі, дозволяють позбавитись великої кількості ускладнень, що виникають при розв'язанні задачі. Наприклад, обмеження на довжину покриваючої послідовності дозволяє обмежитись зберіганням останньої тисячі оптимальних покриттів, аналіз тільки мінімальних справа покриваючих послідовностей — пошуком тільки найкоротших покриваючих послідовностей.

28.1.3 «Підземне місто»

Загальним підходом до розв'язання таких задач є побудова дерева, вершинам якого відповідають запитання, а листкам — знайдені відповіді. Якщо систему запитань сформовано таким чином, що дерево виходить достатньо збалансованим — отримаємо більш ефективну в гіршому випадку програму розв'язку. Але побудова збалансованого дерева є достатньо складною задачею.

Замість побудови збалансованого дерева можна будувати дерево, збалансоване не за довжиною гілок, а за кількістю відповідей у кожній з гілок. Для цього на кожному кроці серед усіх можливих запитань будемо шукати таке, відповідь на яке розбиває множину можливих квадратів на дві частини, що найменш розрізняються за розміром. Якщо є декілька таких запитань, оберемо довільне.

Якщо за допомогою запитань визначити положення не вдалося, або жодне запитання не призводить до ефективного розділення відповідей, можна використовувати переміщення по карті.

28.2 Завдання другого туру

28.2.1 «Світлофори»

В цій задачі описано деяку дискретну систему. Запрограмувавши роботу цієї системи можна моделювати переміщення транспортного засобу між перехрестями. Суттєвими при моделюванні є те моменти часу, коли змінюється стан деякого світлофора або транспортний засіб досягає перехрестя. Якщо змінюється світло на деякому світлофорі — необхідно розглянути можливість руху тих транспортних засобів, які знаходяться на цьому перехресті, та на тих перехрестях, які безпосередньо з'єднані з ним дорогами. Якщо

ж транспортний засіб прибуває на перехрестя, то необхідно або залишити його там чекати, або, якщо дозволяють стани світлофорів, спрямувати до іншого перехрестя.

Будемо вважати, що в нас є необмежена кількість уявних транспортних засобів, які на початку моделювання всі зосереджені на початковому перехресті. Розділимо їх на частини у відповідності до кількості сусідніх перехресть, та відправимо ті з них, пересування яких дозволено станами світлофорів, інші залишимо чекати. В момент, коли одна з груп досягне свого перехрестя, знову розділимо її на частини і відправимо у напрямку сусідніх перехресть.

Таким чином транспортні засоби будуть розповсюджуватись по перехрестях, і час прибуття першого транспортного засобу до перехрестя є мінімальним часом подорожі з початкового перехрестя. Момент часу, коли перший транспортний засіб досягне кінцевого перехрестя є розв'язком задачі.

Додаткова річ, про яку треба потурбуватись — розпізнавання недосяжності кінцевого перехрестя. Це може статись у тому випадку, коли не існує шляхів між перехрестями, або всі вони містять дороги, по яким не можна їздити. Такі дороги вирізняються тим, що світлофори на з'єднуваних перехрестях завжди різного кольору, і їх можна видалити одразу підчас читання вхідних даних або безпосередньо після.

28.2.2 «Вирівнювання»

Одним з можливих варіантів розв'язання цієї задачі є наступний.

Приберемо з усіх стопок по однаковій, максимально можливій кількості карток. Якщо всі стопки стали порожніми — знайдено розв'язок. Інакше є декілька порожніх стопок, і їх необхідно заповнити картками з переповнених стопок. Після наповнення усіх порожніх стопок знову заберемо максимальну можливу кількість карток. Цей процес будемо продовжувати до забирання усіх карток.

Єдине, що треба довести — можливість заповнення пустих стопок хоча б одною карткою. Якщо поруч з пустою стопкою є стопка з принаймні трьома картками, цю стопку можна використовувати для заповнення пустої. Інакше неважко знайти процедуру, яка буде заповнювати пусту стопку, не створюючи при цьому нових пустих стопок. Розглянемо частковий випадок такої процедури, в загальний можна побудувати аналогічно.

Нехай маємо таку конфігурацію стопок: $(a, 1, 1, 1, 0, 0, \ldots)$. Необхідно заповнити перший 0 біля 1. Перенесемо одну картку с першої стопки на другу, отримаємо $(a - 1, 2, 1, 1, 0, 0, \ldots)$. Необхідну кількість разів повторимо перекладання зі стопки з двома картками, отримаємо $(a, 1, 1, 0, 1, 0, \ldots)$. Пі-

сля наступного аналогічного перетворення маємо $(a, 1, 0, 1, 1, 0, \ldots)$, потім $(a, 0, 1, 1, 1, 0, \ldots)$ і нарешті $(a-1, 1, 1, 1, 1, 0, \ldots)$. Повторюючи цю процедуру можна заповнити всі пусті стопки, а, отже, і зрівняти кількості карток в стопках.

28.2.3 «Смужка землі»

Знову, зважаючи на великий розмір вхідних даних, потрібно побудувати алгоритм, який не буде зберігати усі вхідні дані, а буде накопичувати необхідну для розв'язку інформацію в процесі зчитування вхідних даних. Розмір цієї інформації не може мати розмір зрівняний з розміром вхідних даних. В цій задачі вдіється накопичувати інформацію про кожний стовпчик карти, не розбиваючи його на квадрати.

В процесі роботи будемо додавати до стовпчиків по клітині, починаючи з північно-західного кута, в порядку слідування даних у файлі. Для кожного стовпчика будемо зберігати інформацію про довжини смужок, які починаються в деякій клітині стовпчика та висоти в яких відрізняються не більше ніж на C та містять цю клітину. Таких смужок буде $C+1$, з діапазонами висот $[C, 0]$, $[C-1, 0-1]$,. . . ,$[0, -C]$. Переходячи до розглядання наступної клітини в стовпчику, в залежності від висоти нової клітини, деякі смужки будуть відкинуті, деякі продовжені до нової, створені нові, що складаються лише з цієї клітини.

Використовуючи інформацію про стовпчики, можна шукати прямокутники наступним чином. Проглядаючи рядок карти, паралельно з добудовою смужок, будемо об'єднувати їх в прямокутники. Для цього необхідно зберігати інформацію про прямокутники, південно-східним кутом яких є остання розглянута клітина, способом, схожим на спосіб зберігання смужок. Додаючи до них смужки, які закінчуються в поточній клітині, будемо отримувати нові прямокутники. Якщо серед них знайдеться прямокутник більшої площі ніж поточна оптимальна — знайдено кращій розв'язок, інакше після перебудови поточного набору прямокутників відбувається перехід до наступної клітини без зміни поточного оптимального розв'язку.

Наведені в цій статті рекомендації потребують подальшої деталізації для того, щоб бути закінченими розв'язками. Можливо для деяких задач існують більш ефективні алгоритми, ніж наведені. Доробку та удосконалення залишаємо читачеві. Це буде добрим тренуванням для тих, хто бажає приймати участь у олімпіадах з інформатики або просто цікавиться цією

наукою.

Розділ 29. Китай'2000

29.1 Завдання першого туру

29.1.1 «Паліндром»

Добрим методом розв'язання цієї задачі буде знайти довжину L самої довгої спільної підпослідовності для вхідного рядка та його обернення. Відповіддю задачі тоді буде $N - L$. Альтернативним підходом є порівняння початку рядка з оберненням його закінчення. Довжина самої довгої спільної підпослідовності може бути обчислена з використанням динамічного програмування. Час виконання такої програми можна оцінити як $O(N^2)$.

29.1.2 «Паркування»

Фінальний стан паркування може бути визначеним сортуванням списку типів машин. Це можна зробити за лінійний час.

Для побудови послідовних етапів пересування автомобілів можна використовувати жадібний алгоритм, оскільки на кожному кроці можна гарантувати що на кінцеві позиції буде розміщено щонайменше $W-1$ автомобілів. Тоді кількість етапів що використовує жадібний алгоритм є $\left\lceil \frac{N}{W-1} \right\rceil = Q$, де D — кількість автомобілів, що розташовано не на своїх місцях. Взагалі, знаходження мінімальної кількості кроків є $\mathcal{NP}$-складною задачею. Навіть зменшення кількості етапів жадібного алгоритму на 1 є, взагалі, таким же складним як знаходження мінімуму.

Жадібний алгоритм може бути реалізовано, щоб він виконувався за час, пропорційний N.

29.1.3 «Медіанна енергія»

Розглянемо декілька методів що можна застосовувати для розв'язання цієї задачі.

1. Сортування вставками. Будемо зберігати сортований список вже розглянутих об'єктів та послідовно додавати до нього наступні елементи. Місце для вставки можна шукати лінійним, бінарним або тернарним пошуком. Лінійні вставки мають квадратичну оцінку, бінарні та тернарні вставки — $O(N \cdot \log N)$. Замість повного списку оброблених

об'єктів достатньо зберігати тільки половину від загальної кількості об'єктів. Далі, під час перегляду другої половини об'єктів можна вилучати зі списку два крайніх елементи, таким чином на кожному кроці кількість елементів буде зменшуватись на 1.

2. Знаходження розбиття. Метод нагадує вибір медіани (як швидке сортування, з відкиданням частини що не містить медіани), але розглядається розбиття на три частини (двома елементами). Ці елементи можна брати з одного краю, з обох країв, розташовані на відстані одна третина та дві третини від краю або випадковим чином.

3. Впорядковані групи. Будемо підтримувати групи об'єктів деякого розміру (наприклад, 8). Кожна група повинна бути відсортована, а список груп має бути відсортованим за мінімальним елементом в групі. Така структура даних дозволяє запобігти деяких порівнянь.

29.2 Завдання другого туру

29.2.1 «Поштові відділення»

Точний розв'язок може бути отриманий динамічним програмуванням базуючись на двовимірній таблиці. Необхідно зберігати всю таблицю розміром, $V \times P$, кожен елемент якої може бути обчислений за час $O(V)$. Складність алгоритму — $O(P \cdot V^2)$.

29.2.2 «Стіни»

Прямолінійний метод розв'язку може базуватись на графі, який є дуальним до планарного графа, що представляє карту міст та з'єднуючі стіни. Дуальний граф можна отримати, якщо розглядати області як вершини графа, які з'єднані якщо відповідні області мають спільну стіну. Прохід по ребру в дуальному графі відповідає перетину стіни, таким чином, мінімізація перетинів відповідає пошуку найкоротшого шляху в дуальному графі.

Перебравши всі області можна знайти ту, до якої сумарна кількість перетинів мінімальна.

29.2.3 «Конструювання з блоків»

Нижньою оцінкою кількості блоків в декомпозиції є $V/4$ заокруглене догори. Знаходження мінімальної декомпозиції простим пошуком з поверненням є повільним тому що існує багато розв'язків з маленькими блоками.

Техніка гілок та границь допомагає значно скоротити час виконання програми. Коли отримано часткову декомпозицію з T блоками, нижня границя для кількості блоків в закінченій декомпозиції $\lceil T + W/4 \rceil$, де W є об'ємом для якого треба провести декомпозицію. Таким чином рекурсія може бути припинена коли $T + W/4$ перевищує отриманий мінімум.

Розділ 30. Фінляндія'2001

30.1 Завдання першого туру

30.1.1 «Мобільні телефони»

Очевидним є метод, який зберігає у масиві для кожного квадрата відповідні кількості телефонів. Відповідь на кожен запит обчислюється як сума кількостей телефонів у всіх квадратах прямокутника. Такий алгоритм потребує $S \times S$ операцій і є досить повільним.

