Завдання XXV Всеукраїнської олімпіади з інформатики

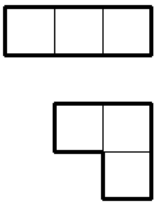
В. В. Бондаренко Р. С. Єдемський А. С. Коротков Д. П. Мисак
І. М. Порубльов Я. О. Твердохліб Ш. І. Ягіяєв

27 березня 2012 р.

Зміст

1 Завдання першого туру

1.1 «Триоміно» (Ілля Порубльов).



Фігуркою триоміно називають зв'язну фігуру, що складається з трьох квадратів. Існує лише два суттєво різних типи таких фігурок — вони зображені на рисунку. Всі інші відрізняються лише поворотами. Прямокутне поле розмірами M рядків та N стовпчиків вважається заповненим фігурками триоміно, коли:

- Кожна фігурка знаходиться у межах поля і накриває рівно три клітинки.
- Фігурки не накладаються.
- Порожніх (не покритих фігуркою) клітинок не більше ніж дві.

Заповнене фігурками поле можна подати у вигляді прямокутної таблиці. Значення 0 позначають не заповнену клітинку. Однакові натуральні значення позначають, що клітинки належать одній фігурці, різні — різним.

Завдання Напишіть програму **triomino**, яка, проаналізувавши вміст кількох числових двовимірних таблиць, визначить для кожної з них, чи задає вона правильно заповнене фігурками триоміно прямокутне поле.

Вхідні дані Перший рядок вхідного файлу **triomino.dat** містить єдине ціле число T ($2 \leqslant T \leqslant 10$) — кількість прямокутних таблиць. Далі йдуть T блоків такої структури. Перший рядок блоку містить два цілих числа M та N ($1 \leqslant M \leqslant 200$, $1 \leqslant N \leqslant 200$) — кількість рядків та кількість стовпчиків відповідної таблиці. Далі йдуть M рядків по N цілих чисел у кожному. Значення цих чисел від 0 до $[M \times N/3]$ включно. Розмір вхідного файлу не перевищуватиме 512 КБ.

Вихідні дані Вихідний файл **triomino.sol** повинен містити T рядків, у кожному — слово `YES` або слово `NO` (великими латинськими літерами) на позначення того, чи задає відповідна таблиця правильно заповнене фігурками триоміно поле.

Оцінювання Щонайменше у 30 % тестів не буде фігурок 2-го типу — будуть або «прямі» фігурки вигляду 1×3 чи 3×1, або взагалі не фігурки триоміно. Щонайменше у 40 % тестів обидва значення M та N кратні 3.

Приклад вхідних та вихідних даних

triomino.dat
2 4 8 1 2 2 2 6 7 7 9 1 4 4 3 6 7 8 9 1 0 4 3 6 8 8 9 10 10 10 3 0 5 5 5 2 6 1 1 2 2 1 1 0 1 0 2 0 1

triomino.sol
YES NO

Рисунок до першого блоку:

1	2	2	2	6	7	7	9
1	4	4	3	6	7	8	9
1		4	3	6	8	8	9
10	10	10	3		5	5	5

1.1.1 Розв'язання

якщо таблиця заповнена правильно, але не фыгурками триомино, то такий тест зарахуэться?

З питань учасників

Розглянемо два різні правильні та ефективні способи, кожен з яких працює за час, пропорційний розміру вхідних даних. Спосіб (Б) очевидніший (більш стандартний), але обсяг коду виявляється меншим у способі (А).

Спосіб (А) Легко переконатися, що масив відповідає правильному покриттю фігурками триоміно тоді й тільки тоді, коли виконуються такі дві вимоги:

1. Значень «`0`» є рівно $(N * M) \bmod 3$ штук, а кожного зі значень від «`1`» до `(M*N)div 3` — рівно по три штуки.
2. Для кожного зі значень від «`1`» до `(M*N)div 3`, зайняті ним клітинки утворюють фігурку триоміно.

Твердження 1. *Три різні клітинки, «координати» (пари індексів) яких подані в парах $(A.i, A.k)$, $(B.i, B.k)$ та $(C.i, C.k)$, утворюють фігурку триоміно тоді й тільки тоді, коли $\big(|A.i - B.i| + |A.k - B.k|\big) + \big(|A.i - C.i| + |A.k - C.k|\big) + \big(|B.i - C.i| + |B.k - C.k|\big) = 4$.*

Доведення. Для кожної із двох можливих фігурок триоміно дійсно маємо $1 + 1 + 2 = 4$.

Оскільки A, B та C — різні клітинки, то кожна з узятих в дужки сум двох модулів (вони називаються *манхетенськими відстанями* між клітинками) ціла додатна. Число 4 можна розкласти в суму трьох цілих додатних лише як $1 + 1 + 2$ (або $1 + 2 + 1$, або $2 + 1 + 1$). З іншого боку, манхетенська відстань дорівнює 1 тоді й тільки тоді, коли клітинки мають спільну сторону. Зразу дві одиниці для пар (A, B), (A, C) та (B, C) означають, що якась із клітинок має спільні сторони з обома іншими (а ті дві між собою — не мають). А це і є всі можливі фігурки триоміно, й лише вони. □

Зрозуміло, перед застосуванням розглянутого критерію (для кожного зі значень від «`1`» до `(M*N)div 3`) треба переконатися, що виконується перша вимога (щодо кількостей), а також сформувати самі трійки «координат». Це можна зробити, наприклад, так. Створимо масиви `num : array[0..(200*200)div 3] of integer` та `C : array[1..(200*200)div 3] of array[1..3] of record i, j : integer end`. В елементах `num` зберігатимемо кількості відповідних значень. Кожен елемент `C[v]` буде масивом, що міститиме перелік «координат» різних клітинок зі значенням `v`. Масиви `num` та `C` при бажанні можна формувати одразу при читанні вхідних даних (прочитали з файлу `v`, збільшили `num[v]`, якщо воно не перевищило 3, то дописали поточні «координати» до `C[v]`). Але тоді не забуваємо, що у вхідному файлі кілька тестових блоків, тому навіть якщо вже знаємо, що поточна відповідь «NO», дані блоку треба дочитати до кінця.

Спосіб (Б) Задача виділення фігурок у клітчатому полі (за допомогою, наприклад, пошуку в ширину або в глибину) досить стандартна, вказівки щодо її розв'язання можна знайти у багатьох джерелах. Вважаючи задачу виділення однієї фігурки відомою, дану задачу можна розв'язати так:

1. Завести допоміжний масив `[1..(200*200)div 3] of boolean`, смисл якого — «чи відбувався вже пошук фігурок, утворених даним числом».
2. Прочитати вхідні дані блоку в двовимірний масив.
3. Переглядати елементи двовимірного масиву рядок за рядком, і щоразу, знайшовши додатне число:
   (a) Якщо фігурку, утворену таким самим значенням, вже шукали — відповідь на поточний блок «NO», перевірку можна обривати.
   (b) Виділити фігурку, утворену поточним числом, замінюючи входження цього числа, наприклад, на протилежне (`mass[i][j]:=-mass[i][j]`), і рахуючи кількість клітинок у фігурці. Якщо кількість $\neq 3$ — відповідь на поточний блок «NO», перевірку можна припинити.
4. Якщо перевірка не була припинена, перевірити будь-яку одну з властивостей: або «кількість нулів у масиві від 0 до 2», або «для всіх значень від 1 до $(M*N)$div3 фігурка була виділена» — це дасть остаточну відповідь «YES» або «NO» для поточного блоку.

Технічне зауваження Вхідні дані досить великі за розміром, тому слід вибрати ефективний спосіб читання. Read мови Паскаль, scanf мови C та оператор `>>` класу `ifstream` мови C++ достатньо ефективні. А поєднання оператора `>>` об'єкта `cin` з `freopen`-ом — ні.

1.2 «Мутація» (Роман Єдемський).

Вчені планети Олімпія впритул наблизилися до відкриття характерного геному для виду олімпійських амеб, а саме: вони знайшли *послідовність генів*, про яку відомо, що вона містить рівно *один* зайвий ген. На поточному етапі вченим необхідно знайти усі гени, які можуть виявитися зайвими у цій послідовності. Для цього вчені використовують геном організму, який гарантовано є представником цього виду.

