Завдання XXVII Всеукраїнської учнівської олімпіади з інформатики

2013/14 н. р.

Віталій Бондаренко, Данило Мисак, Шаміль Ягіяєв

Зміст

Завдання першого туру

1.1. **Шоколад** (Данило Мисак)

Умова 2

Розв'язання 3

1.2. **Відрізки** (Роман Рубаненко, Роман Фурко)

Умова 4

Розв'язання 5

1.3. **Торговельний центр** (Роман Єдемський)

Умова 6

Розв'язання 7

1.4. **Граф зсередини** (Данило Мисак)

Умова 9

Розв'язання 12

Завдання другого туру

2.1. **Максимум** (Данило Мисак)

Умова 15

Розв'язання 16

2.2. **Числові операції** (Данило Мисак)

Умова 17

Розв'язання 18

2.3. **Декартівщина** (Роман Єдемський)

Умова 21

Розв'язання 23

2.4. **Археологи** (Ярослав Твердохліб)

Умова 25

Розв'язання 27

Завдання першого туру

1.1. Шоколад (Данило Мисак)

Умова

Шоколад, який виробляє кондитерська компанія «Ням & Ням», має вигляд довгої плитки $1 \times N$, що складається з N квадратиків. На кожному квадратику зображено одного з N відомих кондитерів, причому на різних шоколадках, які виробляє компанія, зображені одні й ті самі N кондитерів, але в різному порядку.

Завдання. Напишіть програму `chocolate`, що для заданого порядку портретів на двох плитках шоколаду визначає, на яку найменшу кількість частин доведеться розламати першу плитку, щоб шляхом переставляння частин з неї можна було утворити другу. Ламати плитку можна тільки по межах квадратиків, а перегортати плитку чи її частини не можна.

Вхідні дані. Перший рядок вхідного файла `chocolate.dat` містить натуральне число N ($2 \leqslant N \leqslant 10^5$), що задає розмір плитки шоколаду, тобто кількість у ній квадратиків. Усіх кондитерів занумеровано числами від 1 до N. Другий та третій рядки файла містять по N різних натуральних чисел кожен (усі числа не перевищують N) — порядок портретів відповідно на першій і на другій плитках. Відомо, що ці порядки різні.

Вихідні дані. У вихідний файл `chocolate.sol` слід записати єдине число — найменшу кількість частин, на які доведеться розламати першу плитку так, щоб, якимось чином переставивши частини місцями, отримати порядок портретів на другій плитці.

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 20 балів: $2 \leqslant N \leqslant 3$.
- 20 балів: $3 < N \leqslant 6$.
- 30 балів: $6 < N \leqslant 1000$.
- 30 балів: $1000 < N \leqslant 10^5$.

Приклад вхідних та вихідних даних.

chocolate.dat	chocolate.sol
5 4 3 2 5 1 1 2 5 3 4	4

Пояснення. Першу плитку можна розламати на такі чотири частини: квадратик 4, квадратик 3, два квадратики 2 і 5, квадратик 1. Потрібним чином переставивши ці частини місцями, отримаємо такий самий порядок портретів, як і на другій плитці. Водночас жоден спосіб розламування на три або дві частини не дозволить скласти такий самий порядок портретів, як на другій плитці.

Розв’язання

Розгляньмо довільні два сусідніх квадратики першої плитки. Якщо на другій плитці вони не стоять поряд, причому в такому ж порядку, то в місці з’єднання цих квадратиків першу плитку так чи інакше доведеться розламати. Разом із тим, розламавши плитку в усіх таких місцях і лише в них, одержимо частини, кожна з яких є фрагментом послідовних квадратиків другої плитки. Тому, склавши їх правильним чином, одержимо той самий порядок портретів, що й на другій плитці. Отже, нам залишилося для кожного числа з’ясувати, чи на першій і на другій плитці перед ним іде одне й те саме значення, чи ні. Щоб це зробити, можна для всіх чисел проводити окремий пошук по одній чи обох плитках (це дасть складність $O(N^2)$). Або ж за один лінійний прохід створити для однієї з плиток індексний масив $P[i]$, де для кожного i, $1 \leqslant i \leqslant N$, зберігати число, що на даній плитці йде перед i (чи, скажімо, 0, якщо i — перше число на плитці). Далі, проходячи зліва направо по другій плитці, уже не доведеться щоразу здійснювати пошук відповідного числа на першій. Таким чином, алгоритм відпрацює за $O(N)$ часу.

1.2. Відрізки (Роман Рубаненко, Роман Фурко)

Умова

Петрик дуже любить іграшки у формі геометричних фігур. Нещодавно він помітив, що серед його іграшок немає жодного трикутника. Це дуже засмутило Петрика, тому він пішов до найближчого магазину, щоб придбати новісінький трикутник. В магазині Петрику сказали, що всі трикутники вже давно розкупили, але в наявності є N прямих відрізків.

Відрізки пронумеровані послідовними натуральними числами, починаючи з одиниці. Відрізок номер i характеризується двома числами — довжиною L_i та ціною C_i. Петрик дуже розумний, тому знає, що бажаний трикутник він може скласти з трьох відрізків. Більше того, наш герой знає, що трикутник можливо скласти лише з таких відрізків, що довжина будь-якого з них має бути строго меншою за сумарну довжину інших двох. Отже, хлопчик вирішив придбати рівно три таких відрізки. Звичайно, він хоче заощадити якомога більше коштів на морозиво, тому хоче витратити якнайменше на покупку відрізків для свого трикутника.

Завдання. Напишіть програму `segments`, яка за інформацією про відрізки визначить мінімальну вартість трьох відрізків, з яких хлопчик зможе скласти трикутник, або визначить, що це зробити неможливо.

Вхідні дані. В першому рядку вхідного файла `segments.dat` записано єдине число N — кількість відрізків. Далі в N рядках записана інформація про самі відрізки. Кожен такий рядок містить відповідні L_i ($1 \leqslant L_i \leqslant 10^9$) та C_i. Ціни утворюють перестановку чисел від 1 до N, тобто є попарно різними натуральними числами, не більшими за N.