Розглянемо одномірний випадок цієї задачі. Якщо у масиві замість кількості телефонів у відповідному квадраті I зберігати сумарну кількість телефонів у квадратіх з номерами від $I - 2^K + 1$ до I, то підтримувати такий масив та знаходити необхідні суми у ньому можна за логаріфмічний час. Якщо цю ідею використати у двовимірному випадку, то загальна складність оновлення та пошуку буде $O(\log^2 S)$.

30.1.2 «Гра ioiwari»

Ця задача є типовою ігровою задачею. Для ефективного її розв'язання достатньо побудувати граф цієї гри та знайти шлях у графі, який призведе до виграшу. Текож можливе використання жадібного алгоритму (яий робить локально-оптимальний хід) або алгоритму, який робить ходи випадково (але за правилами).

30.1.3 «Twofive»

Розв'язок базується на функції, яка за довільним фіксованим розташуванням букв у таблиці буде обчислювати кількість способів припустимого розташування букв що залишились у таблиці. Ця функція буде розташовувати букви у припустимих вільних позиціях таблиці, рекурсивно обчислювати кількість способів для меншої кількості вільних літер та сумувати отримані результати.

Можна довести, що кількість розташувань вільних літер залежить тільки від конфігурації розташування зафіксованих літер у таблиці, і не залежить безпосередньо від розташування. Спираючись на цей факт можна зберігати кількість варіантів розташування для кожної конфігурації зафіксованих букв та використовувати ці значення щоб уникнути повторного рекурентного обчислення, суттєво скорочуючи перебір.

Маючи таку функцію, досить просто розв'язувати обидві підзадачі. Для того, щоб обчислити номер слова, необхідно послідовно проходити по літера слова від першої до останньої, на кожному кроці обчислюючи кількість слів що необхідно пропустити щоб досягти відповідну літеру. Наприклад, якщо слово починається з літер `ACG`, то необхідно пропустити всі слова що передують словам що починаються з `A` (0 слів), всі слова що передують словам, що починаються з `AC` (тобто слова що починаються `AB`), всі слова передують словам, що починаються з `ACG` (`ACD`, `ACE`, `ACF`) і так далі.

У зворотньому випадку процудура схожа. Необхідно послідовно збільшувати кожну букву, доки кількість пропущених слів не перевищіть задане число. Потім повернутися до попереднього значення букви, перейти до наступної позиції у слові то продовжити підбираючи букву у цій позиції, доки не буде отримане все слово.

30.2 Завдання другого туру

30.2.1 «Score»

Виграшну стратегію у цій грі можна отримати, використовуючи алгоритм пошука у глибину. За допомогою нього можна дослідити усі можливі варіанти ходів, оскільки таких варіантів скінчена кількість (гарантується, що немає нескінчених ігор) і обрати той, просування по якому призводить до виграшу. Також, як і у задачі «Гра ioiwari», можливе використання таких стандартних підходів, як жадібний та випадковий.

30.2.2 «Подвійне кодування»

Ця задача мала нестандартне формулювання. Учасникам не потрібно було здавати для перевірки програму-розв'язок. Замість того було надано тестові файли і необхідно було знайти довільним методом (можливо вручну, ала значно ефективніше було використовувати допомідну програму) відповіді для цих тестів.

На перший погляд здається, що використання подвійного ключа забезпечує значно більшу стійкість до розкодування. Дійсно, на ключах довжини N повний перебір вимагає 2^N операцій перевікки ключа. Може здаватись,

що подвійне кодування відповідає кодуванню з використання ключа довжини $2N$, і відповідно вимагає часу 2^{2N}. Насправді, використовуючи метод «зустріч посередині», можна знайти ключі за час $2N + 1$, що відповідає ключу довжини $N + 1$.

Метод працює натупним чином. Вихідний текст кодується один раз усіма можливими ключами та результати кодування зберігаються. Після цього закодований текст розкодується усіма можливими ключами та результат розкодування шукається серед збережених. Знайдене співпадіння і визначає пару ключів, які могли використовуватись при кодуванні.

30.2.3 «Склад»

Простим метод розв'язання цієї задачі є перебір усих можливих послідовностей контейнерів та вибір тих, які призводять то необхадного розташування контейнерів на складі. Цей метод розв'язує задачу для невеликої кількості контейнерів.

Більш ефективним є метод, який послідовно видаляє контейнери зі складу, відновлюючи попередній стан складу, ніби видалений контейнер був останнім, що надійшов. Такий перебір, після деяких оптимізацій працює значно швидше, ніж у першому методі.

Розділ 31. Південна Корея'2002

31.1 Завдання першого туру

31.1.1 «Шкідлива жаба»

Простий алгоритм розв'язання цієї задачі, що потребує часу $O(N^3)$, перебирає усі $O(N^2)$ сегментів ліній, що задаються множиною точок, та намагається розширити ці сегменти в усіх напрямках.

Ефективний $O(N^2)$ алгоритм для цієї задачі базується на алгоритмі пошуку рівновіддаленої колінеарної підмножини. Алгоритм виконує перекриття всіх рівновіддалених трійок для того щоб визначити всі максимальні рівновіддалені колінеарні підмножини. Перекриття виконується побудовою неорієнтованого графа де для кожної рівновіддаленої трійки (p_A, p_B, p_C) створюються вершини $< A, B >$ та $< B, C >$ та ребро $(< A, B >< B, C >)$. Компоненти зв'язності цього графу відповідають максимальним рівновід-

даленим колінеарним підмножинам початкової множини. Зауважимо, що шлях жаби є просто лінійним ланцюгом з'єднаних вершин (з принаймні одним ребром та двома вершинами, що відповідають трьом пошкодженим кущам) цього графу. Кожна вершина цього графу має ступінь не більше 2, звідки кількість вершин та ребер цього графу мають розмір $O(N^2)$. Використовуючи ці факти можна знайти всі максимальні рівновіддалені колінеарні підмножини віхідної множини за час $O(N^2)$ використовуючи побудований граф.

Єдиною проблемою залишається ефективний пошук рівновіддалених трійок, з яких будується граф. Очевидний метод перебору усіх трійок пошкоджених кущів має неприйнятну складність $O(N^3)$. Якщо замість цього поле зберігається як двовимірний масив, що містить номер приземлення на кущі (наприклад, якщо 100-й пошкодженний кущ знаходиться у позиції $(10, 12)$, тоді значенням елемента масива $(10, 12)$ є 100), можна перебрати усі пари пошкоджених кущів та для кожної пари за константний час визначити чи є третій рівновіддалений пошкоджений кущ. Ця стратегія потребує $O(N^2)$ часу, але використовує $5\ 000 \times 5\ 000 \times 2 \approx 50$ MB пам'яті. Оскільки граф потребує для свого зберігання також $O(N^2)$ пам'яті , ця стратегія перевищіть ліміт в 64 MB. Однак, якщо масив дуже розріджений, він може зберігатись у пам'яті за допомогою хеш-таблиць, що не збільшує часову складність. Третім та найкращим варіантом є побудова графа за лінійний час сортуванням позицій пошкоджених кущів (наприклад, спочатку по рядкам потім по стовпчикам) та зберігання 3-х показчиків у список (A, B, C для p_A, p_B, p_C, $A < B < C$) таким чином: змінюючи A по всіх значеннях, для кожного A просуваємо B та C по списку, на кожному кроці пересуваючи B або C, намагаючись залишатись якомога ближче до рівних відстаней. Коли знайдено три рівновіддалених точки — вершини та ребро додається до графа.

Існує також динамічний $O(N^2)$ алгоритм розв'язання цієї проблеми, якому також властива проблема з пам'яттю описана вище. Додатково до описаної матриці зберігається інша $O(N^2)$ матриця що містить індексовані по pA рядки. Кожен рядок містить N елементів, один на кожен кущ p_B, що позначає кількість приземлень у віріанті пляху жаби, що проходить через кущі p_A та p_B і який використовує тільки кущі з початку списку (до p_A). Припустимо, що таблицю заповнено до рядка A, рядок $A+1$ заповнюється переглядом усіх $O(N)$ пошкоджених кущів B перед p_A, і якщо є пошкоджений кущ C такий що A, B та C є рівновіддаленими, знаходиться в масиві кількість приземлень з C через B, збільшується на 1 та записується як B-й елемент рядка $A+1$. Якщо C опиняється за межами поля, то вважається що шлях має 2 куща. В той же час перевіряється, чи вийде наступний пошкоджений кущ за межі поля, і якщо так, вважається що знайдено шлях жаби.

Для ефективного знаходження C потрібен масив розміром 50 MB. Знову можна використати хеш-таблиці, що, в середньому, не погіршить швидкість алгоритму.

31.1.2 «Утопія»

Задача є двовимірною, але може бути ров'язана як дві незалежні одновимірні. Розглянемо одновимірну задачу: задано N чисел і послідовність N знаків регіонів (+ або −), визначити послідовність N чисел зі знаками x_i, так що знак $\sum_{i<k} x_i$ відповідає знаку i-го регіону.

Загальна ідея є достатньо інтуітивною, хоча потрібні зусилля щоб переконатись що вона працює. Спочатку необхідно відсортувати числа у зростаючому порядку а потім присвоїти числам знаки, що чергуються (при цьому залишається визначити знак x_1, а відповідно і всіх інших). Потім ми починаємо в деякому місці посередині послідовності, та рухаємось назовні, використовуючи великі числа для зміни знаку суми послідовності та малі для для того щоб зберегти знак. Розглянемо декілька тверджень.

Означення. Назвемо послідовність ненульових цілих

$$X = (x_a, x_{a+1}, \ldots, x_b), a \le b$$

знакозмінною послідовністю, якщо:

1. $|x_a| < |x_{a+1}| < \ldots < |x_b|$
2. для всіх i, $a < i \le b$, знак x_i відрізняється від знаку x_{i-1}

Лема 1. *Нехай $X = (x_a, x_{a+1}, \ldots, x_b)$ — знакозмінна послідовність. Знак x_b співпадає зі знаком $\sum_{a\le i\le b} x_i$, загальною сумою елементів X.*

Доведення. Нехай $x_b > 0$. Якщо кількість $b - a + 1$ елементів послідовності є парною (відповідно непарною) часткові суми $x_a + x_{a+1}, x_{a+1} + x_{a+2}, \ldots, x_{b-1} + x_b$ (відповідно $x_a, x_{a+1} + x_{a+2}, \ldots, x_{b-1} + x_b$) є додатніми. І загальна сума $\sum_{a\le i\le b} x_i$ таких часткових сум також є додатньою. Нехай $S = (s_a, s_{a+1}, \ldots, s_b)$, $a \le b$ буде послідовністю знаків плюс та мінус. Послідовність $X' = (x_{i_a}, x_{i_{a+1},\ldots,x_{i_b}})$ буде вірною по відношенню до S, якщо знак $\sum_{a\le j\le k} x_{i_j}$ дорівнює S_k для кожного k, $a \le k \le b$. □

Теорема 1. *Нехай $X = (x_a, x_{a+1}, \ldots, x_b)$ — знакозмінна послідовність. $S = (s_a, s_{a+1}, \ldots, s_b)$, $a \le b$ — послідовність знаків Якщо знак x_b співпадає зі знаком s_b, то існує послідовність $X' = (x_{i_a}, x_{i_{a+1},\ldots,x_{i_b}})$, така що*

1. $\{x_{i_a}, x_{i_{a+1},\ldots,x_{i_b}}\} = x_a, x_{a+1}, \ldots, x_b$

2. X' є вірною по відношенню до S

Доведення. Проведемо доведення за індукцією по кількості k елементів в X. Якщо $k = 1$, легко бачити що твердження теореми виконується. Припустимо, що $k \geq 2$. Нехай $S_1 = S - S_b$, тобто $S_1 = (S_a, S_{a+1}, \ldots, S_{b-1})$. Розглянемо два випадки:

1. Знак x_b співпадає з s_{b-1}. Покладемо $X_1 = X - x_a$, тобто $X_1 = (x_{a+1}, x_{a+2}, \ldots, x_b)$, та $t = x_a$.
2. Знак x_{b-1} співпадає з s_{b-1}. Покладемо $X_1 = X - x_b$, тобто $X_1 = (x_a, x_{a+1}, \ldots, x_{b-1})$, та $t = x_b$.