Послідовність генів, знайдена вченими, та геном організму можуть бути представлені у вигляді рядка, складеного з маленьких літер англійського алфавіту. Кожна літера відповідає окремому гену. Відомо, що організм A належить до певного виду X, якщо з рядка, що представляє *геном* організму A, можна викреслити певні символи та отримати рядок, що представляє характерний геном для виду X.

Завдання Напишіть програму **mutation**, що за заданою послідовністю генів, що знайшли вчені, та геному представника виду, знайде індекси усіх генів, які можуть бути зайвими у послідовності генів, знайденої вченими.

Вхідні дані Вхідний файл **mutation.dat** містить два рядки. Перший рядок представляє знайдену вченими послідовність генів. Другий рядок — геном представника виду. Обидва рядки непорожні, складаються із маленьких літер латинського алфавіту, та довжина кожного з них не перевищує 40 000 символів.

Вихідні дані Перший рядок вихідного файлу **mutation.sol** має міститити одне ціле число — кількість генів, кожен з яких, можливо, є зайвим у знайденій вченими послідовности генів. У другому рядку виведіть *індекси* усіх таких генів в порядку зростання. Гарантовано, що існує хоча б один зайвий ген.

Оцінювання У 50 % тестів довжини рядків у вхідних даних не перевищують 2000.

Приклад вхідних та вихідних даних

mutation.dat
adca abcdaba

mutation.sol
2 2 3

З послідновності `adca` потрібно видалати або ген `d`, або ген `c`. Таким чином, у першому випадку ми отримаємо послідновність `aca`, яку можна отримати з геному, наприклад, так: ~~a~~b~~c~~d~~a~~b~~a~~, а у другому випадку - послідовність `ada`, яку можна отримати, наприклад, так: ~~a~~b~~c~~d~~a~~b~~a~~.

1.2.1 Розв'язання

Після формалізації отримаємо таку постановку задачі:

Дано два рядки A та B довжиною до 40 000. Необхідно знайти усі такі символи рядка B, що після видалення одного з них залишок B був би підпослідовністю рядка A.

Є відома задача про перевірку того, що один рядок є підпослідовністю іншого. Тобто нехай ми перевіряємо, що B є підпослідовністю A. Розв'язати таку задачу можна за допомогою жадібного алгоритму: перебираємо символи рядка A, починаючи з початку, і якщо зустрічаємо символ, рівний першій літері рядка B, то видаляємо цю літеру з нього. Якщо по закінченні виконання цього алгоритму рядок B залишився порожнім, то початковий рядок B був підпослідовністю рядка A. Складність такого алгоритму є лінійною відносно суми довжин двох рядків.

Інтерпретуємо цей жадібний алгоритм як алгоритм динамічного програмування: для кожного префікса рядка $A[1 \dots i]$ підрахуємо $P[i]$ — найбільшу довжину префікса рядка B, що $B[1 \dots P[i]]$ є підпослідовністю префікса $A[1 \dots i]$. Нехай $P[0] = 0$, тоді для $i > 0$: $P[i] = P[i-1]$, якщо $P[i-1] = |B|$ або $A[i] \neq B[P[i-1]+1]$ та $P[i] = P[i-1]+1$ у випадку $A[i] = B[P[i-1]+1]$ та $P[i-1] < |B|$. Якщо $P[|A|] = |B|$, то B є підпослідовністю A.

Адаптуємо цю ідею на випадок, коли нам треба видалити одну літеру з рядка B, а саме: підрахуємо $P[i]$ та аналогічну їй величину $S[i]$ — найбільшу довжину суфікса рядка B, при якому він є підпослідовністю суфікса $A[i \dots |A|]$. Припустимо, що ми видалили деякий символ x з рядка B і отримали рядок $C(x) = B[1 \dots x-1]+B[x+1 \dots |B|]$. Нескладно зрозуміти, що $C(x)$ є підпослідовністю A тоді і тільки тоді, коли знайдеться префікс $A[1 \dots i]$, що $B[1 \dots x-1]$ є підпослідовністю $A[1 \dots i]$, а $B[x+1 \dots |B|]$ є підпослідовністю $A[i+1 \dots |A|]$. Тобто для деякого індексу i виконується $P[i] \geqslant x-1$ та $S[i+1] \geqslant |B|-x$, а з властивостей монотонності величин P, S можна стверджувати, що існує також індекс $0 \leqslant j \leqslant |A|$ такий, що $P[j] = x-1$ та $S[j+1] \geqslant |B|-x$. Вважаємо, що $S[|A|+1] = 0$.

Звідси маємо такий алгоритм:

1. Побудуємо величини P та S.
2. Перебираємо індекс j $(0 \leqslant j \leqslant |A|)$:
   якщо $P[j] + S[j+1] \geqslant |B|-1$ та $P[j]+1 \leqslant |B|$:
   індекс $P[j]+1$ додаємо до набору шуканих індексів.

1.3 «База даних» (Андрій Коротков).

На підприємстві працює N співробітників. Між ними існує M зв'язків «керівник — підлеглий». У співробітника може бути декілька керівників та підлеглих. Не існує послідовності зв'язків «керівник — підлеглий», яка починається та закінчується одним і тим самим співробітником.

Доступ до *корпоративної бази даних* регулюється системою прав. У кожен момент часу для кожного співробітника однозначно відомо: має він права на доступ до бази даних, чи ні. Певний час тому, коли базу даних було встановлено у підприємстві, ніхто не мав прав на доступ до неї. В процесі роботи з базою даних права на доступ змінювались за допомогою операцій вигляду:

1. Адміністратор надає права співробітнику X.
2. Адміністратор позбавляє прав співробітника X.
3. Співробітник починає ділитися правами з усіма безпосередніми підлеглими.
4. Співробітник починає ділитися правами з усіма безпосередніми підлеглими. Потім для кожного безпосереднього підлеглого співробітника виконується операція 4.
5. Співробітник припиняє ділитися своїми правами з усіма безпосередніми підлеглими.

Співробітник X має права на доступ до корпоративної бази даних, якщо виконується хоча б одна з таких умов:

1. Останньою операцією Адміністратора відносно співробітника X було надання йому прав.
2. Хоча б один з безпосередніх керівників, що має права на поточний момент, ділиться з ним правами на доступ.

Зверніть увагу. Якщо співробітник поділився правами доступу зі своїми підлеглими, а потім їх втратив, він все рівно продовжує ділитись правами зі своїми підлеглими. Однак вони не зможуть цим скористатися для одержання прав, поки цей співробітник знову не отримає прав на доступ (див. другий приклад з умови).

Завдання Напишіть програму **database**, яка по заданій послідовності операцій по зміні прав на доступ до корпоративної бази даних для кожного співробітника визначатиме, чи буде він мати права на доступ після виконання цієї послідовності операцій.

Для заданої послідовності операцій виконуються такі обмеження. Для кожного співробітника жодна з операцій 1 та 2 не зустрічається двічі поспіль. Не зустрічаються операції 3 та 4, коли у відповідний момент часу співробітник не має прав на доступ. Не зустрічаються операції 5, коли у відповідний момент часу співробітник не ділиться правами зі своїми підлеглими.

Вхідні дані Перший рядок вхідного файлу **database.dat** містить два цілих числа: N ($1 \leqslant N \leqslant 10\,000$) — кількість співробітників підприємства, та M ($1 \leqslant M \leqslant 50\,000$) — кількість зв'язків «керівник — підлеглий». Наступні M рядків містять по два натуральних числа X та Y ($1 \leqslant X, Y \leqslant N$), визначаючи, що X є безпосереднім керівником Y. Наступний рядок містить число K ($1 \leqslant K \leqslant 20\,000$) — кількість операцій зміни прав. Наступні K рядків містять по два натуральних числа T та X ($1 \leqslant T \leqslant 5, 1 \leqslant X \leqslant N$) — відповідно тип операції та номер співробітника, якого вона стосується.