Вихідні дані. Вихідний файл `segments.sol` має містити єдине число — мінімальну вартість трьох відрізків, з яких можна скласти трикутник, або «-1» (лапки для наочності) в тому випадку, якщо вибрати рівно три такі відрізки неможливо.

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 20 балів: $1 \leqslant N \leqslant 100$.
- 20 балів: $100 < N \leqslant 3000$.
- 30 балів: $3000 < N \leqslant 10^4$.
- 30 балів: $10^4 < N \leqslant 10^5$.

Приклади вхідних та вихідних даних.

segments.dat	segments.sol
4 1 1 2 2 3 3 4 4	9
3 3 1 5 3 10 2	-1

Розв'язання

Розглянемо **найпростіший алгоритм розв'язання** задачі. Перебиратимемо всі трійки відрізків. Для кожної трійки треба перевірити можливість існування відповідного трикутника. Якщо трикутник існує, підрахуємо сумарну вартість трьох відрізків, а з усіх знайдених вартостей виберемо найменшу. Складність алгоритму — $O(N^3)$.

А от **швидше розв'язання**. Розглядатимемо всі відрізки в порядку від найкоротших до найдовших і підтримуватимемо таку структуру даних: масив F, у комірці $F[c]$ якого зберігається найбільша сумарна довжина двох відрізків (з числа перших k відрізків) із загальною вартістю c. Оскільки ціни відрізків не перевищують N, сумарна вартість двох відрізків не буде більшою за $2N$. Переходячи до чергового ($(k+1)$-го) відрізка — нехай цей відрізок має довжину L та вартість C — ми шукаємо найменше таке значення c, для якого $F[c] > L$. Це можна зробити простим лінійним пошуком. Для знайденого значення c сума $c + C$ буде найменшою можливою вартістю трикутника, найбільшою стороною якого є розглядуваний відрізок. Якщо ця сума менша за поточний мінімум знайдених вартостей, відповідним чином цей мінімум оновлюємо. Крім того, оновлюємо і сам масив $F[c]$. Для цього пробуємо взяти в пару з $(k+1)$-м відрізком почергово всі відрізки від першого до k-го і оновлюємо, якщо потрібно, відповідну комірку масиву F. Таким чином, на кожному з N кроків ми виконуємо $O(N)$ операцій, тож загальна складність алгоритму — $O(N^2)$.

Тепер розглянемо **оптимальне розв'язання**. Воно базується на такій нескладній властивості:

Властивість. Якщо серед натуральних чисел L_1, L_2, ..., L_k немає трьох таких, що утворюють сторони трикутника, то найбільше з цих чисел не може бути меншим за k-й член послідовності Фібоначчі. Послідовність Фібоначчі задається таким чином: $F_1 = F_2 = 1$, $F_k = F_{k-1} + F_{k-2}$, $k \geqslant 3$.

Це твердження можна довести методом математичної індукції, попередньо впорядкувавши числа L_1, L_2, ..., L_k за неспаданням.

Оскільки $F_{45} > 10^9$ (у чому можна переконатися, написавши спеціальну «дослідницьку» програму), серед будь-яких 45 натуральних чисел, які не перевищують 10^9, обов'язково знайдуться три, що є сторонами трикутника. Отже, зокрема і серед 45 найдешевших відрізків знайдуться три, що утворюють трикутник. Але ціни відрізків — перестановка чисел від 1 до N, тому сумарна ціна цих трьох відрізків не може перевищувати $45 + 44 + 43 = 132$. Таким чином, усі відрізки вартістю більше за 132 можна відкинути й шукати потрібну трійку лише серед тих, що залишилися. Маємо фактично лінійне від N розв'язання з додатком, що складає порядку 132^3 операцій.

1.3. Торговельний центр (Роман Єдемський)

Умова

У країні Олімпії вирішили побудувати великий торговельний центр. Задля цього було виділено квадратну ділянку $N \times N$ метрів. Існують певні обмеження щодо висоти будівлі. А саме: якщо розбити схему ділянки на вертикальні та горизонтальні смуги шириною 1 метр, то будівля в межах однієї смуги буде мати певне своє обмеження на висоту. Архітектори бажають побудувати торговельний центр у формі прямокутного паралелепіпеда.

Завдання. Напишіть програму `shopping`, що за даними про розмір ділянки та обмеження по висоті по кожній зі смуг знайде максимальний об'єм будівлі у формі прямокутного паралелепіпеда, яку можна збудувати на цій ділянці.

Вхідні дані. Вхідний файл `shopping.dat` складається з трьох рядків. У першому рядку міститься натуральне число N ($2 \leqslant N \leqslant 5 \cdot 10^4$) — розмір ділянки. У другому рядку записано N невід'ємних цілих чисел, жодне з яких не перевищує 10^5, — обмеження висоти на вертикальних смугах. Третій рядок також містить N невід'ємних цілих чисел, жодне з яких не перевищує 10^5, — обмеження висоти на горизонтальних смугах.

Вихідні дані. Вихідний файл `shopping.sol` повинен містити єдине число — максимальний об'єм будівлі торговельного центру, яку можна збудувати на описаній ділянці. Вхідні дані гарантують можливість збудувати на ділянці будівлю ненульового об'єму.

Оцінювання. Набір тестів складається з 5 блоків, для яких додатково виконуються такі умови:

- 10 балів: $1 \leqslant N \leqslant 10$.
- 10 балів: $10 < N \leqslant 30$.
- 10 балів: $30 < N \leqslant 70$.
- 30 балів: $70 < N \leqslant 1000$.
- 40 балів: $1000 < N \leqslant 5 \cdot 10^4$.

Приклади вхідних та вихідних даних.

shopping.dat	shopping.sol
3 0 1 0 0 2 0	1
3 3 2 1 1 2 3	9

Розв'язання

Нехай паралелепіпед займає вертикальні смуги з x_1-ї до x_2-ї та горизонтальні смуги з y_1-ї до y_2-ї. Тоді його максимально можлива висота дорівнює найменшому з мінімумів на цих смугах. Мінімум серед смуг з x_1-ї до x_2-ї позначатимемо через $m(x_1, x_2)$.