Зауважимо, що X_1 є знакозмінною послідовністю, а S_1 є такою послідовністю знаків, що знак останнього елементу X_1 співпадає з останнім знаком у S_1. Отже, за припущенням індукції, існує вірна послідовність X'_1 по відношенню до S_1. З цього випливає, що $X' = (X'_1, t)$ є вірною по відношенню до S. □

Тепер можна сформулювати алгоритм розв'язання задачі.

I *Прочитати вхідні дані*

(a) прочитати N

(b) прочитати $2N$ кодових чисел та розділити їх на дві рівні групи, $|A| = |B|$

(c) прочитати послідовність регіонів $R = (r_1, r_2, \ldots, r_N)$

II *знайти x-координати кодових пар*

(a) побудувати послідовність знаків $S_1 = (s_1, s_2, \ldots, s_N)$, таку що для всіх j, $1 \leq j \leq N$, $s_j =$"+" якщо $r_j = 1, 4$, інакше $s_j =$"−".

(b) побудувати знакозмінну послідовність $X = (x_1, x_2, \ldots, x_n)$ з елементів A, щоб знак x_N дорівнював s_N.

(c) за X та S побудувати послідовність $X' = (x_{i_1}, x_{i_2}, \ldots, x_{i_N})$ вірну відносно S за доведенням теореми.

III *знайти y-координати кодових пар*

(a) побудувати послідовність знаків $S_1 = (s_1, s_2, \ldots, s_N)$, таку що для всіх j, $1 \leq j \leq N$, $s_j =$"+" якщо $r_j = 1, 2$, інакше $s_j =$"−".

(b) побудувати знакозмінну послідовність $Y = (y_1, y_2, \ldots, y_n)$ з елементів A, щоб знак y_N дорівнював s_N.

(c) за Y та S побудувати послідовність $Y' = (y_{i_1}, y_{i_2}, \ldots, y_{i_N})$ вірну відносно S за доведенням теореми.

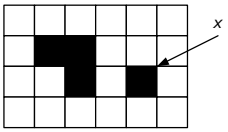
IV *вивести результат* $(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \ldots, (x_{i_N}, y_{i_N})$

Складність цього алгоритму $O(N \log N)$.

31.1.3 «XOR»

Назвемо точкою сітки точку перетину двох ліній сітки. Кутом зображення будемо називати точку сітки, що має непарну кількість заповнених сусідніх пікселів зображення.

Якщо точка сітки є кутом зображення, будемо казати, що її ic-значення дорівнює 1, інакше — 0.



Мал. 31.1:

Наприклад, точка x на малюнку вище має 4 сусідні пікселі, один з яких заповнений, інші 3 — ні. Таким чином ця точка є кутом зображення. Всього на малюнку є 10 кутів зображення.

Не важко помітити, що зображення є повністю незаповненим тоді і тільки тоді, коли немає кутів зображення. Наступне твердження буде буде базою розв'язання задачі.

Лема 2. *Нехай виконано операцію* $\mathsf{XOR}(L, R, T, B)$. *Нехай, Q буде прямокутником, до якого була застосована ця операція, з кутами в точках сітки a, b, c, d. Тоді* $\mathsf{XOR}(L, R, T, B)$ *змінила ic-значення тільки для точок сітки a, b, c, d, для всіх інших ic-значення залишилися незміненими.*

Доведення. Рівно один піксел, сусідній до a, b, c, d змінив свій колір. З цього випливає, що для цих точок ic-значення змінилися на протилежні. Для інших точок на границі Q два пікселі змінили свій колір (ті що всередині Q), таким чином ic-значення не змінилися для цих точок сітки. Точки всередині Q також не змінили ic-значення, оскільки всі сусідні пікселі змінили свій колір, не вплинувши на парність.

Підчас зменшення кількості кутів зображення бажано робити це максимально кожним викликом XOR. Але не завжди можливо знайти параметри

операції XOR так що вона змінює *ic*-значення усіх кутів області на якій вона діє. □

Лема 3. *Якщо зображення не порожнє, завжди можливо обрати такий виклик* XOR*, що кількість кутів зображення зменшується на 2 або 4.*

Доведення. Поперше зауважимо, що принаймні два кута зображення знаходяться на одній лінії. Можна розглянути перший верхній рядок, в якому є заповнені піксели. Є кут зображення в правому верхньому куті самого правого пікселя та кут зображення в верхньому лівому куті самого лівого пікселя. Ці кути зображення не співпадають і лежать на одній лінії.

Подруге, всі кути зображення не можуть знаходитись на одній лінії. В цьому можна переконатися, розглянувши найверхні та найнижчі кути зображення. Тепер можна вибрати такий прямокутник, що три з його кутів були кутами зображення наступним чином. Виберемо самий верхній з самих правих кутів зображення. Другим виберемо ще один самий правий кут зображення. Третій може бути знайдений так. Починаємо з обраного кута зображення. Тільки піксель що знаходиться знизу ліворуч заповнений, інші —- вільні. Далі рухаємось ліворуч по точкам сітки. На деякому кроці або знизу закінчаться заповнені пікселі, або зверху з'являться заповнені. Якщо це відбулось одночасно — ситуація змінюється на зворотню та продовжується пошук. Якщо потрібна ситуація знайдена (а це має трапитись оскільки в деякому стовпчику мають з'явитись незаповнені клітини зверху та знизу) — знайдено кут.

Таким чином завжди можна знайти параметри виклику XOR так що чотири або три точки сітки в кутах прямокутника на який діє XOR є кутами зображення. Якщо всі 4 кути є кутами зображення, то за Лемою 2 кількість кутів зображення зменшиться на 4. Якщо рівно три є кутами зображення, то за Лемою 2 три кута зображення зникне а один з'явиться, тобто загальна кількість кутів зменшиться на 2. □

Цей розв'язок є 2-оптимальним, тобто викликає функцію XOR не більше ніж в два рази більше ніж теоретично можливий мінімум.

Також можна використати жадібний алгоритм, який буде проходити пікселі по рядках, зверху до низу зліва на право, і для кожного знайденого заповненого пікселя знаходити неперервні послідовності заповнених пікселів праворуч та вниз та застосовувати XOR над утвореним прямокутником.

31.2 Завдання другого туру

31.2.1 «Пакетна обробка завдань»

Ця задача може бути розв'язана за допомогою принципу динамічного програмування. Нехай C_i це мінімальна загальна вартість серед всіх розбиттів завдань $J_i, J_{i+1}, \ldots, J_n$ на групи. Нехай $C_i(k)$ це мінімальна загальна вартість при умові, що перша група сформована як $J_i, J_{i+1}, \ldots, J_{k-1}$. Тобто,

$$C_i(k) = C_k + (S + T_i + T_{i+1} + \ldots + T_{k-1}) \times (F_i + F_{i+1} + \ldots + F_n)$$

Таким чином отримаємо: $C_i = \min\{C_i(k) \mid k = i+1, \ldots, n+1\}$, де $1 \le i \le n$, і $C_{n+1} = 0$. Складність наведеного алгоритму — $O(n^2)$.

Однак, дослідивши властивості $C_i(k)$ можна знайти і більш ефективний — лінійний алгоритм. Можна розмірковувати таким чином. З

$$C_i(k) = C_k + (S + T_i + T_{i+1} + \ldots + T_{k-1}) \times (F_i + F_{i+1} + \ldots + F_n)$$

маємо для $i < k < l$,

$$C_i(k) \le C_i(l) \Leftrightarrow C_l - C_k + (T_k + T_{k+1} + \ldots + T_{l-1}) \times (F_i + F_{i+1} + \ldots + F_n) \ge 0$$

$$(C_k - C_l)/(T_k + T_{k+1} + \ldots + T_{l-1}) \le (F_i + F_{i+1} + \ldots + F_n)$$

Нехай $g(k,l) = (C_k - C_l)/(T_k + T_{k+1} + \ldots + T_{l-1})$, і $f(i) = (F_i + F_{i+1} + \ldots + F_n)$

Властивість 1. Припустимо, що $g(k,l) \le f(i)$, для $1 \le i < k < l$. Тоді $C_i(k) \le C_i(l)$.

Властивість 2. Припустимо, що $g(j,k) \le g(k,l)$, для $1 \le j < k < l \le n$. Тоді для кожного i такого, що $1 \le i < j$, $C_i(j) \le C_i(k)$ або $C_i(l) \le C_i(k)$.

Друга властивість озачає, що якщо $g(j,k) \le g(k,l)$, для $j < k < l$, то C_k можна не обчислювати для знаходження F_i. На основі цієї властивості можна розробити лінійний алгоритм, який полягає у наступному.

Алгоритм Алгоритм обчислює значечення C_i для i від n до 1 (з кроком -1). Він використовує черго-подібний список $Q = (i_r, i_{r-1}, \le, i_2, i_1)$ з хвістом i_r і головою i_1, який задовільняє наступним властивостям:

$$i_r < i_{r-1} < \ldots < i_2 < i_1$$

і

$$g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \ldots > g(i_2, i_1) \qquad (*)$$

Коли C_i обчислене,

1. *З використанням $f(i)$, видалити непотрібний елемент з голови Q.*
   Якщо $f(i) \geq g(i_2, i_1)$, видалити i_1 з Q оскільки для кожного $h \leq i$, $f(h) \geq f(i) \geq g(i_2, i_1)$ і $C_h(i_2) \leq C_h(i_1)$ за властивістю 1. Продовжувати цю процедуру доки для деяких $t \geq 1$,

   $$g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \ldots > g(i_{t+1}, i_t) > f(i)$$

   Тоді за властивістю 1, $C_i(i_{v+1}) > C_i(i_v)$ для $v = t, \ldots, r-1$ або $r = t$ і Q містить тільки i_t. Таким чином, $C_i(i_t) = \min\{C_i(k) \mid k = i+1, \ldots, n+1\}$

2. *При вставці i у хвіст Q, зберігати Q так, щоб умова (*) виконувалася.*
   Якщо $g(i, i_r) \leq g(i, i_{r-1})$, видалити i_r з Q за властивістю 2. Продовжувати цю процедуру, доки $g(i, i_v) > g(i_v, i_{v-1})$. Добавити i як новий хвіст Q.

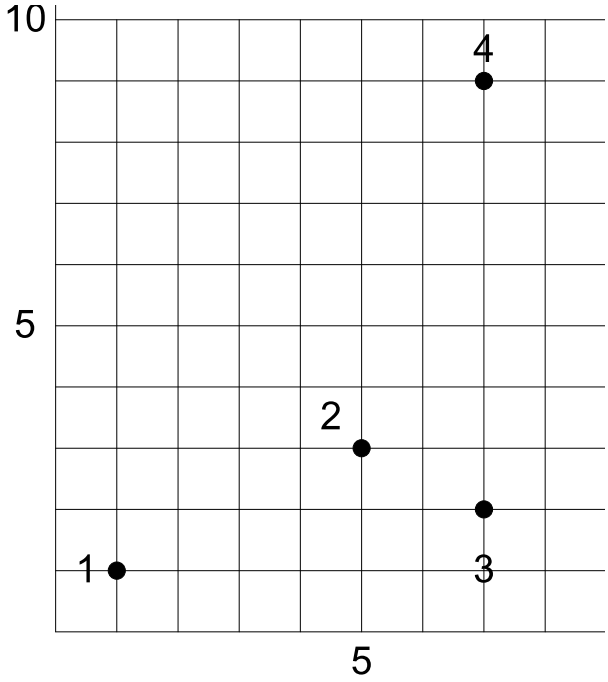
Аналіз Кожне i може бути додане до Q і видалене з Q лише один раз. Кожна операція додавання та видалення займає константний час. Таким чином, оцінка часу є $O(n)$.