Вихідні дані Єдиний рядок вихідного файлу **database.sol** має містити N цілих чисел, розділених пропусками. i-те число рівне 1 або 0, в залежності від того, матиме i-й співробітник права доступу після виконання заданих команд (1), чи ні (0).

Оцінювання Принаймні у 20 % тестів присутні лише операції 1—3. Принаймні у 35 % тестів присутні лише операції 1—4. Принаймні в 40 % тестів $N \leqslant 500$, $M \leqslant 500$, $K \leqslant 1000$.

Приклади вхідних та вихідних даних

database.dat
5 5 1 2 1 3 2 4 3 4 3 5 4 1 1 4 1 1 2 5 1

database.sol
1 1 0 1 0

database.dat
2 1 1 2 4 1 1 3 1 2 1 1 1

database.sol
1 1

Пояснення до прикладу 1

1. 1-й співробітник отримує права на доступ від адміністратора.
2. 1-й співробітник починає ділитися правами на доступ із 2-м та 3-м співробітниками. У свою чергу 2-й ділиться із 4-м, а 3-й ділиться з 4-м і 5-м співробітниками.
3. 2-й співробітник отримує права доступу від адміністратора.
4. 1-й співробітник припиняє ділитися правами на доступ із 2-м та 3-м співробітниками. В результаті 3-й співробітник втрачає права на доступ, а 2-й співробітник — ні, оскільки він має права на доступ від адміністратора. 4-й співробітник також не втрачає права на доступ, оскільки 2-й співробітник має права на доступ і ділиться з ним правами. 5-й співробітник втрачає права, тому що 3-й співробітник хоч і ділиться правами, на поточний момент не має прав.

Пояснення до прикладу 2

1. 1-й співробітник отримує права на доступ від адміністратора.
2. 1-й співробітник починає ділитися правами на доступ із 2-м співробітником. 2-й співробітник отримує права, тому що 1-й має права і ділиться правами з ним.
3. Адміністратор позбавляє прав 1-го співробітника. 2-й співробітник також втрачає права, оскільки 1-й хоч ділиться з ним правами, на даний момент їх не має.
4. 1-й співробітник отримує права на доступ від адміністратора. 2-й співробітник отримує права, оскільки 1-й співробітник має права на поточний момент часу і ділиться з ним правами.

1.3.1 Розв'язання

Введемо позначення: $\Gamma = (V, R)$ — граф, що задає ієрархію на підприємстві, V — множина вершин (співробітників), R — множина ребер (начальник, підлеглий). $last_0[t][x]$ — найбільший час виконання команди tx. $last[t][x]$ — найбільший момент часу, коли для вершини x була виконана операція t. $ans[x]$ — чи матиме права x після виконання заданої послідовності операцій.

$$last[t][x] = \begin{cases} last_0[t][x], & \text{якщо } t \neq 4; \\ \min(last_0[t][x], \min_{(y,x)\in R}(last[t][y])), & \text{якщо } t = 4. \end{cases}$$

Топологічно відсортуємо вершини графа. Обчислимо відповідь для вершини графу, починаючи з топологічно старших. В такому разі, при розгляді довільної вершини x всі її начальники будуть розглянуті попередньо. Умови володіння правами для x будуть виглядати наступним чином:

1) $last[1][x] > last[2][x]$;

2) $ans[y]$ AND $max(last[3][y], last[4][y]) > last[5][y]$, $(y, x) \in R$.

Складність алгоритму $O(N + M + K)$.

1.4 «Автомагістраль» (Ілля Порубльов).

Команда Логарифмічної області країни Олімпія збирається на олімпіаду, що відбудеться у далекому місті Експоненціальську. Їхати команда буде власним автобусом. Автомагістраль, що з'єднує Логарифмічну область з Експоненціальськом, складається з N послідовних фрагментів. Всередині кожного фрагмента можна їхати або по безплатній дорозі, витрачаючи a_i секунд, або по платній, витрачаючи c_i олімпійських центів та b_i секунд. Між цими фрагментами є транспортні розв'язки, через які можна з'їхати з однієї дороги на іншу. Такі перебудови вимагають q_i секунд (без різниці, з платної дороги на безплатну, чи з безплатної на платну). При продовженні руху по тій самій дорозі аналогічних затримок не виникає. Спочатку можна виїхати хоч на безплатну дорогу, хоч на платну. Завершити шлях теж можна по будь-якій з двох доріг останнього фрагмента. Тому, розв'язки є лише між 1-м та 2-м фрагментами, між 2-м та 3-м, ..., між $(N-1)$-м та N-м.

При поїздці на олімпіаду дуже важливо встигнути вчасно, тому команду цікавить, за яку найменшу вартість можна дістатися з Логарифмічної області до Експоненціальська, витративши сумарно не більш як T секунд. При поверненні з олімпіади час не настільки критичний, і перед командою була поставлена вимога сплатити на зворотньому шляху не більш як S олімпійських центів дорожніх зборів. Усі витрати часу та дорожні збори однакові при руху в обох напрямках.

Завдання Напишіть програму **highway**, яка за даними про безплатні та платні дороги автомагістралі знаходитиме:

1. Найменшу ціну, за яку можна дістатися на олімпіаду за час, не більший T секунд.
2. Найменший час, за який можна повернутися з олімпіади, сплативши не більш ніж S центів дорожніх зборів.

Вхідні дані Перший рядок вхідного файлу **highway.dat** містить три цілих числа N ($2 \leqslant N \leqslant 40$) — кількість фрагментів магістралі, T ($0 \leqslant T \leqslant 10^{16}$) та S ($0 \leqslant S \leqslant 10^{16}$) — обмеження на сумарний час при пошуку першої відповіді та обмеження на сумарну вартість при пошуку другої. Другий рядок містить три цілі числа a_1, b_1 та c_1 — час руху по безплатній та платній дорогах 1-го фрагмента, та ціну проїзду по платній. Кожен з наступних $N-1$ рядків містить по чотири цілі числа q_i, a_i, b_i та c_i — спочатку час на перебудову між дорогами, потім час руху по безплатній та платній дорогах цього фрагмента, потім ціну проїзду по платній. Зауважимо, що на шляху на олімпіаду q_i — час, необхідний на перебудову з $(i-1)$-го фрагмента на i-й, а на зворотньому шляху q_i — час, необхідний на перебудову з i-го фрагмента на $(i-1)$-й (за умови, що автобус з'їжджає з платної дороги на безплатну або навпаки). Усі значення a_i, b_i та c_i ($1 \leqslant i \leqslant N$) у межах від 1 до 10^{15}. Усі значення q_i ($2 \leqslant i \leqslant N$) у межах від 0 до 10^9.

Вихідні дані Єдиний рядок вихідного файлу **highway.sol** має містити два цілих числа — вартість найдешевшого серед способів дістатися на олімпіаду за час, не більший T секунд, та тривалість найшвидшого серед способів повернутися з олімпіади, сплативши не більш

ніж S центів дорожніх зборів. Якщо жодного відповідного способу не існує, слід виводити -1 як одне із чисел.

Оцінювання Щонайменше у 30 % тестів виконується додаткове обмеження $2 \leqslant N \leqslant 17$. Щонайменше у 40 % тестів виконується додаткове обмеження вигляду: T, S, усі q_i, a_i, b_i та c_i не перевищують 10^5.

Приклад вхідних та вихідних даних

highway.dat
5 2012 2012 10000 17 10000 4 1000 17 1000 3 100 17 100 2 10 17 10 1 1 17 1

highway.sol
10000 10051

Пояснення

Найдешевший спосіб за час, не більший 2012 — проїхати 1-й фрагмент по платній дорозі, далі по безплатній: сумарна вартість $10000+0+0+0+0 = 10000$ за час $17+4$ (перебудова)$+$ $1000 + 100 + 10 + 1 = 1132$. Найшвидший спосіб повернутися з сумою зборів не більше 2012 — 5-й і 4-й фрагменти по безплатній, 3-й і 2-й по платній, 1-й по безплатній: час $1+10+2$ (перебудова)$+17+17+4$ (перебудова)$+10000 = 10051$ при сумі зборів $0+1000+$ $100+0+0 = 1100$.