Тривіальний підхід з перебором усіх варіантів четвірок (x_1, x_2, y_1, y_2) з безпосереднім підрахунком найбільшої можливої висоти для кожної четвірки набирає 20 балів. Ще 10 балів можна дібрати, знаходячи для кожної четвірки найбільшу можливу висоту відразу, без перебору чисел. Це можна зробити, наприклад, підрахувавши й запам'ятавши мінімуми для кожної пари (x_1, x_2) та для кожної пари (y_1, y_2) заздалегідь.

Тепер дамо таке означення: відрізок $[x_1, x_2]$ називатимемо *цікавим*, якщо не існує жодного такого відрізка $[x_1', x_2']$, що повністю містить відрізок $[x_1, x_2]$, але не збігається з ним, для якого $m(x_1', x_2') = m(x_1, x_2)$. Інакше кажучи, неможливо розширити відрізок $[x_1, x_2]$, не змінивши на ньому мінімального значення. Аналогічним є визначення цікавого відрізка $[y_1, y_2]$.

Зауважимо таке: якщо деякий прямокутний паралелепіпед має найбільший можливий об'єм і займає смуги з x_1-ї до x_2-ї та з y_1-ї до y_2-ї, то відрізки $[x_1, x_2]$ та $[y_1, y_2]$ цікаві. Якби хоча б один з них не був цікавим, ми б розширили його, не змінивши мінімуму, й отримали би паралелепіпед ще більшого об'єму.

Нехай мінімум деякого цікавого відрізка $[x_1, x_2]$ досягається для смуги i $(x_1 \leqslant i \leqslant x_2)$. Тоді смуги з x_1-ї до x_2-ї мають обмеження, не менші за смугу i, а от $(x_1 - 1)$-ша та $(x_2 + 1)$-ша смуги, якщо такі є, мають за означенням цікавого відрізка менші, ніж смуга i, обмеження. Тому цікавих відрізків однієї орієнтації не більше ніж N (по одному для кожної потенційної смуги-мінімуму) і знайти їх можна, відшукавши для кожної потенційної смуги-мінімуму i найбільше число $l_i < i$ таке, що обмеження на l_i-й смузі менше, ніж на i-й, а також найменше число $r_i > i$ таке, що обмеження на r_i-й смузі менше, ніж на i-й. Якщо відповідних смуг немає, вважаємо, що $l_i = 0$, $r_i = N + 1$ (при цьому нумерація самих смуг починається з одиниці і закінчується числом N). Цікавий відрізок з мінімумом на смузі i утворюватиме пара $[l_i + 1, r_i - 1]$.

Якщо на цьому етапі визначити l_i та r_i для всіх i та обох орієнтацій безпосереднім перебором, розв'язок заробить 60 балів. Повний бал дозволяє набрати такий лінійний алгоритм знаходження значень l_i (значення r_i рахуються аналогічно):

1. Заводимо стек, що в кожен момент часу міститиме число 0 та деяку підпослідовність індексів масиву обмежень однієї з орієнтацій.
2. Кладемо в порожній стек число 0.
3. Розглядаємо індекси i масиву обмежень в порядку від найменшого (1) до найбільшого (N) та для кожного робимо таке:
    a. Доки верхній елемент стека більший за 0, а обмеження, записане у відповідній комірці, не менше за обмеження, записане в i-й комірці, видаляємо зі стека верхній елемент.
    b. Тепер l_i — це верхній елемент стека (тобто перший елемент, який ми не відкинули).
    c. Додаємо у стек саме число i.

Лишається ефективно знайти пару цікавих відрізків, що дає найбільший об’єм паралелепіпеда. Для цього спільно (в одному масиві) відсортуємо всі цікаві відрізки обох орієнтацій за величиною їхніх мінімумів. Будемо розглядати відрізки в порядку від відрізка з найбільшим мінімумом до відрізка з найменшим мінімумом. На кожному кроці пам’ятатимемо, відрізок якої найбільшої довжини нам на даний момент трапився серед відрізків горизонтальної орієнтації та який найдовший відрізок нам трапився серед відрізків вертикальної орієнтації. Для кожного нового розглядуваного відрізка рахуємо добуток його мінімуму, його довжини та довжини найдовшого відрізка протилежної орієнтації, який нам трапився. З усіх таких значень вибираємо найбільше.

Складність алгоритму — $O(N \log N)$, більшість часу йде на сортування цікавих відрізків.

1.4. Граф зсередини (Данило Мисак)

Умова

Якось Ліса Сімпсон, персонаж мультсеріалу «Сімпсони», опинилася в одній з вершин деякого зв'язного неорієнтованого графа. Ліса може пересуватися між вершинами графа, які сполучено ребром, однак, окрім вершини, у якій вона на даний момент перебуває, а також усіх суміжних з нею вершин, дівчині нічого не видно. До того ж у Ліси нема з собою ручки та паперу, щоб занотовувати маршрут своїх пересувань або іншу інформацію. Зате вона має необмежену кількість камінців, які дівчина може залишати у вершинах графа, і саме цим вона планує скористатися, щоб дослідити деякі його властивості. На жаль, в одній вершині графа поміщається не більше ніж 1000 камінців, але Ліса вірить, що цього їй вистачить. Відомо, що кількість вершин у графі не менша за 2 і не більша за 500.

Завдання. Дана задача складається з двох підзадач:

- У першій підзадачі у жодній вершині графа спочатку немає камінців, а Ліса хоче з'ясувати, скільки вершин містить граф.
- У другій підзадачі в одній з вершин графа (відмінній від початкової) лежить один камінець, а в решті вершин — жодного. Ліса хоче з'ясувати, якою є довжина найкоротшого шляху по ребрах між початковою вершиною і тією, де лежить камінець (якщо вважати довжину кожного ребра графа одиничною).