31.2.2 «Автобусні зупинки»

Діаметром мережі маршрутів назвемо довжину самого довгого шляху між довільними двома зупинками в мережі. Нашою метою є знаходження мінімального значення діаметрів серед усіх допустимих виборах пересадних станцій та з'єднань зупинок з ними. Нехай D_1 — мінімальне значення максимальної відстані між всіма парами зупинок, що з'єднані через одну пересадну станцію при всіх варіантах вибору пересадочної станції. D_2 – мінімальне значення максимальної відстані між всіма парами зупинок, що з'єднані через рівно дві пересадні станції при всіх варіантах вибору пересадних станцій та з'єднання зупинок з ними. Діаметром мережі буде мінімум значень D_1 та D_2.

Спочатку розглянемо метод обчислення D_1. Якщо точка p буде виступати пересадочною станцією, через яку проходить найдовжчій шлях, довжина цього шляху є $d(p,q) + d(p,r)$, де точки q та r є найвіддаленішою та другою найвіддаленішою зупинками від p, відповідно. Тоді $D_1 = \min_p\{d(p,q) + d(p,r)\}$ по всім точкам p з вхідних даних. Це значення може бути обчислене за час $O(n^2)$, тому що найвіддаленіша та друга найвіддаленіша зупинки можуть бути знайдені за $O(n)$. Час обчислення можна ще зменшити, зробивши його $O(n \log n)$ використовуючи діаграму Вороного 2-го порядку, але це не варто робити з огляду на те, що загальній алгоритм буде мати складність $O(n^3)$.

Розглянемо на простому прикладі як знайти D_2. Зауважимо, що у цьому випадку найдовший шлях має проходити через дві пересадні станції H_1 та H_2.

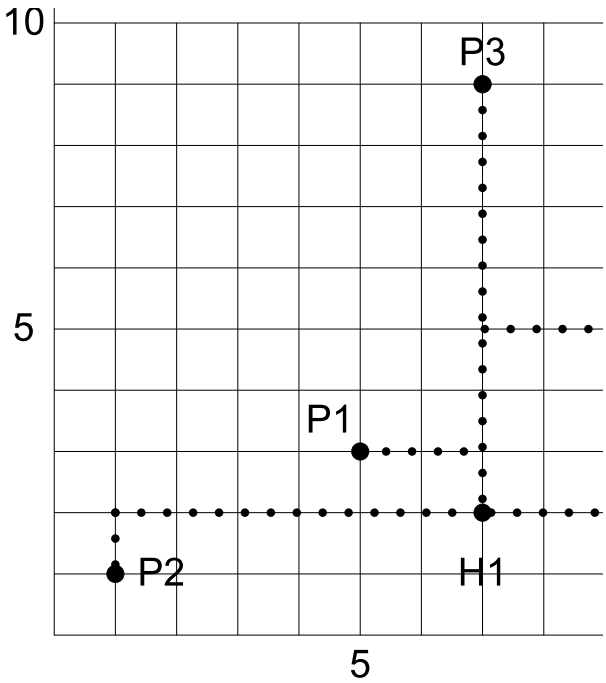


Мал. 31.2:

На мал. 31.2 зображено розміщення 7 зупинок. Ми розглядаємо всі пари автобусних зупинок як можливі дві станції пересадок H_1 та H_2, та обираємо пару автобусних зупинок, що дають мінімальній діаметр. Нехай на початку D_2 суттєво велике. Зафіксуємо дві станції пересадок H_1 та H_2, всі інші зупинки приєднаємо до однієї станції пересадок, скажімо до H_1. Відсортуємо зупинки в масиві P за неспаданням відстані до станції пресадок H_1 (Мал 31.3).

Позначимо $r_1 = d(H_1, P[n-3])$, $r_2 = d(H_2, P[n-2])$, $d_{12} = d(H_1, H_2)$. Якщо $r_1 + d_{12} + r_2 < D_2$, то станція $P[n-2]$ приєднується до пересадної станції H_2, а D_2 отримує нове значення $D_2 = r_1 + d_{12} + r_2$. Мал 31.4 ілюструє ситуацію після виконання цих дій.

Після цього виконаємо аналогічні процедури з $P[n-3]$ та $P[n-4]$. В цьому випадку нова відстань не зменшить D_2, і $P[n-3]$ залишится при-



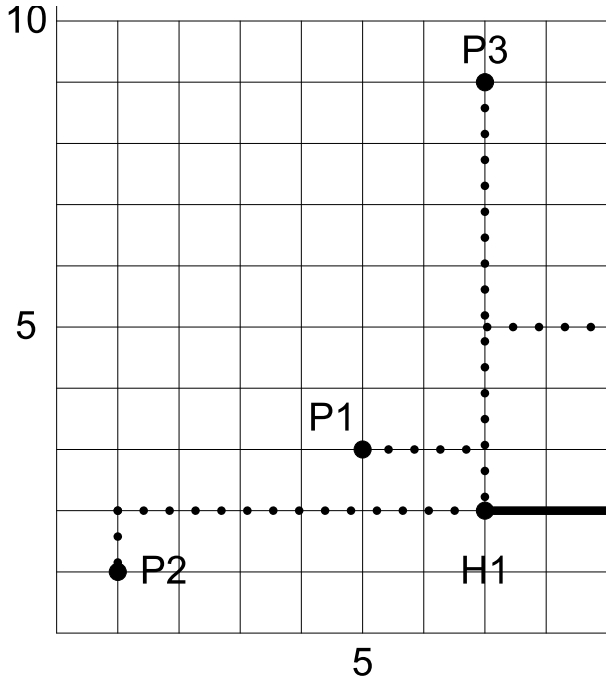
Мал. 31.3:

єднаним до H_1. Повторимо процедуру перевірки просуваючись до $P[1]$, та змінюючи приєднання зупинок при потребі.

31.2.3 «Дві стежки»

Найефективніший з відомих підходів — це застосування шести бінарних пошуків.

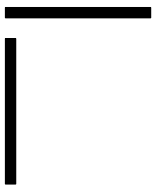
- При використанні цілих рядків та стовпчиків як прямокутних областей для запиту, найверхній і найнижчий рядок та найправіший і найлівіший стовпчик, що містять клітини, які належать стежкам, можуть бути знайдені за допомогою 4-ох бінарних пошуків, або за $4\lceil \log N \rceil$ викликів rect.
- Таким чином, ми отримуємо найменьший прямокутник, що містить обидві стежки. Перевіривши кути цього прямокутника (4 виклики rect, кожен з прямокутником 1×1) ми можемо з'ясувати загальну



Мал. 31.4:

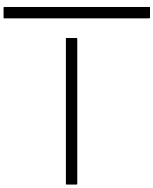
структуру розташування стежок, яка може співпадати із однією з зображених на малюнку, або бути ротацією такої структури.

- Нарешті розв'язок може бути знайдений за один або два бінарних пошука, в залежності від структури.

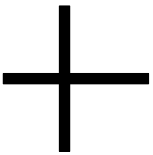


Мал. 31.5:

Мал. 31.6:



Мал. 31.7:



Мал. 31.8:

Це приводить до розв'язку з кількістю операцій: $6 \times \lceil \log N \rceil + 4$. Максимальне значення $\lceil \log N \rceil$ у тестах 14, тобто такий розв'язок використовує 88 викликів rect. Якщо уважно реалізувати алгоритм, оцінка може бути покращена до $6 \times \lceil \log N \rceil + 1$. Існує всього $N^4(N-1)^2/4$ можливих розташувать стежок, тобто $\lceil \log(N^4(N-1)^2) \rceil - 2 \approx \lceil 6 \times \log N \rceil - 2$ викликів необхідні, в середньому, будь-якому алгоритму.

Існують інші варіанти запропонованого підходу. Наприклад, можна намагатися шукати прямокутник, що обмежує стежки, швидче, у для випадку коли він великий. Підходи такого типу призводять до збільшення кількості викликів rect удвічи. Такі розв'язки отримували повний бал на 4 маленьких тестах, та 3 бали на кожному з великих тестів.

Найбільш наївний підхід полягає у скануванні кожної клітини окремо, для знаходження частини стежки, а потім пошуку навколо неї решти. Це може призвести до кількості запитів порядку N^2, або, навіть, трохи більше якщо бути необережним. Такий підхід отримував повний бал на чотирьох тестах розміру 10. Інший метод використовує цілі рядки (або стовпчики), як області для запиту, для локалізації стежок у прямокутнику, і потім застовує подібний підхід для знаходження стежок у цьому прямокутнику. У такому разі буде використано порядку $O(N)$ викликів, де константа буде залежати від деталей реалізації. В залежності від деталей реалізації, такий підхід може отримати непогані оцінки на деяких великих тестах у яких стежки маленькі, і розташовані ближче до границь.

Розділ 32. США'2003

32.1 Завдання першого туру

32.1.1 «Вибір доріг»

Алгоритм 1 На початку кожного тижня знаходити мінімальне остовне дерево (МОД), розглядаючи при цьому тільки стежки що входили в попереднє дерево та нову стежку. Складність алгоритму $O(NY \log N)$.

Алгоритм 2 Використовується справжне зростаюче мінімальне остовне дерево. Кожен тиждень необхідно знаходити шлях між кінцями нової стежки у остовному дереві та знаходити стежку максимальної довжини у цьому шляху. Якщо довжина максимальної стежки більше ніж довжина нової — видалити максимальну стежку і додати нову. Складність алгоритму $O(NY)$.

Алгоритм 3 Необхідно переобчислювати МОД на початку кожного тижня, розглядаючи тільки найкорочші стежки між парами полів.

Складність алгоритму $O(YY \log N)$.

Алгоритм 4 Необхідно переобчислювати МОД на початку кожного тижня, розглядаючи всі відомі стежки.

Складність алгоритму $O(YY \log Y)$.

32.1.2 «Reverse»

Алгоритм 1 Співставимо кожному з перших 8-ми регістрів відповідні біти числа 256. Встановимо 9-й регістр рівним 0. Мета кожного регістра зберігати наступне число, яке буде виведене, і в якому останній не нульовий біт, це біт співставлений з цим регістром. Наприклад, стартовими значеннями є: 128, 192, 224, 240, 248, 252, 254, 255 і 0. Як тільки значення 248 буде виведене, метою регістра буде число 232. Виберемо обмеження на кількість послідовних S-операцій. Після кожної P-операції виконаємо цю кількість S-операцій, роблячи регістри ближче до їхніх цільових значень. Для кожної S-операції, для регістрів, які ще не дорівнюють своїм цільовим значенням, виберемо максимальну ціль і будемо використовувати S-операції для того щоб до неї наблизитись. Якщо в якийсь момент виконання програми ціль не досягнута коли вона потрібна, необхідно збільшити кількість S-операцій та почати спочатку.