1.4.1 Розв'язання

Спосіб з *гарантовано* допустимими часом роботи і об'ємом пам'яті використовує «meet in the middle», «2 вказівника» та модифікації злиття і має асимптотичну складність $O(2^{N/2})$ часу, $O(2^{N/2})$ пам'яті. Значно ефективнішого правильного розв'язку, мабуть, не існує, бо задача NP-повна (як узагальнення NP-повної задачі «є сукупність елементів, вибрати деякі з них так, щоб сума вибраних була якомога ближчою знизу до S»).

Враховуючи спосіб оцінювання даної оліміпади, непоганих результатів можна досягти також і перебірними (які не гарантують, що програма завжди вкладатиметься у обмеження по часу) та/або евристичними (які не гарантують правильності) методами — можна перепробувати кілька переборів та/або евристик і вибрати з них варіант, який приносить максимальні бали.

Оптимальний розв'язок — підтримка сукупності не домінованих пар Побудуємо для фрагментів магістралі по дві сукупності пар (t, p), тобто (час; вартість) — одна сукупність для шляхів (від початку), що закінчуються на безплатній дорозі даного фрагмента, інша — на платній. На попередніх фрагментах дороги можна міняти.

Якщо для $(a.t, a.p)$ та $(b.t, b.p)$ виконується $(a.t \leqslant b.t)$ and $(a.p \leqslant b.p)$, будемо називати пару $(b.t, b.p)$ *домінованою* — її можна викинути, бо пара $(a.t, a.p)$ гарантовано не гірша.

Твердження 1. *Сукупність не домінованих пар можна впорядкувати* $a_1.p < a_2.p < \ldots <$ $< a_L.p$*; при цьому одночасно виконуватиметься* $a_1.t > a_2.t > \ldots > a_L.t$.

Доведення. Впорядкувати $a_1.p \leqslant \ldots \leqslant a_L.p$ можна будь-яку сукупність. Різних пар $(a.t, a.p)$ та $(b.t, b.p)$ з $a.p = b.p$ не буде, бо при $a.t < b.t$ пара b буде домінованою, при $a.t > b.t$ — пара a, а при $a.t = b.t$ нема смислу зберігати дві копії однакової пари. Далі, при $a.p < b.p$ не може бути $a.t \leqslant b.t$, інакше пара a була б домінованою. □

Будуватимемо послідовності не домінованих пар, впорядковані $p\nearrow$, $t\searrow$. Для 1-го фрагмента ці послідовності тривіальні — єдина пара $\langle(a_1, 0)\rangle$ для безплатної дороги і єдина $\langle(b_1, c_1)\rangle$ для платної. Надалі, є впорядковані послідовності не домінованих пар для $(s-1)$-го фрагмента (позначимо їх $lastfree$ та $lastpaid$), будемо будувати послідовності не домінованих пар для s-го (позначимо $nextfree$ та $nextpaid$).

Аналогічно злиттю, будемо рухатися індексом i по $lastfree$, індексом j по $lastpaid$, щоразу порівнювати поточні пари $lastfree[i]$ та $lastpaid[j]$, і вибирати ту з них, де менше p. Тільки, на відміну від злиття:

1. Кандидатом на додавання у результат є не просто вибрана пара, а пара, побудована за такими правилами:
   (a) При побудові $nextfree$ поле p копіюється з вибраної пари, а при побудові $nextpaid$ до нього ще додається вартість проїзду по платній дорозі поточного фрагмента c_s.
   (b) Поле t збільшується: на час проїзду по самому фрагменту (a_s для $nextfree$ або b_s для $nextpaid$); якщо відбулася зміна дороги (платна→безплатна або навпаки), то додатково ще на q_s.
2. В разі рівності $lastfree[i].p = lastpaid[j].p$, вибираємо пару з меншим (з урахуванням всіх поправок п. 1b) часом.
3. Дописуємо побудовану пару до результату тільки коли або результат ще порожній, або t побудованої пари строго менше за t останньої наразі пари результату.

Вибір меншого p та п. 2 гарантують, що домінована пара не потрапить у результат раніше пари, яка її домінує. А п. 3 — що не потрапить пізніше.

Зберігання лише не домінованих пар, *як правило*, суттєво зменшує розміри сукупностей. Але можливі вхідні дані (наприклад: всі $q_i = 1$, всі $b_i = 3$, решту визначити як $a_i = c_i = 7 \cdot 2^i$), для яких розміри кожної з сукупностей не домінованих пар для i-го фрагменту будуть 2^{i-1}, що при $i \approx N \approx 42$ забагато. Тому, *крім* вже розглянутого етапу, треба користуватися також ідеєю meet in the middle. При розробці тестів планувалося, щоб ефективна побудова не домінованих пар (без meet in the middle) набирала 60–70% балів.

Оптимальний розв'язок — meet in the middle (з «2-ма вказівниками») Сукупності не домінованих пар можна будувати не для всіх фрагментів зразу, а двічі по $N/2$. Візьмемо $N_1 := N \operatorname{div} 2$, $N_2 := N - N_1$, і побудуємо сукупності не домінованих пар для N_1 фрагментів, починаючи від початку («ліва половина») та N_2 фрагментів, починаючи від кінця («пра́ва»). Тоді розміри кожної з послідовностей не перевищуватимуть $2^{20} \approx 10^6$. Остато́чні відповіді задачі будемо шукати, поєднуючи ці послідовності для «половинок». Причому, для поєднань дуже зручно, що послідовності задані в порядку $p\nearrow$, $t\searrow$.

Твердження 2. *«Ліва половинка» з характеристиками $(left[i].t, left[i].p)$ може бути частиною оптимального маршруту (вигляду «*min *час при вартості $\leqslant S$») лише при поєднанні з «правою половинкою», яка має значення $right[j].p$, найближче знизу до $S - left[i].p$.*

Доведення. «Праву половинку» з більшим значенням p брати не можна, бо шлях буде не допустимим (вартість $> S$). А взявши «половинку» зі значенням $right[k].p < right[j].p$, отримаємо (завдяки $p\nearrow$, $t\searrow$) $right[k].t > right[j].t$, тоді як час треба мінімізувати. □

Отже, при переборі i у порядку спадання, кожне відповідне за твердженням 2 значення j буде або тим самим, що для попереднього i, або більшим ($left[i].p$ зменшилося $\Rightarrow S - left[i].p$ збільшилося). Тобто, застосовна ідея «два вказівники»: організувавши цикл пошуку j всередині циклу перебору i, можна продовжувати пошук j з того місця, де спинилися при попередньому i, й усе проходження двома вказівниками працюватиме за $O(|left| + |right|)$.

Всього проходів 2-ма вказівниками треба зробити $2 \times 2 \times 2 = 8$ штук: min вартість для часу $\leqslant T$ чи min час для вартості $\leqslant S$; якою (безплатною чи платною) доро́гою закінчується

ліва половина; якою права половина. Якщо одна з доріг безплатна, а інша платна, треба безпосередньо в даному проході врахувати $q[N_1]$.

Бажано *не* переписувати код 8 разів, а оформити як підпрограму і багатократно ви́кликати. Наприклад, підпрограм може бути дві: min вартість для часу $\leqslant T$ та min час для вартості $\leqslant S$; яка дорога з якою перевіряються, варто задати параметрами. Аналогічно, варто зробити однією процедурою (з різними параметрами) побудову сукупностей не домінованих пар $nextfree$ та $nextpaid$.

Інші способи формування не домінованих переліків Замість модифікацій злиття, можна формувати кожен перелік не домінованих пар так: включити до переліку пари, відповідні абсолютно всім 2^{N_i} шляхам; відсортувати за неспаданням p, а при рівних p за неспаданням t. Після цього, за один прохід, легко вибрати лише не доміновані пари, порівнюючи кожну поточну пару з останньою вибраною.

Такий підхід простіший для написання, але менш ефективний, бо сортування переліка довжиною $2^{N/2}$ потребує $O(2^{N/2}\log(2^{N/2})) = O(2^{N/2} \cdot N)$ дій, а рекомендований алгоритм формує правильно впорядкований перелік за $O(2^{N/2})$.