Напишіть процедуру `graph`, що за інформацією про кількість камінців у поточній вершині графа та всіх суміжних з нею вершинах визначає наступну дію Ліси: скільки камінців залишити в поточній вершині (або забрати з неї) і в яку з суміжних вершин піти далі; або, коли Ліса готова назвати відповідь на питання підзадачі, дає цю відповідь.

Деталі реалізації. Ви маєте надіслати файл, що містить реалізацію процедури `graph`, детально описану нижче, та, за потреби, інший код, необхідний для коректної роботи цієї процедури, але не містить самого тіла програми (тобто функції `main` у C++ або блоку `begin/end.` у Pascal). При тестуванні ваш файл буде доповнено спеціальним тілом програми, написаним журі. Тіло відповідним чином викликатиме написану вами процедуру та перевірятиме коректність алгоритму, який вона втілює.

Реалізована вами процедура повинна мати такий вигляд:

C++:
```
void graph(int subproblem, int neighborsCount, int neighbors[],
           int &current, int &whereToGo, int &answer);
```

Pascal:
```
procedure graph(subproblem, neighborsCount: longint;
                var neighbors: array of longint;
                var current, whereToGo, answer: longint);
```

Параметри процедури:

- `subproblem` — номер підзадачі: 1 для підзадачі визначення кількості вершин, 2 для підзадачі пошуку довжини найкоротшого шляху. Номер підзадачі не змінюється під час запусків процедури на одному й тому ж графі.

- `neighborsCount` — кількість суміжних вершин, тобто кількість вершин, сполучених з поточною вершиною ребром.
- `neighbors` — масив, що містить рівно `neighborsCount` елементів (індексація з нуля) — кількості камінців у суміжних вершинах. Зверніть увагу, що порядок суміжних вершин може бути різним (довільним чином переставленим) під час різних викликів процедури, навіть якщо поточна вершина є однаковою. Вміст цього масиву процедура в разі потреби може змінювати, однак на реальній кількості камінців у відповідних вершинах це ніяк не позначається: Ліса може змінювати кількість камінців виключно в тій вершині, де вона на даний момент перебуває.
- `current` — кількість камінців у поточній вершині. Цю кількість у процедурі можна змінити (як збільшити, так і зменшити, але так, щоб кількість не стала від'ємною або більшою за 1000). При цьому зміниться й реальна кількість камінців у відповідній вершині графа.
- `whereToGo` — початкове значення цього параметра (аргумент, який передає тіло програми) дорівнює -1. Якщо Ліса продовжує дослідження графа, це значення слід переписати, зробивши рівним кількості камінців у вершині, куди Ліса повинна піти далі (зверніть увагу: це не індекс в масиві `neighbors`, а значення!). Якщо в масиві `neighbors` є відразу кілька елементів, що мають таке значення, то Ліса піде в одну з відповідних вершин, але те, в яку саме, процедура обмежити не може. Якщо Ліса на даному виклику процедури вже готова дати відповідь, початкове значення -1 цього параметра переписувати не потрібно (і не можна).
- `answer` — початкове значення цього параметра (аргумент, який передає тіло програми) також дорівнює -1. Якщо Ліса готова дати відповідь, це значення слід переписати, зробивши його рівним відповіді. Якщо Ліса на даному виклику процедури ще не готова дати відповідь, початкове значення -1 цього параметра переписувати не потрібно (і не можна).

На кожному кроці процедура повинна визначати дії дівчини, **виходячи виключно з даних, переданих їй на цьому кроці**. На результат роботи процедури жодним чином не повинна впливати попередня взаємодія тіла програми та процедури. Крім того, процедура повинна бути детермінованою, тобто для сталих вхідних даних завжди повертати одні й ті самі значення.

Власноручне тестування. В електронному варіанті умов наведено приклад основного тіла програми `graph_tester.cpp`/`graph_tester.pas`, що запускає процедуру `graph` на графі, заданому у створеному вами вхідному файлі, і виводить відповідь, повернену процедурою, у вихідний файл. Щоб використати цю програму, розташуйте її в одному каталозі з файлом `graph.cpp`/`graph.pas`, який містить вашу реалізацію процедури, та створіть у цьому ж каталозі файл `graph.dat` зі структурою, описаною в наступному абзаці. Зауважте, що основне тіло програми, яке буде використано для оцінювання вашої процедури, відрізнятиметься від наданого вам у `graph_tester.cpp`/`graph_tester.pas` прикладу.

Вхідні дані `graph_tester`. Перший рядок містить чотири цілих числа. Перші три — кількість N вершин графа, кількість M ребер графа та номер початкової вершини. При цьому вважаємо, що вершини графа занумеровано натуральними числами від 1 до N. Четверте число дорівнює 0, якщо ви тестуєте процедуру на першій підзадачі, або дорівнює номеру вершини, в якій ле-

жатиме один камінець, якщо ви тестуєте процедуру на другій підзадачі. Наступні M рядків файла задають ребра графа: по два числа в рядку у довільному порядку — номери сполучених ребром вершин.

Вихідні дані `graph_tester`. Єдиний рядок вихідного файла міститиме відповідь, яку після дослідження графа повернула процедура, або словесну діагностику помилки в разі, якщо на якомусь кроці процедура порушила технічні вимоги.

Приклади вхідних та вихідних даних `graph_tester`.

graph.dat	graph.sol
7 8 3 0 3 1 3 7 1 5 1 6 1 2 7 4 5 7 1 4	7
7 8 3 4 3 1 3 7 1 5 1 6 1 2 7 4 5 7 1 4	2

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 30 балів: номер підзадачі — 1, граф є деревом.
- 20 балів: номер підзадачі — 1, граф не обов'язково є деревом.
- 10 балів: номер підзадачі — 2, граф є деревом.
- 40 балів: номер підзадачі — 2, граф не обов'язково є деревом.

Зауваження. Якщо під час тестування на сервері написана вами процедура порушить описані вимоги, діагностикою може бути зокрема помилка виконання (runtime error) або помилка компіляції. Обмеження на пам'ять у цій задачі явно не задано, але ви можете сподіватися принаймні на 16 МБ. Користування глобальними змінними не забороняється.