Алгоритм 2 Розглянемо випадок коли кожне число можна отримати тільки однією S-операцією. Регістр 1 може бути ініціалізований N, Регістр 2 може бути $N-2$. Після друку N одна S-операція може записати в регістр 1 значення $N-1$. Регістр 3 може бути $N-5$. Після друку $N-1$ одна S-операція може записати в регістр 1 значення $N-3$, а після друку $N-2$ S 1 2 записує в регістр 2 значення $N-3$, наступне значення для виводу. 44 — це максимальне значення N для якого достатньо обмеження в 1 операцію.

Алгоритм 3 Виконувати алгоритм 1, але мати фіксовану межу в 5 операцій.

Алгоритм 4 Ініціалізувати перший регістр в 0, наступні сім в $[kN/8]$. Залишити останній для роботи, але проініціалізувати його в N. На кожному кроці, вибрати регістр, що найближчий знизу до необхідного значення, та за допомогою S-операції, якщо не рівний, записати значення в робочий регістр. Продовжувати використовувати S-операції до отримання необхідного значення.

32.1.3 «Порівняння кодів»

Задачу можна розбити на дві частини: визначення чи існує відповідність між деякою частиною однієї програми та деякою частиною другої програми (внутрішній алгоритм), та знаходження найкращої такої відповідності (зовнішній алгоритм). Ефективність зовнішнього алгоритму залежить від функцій, що забезпечує внутрішній.

Якщо є два набори рядків, можна достатньо ефективно з'ясувати, чи є відповідність між цими наборами. Кожен рядок це рівняння. Рівняння, що відповідає рівнянню — це рівняння з відповідного рядка другої програми. Для кожної змінної, що згадується в програмі, розглянемо всі рядки де вона зустрічається. Якщо вона зустрічається зліва, то ця змінна має бути поставлена у відповідність змінній, що стоїть у лівій частині відповідного рівняння з другої програми. Якщо вона зустрічається двічі у правій частині одного рівняння, то вона має відповідати змінній у правій частині відповідного рівняння іншої програми (і там має бути одна змінна). Якщо змінна зустрічається тільки у правих частинах рівнянь програми, то відповідні праві частини для усіх входжень цієї змінної мають містити спільну змінну. Зрозуміло, що якщо змінна має відповідати декільком змінним - то відповідності не існує. Змінні з обох програм мають бути протестовані.

Всі ці умови є очевидно необхідними, але треба переконатися що вони є достатніми. Припустимо, не втрачаючи загальності, що $R \leq H$.

Алгоритм 1 Для кожного зсуву від $-R$ до R, почнемо з першої пари рядків які зсунуті на задане значання. Будемо намагатись додавати пари рядків в кінець кожної з підпрограм. Якщо пара рядків не може бути доданою, видалимо пар рядків на початку підпрограм до тих пір, поки пару рядків не можна бути додати або коли підпрограми не стануть порожніми. Цей алгоритм має оцінку складності $O(RH)$, допускаючи що умови додавання можуть бути перевірені за час $O(1)$.

Це найкращий алгоритм.

Алгоритм 2 Будемо починати з кожної пари рядків програм, та намагатись додавати пари рядків поки не виникне конфлікт. Цей алгоритм має оцінку складності $O(R^2H)$.

Алгоритм 3 Для кожної пари початкових розташувань будемо розглядати кожну можливу відповідність програм, що включає задану пару рядків, шукаючи оптимальну довжину бінарним пошуком. Цей алгоритм має оцінку складності $O(R^2H \log R)$, допускаючи що існує лінійний алгоритм перевірки існування відповідності між частинами програм.

32.2 Завдання другого туру

32.2.1 «Вгадайте корову»

Алгоритм 1 Здійснюється пошук у ширину на множині питань, у якому кожний стан являє собою множину можливих корів, що залишилися.

Може здаватися, що цей простір складається з 2^{50} можливих станів, але це не так. Стан може розглядатися як множина можливих значень для кожної ознаки. Усього $2^3 - 1$ значень для множини можливих значень для кожної ознаки, та в цілому менше ніж 2^{24} станів у сукупності (більш схоже на $2^{22.5}$ станів).

Виконання цього алгоритм потребує порядку $O(NP \cdot 2^3 \cdot 2^{3P})$ часу.

Алгоритм 2 На кожному кроці обирається питання яке розбиває корів на найбільш рівні частини.

Цей алгоритм потребує порядку $O(NP \cdot 2^3)$ часу.

Алгоритм 3 Задається два питання про кожну ознаку для того, щоб отримати її точне значення. Алгоритм зупиняється, коли корова відома. Враховується те, що іноді після першого питання можна встановити точне значення ознаки.

Алгоритм 4 Задається два питання про кожну ознаку для того, щоб отримати її точне значення.

32.2.2 «Роботи в лабіринті»

Зауважимо, що охоронці одночасно повертаються на своїх початкові позиції кожні 12 хвилин.

Для спрощення аналізу, будемо вважати, що лабіринти мають однаковий розмір. Нехай n це кількість квадратів в лабіринті ($R \times C$).

Алгоритм 1 Здійснюється пошук у ширину, де стан пошуку це позиція двох роботів та поточна хвилина з початку 12-ти хвилинного циклу охоронців. Відвідані стани запам'ятовується для того, щоб запобігти їх повторного відвідання.

Цей алгоритм потребує порядку $O(12n^2)$ часу.

Алгоритм 2 Здійснюється пошук у ширину, де стан пошуку це позиція двох роботів та поточна хвилина. Відвідані стани запам'ятовується (у хеш-таблиці) для того, щоб запобігти їх повторного відвідання.

Цей алгоритм потребує порядку $O(12T \cdot n^2)$ часу, де T — це мінімальний час, за який обидва роботи виберуться з своїх лабіринтів.

Алгоритм 3 Здійснюється пошук у ширину, де стан пошуку це позиція двох роботів та поточна хвилина. Повторне відвідання стану допустиме.

Цей алгоритм потребує порядку $O(4^T)$ часу.

Алгоритм 4 Здійснюється пошук у ширину, де стан пошуку це позиція двох роботів та мінімальний час, необхідний для досягнення цього стану. Кожен стан відвідується лише один раз.

Цей алгоритм потребує порядку $O(n^2)$ часу, але він не завжди дає вірну відповідь. Він лише розглядає найкоротші шляхи між кожною парою позицій. Іноді необхідно добратися до деякої позиції пізніше, ніж це можливо, так як не можна проводити час на відкритому просторі.

32.2.3 «Паркан»

Зауважимо, що існує багато способів зробити помилку у обчислювальній геометрії. Також необхідно бути вкрай обережним для того, щоб запобігти помилок округлення при роботі з числами з плаваючою точкою. Розв'язки, що роблять подібні помилки, не наведено нижче.

Нехай, d це максимальне число вершин у камені.

Алгоритм 1 Розглянемо паркан як циклічний список. Кожен камінь закриває від Фермера Дона діапазон стовпів паркана. Знайти цей діапазон для кожного каміння можна за допомогою бінарного пошуку, знайшовши найменший та найбільший кут, що утворюють вершини камінь та позиція Фермера. Потім ці діапазони необхідно відсортувати, та за допомогою послідовного перегляду знайти діапазони стовпів паркана, які не перекриваються жодним камінням. Цей алгоритм вимагає $O(R \log R + R \log d)$ часу.

Алгоритм 2 Зберігається двійковий масив стовпів паркана. Для кожного каміння визначається діапазон що перекривається цим камінням, та встановлюються відповідна інформація. Цей алгоритм вимагає порядку $O(RN)$ часу.

Алгоритм 3 Для кожного стовпа та каміння проводиться промінь, щоб визначити чи перекриває цей камінь стовп паркана. Цей алгоритм вимагає порядку $O(RN)$ часу.

Розділ 33. Греція'2004

33.1 Завдання першого туру

33.1.1 «Гермес»

Опишемо алгоритм, що потребує квадратичного часу. Нехай координати точок — $(x_0, y_0), \ldots, (x_n, y_n)$. Точка $(x_0, y_0) = (0, 0)$ — початкова. Будемо рекурентно обчислювати значення $A_{i,j}$ та $B_{i,j}$. $A_{i,j}$ — це мінімальна довжина шляху за яку можна обійти i точок та зупинитися у (x_i, y_j), а $B_{i,j}$ —

це мінімальна довжина шляху за яку можна обійти i точок та зупинитися у (x_j, y_i). Маємо:

$$\begin{aligned} A_{i+1,j} &= \min\{A_{i,j} + d_{x_i,x_{i+1}}, B_{i,i+1} + d_{y_i,y_j}\} \\ B_{i+1,j} &= \min\{B_{i,j} + d_{y_i,y_{i+1}}, A_{i,i+1} + d_{x_i,x_j}\} \end{aligned}$$

Відповіддю буде: $\min_j\{A_{n,j}, B_{j,n}\}$, а складність алгоритму можна оцінити як $O(n^2)$.

33.1.2 «Артеміда»

Нехай $f(x, y)$ — це кількість дерев, що знаходяться "нижче"та "лівіше"за точку (x, y). Помітимо, що в такому випадку кількість дерев між у прямокутнику з кутами t_1 та t_2 можна обчислити за формулою: $f(t_1.x, t_1.y) + f(t_2.x, t_2.y) - (t_1.x, t_2.y) - (t_2.x, t_1.y) + 1$ якщо t_1 знаходиться "нижче"та "лівіше"за t_2, і за подібною формулою в іншому випадку. Розглянемо декілька підходів до розв'язання.

1. Тривіальний алгоритм. Перегляд усіх можливих прямокутників, та підрахунок дерев, що лежать всередині кожного з них. Оцінка складності $O(n^3)$.

2. Динамічне програмування для обчислення $(t_1.x, t_2.y)$ для всіх t_1 та t_2.Після цього обробляємо всі прямокутники з використанням формули. Оцінка складності $O(n^2)$, але з використанням порядку $O(n^2)$ пам'яті.

3. Створити зовнішній цикл t по деревах, як кутів шуканої області. Для того щоб обробити прямокутник з кутом у t необхідні тільки значення $f(t.x, *)$ та $f(*, t.y)$. Вони можуть бути обраховані за допомогою алгоритму динамічного програмування (як і у другому алгоритмі). У цьому випадку складність алгоритму складатиме $O(n^2)$, але буде використано лінійний об'єм пам'яті.

4. Відсортувати дерева зліва направо, а потім обробляти їх у цьому порядку. Коли обробляється нове дерево t_n, воно додається до списку дерев, що відсортований вертикально. Маючи цю інформацію можна підрахувати $f(t.x, t_n.y)$ та $f(t_n.x, t.y)$ для кожного дерева t, яке лівіше за t_n за лінійний час. Після цього можна обчислити кількість дерев у всіх прямокутниках з кутом у t_n - подібно до алгоритму 3.

33.1.3 «Багатокутник»

Хоча обчислити суму Мінковського за двома багатокутниками нескладно, визначення чи є певний багатокутник сумою Мінковського двох інших багатокутників — NP-повна задача. Однак для неї існує псевдо-поліноміальний алгоритм, який частково описано нижче.

Позначимо вихідний багатокутник як P. Вважатимемо що він складається з N вершин з невід'ємними координатами вигляду $v_i = (x_i, y_i)$, $0 \le i \le N-1$, що перелічені у напрямку проти часової стрілки.

Означення. Будемо називати вектор $v = (a, b)$ примітивним, якщо НСД$(a, b) = 1$, де a та b — цілі невід'ємні числа. Або, що еквівалентно, v — примітивний тоді і тільки тоді, коли він не містить цілих точок всередині.