Ще один спосіб побудови не домінованих пар — виконання додаткових дій з `map`-ом (STL мови C++); див., зокрема, вказівки до задачі «Choosing a camera» студентської олімпіади SEERC–2011 (`codeforces.ru/blog/entry/2880`). Цей спосіб теж потребує $O(2^{N/2} \cdot N)$ дій.

При розробці тестів планувалося, щоб дані способи (при використанні meet in the middle) набирали 75–80% балів.

Розв'язання динамічним програмуванням при S, T до 10^5–10^6 Якщо S і T досить малі, щоб можна було підтримувати кілька масивів від 0 до $\max(S,T)$, задачу можна вирішити, розв'язавши серії підзадач *«Яку мінімальну суму $S(i,r,t)$ доведеться сплатити, щоб дістатися від початку до кінця i-го фрагменту, витративши не більш ніж t центів, причому $r=0$ означає, що останній i-й фрагмент їхали по безплатній дорозі, $r=1$ — по платній»*, та аналогічну серію $T(i,r,s)$.

При $S, T \sim 10^{16}$, цей спосіб практично неможливий. Але він гарантовано проходить 40% тестів, де значення до 10^5. В інших тестах можна усі значення відповідної природи поділити (з заокругленням до цілого), щоб поділені S та T були в межах до, наприклад, $4 \cdot 10^6/N$ (приблизно максимальне значення, при якому програма ще поміщається у обмеження по часу і пам'яті). Сума заокруглених може відрізнятися від заокруглення суми, тому такий спосіб не завжди даватиме правильну відповідь. Але є ймовірність, що якась частина відповідей інших тестів теж буде правильною. Причому, якщо значення ділити саме з заокругленням (до найближчого), ця ймовірність вища, ніж коли ділити цілочисельно (без остачі).

Гілки та межі, інші перебірні методи Бажаючі можуть знайти інформацію про ці методи за ключовими словами «метод гілок та меж», він же «метод розгалужень та обмежень», «метод ветвей и границ», «branch and bound», а також «backtracking». Можна пробувати й інші перебірні або перебірно-евристичні методи («метод отжига», генетичні алгоритми, і ще багато інших).

При розробці тестів ставилася мета, щоб очевидні варіанти гілок та меж набирали 60–70% балів. Але поведінка гілок та меж сильно й малопрогнозовано залежить від дрібних змін, тому можливі значні відхилення від оголошених відсотків. Ще раз нагадаємо, що при використаному лояльному способі перевірки можна було пробувати міняти оціночні функції та інші деталі перебору й вибирати варіант, що набере найбільше балів.

Більш ефективними виявилися переборні підходи, які починаються з сортування за певною евристичною властивістю, яка визначає, припущення щодо якого фрагменту варто

робити на зовнішньому рівні рекурсії, щодо якого (не обов'язково сусіднього) — на наступному, і т. д. Інше евристичне порівняння визначає, в якому порядку робити рекурсивні виклики: спочатку розглянути, що даний фрагмент їдуть по платній дорозі, а потім, що по безплатній, чи навпаки).

2 Завдання другого туру

2.1 «Розфарбування» (Роман Єдемський).

На планеті Олімпія щорічно проводиться традиційна гра у «Фарбувашки». Місцем проведення цієї гри є велике олімпійське поле $N \times M$, кожна клітинка якого є або білою, або чорною. Двоє гравців ходять по черзі та загалом здійснюють на двох рівно K ходів. Під час свого ходу гравцю дозволяється замалювати власним кольором усі клітинки будь-якого одного прямокутника, чия висота та ширина не перевищує D. Перший гравець замальовує клітинки у білий колір, а другий — у чорний. По закінченні K-го ходу підраховується остаточний рахунок гри — кількість клітинок кожного кольору. Переможцем оголошується той гравець, в чий колір розмальовано більшу кількість клітинок, ніж у супротивника. Тому метою кожного гравця є максимізація кількості клітинок його власного кольору в остаточному розфарбуванні поля.

Завдання Напишіть програму **paint**, що за заданим початковим полем, кількістю ходів та обмеженням на прямокутник, який може зафарбувати гравець, знайде остаточний рахунок гри при оптимальній стратегії обох гравців.

Вхідні дані Перший рядок вхідного файлу **paint.dat** містить чотири натуральних числа: N, M, D і K ($N \leqslant 400$, $M \leqslant 400$, $D \leqslant 400$, $K \leqslant 10^9$) — висота та ширина поля, обмеження на розмір прямокутника, який може зафарбовувати гравець, кількість ходів, що належить виконати до підведення остаточного рахунку. В наступних N рядках розташовано по M символів: `W` якщо відповідну клітинку розфарбовано у білий колір і `B` у випадку, якщо клітинку поля розфарбовано у чорний колір.

Вихідні дані Єдиний рядок вихідного файлу **paint.sol** має містити два цілих числа — кількість білих та кількість чорних клітинок в остаточному розфарбуванні при умові оптимальної гри обох гравців.

Оцінювання Принаймні у 50 % тестів N, M, D і K не перевищують 100.

Приклад вхідних та вихідних даних

paint.dat
3 3 2 1 BWB BBW WBB

paint.sol
6 3

2.1.1 Розв'язання

Таблиця Н на М.
Н стовпчикыв чи стовпцыв?

Ходит первым первый игрок?

З питань учасників

Нехай деяке початкове розфарубвання P містить хоча б одну клітинку чорного кольору і гравцям в сумі належить зробити K ходів до завершення гри. Позначимо через $F(R)$ максимальну кількість білих клітинок в остаточному розфарбуванні, що може досягнути перший гравець при оптимальній грі обох, якщо початкове розфарбування — R. Оберемо клітинку чорного кольору у P і замалюємо її у білий колір, позначимо отримане розфарбування P'.

Твердження 1. *Для будь-якої фіксованої послідовності ходів кількість клітинок білого кольору в остаточному рахунку при початковому розфарбуванні P' не менша, аніж при початковому розфарбуванні P.*

Доведення. Справді, розглянемо будь-яку клітинку дошки. Якщо внаслідок послідовності ходів відбувається перефарбування цієї клітинки, то її остаточний колір не залежить від початкового розфарбування. Якщо ж унаслідок послідовності ходів не відбувається перефарбування клітинки, то її колір залежить лише від початкової позиції. Але за побудовою усі білі клітинки P є білими і у P', тому кількість клітинок білого кольору в остаточному рахунку не зменшиться при заміні початкового розфарбування P на P'. □

Твердження 2. $F(P') \geqslant F(P)$.

Доведення. Розглянемо дві гри: гру A «фарбувашки» при початковому розфарбуванні P і гру B «фарбувашки» при початковому розфарбуванні P'. Нехай ми знаємо оптимальну стратегію для гри A і намагаємось побудувати стратегію, що дає не гірший результат для гри B. Зробимо хід у грі B, який є оптимальним першим ходом у грі A, позначимо його u. Нехай другий гравець, який слідує оптимальній стратегії в B, відповів нам деяким ходом v (якщо, звісно, $K \neq 1$). Тепер припустимо, що у грі A було здійснено два ходи u та v. Нехай тепер оптимальним ходом є x. Здійснимо x у грі B. Якщо $K \neq 3$, отримаємо у відповідь деякий хід z, і знову оберемо хід, виходячи з того, що у грі A здійснено вже u, v, x, z. Тобто для будь-якої послідовності ходів непарної довжини, здійсненої у грі B, ходом у відповідь ми обираємо оптимальний (за мірками гри A) хід у відповідь на цю ж саму послідовність ходів. Зауважимо, що оскільки перший гравець грав оптимально на кожному кроці (за мірками гри A), то отримана послідовність ходів призводить до не гіршого рахунку у грі A, аніж $F(P)$. В силу твердження 1 ця послідовність ходів у B призводить до не гіршого рахунку, аніж у грі A. Тобто, перший гравець може гарантувати собі рахунок не гірший, аніж $F(P)$, незалежно від ходів суперника, а отже $F(P') \geqslant F(P)$. □

Твердження 3. *Для обох гравців існує оптимальна стратегія, що включає лише перефарбування прямокутників розміром* $\min(N, D) \times \min(M, D)$.