Розв'язання

Для зручності, коли Ліса залишає в деякій вершині певну кількість камінців, казатимемо, що ми *записали* або *поставили* в цій вершині відповідне число.

Першу підзадачу можна розв'язувати симулюванням пошуку в глибину. У випадку, **якщо граф є деревом**, спрацює такий алгоритм. У початкову вершину (корінь дерева пошуку) ставимо число 500, після чого «спускаємося»: ідемо в першу ліпшу вершину, ставимо там число 499, ідемо з неї в першу ліпшу вершину з нулем камінців і ставимо там 498 і т. д., поки не станеться так, що в деякої вершини немає нульових сусідів. У такій вершині ставимо 1 і «піднімаємося»: йдемо у суміжну вершину з найменшою кількістю камінців. З тієї вершини продовжуємо спуск, якщо є куди спускатися (тобто якщо у вершини ще залишився хоча б один нульовий сусід). Коли дійдемо до вершини без нульових сусідів, записуємо туди 1 і піднімаємося та продовжуємо аналогічні дії. Коли станеться так, що у вершини, куди ми піднялися, більше немає нульових сусідів, у неї ми записуємо збільшену на 1 суму всіх сусідів, де кількість камінців менша, ніж у даній, — фактично кількість вершин у піддереві з коренем у даній вершині. Після цього піднімаємося ще на рівень вище (у вершину з найменшою кількістю камінців, більшою за кількість камінців у даній вершині). Якщо далі підніматися нікуди, тобто ми повернулися в початкову вершину (корінь), даємо відповідь — це число, записане в дану вершину.

Якщо граф не є деревом, цей алгоритм дасть неправильну відповідь, адже на етапі підрахунку суми вершин на дочірніх піддеревах інколи будуть враховуватися зайві числа, що стоять у вершинах, які не є безпосередніми дочірніми вершинами (в дереві пошуку) даної вершини, але тим не менше з нею суміжні і містять менші, ніж у ній, числа. Щоб це виправити, всі вершини в графі, що розташовані на два або більше рівнів глибше за поточну вершину, маємо «відкидати» — заповнювати спеціальним значенням (наприклад, числом 1000). Для цього щоразу після підрахунку суми чисел у дочірніх вершинах повертаємося почергово в кожну з цих вершин і оновлюємо значення в ній до числа 1000. Щоб знати, чи потрібно на даному кроці оновлювати число у вершині до 1000, чи просто піднятися вгору, не змінюючи числа, нам доведеться запровадити додатковий стан батьківської вершини: якщо батьківська вершина перебуває у звичайному стані, ми просто піднімаємося вгору, а якщо у спеціальному, то замінюємо число на 1000. Стан самої батьківської вершини ми змінюємо зі звичайного на спеціальний, щойно підрахуємо для неї суму вершин в дочірніх піддеревах, а назад зі спеціального стану у звичайний вершина переходить, коли в усіх її нащадках уже записано 1000 і треба підніматися вгору. Спеціальний стан можна закодувати, наприклад, шляхом додавання до кількості камінців у вершині числа 499. Тоді з числа, записаного у вершині, можна однозначно встановити і те, в якому стані вона перебуває, і те, яка насправді кількість камінців повинна бути в цій вершині.

Можна показати, що загальний час виконання програми з процедурою, що втілює даний алгоритм, на графі з N вершинами складає $O(N^2)$.

Другу підзадачу для **випадку дерев** можна, як і першу підзадачу, розв'язати симуляцією пошуку в глибину, адже найкоротший шлях між двома вершинами в дереві збігається з глибиною однієї з вершин у дереві пошуку з коренем у другій. У даному випадку нам достатньо пам'ятати лише глибину кожної вершини, а підраховувати суму вершин на піддеревах, як ми це робили у першій підзадачі, не потрібно.

Якщо граф не є деревом, знайти довжину найкоротшого шляху стає значно складніше. Для цього доведеться провести кілька ітерацій, деталі реалізації яких буде описано далі. На нульовій ітерації ми ставимо в початкову вершину число 2; на першій ітерації ставимо в усі вершини, суміжні з початковою, число 3, а після ітерації опиняємось у початковій вершині; на другій ітерації в усі вершини, найкоротший шлях у які з початкової вершини має довжину 2, записуємо число 4, а після ітерації знов опиняємось у початковій вершині і т. д. Інакше кажучи, маємо таку ситуацію: перед k-ю ітерацією в усіх вершинах, до яких можна дійти з початкової вершини менше ніж за k кроків, уже стоїть відповідне число (довжина найкоротшого шляху, збільшена на 2), а на k-й ітерації в усі вершини, найкоротший шлях у які має довжину рівно k, проставляємо число $k+2$, після чого повертаємось у початкову вершину і відразу переходимо до $(k+1)$-ї ітерації. Додамо, що, для того щоб мати змогу здійснювати такі ітерації, нам, як і в попередній підзадачі, доведеться ввести два стани для вершин. Після парних ітерацій усі вершини матимуть один стан, а після непарних — інший. При цьому стани зручно кодувати так: для одного стану кількість камінців у вершині саме така, як і має бути, а для іншого стану додаємо до оригінальної кількості камінців число 500. Таке кодування дає за кількістю камінців у вершині однозначно встановити і її стан, і оригінальну кількість камінців.