Ребра P представлені векторами $E_i = v_i - v_{i-1} = (a_i, b_i)$, $1 \le i \le N$, де a_i, b_i — цілі, та усі індекси беруться за модулем N. Замість ребра (a_i, b_i), якщо n_i =НСД(a_i, b_i), ми розглядаємо відповідне примітивне ребро $e_i = (a_i/n_i, b_i/n_i)$. Назвемо послідовністю векторів $\{E_i\}_{1\le i\le m} = \{n_i e_i\}_{1\le i\le m}$ послідовність ребер багатокутника, де e_i — примітивний вектор.

Наступне спостереження є ключовим в обчислені доданків A, B таких, що $P = A + B$.

Твердження. Кожне примітивне ребро P повинно бути ребром або A або B. Якщо ребро не примітивне, це ребро може також бути сумою Мінковського паралельних ребер з A та з B.

Тепер можна зробити висновок, що багатокутник A є доданком P, тоді і тільки тоді, коли послідовність його ребер має вигляд $\{k_j e_j\}_{j\in J}$, де $J \subseteq \{1, \ldots, N\}$, $0 \le k_j \le n_j$, k_j — ціле, $\sum_{j\in J} k_j e_j = (0, 0)$, тобто сума векторів, що представляють його ребра рівна нулю.

Нижче описані алгоритми пошуку доданків. При обчисленні багатокутників-доданків можуть бути отримані від'ємні координати. В цьому випадку ми повинні здвинути обидва доданки, щоб позбутися від'ємних координат. Це можна робити, тому що ніякий здвиг не змінює багатокутника.

Пошук відрізка як доданка

Багатокутник має відрізок-доданок тоді і тільки тоді, коли як мінімум одна пара векторів його ребер має нульову суму. Базуючись на цьому можна навести три алгоритми, за зростанням швидкості.

Описані алгоритми починають з пошуку примітивних векторів для кожного з ребер, виконуючи обчислення НСД. Для багатокутників, що складаються лише з примітивних ребер цей крок непотрібний.

Найпростіший алгоритм Спочатку обчислимо ребра багатокутника. Кожне ребро має вигляд:

$$E_i = (x_{i+1} - x_i, y_{i+1} - y_i) = (a_i, b_i)$$

де індекси беруться за модулем N. Далі ми обчислюємо всі НСД вигляду НСД(ai, bi), та формуємо послідовність ребер, як це було описано вище. Ці обчислення можуть бути проведені за час $O(mN)$, де $m = \max n_i = max\,\text{НСД}(a_i, b_i)$.

Для кожного вектора з цієї послідовності ми обчислюємо його суму з кожним з інших. Це можна зробити за час $O(m^2N^2)$.

Покращений алгоритм Обчислюємо послідовність ребер як раніше за час $O(mN)$. Впорядковуємо послідовність за x-координатою за час $O(mN\log(mN))$. Для кожного вектора з послідовності шукаємо на протилежний до нього за час $O(\log(mN))$. Загальна складність алгоритму $O(mN\log(mN))$.

Найшвидший алгоритм Обчислюємо послідовність ребер за час $O(mN)$. Додаємо ребра в хеш-таблицю, використовуючи їхню x-координату як ключ. Додавання виконується за $O(1)$.

Для кожного ребра у послідовності шукаємо у хеш-таблиці на інше ребро із зворотною x-координатою, що при додаванні дає нуль. Пошук виконується за час $O(1)$. Загальна складність алгоритму $O(mN)$.

Пошук трикутника як доданка

Найпростіший алгоритм Обчислюємо послідовність ребер як раніше за час $O(mN)$. Формуємо усі можливі трійки, та перевіряємо чи дають вони в сумі нуль за час $O(m^3N^3)$.

Покращений алгоритм Обчислюємо послідовність ребер як раніше за час $O(mN)$. Впорядковуємо послідовність за x-координатою за час $O(mN\log(mN))$. Формуємо всі можливі суми з двох векторів за час $O(m^2N^2)$. Для кожної такої суми за допомогою двійкового пошуку шукаємо на інший вектор який в сумі з обраними двома дає нуль. Загальна складність — $O(m^2N^2\log(mN))$.

Найшвидший алгоритм Обчислюємо послідовність ребер за час $O(mN)$. Можемо впорядкувати послідовність за x-координатою за час $O(mN\log(mN))$. Для кожного ребра k з послідовності, продивляємося послідовність зліва щоб знайти i, та справа, щоб знайти j, такі, що $E_k + E_i + E_j = 0$. Оцінка часу цього алгоритму складає $O(m^2N^2)$.

Пошук чотирикутника як доданка

Найпростіший алгоритм Обчислюємо послідовність ребер як раніше за час $O(mN)$. Формуємо усі можливі четвірки, та перевіряємо чи дають

вони в сумі нуль за час $O(m^4N^4)$.

Покращений алгоритм Обчислюємо послідовність ребер як раніше за час $O(mN)$. Формуємо всі можливі суми з трьох векторів за час $O(m^3N^3)$. Для кожної трійки шукаємо на інший вектор, який в сумі з нею дає нуль. Якщо зберегти ребра у хеш-таблиці, використавши їхню x-координату у якості ключа, можна шукати за час $O(1)$, тобто загальна складність складе $O(m^3N^3)$.

Найшвидший алгоритм Обчислюємо послідовність ребер за час $O(mN)$. Формуємо всі можливі суми з двох векторів за час $O(m^2N^2)$, та зберігаємо їх у хеш-таблиці використавши їхню x-координату у якості ключа. Для кожної такої суми шукаємо у хеш-таблиці на вектор, який в сумі з поточним дає нуль. Оцінка часу цього алгоритму складає $O(m^2N^2)$.

33.1.4 «Фідій»

Для розв'язання застосуємо принцип динамічного програмування. Нехай $a_{x,y}$ — не використана площа для прямокутника (x, y), $1 \le x \le W$, $1 \le y \le H$. З початку надамо $a_{x,y} = xy$, для кожного (x, y), за виключенням тих, що відповідають потрібним плиткам. Тобто, для кожного $a_{x,y}$ де $x = w_i$ та $y = h_i$ $(1 \le i \le N)$ $a_{x,y} = 0$. Для плитки (x, y) розглянемо всі вертикальні розрізи $c = 1, 2, \ldots, x-1$, та всі горизонтальні розрізи $c = 1, 2, \ldots, y-1$ та обираємо розріз, завдяки якому залишається найменша площа плитки, що не використовується: $a_{x,y} = a_{c,y} + a_{x-c,y}$ або $a_{x,c} + a_{x,y-c}$ для деякого c.

33.1.5 «Фермер»

Нехай g_i - це поле або смуга. Позначимо як n_i кількість кипарисів в полі або смузі. Якщо ми позначимо як e_i кількість оливкових дерев в g_i, матимемо $e_i = n_i$ якщо g_i це поле, та $e_i = n_i - 1$, якщо g_i — смуга.

Розглянемо наступну задачу (відому під назвою "Рюкзак"). Знайти $\max \sum_{i=1}^{n+m} e_i x_i$ де $\sum_{i=1}^{n+m} n_i x_i \le Q$ та $x_i \in \{0, 1\}$, де n, m — кількість полів та смуг відповідно.

Оптимальний розв'язок x_i^*, $1 \le i \le n+m$ складається з підмножини g_i такої, що сумарна кількість кипарисів у ній як максимум Q, і загальна кількість оливкових дерев максимальна. Однак, у загальному випадку $Q\prime = \sum_{i=1}^{n+m} n_i x_i < Q$. Якщо це так, то тоді існує g_i таке, що $x_i^* = 0$ та $n_i > Q - Q\prime$, в іншому випадку оптимум може бути покращений при додаванні до розв'язку g_i (що суперечить його оптимальності). Таким чином, додавання ланцюга з $Q - Q\prime$ кипарисів з g_i, та $Q - Q\prime - 1$ оливкових дерев до оптимального розв'язку задачі "Рюкзак", описаної вище, призводить до

розв'язку вихідної задачі. Задача "Рюкзак"може бути розв'язана за час порядку $O((n+m)Q)$ за допомогою алгоритму динамічного програмування.

33.1.6 «Емподіо»

Ця задача виникла при дослідженні "відстані"між геномами рослин на межі наук інформатики та біології. Було досліджено багато методів розв'язання цієї задачі, та найшвидший відомий з них потребує лише лінійного часу. Читачів, яким цікаво вивчити цю проблему, відсилаємо до відповідних наукових статей, які можна знайти у мережі Інтернет:

- Bader, Moret, and Yan. A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study. In Lecture Notes in Computer Science, editor, WADS: 7th Workshop on Algorithms and Data Structures, volume 2125, pages 365-376, 2001.
- Piotr Berman and Sridhar Hannenhalli. Fast Sorting by Reversal. Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching, pages 168-185, 1996.
- Sridhar Hannenhalli and Pavel Pevzner. Transforming Cabbage into Turnip. Proceedings of the 27th Annual Symposium on Theory of Computing (STOC95), pages 178-189, 1995.

Розділ 34. Польща'2005

34.1 Завдання першого туру

34.1.1 «Садок»

Назвемо прямокутну область, що містить рівно k троянд k-пря-мо-кутником. Також позначимо $r_{x,y}$ кількість троянд в прямокутнику (x,y). Задачею є знайти два k-прямокутника, що не перетинаються, з мінімальною сумою периметрів.

Простий розв'язок

Найпростішим розв'язком є розглянути всі можливі прямокутні області в садочку та для кожного підрахувати кількість троянд всередині. У такий спосіб можна перебрати всі k-прямокутники за час $O(w^3 \cdot l^3)$. Всього може бути до $O(w^2 \cdot l^2)$ k-прямокутників. Наступним шагом буде перебрати всі

пари k-прямокутників та вибрати ту, що складається з прямокутників з мінімальною сумою периметрів. Такий розв'язок отримає біля 50% балів, не зважаючи на його жахливу складність $O(w^4 \cdot l^4)$.

Гарний розв'язок

Розглянемо декілька поступових покращень, що приведуть до гарного розв'язку. Ми можемо оптимізувати перевірку, чи є прямокутна область k-прямокутником. Позначимо $R_{x,y}$ кількість троянд в області з протилежними кутами в кутах $(1,1)$ та (x,y). Ми можемо попередньо обчислити $R_{x,y}$ за час $O(w,l)$ використовуючи таку формулу:

$$R_{x,y} = \begin{cases} 0, & \text{якщо } x = 0 \text{ або } y = 0; \\ R_{x-1,y} + R_{x,y-1} + R_{x-1,y-1} + r(x,y), & \text{інакше.} \end{cases}$$

Маючи це, ми можемо виразити $\mathcal{R}(x_1, y_1, x_2, y_2)$ – кількість троянд в прямокутному регіоні з кутами (x_1, y_1) та (x_2, y_2) як:

$$\mathcal{R}(x_1, y_1, x_2, y_2) = R_{x_2,y_2} - R_{x_2,y_1-1} - R_{x_1-1,y_2} + R_{x_1-1,y_1-1}$$

Таким чином $\mathcal{R}(x_1, y_1, x_2, y_2)$ може бути обчислене за час $O(1)$. Всі k-прямокутники можна перерахувати за час $O(w^2 \cdot l^2)$. Нажаль, це не розв'язує проблему перебору всіх пар k-прямокутників.

На щастя, існує інший метод боротьби з цими проблемами. Зрозуміло, що для двох прямокутників, що не перетинаються, має існувати горизонтальна або вертикальна лінія, такі що один прямокутник знаходиться з одного боку, а другий - з іншого. Для кожної горизонтальної лінії можна знайти два k-прямокутника з мінімальними периметрами, що лежать з протилежних боків лінії, аналогічно - для вертикальних ліній. Коли це зроблено – результат можна обчислити за час O(w+l) розглянувши всі лінії та вибравши оптимальну суму периметрів.