Доведення. Справді, нехай це не так, і на деякому кроці одному з гравців *необхідно* для досягнення оптимального результату перефарбувати прямокутник Q менших розмірів. Але з твердження 2 випливає, що використавши для перефарбування прямокутник з розміром $\min(N, D) \times \min(M, D)$, який накриває Q, гравець отримає результат не гірший, аніж при перефарбуванні прямокутника Q. □

Твердження 4. *Нехай у початковому розфарбуванні було W білих клітин та перший гравець зробив хід, яким перефарбував X чорних клітин у білий колір. Тоді кількість білих клітин в остаточному розфарбуванні при оптимальній грі обох буде рівною $W + X$ при непарному K та $W + X - \min(N, D) \cdot \min(M, D)$ при парному.*

Доведення. Нехай баланс розфарбування — це різниця між кількістю білих та кількістю чорних клітинок. Тоді перший гравець намагається максимізувати баланс, а другий — мінімізувати. Нехай у початковому розфарбуванні баланс дорівнює T, тоді після першого ходу баланс дорівнюватиме $T + 2 \cdot X$. Унаслідок твердження 3 можна вважати, що обидва гравці

на кожному своєму ході перефарбовують прямокутник розмірами $\min(N, D) \times \min(M, D)$. Тому починаючи з другого ходу, незалежно від ходів суперника, у кожного з гравців є можливість змінити баланс на $2 \cdot \min(N, D) \cdot \min(M, D)$ у вигідну для нього сторону.

Розглянемо кінцевий баланс: $T + 2 \cdot X - A_2 + A_3 - A_4 + \ldots + (-1)^{k+1} \cdot A_k$, де A_i — це зміна балансу на i-му кроці. Позначимо $\min(N, D) \cdot \min(M, D)$ за U. Зрозуміло, що перший гравець може собі гарантувати щонайменше $Q = T + 2 \cdot X - 2 \cdot U + 2 \cdot U - 2 \cdot U + \ldots + 2 \cdot (-1)^{k+1} \cdot U$, змінюючи на кожному своєму кроці баланс на $+2 \cdot U$. З іншого боку, другий гравець може гарантувати собі результат, не більший від Q, змінюючи на кожному своєму кроці баланс на $-2 \cdot U$. Тому при оптимальній грі обох баланс у кінцевому розфарбуванні буде рівний Q. З цього маємо, що при парному K баланс дорівнює $T + 2 \cdot X - 2 \cdot U$, а при непарному — $T + 2 \cdot X$. Тобто при оптимальній грі при парному K маємо $W + X - U$, а при непарному — $W + X$ білих клітин. □

Для розв'язання задачі залишається знайти прямокутник розмірами $\min(N, D) \times \min(M, D)$ з максимальною кількістю чорних клітин.

1. Наївний розв'язок $O(N \cdot M \cdot \min(N, D) \cdot \min(M, D))$ набирає 50 балів.
2. Розв'язок $O(N \cdot M \cdot \min(N, D))$ з одномірними частковими сумами набирає 100 балів.
   Для кожного рядка поля i підрахуємо величину $S[i][k]$, що дорівнює кількості клітин чорного кольору з початку рядка до його k-ї клітини включно. Нехай $S[i][0] = 0$. Тоді, щоб за $O(N)$ знайти кількість клітин чорного кольору на прямокутнику $[x, y] \times [c, d]$, необхідно підрахувати суму $S[x][d] - S[x][c-1] + S[x+1][d] - S[x][c-1] + \ldots + S[y][d] - S[y][c-1]$.
3. Розв'язок $O(N \cdot M)$ із двомірними частковими сумами набирає 100 балів.
   Для кожної пари індексів поля (i, j) підрахуємо величину $S[x][y]$, рівну кількості клітин чорного кольору на прямокутнику $[1, x] \times [1, y]$. Нехай $S[0][y] = S[x][0] = 0$. Тоді кількість клітин чорного кольору на прямокутнику $[x, y] \times [c, d]$ можна розрахувати так: $S[y][d] - S[x-1][d] - S[y][c-1] + S[x-1][c-1]$. Щоб підрахувати $S[x][y]$ за $O(N \cdot M)$ можна використати динамічне програмування: $S[x][y] = S[x-1][y] + S[x][y-1] - S[x-1][y-1] + c$, де $c = 1$ у випадку, коли (x, y) чорного кольору, та $c = 0$, якщо ні.

2.2 «Буфер обміну» (Данило Мисак).

Барт Сімпсон, герой мультсеріалу «Сімпсони», як покарання за те, що прогулює уроки, має надіслати директору школи електронного листа. У листі Барт повинен N разів надрукувати фразу «Я більше не буду прогулювати уроків». На думку директора, написання такого листа справить на Барта вплив, і він більше ніколи не буде прогулювати школу. Наївний директор!

Замість того, щоб N разів друкувати потрібну фразу, Барт надрукував її один раз, після чого вирішив скористатися буфером обміну. За одну операцію Барт або копіює увесь поточний вміст листа до буфера обміну, або вставляє скопійований до буфера текст у лист.

Завдання Напишіть програму **copypast**, яка знаючи, скільки фраз має бути в листі директору, визначить, за яку найменшу кількість операцій з буфером обміну Барт зможе скласти листа, кількість фраз у якому дорівнює потрібній.

Вхідні дані Вхідний файл **copypast.dat** містить єдине число N — кількість фраз «Я більше не буду прогулювати уроків», які мають потрапити до електронного листа. Число N натуральне та не перевищує $2 \cdot 10^9$.

Вихідні дані Вихідний файл **copypast.sol** повинен містити єдине ціле число — найменшу можливу кількість операцій із буфером обміну, після яких у листі буде рівно N фраз.

Оцінювання У 25 % тестів N не перевищує 10. У 65 % тестів N не перевищує 3000.

Приклад вхідних та вихідних даних

copypast.dat
12

copypast.sol
7

Пояснення

Після того, як Барт надрукував першу фразу, він може виконувати такі операції з буфером обміну:

1. Копіює поточний вміст листа (1 фразу).
2. Вставляє скопійований текст (фраз у листі стає 2).
3. Вставляє скопійований текст (фраз стає 3).
4. Вставляє скопійований текст (фраз стає 4).
5. Копіює поточний вміст листа (4 фрази).
6. Вставляє скопійований текст (фраз стає 8).
7. Вставляє скопійований текст (відтак лист містить необхідні 12 фраз).

2.2.1 Розв'язання

Барт діє в такий спосіб: копіює початкову фразу, вставляє її a_1 разів, копіює утворені a_1+1 фразу, вставляє їх a_2 разів, копіює утворені $(a_1+1)(a_2+1)$ фраз, вставляє їх a_3 разів, ..., копіює утворені $(a_1+1)(a_2+1)\dots(a_{k-1}+1)$ фраз, вставляє їх a_k разів, унаслідок чого дістає $(a_1+1)(a_2+1)\dots(a_k+1) = n$ фраз. Для зручності позначимо a_i+1 через p_i. Неважко підрахувати, що Барт зробив усього $p_1+p_2+\ldots+p_k$ операцій і отримав число $n = p_1p_2\dots p_k$. Отже, початкова задача зводиться до такої: розкласти число n на множники $p_1, p_2, \dots, p_k$ так, щоб сума $p_1+p_2+\ldots+p_k$ була мінімальною. Залишається помітити, що якщо хоча б одне із чисел $p_1, p_2, \dots, p_k$ у відповідному розкладі не буде простим, то його можна розкласти хоча б на два множники, більші за 1, зменшивши при цьому суму: якщо, наприклад, $p_1 = d_1d_2$ і $d_1 > 1, d_2 > 1$, то $d_1 + d_2 < p_1$, а тому $d_1 + d_2 + p_2 + p_3 + \ldots + p_k < p_1 + p_2 + \ldots + p_k$. Таким чином, сума $p_1+p_2+\ldots+p_k$ є найменшою тоді, коли $n = p_1p_2\dots p_k$ — розклад числа n на прості множники (при цьому деякі з чисел $p_1, p_2, \dots, p_k$ можуть збігатися). Значить, щоб отримати відповідь, число n треба розкласти на прості множники, а потім знайти їхню суму.