Тепер опишемо, як здійснювати ітерацію. Від початкової вершини ми «спускаємось» до щораз на одиницю більших вершин (від 2 до 3, від 3 до 4 і т. д.), поки не дійдемо до кінця такого ланцюжка. Тоді, якщо у вершини, куди ми прийшли, є сусідня нульова вершина, ідемо туди, записуємо в нею на одиницю більше число, причому в стані, протилежному до даного, після чого повертаємося назад. Здійснюємо таку операцію з усіма суміжними нулями. Далі маємо ситуацію, коли всі суміжні вершини, в яких записано число, що є більшим, ніж у даній вершині, мають протилежний до даної вершини стан. У такій ситуації ми змінюємо стан поточної вершини на протилежний та «піднімаємося»: переходимо від поточної вершини до тієї, в якій записано на одиницю менше число у тому стані, в якому перебувала поточна вершина раніше (зауважимо, що може вийти так, що ми підніемемося не в ту саму вершину, з якої свого часу спустилися в дану, а в іншу вершину з тією самою кількістю камінців). Далі спускаємося в усі вершини, які мають той самий стан, що й поточна, але на один більшу кількість камінців, а дійшовши до кінця кожного такого ланцюжка, спускаємося в нулі та відразу ж піднімаємося з нулів назад і т. д. У підсумку, спустившись по всіх напрямках, додавши новий рівень вершин та піднявшись назад, ми опинимося в початковій вершині, причому всі суміжні з нею вершини матимуть протилежний порівняно з нею стан. Це означатиме, що треба змінити стан самої початкової вершини та розпочати нову ітерацію.

Щоб знайти відповідь, під час визначення того, в яку вершину далі переходити, зручно вважати вершину з одним камінцем нульовою, але, коли один камінець опиняється в поточній вершині, замість того щоб записувати в неї потрібне число, відповідне число (зменшене на 2) потрібно повернути як відповідь.

Можна показати, що загальний час виконання програми з процедурою, що втілює даний алгоритм, на графі з N вершинами складає $O(N^3)$.

Пропонуємо читачу самостійно довести правильність наведеного алгоритму, яка, зважаючи на можливі перескакування з одного піддерева на інше, не є очевидною.

Насамкінець додамо, що цікавими можуть виявитися спроби розв’язати задачу, якщо кількість камінців, які дозволено розміщувати у вершині, менша за подвоєну максимальну кількість вершин у графі. Наприклад, чи завжди вдасться знайти відповідь, якщо в графі може бути до 500 вершин і камінців також можна розмістити щонайбільше по 500 (або трохи більше) у кожній вершині? Автору вдалося розв’язати з таким обмеженням другу підзадачу. А от чи допускає розв’язання з цим обмеженням перша підзадача, залишається відкритим питанням. Прохання надсилати свої ідеї з цього приводу автору: danmysak@gmail.com.

Завдання другого туру

2.1. Максимум (Данило Мисак)

Умова

Назвемо фрагментом числової послідовності будь-яку непорожню підпослідовність цієї послідовності без пропусків. Наприклад, фрагментами послідовності чисел $1, 7, 3$ є сама послідовність $1, 7, 3$, її двоелементні підпослідовності $1, 7$ та $7, 3$ (але не підпослідовність $1, 3$), а також три одноелементні підпослідовності $1, 7$ і 3.

Завдання. Напишіть програму `maximum`, що для послідовності чисел і величини M визначить, скільки існує фрагментів заданої послідовності, максимум на яких дорівнює M. Фрагменти, що містять однакові числа, але розташовуються в різних місцях послідовності, ми вважаємо різними.

Вхідні дані. У першому рядку вхідного файла `maximum.dat` записано два натуральних числа: N ($2 \leqslant N \leqslant 10^5$) — довжину послідовності чисел — та M ($1 \leqslant M \leqslant 10^9$). У другому рядку міститься послідовність з N натуральних чисел, кожне з яких не перевищує 10^9.

Вихідні дані. Вихідний файл `maximum.sol` повинен містити єдине число — кількість фрагментів послідовності, найбільше число яких дорівнює M.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 25 балів: $2 \leqslant N \leqslant 100$.
- 25 балів: $100 < N \leqslant 1000$.
- 50 балів: $1000 < N \leqslant 10^5$.

Приклади вхідних та вихідних даних.

maximum.dat	maximum.sol
4 5 1 5 1 2	6
2 2 3 4	0

Пояснення. У першому прикладі можна взяти такі шість фрагментів з максимумом, що дорівнює п’яти: $(1, 5, 1, 2)$, $(1, 5, 1)$, $(5, 1, 2)$, $(1, 5)$, $(5, 1)$ та (5). У другому прикладі жоден з фрагментів послідовності, очевидно, не матиме максимуму, що дорівнює 2.

Розв’язання

Чверть балів можна набрати шляхом перебору всіх фрагментів і безпосереднім визначенням для кожного з них максимуму. Ще чверть балів можна дібрати, якщо перебирати фрагменти, наприклад, у порядку збільшення лівого краю, а для однакового лівого краю в порядку збільшення правого, при цьому, додаючи до фрагмента новий елемент, користуватися максимумом, знайденим на попередньому кроці.

Щоб заробити повний бал, будемо відштовхуватися від такого зауваження: максимум M мають ті й лише ті фрагменти, на яких є хоча б одне число, що дорівнює M, але немає чисел, більших за M. Таким чином, числа, більші за M, розбивають задану послідовність на фрагменти, для кожного з яких можна незалежно знайти кількість підфрагментів з максимумом M, після чого ці кількості залишиться тільки скласти. Для кожного ж утвореного фрагмента, що не містить чисел, більших за M, максимум, відмінний від M, матимуть усі ті й лише ті підфрагменти, які не містять чисел M. Отже, достатньо ще раз розбити кожен фрагмент на менші фрагменти (тепер уже числами, рівними M) і відповідним чином підрахувати відповідь.

Розглянемо приклад, де $M = 6$:

$$\overbrace{4\,3\,1}^{3}\ \boxed{6}\ \overbrace{2\,2}^{2}\ \boxed{\mathbf{9}}\ \overbrace{1\,4}^{2}\ \boxed{6}\ \overbrace{2}^{1}\ \boxed{6}\ \overbrace{3\,5}^{2}\ \boxed{\mathbf{7}}\ \boxed{\mathbf{8}}\ \overbrace{4\,5\,2\,1}^{4}$$

Тут у рамку взято числа, не менші за $M = 6$, при цьому числа, що більші за M, додатково виділено грубим шрифтом.