Розглянемо метод пошуку k-прямокутнику з оптимальним периметром над горизонтальною лінією, інші випадки розглядаються аналогічно. Позначимо A_y мінімальний периметр k-прямокутника, що лежить над горизонтальною лінією з координатою y, чия нижня координата більше або дорівнює y. Позначимо a_y мінімальний периметр k-прямокутника, з нижньою координатою a_y. Помітимо, що:

$$A_y = \min(a_y, \ldots, a_w)$$

Простий метод підрахування a_i - встановити їх на початку на нескінченність, після цього модифікувати перебираючи всі k-пря-мо-кут-ники. Покращений алгоритм працює за час $O(w^2, l^2)$.

І це ще не все. Можна помітити, що не потрібно знаходити всі k-прямокутники. Ми можемо розглядати тільки ті k-прямокутники, що не містять в собі інші k-прямокутники. Щоб перебрати всі k-прямокутники, що перетинаються, розглянемо всі пари (y_1, y_2), $1 \le y_1 \le y_2 \le w$. Для кожної такої пари будемо використовувати плаваюче вікно, що є прямокутником з кутами (x_1, y_1) та x_2, y_2. На початку $x_1 = x_2 = 1$. Ми будемо рухати вікно відповідно до наступних правил, до $x_2 > l$:

- якщо є рівно k троянд у вікні (тобто $\mathcal{R}(x_1, y_1, x_2, y_2) = k$, то ми знайшли новий k-прямокутник. Після оновлення чотирьох послідовностей (a_i та трьох інших), x_1 збільшується на одиницю;
- Якщо $\mathcal{R}(x_1, y_1, x_2, y_2) < k$ то x_2 збільшується на одиницю, щоб розширити вікно;
- Якщо $\mathcal{R}(x_1, y_1, x_2, y_2) > k$ то x_1 збільшується на одиницю, щоб звузити вікно.

Наведений алгоритм працює за час $O(w^2 \cdot l)$, та перераховує (серед інших) всі цікаві k-прямокутники.

34.1.2 «Послідовність середніх»

На початку, позначимо I кількість вхідних рядків, що починаються з літери `I`. Аналогічно, нехай Q будуть кількістю запитів.

Простий розв'язок

Можливим простим, але не оптимальним розв'язком буде подати стежку як вектор A, де $A[i]$ позначає підйом після i-ї ланки. Складність обробки зміни конфігурації (назвемо цю операцію вставкою) є $O(n)$. Оскільки запит також потребує $O(n)$, загальна складність буде $O((I + Q) \cdot n)$. Тим більш, складність за пам'яттю є $O(n)$, що занадто багато щоб уміститися у пам'ять для тестових випадків великого розміру. Іншим простим підходом є подати трек як сортований список інтервалів, що не перетинаються, так що по кожному інтервалу різниця в висоті між усіма ланками є сталою. У цьому випадку ми маємо складність вставки та запиту $O(I)$, таким чином весь розв'язок буде мати складність $O((I + Q) * I)$. Вартість по пам'яті - $O(I)$.

Гарний розв'язок

Структурою даних, що використовується в розв'язку є бінарне дерево. Кожен з його вузлів описує інтервал (тобто деяку кількість послідовних ла-

нок), $J = [2^k t, 2^k(t+1))$, для деяких цілих k та t. Інформація, що міститься у вузлі є

- $S_J = \sum_{i \in J} d_i$
- $H_J = \max\left(\{0\} \cup \left\{\sum_{2^k \le i \le j} d_i : j \in J\right\}\right)$

Вузол є листком, коли всі значення d_i є рівними. В цьому випадку обчислення S_J та H_J є тривіальним. Інакше, вузол має двох синів, що відповідають інтервалам $J_1 = [2^{k-1}(2t), 2^{k-1}(2t+1))$ та $J_2 = [2^{k-1}(2t+1), 2^{k-1}(2t+2))$. В цьому випадку S_J та H_J обчислюються як $S_J = S_{J_1}$ та $H_J = \max\{H_{J_1}, S_{J_1} + H_{J_2}$. Корінь дерева описує інтервал $[0, 2^{\lceil \log_2 n \rceil})$.

Найбільш важливою операцією є операція вставки. Вона вимагає вставки або модифікації щонайбільше $2\lceil \log_2 n \rceil$ вузлів в дерево. Аналогічно, обробка запита потребує перегляду щонайбільше $\lceil \log_2 n \rceil$ вузлів. Таким чином розв'язок має складність $O((I+Q) \cdot \log n$ та складність по пам'яті $O(I \cdot \log n)$.

Ще кращій розв'язок

Оскільки не потрібно обробляти рядки з файлу один за одним, можна прочитати всі вхідні дані і знати, які частини треку будуть взагалі конфігуруватись. Нехай M буде сортованим вектором кінців інтервалів, що зустрічаються в I-записах на вході. Ми також можемо припустити що довжина M є ступенем двійки.

Тепер використаємо дерево, що дуже схоже на описане вище. Різниця в тому, кожен вузол відповідає інтервалу $I = [M[2^k t], M[2^k(t+1)])$. Його можна зберігати у векторі, як в стандартній реалізації кучі. Складність буде $O((I+Q) \cdot \log I)$ та складність по пам'яті знизиться до $O(I+Q)$.

34.1.3 «Гори»

На початку, позначимо I кількість вхідних рядків, що починаються з літери `I`. Аналогічно, нехай Q будуть кількістю запитів.

Простий розв'язок

Можливим простим, але не оптимальним розв'язком буде подати стежку як вектор A, де $A[i]$ позначає підйом після i-ї ланки. Складність обробки зміни конфігурації (назвемо цю операцію вставкою) є $O(n)$. Оскільки запит також потребує $O(n)$, загальна складність буде $O((I+Q) \cdot n)$. Тим більш, складність за пам'яттю є $O(n)$, що занадто багато щоб уміститися у пам'ять

для тестових випадків великого розміру. Іншим простим підходом є подати трек як сортований список інтервалів, що не перетинаються, так що по кожному інтервалу різниця в висоті між усіма ланками є сталою. У цьому випадку ми маємо складність вставки та запиту $O(I)$, таким чином весь розв'язок буде мати складність $O((I+Q)*I)$. Вартість по пам'яті - $O(I)$.

Гарний розв'язок

Структурою даних, що використовується в розв'язку є бінарне дерево. Кожен з його вузлів описує інтервал (тобто деяку кількість послідовних ланок), $J = [2^k t, 2^k(t+1))$, для деяких цілих k та t. Інформація, що міститься у вузлі є

- $S_J = \sum_{i \in J} d_i$
- $H_J = \max\left(\{0\} \cup \left\{\sum_{2^k \le i \le j} d_i : j \in J\right\}\right)$

Вузол є листком, коли всі значення d_i є рівними. В цьому випадку обчислення S_J та H_J є тривіальним. Інакше, вузол має двох синів, що відповідають інтервалам $J_1 = [2^{k-1}(2t), 2^{k-1}(2t+1))$ та $J_2 = [2^{k-1}(2t+1), 2^{k-1}(2t+2))$. В цьому випадку S_J та H_J обчислюються як $S_J = S_{J_1}$ та $H_J = \max\{H_{J_1}, S_{J_1} + H_{J_2}$. Корінь дерева описує інтервал $[0, 2^{\lceil \log_2 n \rceil})$.

Найбільш важливою операцією є операція вставки. Вона вимагає вставки або модифікації щонайбільше $2\lceil \log_2 n \rceil$ вузлів в дерево. Аналогічно, обробка запита потребує перегляду щонайбільше $\lceil \log_2 n \rceil$ вузлів. Таким чином розв'язок має складність $O((I+Q) \cdot \log n$ та складність по пам'яті $O(I \cdot \log n)$.

Ще кращій розв'язок

Оскільки не потрібно обробляти рядки з файлу один за одним, можна прочитати всі вхідні дані і знати, які частини треку будуть взагалі конфігуруватись. Нехай M буде сортованим вектором кінців інтервалів, що зустрічаються в I-записах на вході. Ми також можемо припустити що довжина M є ступенем двійки.

Тепер використаємо дерево, що дуже схоже на описане вище. Різниця в тому, кожен вузол відповідає інтервалу $I = [M[2^k t], M[2^k(t+1)])$. Його можна зберігати у векторі, як в стандартній реалізації кучі. Складність буде $O((I+Q) \cdot \log I)$ та складність по пам'яті знизиться до $O(I+Q)$.

34.2 Завдання другого туру

34.2.1 «День народження»

Задачею є знайти таке впорядкування дітей, що максимальна кількість стільців, на яку треба перемістити кожного з дітей є мінімальною. По-перше, помітимо що є два класи впорядкування. Наприклад, якщо є перестановка (1,2,3), то дитина 1 може бути сусідом 2 ліворуч або праворуч. Відповідно впорядкування буде проти годинникової стрілки чи за годинниковою стрілкою. В обох випадках підрахунки однакові, розглянемо тільки випадок впорядкування за годинниковою стрілкою.

Простий розв'язок

Першою ідеєю може бути генерація всіх можливих перестановок. Зафіксуємо першу дитину. Тепер, маючи перестановку, ми можемо підрахувати фінальну позицію (і відстані при переміщенні) всіх дітей за час $O(n)$, а оскільки маємо повторити цей крок для всіх позицій - складність алгоритму - $O(n^{@})$.

Оптимальний розв'язок

Існує кращій розв'язок. Позначимо (p_i) задану впорядкування дітей. Розглянемо фінальну перестановку, де дитина p_1 знаходиться на позиції f. Для того, щоб досягти такого впорядкування, деякі (а можливо всі) діти мають переміститися. Ми можемо позначити переміщення дитини i числом d_i^f, де $|d_i^f|$ є відстанню переміщення, воно є додатнім, коли дитина переміщується за годинниковою стрілкою, і від'ємним, коли проти. Крім того, ми припускаємо, що дитина завжди обирає такий напрямок переміщення, що відстань менша ніж протилежному напрямку (або рухається за годинниковою стрілкою, якщо відстані однакові), тобто $1 - \lceil \frac{n}{2} \rceil \le d_i^f \le \lfloor \frac{n}{2} \rfloor$.

Нехай $S_f = \{d_y^f : i = 1, 2, \ldots, n\}$. Ми можемо розглядати S_f як альтернативне подання розглянутого переміщення. Маючи це подання, ми можемо просто порахувати максимальну відстань переміщення використовуючи формулу $R_f = \max(-\min(S_f), \max(S_f))$. Коефіцієнти d_i^f залежать від заданої перестановки (p_i) та f. Вони можуть бути задані такою формулою:

$$d_{p_i}^f = \min(a, n-a) \text{де} a = (f + i - 1 - p_i) \mod n$$

Крім того, по заданому поданню S_f ми можемо просто побудувати $S_(f+1)$ зсуваючи елементи S_f (тобто ми замінюємо x на $x+1$ якщо $x < \lfloor \frac{n}{2} \rfloor$ та замінюємо $\lfloor \frac{n}{2} \rfloor$ на $1 - \lceil \frac{n}{2} \rceil$).

Для ми зацікавлені в обчисленні найменшого можливого значення для всіх f. Зауважимо, що всі подання S_f є зсувами одного базового подання, скажемо S_0. Позначимо C максимальну кількість (за модулем n) елементів з $\{1 - \lceil \frac{n}{2} \rceil, \ldots, \lfloor \frac{n}{2} \rfloor\}$, що не з'являються в S_0. Вона може бути підрахована за лінійний час.

Результатом буде $\lfloor \frac{n-C}{2} \rfloor$.