Щоб розкласти задане число n на прості множники, можна перебрати в порядку збільшення всі потенційні дільники цього числа — числа від 2 до $n-i$, щойно n націло поділиться на якесь із цих чисел k, зменшувати значення n у k разів (і продовжувати перебір із k, а не з $k+1$, на випадок, якщо n ділиться на k у деякому степені). Утім, щоб суттєво покращити час роботи в найгіршому випадку, можна здійснювати перебір не від 2 до n, а від 2 до $\sqrt{n}$. Якщо по закінченні перебору, після всіх ділень, n не дорівнюватиме 1, це означатиме, що поточне значення n — просте число: інакше хоча б один із його дільників не перевищував би $\sqrt{n}$. Під час реалізації не слід забувати також, що початкове значення n може дорівнювати 1, у цьому випадку правильна відповідь — 0.

Знехтувавши часом, потрібним на ділення, можна оцінити складність поданого алгоритму як $O(\sqrt{n})$.

До розв'язання задачі можна підійти з допомогою динамічного програмування. У даному випадку стандартне його застосування буде таким: послідовно для всіх пар чисел

$k, m, 1 \le k \le n, 1 \le m \le k$, ми підраховуємо найменшу кількість операцій $f(k,m)$ із буфером обміну, які приводять до стану (k,m), де k — кількість фраз у листі, а m — кількість фраз, які на даний момент зберігаються в буфері обміну. При цьому $f(1,1) = 1$, а коли $k > 1$, то $f(k,m)$ дорівнює: $f(k-m,m)+1$, якщо $k-m \ge m$; $\min\{f(k,i)+1 \mid 1 \le i \le k-1\}$, якщо $m = k$; ∞ в інших випадках («∞» позначає, що відповідного стану неможливо досягти в принципі; на практиці можна використовувати досить велике число, яке достеменно перевищує правильну відповідь). Коли ми підрахуємо всі значення, відповіддю буде число $\min\{f(n,i) \mid 1 \le i \le n-1\}$.

Складність такого алгоритму можна оцінити як $O(n^2)$; кількість пам'яті, яку використовує алгоритм, теж дорівнює $O(n^2)$. Але якщо зауважити, що $f(k,m) \ne \infty$ лише в тому випадку, якщо m є дільником k, то для кожного фіксованого k можна знаходити всі дільники за $O(\sqrt{k})$ і рахувати $f(k,m)$ лише для відповідних значень m. Це дасть оцінку $O(n\sqrt{n})$ як на час виконання алгоритму, так і на використовувану пам'ять (якщо зберігати результати обчислень для кожного k не індексованим по m масивом, а списком).

Цікавим є той факт, що коли реалізовувати ті самі рекурентні співвідношення з допомогою простої рекурсії, можна зекономити порівняно з простішим підходом динамічного програмування не лише оперативну пам'ять, але й час виконання програми — принаймні, для n достатньо малих, щоб на рекурсію не забракло пам'яті у стеку.

За допомогою динамічного програмування можна створити і такий алгоритм, який працюватиме за $O(\sqrt{n})$. Для цього достатньо зауважити, що якщо виписати кількості фраз, які опинялися в буфері обміну на оптимальному «шляху» до n фраз, матимемо послідовність, у якій кожне наступне число ділиться на попереднє. Якщо ввести функцію $g(k)$, яка дорівнює мінімальній кількості операцій, потрібних для отримання k фраз, зможемо записати співвідношення: $g(1) = 0, g(k) = \min\{g(i) + k/i \mid i \text{ ділить } k, 1 \le i < k\}, k > 1$. Порахувавши $g(k)$ послідовно для всіх дільників n, матимемо відповідь — $g(n)$. Це займе $O(\sqrt{n} + d^2(n)) = O(\sqrt{n})$ часу і стільки ж пам'яті (через $d(n)$ позначено кількість дільників числа n).

2.3 «Перехрестя» (Андрій Коротков).

Країна Квадроляндія являє собою квадрат $N \times N$ клітинок. Її мешканці використовують систему координат, в якій лівий нижній кут квадрата має координати $(0,0)$, правий верхній — (N,N). У Квадроляндії знаходяться K міст, кожне в точці з цілими координатами. Мешканці Квадроляндії пересуваються по країні паралельно осям координат. Для пришвидшення подорожей до сусідніх країн уряд вирішив побудувати дві швидкісні автомагістралі. Автомагістралі мають бути перпендикулярними між собою та паралельними осям координат. Кожна магістраль з'єднуватиме дві протилежні сторони квадрата.

Відстанню від міста до магістралей буде відстань від міста до найближчої з них. Уряд вирішив розмістити магістралі так, щоб максимальна відстань від міст до магістралей була мінімально можливою.

Завдання Напишіть програму **cross**, що за довжиною сторони квадрата та розташуванням міст знайде оптимальну відстань та розташування магістралей.

Вхідні дані У першому рядку вхідного файлу **cross.dat** містяться два цілих числа: N ($1 \leqslant N \leqslant 1\,000\,000$) та K ($1 \leqslant K \leqslant 40\,000$) — довжина сторони квадрата та кількість міст відповідно. Наступні K рядків містять по два цілих числа — координати міст (від 0 до N).

Вихідні дані У єдиному рядку вихідного файлу **cross.sol** виведіть три цілих числа — шукану оптимальну відстань від найвіддаленішого міста до найближчої магістралі та коор-

динати точки перетину магістралей (перехрестя). Якщо оптимальних відповідей декілька, виведіть будь-яку з них.

Оцінювання

1. Принаймні у 35 % тестів $N \leqslant 50$, $K \leqslant 50$.
2. Принаймні у 50 % тестів $K \leqslant 200$.
3. Принаймні у 65 % тестів $K \leqslant 2000$.

Приклад вхідних та вихідних даних

cross.dat
4 3 1 1 2 2 3 3

cross.sol
1 2 2

2.3.1 Розв'язання

Відстань d будемо вважати допустимою, якщо існує розташування перехрестя, при якому максимальна відстань від міст до нього не перевищує d. Допустима відстань має такі властивості.

1. Якщо d_1 — допустима відстань, і $d_2 > d_1$, то d_2 — також допустима відстань.
2. Якщо d_1 — не допустима відстань, і $d_2 < d_1$, то d_2 — також не допустима відстань.

Ці властивості дозволяють скористатися бінарним пошуком для знаходження оптимальної відстані як мінімальної допустимої відстані. Попередньо відсортуємо міста за x-координатою.

Наведемо алгоритм перевірки відстані d на допустимість.

Нехай x-координата вертикальної магістралі рівна x_0. Тоді міста з x-координатою із проміжку $[x_0 - d, x_0 + d]$ будуть задовольняти умови допустимості. Решта міст будуть знаходитися на відстані, більшій за d, від вертикальної магістралі. Отже, відстань від цих міст до горизонтальної магістралі не повинна перевищувати d. Це можливо тоді і лише тоді, коли різниця максимальної та мінімальної y-координат цих міст не перевищує $2d$.

Методом двох вказівників переберемо найлівіше і найправіше міста, які будуть віддалені від вертикальної смуги не більш ніж на d. На кожному кроці це буде певний проміжок $i..j$. Отже, відстань від міст $1..i-1$ та $j+1..k$ до горизонтальної магістралі не повинна перевищувати d. Для ефективної перевірки можливості відповідного розташування горизонтальної магістралі попередньо обчислимо мінімальну і максимальну y-координати міст на проміжках вигляду $1..i$ та $i..k$.

Складність алгоритму — $O(K \log N)$.

2.4 «Опукла оболонка» (Ярослав Твердохліб).

Задано множину точок на площині, а також N операцій, що модифікують цю множину. Кожна операція може бути одного з двох наступних типів:

1. Додати до множини точку з цілими координатами. При цьому абсциса (тобто X-координата) нової точки є строго більшою за абсциси всіх інших точок, які вже знаходяться у множині.
2. Видалити з множини точку з найбільшою абсцисою.