34.2.2 «Гра в прямокутник»

Ключем до розв'язання є пошук характеристики виграшних чи програшних позицій. Першим кроком у пошуку може бути малюнок, на якому позначена виграшність позицій.

```
      x-напрямок
                          1 1 1 1 1 1 1 1 1 2
      1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
y   1 * . * . . . * . . . . . . . * . . . . .
|   2 . * . . * . . . . . * . . . . . . . . .
н   3 * . * . . . * . . . . . . . * . . . . .
а   4 . . . * . . . . * . . . . . . . . * . .
п   5 . * . . * . . . . . * . . . . . . . . .
р   6 . . . . . * . . . . . . * . . . . . . .
я   7 * . * . . . * . . . . . . . * . . . . .
м   8 . . . . . . . * . . . . . . . . * . . .
о   9 . . . * . . . . * . . . . . . . . * . .
к  10 . . . . . . . . . * . . . . . . . . . .
   11 . * . . * . . . . . * . . . . . . . . .
   12 . . . . . . . . . . . * . . . . . . . .
   13 . . . . . * . . . . . . * . . . . . . .
   14 . . . . . . . . . . . . . * . . . . . .
   15 * . * . . . * . . . . . . . * . . . . .
   16 . . . . . . . . . . . . . . . * . . . .
   17 . . . . . . . * . . . . . . . . * . . .
   18 . . . . . . . . . . . . . . . . . * . .
   19 . . . * . . . . * . . . . . . . . . * .
   20 . . . . . . . . . . . . . . . . . . . *
```

Після короткого аналізу малюнку, можна помітити наступне: прямокутник $n \times m$ є програшною позицією, тоді і тільки тоді, коли існує таке k, що

$$m + 1 = 2^k \cdot (n + 1) \tag{34.1}$$

Доведемо це твердження індукцією по прямокутній області.

- Нехай n та m задовольняють умову для $k = 0$, тобто $n = m$. Ми можемо довести, що позиція є програшною, використовуючи просту індукцію по n.
  - Якщо $n = 1$, то позиція є програшною за визначенням гри.
  - Припустимо, що $n > 1$ та перший гравець своїм ходом робіть прямокутник $n \times l$, де $n/2 \le l < n$. Тоді опонент може зменшити прямокутник до $l \times l$, що є програшною позицією.

- Припустимо, що n та m задовольняють умову для $k \neq 0$. Без втрати загальності можна вважати, що $k > 0$, оскільки $m + 1 = 2^k \cdot (n + 1)$ еквівалентно $n + 1 = 2^{-k} \cdot (m + 1)$. Перший гравець може розрізати прямокутник у двох можливих напрямках:
  - Припустимо, що після першого розрізу ми маємо прямокутник $l \times m$, де $n/2 \le l < n$. Оскільки $n = 2^k(m + 1) - 1$, маємо $2^{k-1}(m + 1) - 1 < l < 2^k(m + 1) - 1$. Другий гравець може зменшити прямокутник до $2^{k-1}(m+1)-1 \times m$, що є програшним для першого гравця.
  - Припустимо, що після першого розрізу ми маємо прямокутник $n \times l$, де $m/2 \le l < m$. Ми маємо показати, що $n \neq 2^i(l + 1) - 1$ для всіх цілих i.

    Доведемо від супротивного – припустимо що $n = 2^i(l+1)-1$ для деякого i. Оскільки $l < m$, маємо $n = 2^i(l+1)-1 < 2^i(m+1)-1$, отже $i > k$.

    З іншого боку, оскільки $m/2 \le l$, ми маємо $n = 2^i(l + 1) - 1 \ge 2^i(m/2+1)-1 = 2^{i-1}(m+2)-1 > 2^{i-1}(m+1)-1$, звідки $i-1 < k$, і $i < k + 1$.

    Отримали протиріччя, відповідно $n \times l$ є виграшної позицією.

  Отже, $n \times m$ є програшною позицією.

- припустимо, що n та m не задовольняють умові. Тоді $\log_2\left(\frac{n+1}{m+1}\right)$ не є цілим. Без втрати загальності, можемо припустити що $n \ge m$. позначимо через l значення $2^{\left\lfloor \log_2\left(\frac{n+1}{m+1}\right)\right\rfloor}(m - 1) - 1$.

  Маємо $(n+1)/2 < l+1 < n+1$, звідки отримуємо $n/2 \le l < n$. Таким чином перший гравець може зменшити прямокутник до розмірів $l \times m$, що э програшною позицією для другого гравця.

Модельний розв'язок відповідає доведенню та для кожної виграшної позиції знаходить відповідний хід за логарифмічний час. Однак кількість кроків може бути лінійною. Наприклад, для прямокутника $2i \times 2(i+1)$ єдиним виграшним ходом буде зменшення до $2i \times 2i$, відповідно складність у найгіршому випадку $O(n \log n)$.

Альтернативні розв'язки

Пошук з поверненням, що рекурсивно перебирає всі ходи є простішим розв'язком, який мав набрати половину балів.

Інший розв'язок базується на динамічному програмуванні. Для кожної позиції (n, m) ми можемо порахувати чи є вона виграшною або програшною. Простий динамічний підхід перевіряє всі можливі ходи для заданої позиції (n, m). Цей розв'язок має складність $O(n^3)$.

Є більш швидкий динамічний підхід, який зберігає найбільше $n' < n$ та $m' < m$ для яких (n', m) та (n, m') є програшними позиціями. У такий спосіб ми можемо знайти оптимальний хід для заданої позиції за час $O(1)$, що приводить до складності алгоритму $O(n^2)$.

34.2.3 «Річки»

Не важко помітити, що оскільки для кожного села є тільки один шлях по річці до Байттауна, ми можемо розглядати річки та села як дерево з коренем у Байттауні. Вузли відповідають селам (будемо вважати Байттаун також селом), і вузол v є батьком вузла u коли v є першим селом вниз по річці від u.

Нехай r позначає корінь дерева, тобто r відповідає Байттауну. Як $depth(u)$ будемо позначати кількість ребер на шляху від u до r. Зрозуміло, що значення $depth(u)$ можуть бути обраховані для всіх сел u за лінійний час. Кількість синів вузла u позначимо $deg(u)$, а кількість дерев, що зрізано біля села u як $trees(u)$.

Динамічне програмування

Ми можемо застосувати динамічне програмування для розв'язання задачі. Нехай $A_{v,t,l}$ позначає мінімальну вартість транспортування дерев, зрізаних в піддереві з коренем у v, враховуючи що t додаткових лісопилок може бути побудовано в піддереві, та дерева, не оброблені в v можуть бути оброблені в селі глибини l. Ми обчислюємо $A_{v,t,l}$ для кожного села v та таких чисел t, l, що $0 \le t \le k$ та $0 \le l < depth(v)$. Зрозуміло, що коли дерево з коренем

у v має щонайбільше t вузлів, то $A_{v,t,l} = 0$, оскільки ми можемо поставити лісопилку у кожному селі. Можемо використовувати таку формулу:

$$A_{v,t,l} = \begin{cases} 0, & \text{коли дерево з коренем у } v \\ & \text{має щонайбільше } t \text{ вузлів,} \\ \min(A'_{v,t,l}, A''_{v,t}), & \text{інакше,} \end{cases} \tag{34.2}$$

де $A'_{v,t,l}$ є вартістю транспортування коли немає пилорами у v, та $A''_{v,t}$ є вартістю транспортування коли лісопилка там є. Ці вартості залежать від розташування лісопилок між піддеревами з коренями у синах v. Тоді:

$$A'_{v,t,l} = trees(v) \cdot (depth(v) - l) + \min_{t_1+\ldots+t_d=t} \sum_{i=1}^{d} A_{v_i,t_i,l}, \tag{34.3}$$

$$A''_{v,t} = \min_{t_1+\ldots+t_d=t-1} \sum_{i=1}^{d} A_{v_i,t_i,depth(v)}. \tag{34.4}$$

Ще одне динамічне програмування

Поглянемо уважніше на рекурентності (34.3) та (34.4). Нам не потрібно розглядати кожне розбиття t у суму $deg(v)$ доданків, щоб підрахувати A' та A''. Це буде додавати час у випадку коли дерева містять вершини з багатьма синами. Знову, ми можемо використати динамічне програмування. Нехай $B^{i,s}_{v,l}$ позначає вартість транспортування дерев зрізаних в піддеревах з коренями $v_1, v_2, \ldots, v_i$, враховуючи що s додаткових лісопилок буде збудовано в цих піддеревах і дерева не оброблені в цих піддеревах будуть оброблені в селі глибини l. Можемо використовувати наступну рекурентність:

$$\begin{aligned} &B^{0,s}_{v,l} = 0, \\ &B^{i,s}_{v,l} = \min_{0\le j\le s}(B^{i-1,s-j}_{v,l} + A_{v_i,j,l}) \quad \text{для кожного } s = 1, \ldots, k \end{aligned} \tag{34.5}$$

Визначимо аналогічно $C^{i,s}_v$, але в цей раз припустимо що дерева, не оброблені в піддеревах, будуть оброблені в v. Тоді

$$\begin{aligned} &C^{0,s}_v = 0, \\ &C^{i,s}_v = \min_{0\le j\le s}(C^{i-1,s-j}_v + A_{v_i,j,depth(v)}) \quad \text{для кожного } s = 1, \ldots, k \end{aligned} \tag{34.6}$$

Очевидно, $A'_{v,t,l} = B^{deg(v),t}_{v,l}$ та $A''_{v,t} = C^{deg(v),t-1}_v$. Для того, щоб порахувати значення $A_{v,t,l}$ (відповідно $A''_{v,t}$) для деякого $l \in 0, \ldots, depth(v)$, ми обчислюємо $B^{i,s}_{v,l}$ (відповідно $C^{i,s}_v$) для кожного $i = 0, \ldots, deg(v)$ та $s = 0, \ldots, k$.

Для кожної пари v, l потрібно $O(k^2(deg(v)+1))$ часу, загальний час буде $O(n\sum_v k^2(deg(v)+1)) = 0(k^2n^2)$, оскільки $\sum_v deg(v)$ дорівнює кількості ребер в дереві, тобто $n-1$. Після обчислення всіх значень $A_{v,t,l}$ програма поверне $A'_{r,k,0}$ як відповідь.

Забагато динамічного програмування

Використання двічі динамічного програмування непотрібне якщо вершини в дереві мають малу кількість синів. На щастя, ми можемо просто побудувати бінарне дерево сел, що має таку ж мінімальну вартість транспортування як і оригінальне. Для того щоб це зробити, ми під'єднаємо першого сина кожної вершини до батька як лівого сина, створимо додаткове село і під'єднаємо його як правого сина. Після цього під'єднаємо інших синів батьківської вершини оригінального дерева один за одним, кожного разу під'єднуючи сина як лівого сина додаткового села, створюючи нове додаткове село та під'єднуючи його як як правого сина попереднього села. Додаткові села не виробляють деревини, та річки, що під'єднують їх до батьків мають довжину 0, відповідно не впливають на транспортування. Побудова лісопилок у додаткових селах не дозволяє зменшити мінімальну вартість транспортування, тому що кожна лісопилка у додатковому селі може бути передвинута до наступного не додаткового села у напрямку Байттауна без збільшення (або зменшення) загальної вартості.

Існує $t+1$ розбиттів t на суму 2 невід'ємних доданки. Для кожної пари v, l ми можемо підрахувати $A_{v,t,l}$ для всіх t за час $O(k^2)$. В бінарному дереві буде в 2 рази більше вершин ніж у оригінальному, загальний час обчислення буде $O(n\sum_v k^2) = O(k^2n^2)$.