Завдання Напишіть програму **convex**, яка за заданою послідовністю з N операцій промоделює їх виконання та після кожної з них виведе подвоєну площу опуклої оболонки точок множини. Перед виконанням першої операції множина точок є порожньою. Вважатимемо, що площа опуклої оболонки порожньої множини точок рівна нулю.

Опуклою оболонкою множини точок на площині називається опуклий багатокутник найменшої площі, який містить всі точки з множини. Багатокутник називається опуклим, якщо відрізок, що сполучає будь-які дві його точки, цілком належить цьому багатокутнику. Тут під багатокутником розуміється границя фігури разом з її внутрішністю.

Вхідні дані Перший рядок вхідного файлу **convex.dat** містить одне ціле число: N ($1 \leqslant N \leqslant 100\,000$) — кількість операцій, які потрібно виконати. Наступні N рядків містять по три цілих числа — інформацію про операції у зашифрованому форматі. Для того, щоб отримати параметри операції з номером i, потрібно зчитати три цілих числа T^*, X^* та Y^* ($0 \leqslant T^* \leqslant 1$, $0 \leqslant X^*, Y^* \leqslant 2\,000\,000\,000$) з $(i+1)$-го рядка, після чого отримати число T за наступною формулою: $T = (T^* + S) \bmod 2$, де S — це подвоєна площа опуклої оболонки точок множини до виконання операції з номером i. Можна переконатись, що S завжди є цілим числом.

1. Якщо $T = 0$, то чергова операція першого типу і координати (X, Y) нової точки отримуються за наступними формулами: $X = L + ((S + X^*) \bmod 2\,000\,000\,001), Y = ((Y^* + S) \bmod 2\,000\,000\,001) - 1\,000\,000\,000$. Тут L — це максимальна з абсцис усіх точок з множини. Якщо множина порожня, то $L = -1\,000\,000\,000$.
2. Якщо $T = 1$, то потрібно K разів виконати операцію другого типу. Число K знаходиться за формулою: $K = ((X^* + Y^* + S) \bmod Q) + 1$. Тут Q — це кількість точок у множині.

Гарантується, що при правильній розшифровці всіх операцій виконуються наступні обмеження:

1. Координати усіх точок, які додаються до множини, не перевищують $1\,000\,000\,000$ за модулем.
2. При додаванні нової точки її абсциса є строго більшою за абсциси усіх точок, які вже лежать у множині.
3. Операція видалення застосовується лише до непорожньої множини точок.

Вихідні дані Вихідний файл **convex.sol** має містити N рядків. В рядок з номером i потрібно вивести подвоєну площу опуклої оболонки множини точок, яка утворилася після виконання перших i операцій. Якщо множина точок виявилася порожньою, то площу її опуклої оболонки вважати рівною 0.

Оцінювання Набір тестів складається з 4 окремих блоків, з такими додатковими обмеженнями:

1. 15 % тестів: $N \leqslant 300$.
2. 15 % тестів: $N \leqslant 3000$.
3. 20 % тестів $N \leqslant 100\,000$, кількість рядків, у яких $T = 1$ (після розшифрування) не перевищує 100.
4. 50 % тестів: $N \leqslant 100\,000$.

Приклад вхідних та вихідних даних

convex.dat
6 0 1000000000 1000000000 0 1000000 1001000000 0 1000000 999000000 0 1001500 1000001500 1 85072946 2 1 61619603 2

convex.sol
0 0 3000000000000 6000000000000 3000000000000 0

Пояснення

Після розшифрування тест виглядає таким чином:

- $T = 0, X = 0, Y = 0$;
- $T = 0, X = 1000000, Y = 1000000$;
- $T = 0, X = 2000000, Y = -1000000$;
- $T = 0, X = 3000000, Y = 0$;
- $T = 1, K = 1$;
- $T = 1, K = 2$.

2.4.1 Розв'язання

По-перше, помітимо, що всі операції видалення декількох точок можна розбити на елементарні операції видалення однієї точки. Оскільки кожна точка додається до верхнього ланцюга не більше одного разу, то й у сумі операцій видалення буде не більше ніж N.

Нехай ми вже побудували опуклу оболонку множини точок в певний момент часу. Розглянемо відрізок, що сполучає найлівішу і найправішу точки з множини. Частину опуклої оболонки, яка лежить над цим відрізком разом з двома його кінцями, назвемо верхнім ланцюгом опуклої оболонки. Іншу частину оболонки та два кінці обраного відрізка назвемо нижнім ланцюгом опуклої оболонки. Таким чином, найлівіша і найправіша точки належать обом ланцюгам одночасно. Верхній і нижній ланцюги разом з обраним відрізком утворюють опуклі багатокутники. Площа всієї опуклої оболонки дорівнює сумі площ цих двох багатокутників. Отже, достатньо навчитись виконувати всі необхідні операції з верхнім ланцюгом, а для нижнього все буде аналогічно.

Розв'язок на 50 балів Заведемо стек, в якому будуть зберігатись точки верхнього ланцюга. Додавати нову точку будемо так само, як і в алгоритмі Грехема. А саме, видалятимемо останню точку зі стека до тих пір, поки поворот між двома останніми точками стека (назвемо їх A і B) і новою точкою (D) не стане правим (тобто векторний добуток $AB \times AD < 0$). Після цього додамо нову точку до стека. Для того, щоб окрім самого ланцюга знаходити ще й його площу, потрібно разом з кожною точкою зберігати площу опуклого багатокутника, утвореного всіма точками, які в стеку знаходяться перед нею (включаючи її). Тоді при додаванні нової точки достатньо до вже збереженої площі ланцюга додати площу трикутника, утвореного першою і останньою точками зі стека, а також новою точкою.

Ми вже вміємо додавати точки до верхнього ланцюга і рахувати після цього його площу. Але описаний вище алгоритм не дозволяє видаляти точки з множини. А саме, при додаванні нової точки ми видаляємо декілька останніх точок зі стека. Тепер при операції видалення ми маємо знову повернути ці точки до стека. Для цього заведемо ще один стек, куди ми будемо перекладати точки при видаленні їх з першого стека. Якщо потрібно повернути декілька видалених точок в перший стек, то ми можемо взяти їх із другого. Потрібно лише для кожного запиту додатково запам'ятати розмір другого стека для того, щоб ми знали, скільки точок потрібно повернути до верхнього ланцюга. Цей алгоритм може обробляти всі

типи операцій і має часову складність $O(NM)$, де M — це кількість рядків вхідного файлу, у яких просять видалити декілька останніх точок. Набирає такий розв'язок 50 балів.

Розв'язок на 100 балів Для отримання повного розв'язку потрібно зрозуміти, яке місце є найповільнішим у цьому алгоритмі. Розглянемо наступну ситуацію: нехай в нас є побудований верхній ланцюг якоїсь множини точок (зберігається у стеку). Після цього ми додаємо нову точку, яка лежить набагато вище всіх інших, і тому зі стека видаляються усі точки, крім першої. Тепер ми видаляємо цю точку, і всі ті точки знов перекладаються з одного стека в інший. Отже, за 2 операції алгоритм зробив $O(N)$ дій. Якщо їх повторювати багато разів, то складність буде $O(N^2)$.

Замінимо в нашому розв'язку стек звичайним масивом, для якого будемо додатково пам'ятати кількість точок, яка належить верхньому ланцюгу (розмір стека). При додаванні нової точки знайдемо бінарним пошуком кількість точок, яку потрібно видалити зі стека перед тим, як покласти цю нову точку до нього. Замість перекладання усіх цих точок у другий стек ми покладемо туди лише розмір верхнього ланцюга, а також всю інформацію про останню з точок, що видаляються (тобто її номер у масиві, її координати, а також площа многокутника, пов'язаного з нею). Після цього ми замінимо цю точку на нову, перерахуємо для неї площу і змінимо розмір ланцюга. При цьому всі інші точки, які мали бути видалені, залишаються в масиві без змін (проте вони знаходяться за межами ланцюга). При видаленні останньої точки нам достатньо змінити лише розмір ланцюга і замінити одну точку на ту, що зберігається у другому стеку. Складність такого розв'язку — $O(N \log N)$.