Для першого фрагмента (до дев’ятки), що має довжину 6, усього є $1 + 2 + \ldots + 6 = \frac{6\cdot(6+1)}{2} = 21$ підфрагмент. Із них $\frac{3\cdot(3+1)}{2} + \frac{2\cdot(2+1)}{2} = 9$ підфрагментів не містять числа 6, тож у підсумку для цього фрагмента маємо $21 - 9 = 12$ підфрагментів із максимумом 6. Аналогічно для другого фрагмента (між дев’яткою та сімкою) є $\frac{7\cdot(7+1)}{2} - \frac{2\cdot(2+1)}{2} - \frac{1\cdot(1+1)}{2} - \frac{2\cdot(2+1)}{2} = 21$ підфрагмент із максимумом 6. Третій фрагмент розташовується між сімкою та вісімкою і є порожнім, тому має 0 підфрагментів, що задовольняють умову. Нарешті, останній фрагмент теж має $\frac{4\cdot(4+1)}{2} - \frac{4\cdot(4+1)}{2} = 0$ підфрагментів з максимумом 6. Разом дістали $12 + 21 + 0 + 0 = 33$ фрагменти послідовності, найбільше число на яких дорівнює 6.

Зауважимо, що відповідь можна підрахувати за один лінійний прохід, пам’ятаючи на кожному кроці позицію останнього на даний момент пройденого числа, більшого за M, та останнього числа, яке було не меншим за M. При цьому можна додати спереду та ззаду до послідовності чисел два фіктивних значення $M + 1$, які не впливають на відповідь, але полегшують реалізацію (ініціалізацію та фіналізацію) лінійного проходження. Відповідь слід зберігати у змінній 64-бітового типу.

2.2. Числові операції (Данило Мисак)

Умова

На дошці записано число 1. Кожної секунди Петрик може провести з числом одну з двох операцій: або додати до числа 1, або довільним чином переставити цифри числа (але так, щоб на першому місці опинився не нуль). Після цього Петрик витирає з дошки старе число і записує замість нього утворене.

Завдання. Напишіть програму `numbers`, що для заданого натурального числа визначає, за яку найменшу кількість операцій Петрик може, почавши з одиниці, дійти до цього числа.

Вхідні дані. Перший рядок вхідного файла `numbers.dat` містить число T ($1 \leqslant T < 10^4$), що задає кількість чисел у вхідному файлі, для яких треба знайти відповідь. У наступних T рядках записано по одному натуральному числу N_i, $2 \leqslant N_i < 10^9$, $1 \leqslant i \leqslant T$. Відомо, що серед чисел N_i, $1 \leqslant i \leqslant T$, нема однакових.

Вихідні дані. Вихідний файл `numbers.sol` повинен містити T чисел по одному в рядку — в i-му рядку має бути записано найменшу кількість секунд, які знадобиться витратити Петрику, щоб, почавши з одиниці, записати на дошці відповідне число N_i.

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 25 балів: $2 \leqslant N_i < 100$ для всіх i.
- 25 балів: $T = 1$, $100 \leqslant N_1 < 10^4$.
- 15 балів: $T > 1$, $100 \leqslant N_i < 10^4$ для всіх i.
- 35 балів: $10^4 \leqslant N < 10^9$ для всіх i.

Приклад вхідних та вихідних даних.

numbers.dat	numbers.sol
3 2 955 21	1 48 12

Розв'язання

Стандартний підхід до розв'язування такого роду задач — пошук у ширину по числах — дозволяє заробити лише 50 балів, якщо здійснювати пошук для кожного числа у вхідному файлі окремо, або 65 балів, якщо провести єдиний пошук у ширину для всіх чисел. Натомість, щоб заробити повний бал, треба знайти певну математичну закономірність та скористатися нею.

Спершу зробімо таке зауваження: кількість цифр у числі ніколи не зменшується, а збільшитися може лише при додаванні до числа, що складається з самих дев'яток, одиниці. Отже, для досягнення числа N слід спершу дійти до числа 10, потім до чисел 99 і 100, далі до 999 і 1000 і т. д., поки кількість цифр у числі не буде потрібною. Тепер сформулюємо таке твердження:

Лема. Найменша кількість операцій, необхідна для того, щоб перейти від числа $\underbrace{100\ldots0}_{n\text{ цифр}}$ до деякого n-цифрового числа N, дорівнює:

- Якщо число N містить принаймні одну ненульову цифру, крім першої, то кількість операцій складає $O_N = S_N + G'_N - E'_N - 1$, де через S_N позначено суму цифр числа N, через G'_N позначено кількість ненульових цифр на всіх позиціях числа N, крім останньої, а через E'_N — число, що дорівнює 1, якщо хоча б на одній позиції числа N, крім останньої, є одиниця, або 0, якщо жодної одиниці на цих позиціях немає.
- Якщо всі цифри числа N, крім першої, нульові, а перша цифра більша за одиницю, то кількість операцій дорівнює $O_N = S_{N-1} + G'_{N-1} - E'_{N-1}$ (у позначеннях з попереднього пункту). А якщо $N = 10^{n-1}$, то, очевидно, $O_N = 0$.

Доведення. Для $n = 1$ твердження леми можна перевірити безпосередньо. Припустімо тепер, що $n \geqslant 2$ і для якогось n-цифрового числа N мінімальна кількість операцій відрізняється від тієї, яку дає твердження леми. Позначимо справжню кількість операцій через O'_N (на противагу числу O_N, яке дає лема). Покажемо спершу, що $O'_N \leqslant O_N$. Для цього достатньо навести приклад ланцюжка, який за O_N операцій число 10^{n-1} перетворює на N. Але замість прямого порядку дій розглядатимемо еквівалентний зворотний порядок: починаємо з числа N і за одну операцію можемо або зменшити число на 1, або переставити в ньому цифри так, щоб на першому місці опинився не нуль. Кінцева мета — число 10^{n-1}. Діятимемо так:

- Якщо $N = 10^{n-1}$, очевидно.
- Якщо $N \neq 10^{n-1}$ і число N починається з одиниці, спершу віднімемо від N стільки одиниць, щоб остання цифра числа стала нульовою. Далі переставимо місцями останню цифру з будь-якою іншою ненульовою, крім першої, і знову віднімемо кілька одиниць, щоб остання цифра стала нульовою. Повторюватимемо це доти, доки всі цифри, крім першої, не стануть нулями. Очевидно, загалом ми виконаємо $S_N - 1$ операцію віднімання одиниць. Кількість же операцій переставляння цифр дорівнює кількості G''_N ненульових цифр на всіх позиціях, крім першої та останньої. Оскільки $G'_N = G''_N + 1$, а $E'_N = 1$, разом маємо $(S_N - 1) + G''_N = S_N - 1 + G'_N - 1 = S_N + G'_N - E'_N - 1 = O_N$ операцій.
- Якщо число N починається з цифри, більшої за один, але хоча б на одній з позицій числа, крім останньої, є одиниця, зробимо таке. Спершу зменшуватимемо останню цифру, поки вона не стане нульовою. Далі проведемо таку операцію переставляння цифр: пос-

тавимо на місце першої цифри одиницю, першу цифру зробимо останньою, а нуль, який стояв у кінці числа, поставимо на колишнє місце одиниці. Після цього діятимемо згідно з алгоритмом з попереднього пункту. Загалом виконаємо $S_N - 1$ операцію віднімання одиниць та $1 + (G''_N - 1) = G''_N$ операції переставляння цифр. У сумі це знову дає $(S_N - 1) + G''_N = S_N - 1 + G'_N - 1 = S_N + G'_N - E'_N - 1 = O_N$ операцій.

- Нехай тепер на жодній з позицій числа N, крім, можливо, останньої, нема одиниці, але в числі є хоча б одна ненульова цифра, крім першої. Будемо діяти так. Робитимемо останню цифру нульовою і переставлятимемо її з довільною ненульовою цифрою (крім першої) доти, доки на останньому місці не опиниться остання ненульова цифра (не рахуючи першої). Її зменшимо не до нуля, а до одиниці, переставимо місцями з першою цифрою, а вже нову останню цифру зробимо нульовою. Як і раніше, маємо $S_N - 1$ операцію віднімання одиниць. Кількість же операцій переставляння, як неважко зрозуміти, дорівнює G'_N. Тоді, враховуючи, що в даному випадку $E'_N = 0$, загальна кількість операцій складає $(S_N - 1) + G'_N = S_N + G'_N - E'_N - 1 = O_N$.
- Нарешті, якщо перша цифра N більша за один, а всі інші цифри нульові, віднімемо від N одиницю та побудуємо ланцюжок для числа $N - 1$ (що ми вже вміємо робити). Разом матимемо $1 + O_{N-1} = 1 + (S_{N-1} + G'_{N-1} - E'_{N-1} - 1) = S_{N-1} + G'_{N-1} - E'_{N-1} = O_N$ операцій.

Тепер доведемо, що $O'_N \geqslant O_N$. Нехай це не так, тобто $O'_N < O_N$. Розглянемо відповідний найкоротший (довжини O'_N) ланцюжок від числа 10^{n-1} до числа N та перше (найлівіше) число K в ньому, для якого справжня кількість переходів (віддаленість від початку ланцюжка) O'_K менша за число O_K, яке дає лема. Оскільки ми розглядаємо найлівіше число з такою властивістю, для числа, яке йде перед ним (позначимо його через P), кількість операцій коректно визначено лемою: $O'_P = O_P$. Розгляньмо кілька випадків:

- $P = 10^{n-1}$. Тоді $O'_P = O_P = 0$, $O'_K = O'_P + 1 = 1$, $K = P + 1 = 10^{n-1} + 1$, $O_K = 1$, тобто $O'_K = O_K$, що суперечить нашому припущенню $O'_K < O_K$.
- $P = 2 \cdot 10^{n-1}$. Тоді $O'_P = O_P = 10n - 10$, $O'_K = O'_P + 1 = 10n - 9 > 3$. Але маємо, що $K = P + 1 = 2 \cdot 10^{n-1} + 1$, тому $O_K = 3 < O'_K$. Знову дістали суперечність.
- $P = m \cdot 10^{n-1}$, $3 \leqslant m \leqslant 9$. Тоді $O'_P = O_P = 10n - 11 + m$. Міркуючи так само, як раніше, маємо $O'_K = O'_P + 1 = 10n - 10 + m > m + 1$, але, як і в попередньому пункті, $K = P + 1$, а $O_K = m + 1$, звідки $O_K < O'_K$ — суперечність.
- P має ненульові цифри, крім першої, а остання цифра числа P менша за 9. Якщо $K = P + 1$, то

$$O_K = O_{P+1} = S_{P+1} + G'_{P+1} - E'_{P+1} - 1 =$$
$$= (S_P + 1) + G'_P - E'_P - 1 = O_P + 1 = O'_P + 1 = O'_K,$$

що дає суперечність. Інакше K — перестановка цифр числа P, звідки

$$O_K = S_K + G'_K - E'_K - 1 = S_P + (G'_K - G'_P) + G'_P - (E'_K - E'_P) - E'_P - 1 =$$
$$= O_P + (G'_K - G'_P) - (E'_K - E'_P) = O'_P + (G'_K - G'_P) - (E'_K - E'_P) =$$
$$= O'_K + (G'_K - G'_P) - (E'_K - E'_P) - 1.$$

Оскільки за нашим припущенням $O_K > O'_K$, маємо $(G'_K - G'_P) - (E'_K - E'_P) - 1 > 0$, або $G'_K - G'_P > E'_K - E'_P + 1$. Але за побудовою $G'_K - G'_P \leqslant 1$, а $E'_K - E'_P \geqslant -1$, звідки маємо єдиний можливий варіант $G'_K - G'_P = 1$, $E'_K = 0$, $E'_P = 1$. Утім, з першої рівності випли-

ває, що на останньому місці в числа K має стояти нуль, а з інших двох рівностей — що остання цифра числа K — одиниця. Дістали суперечність.

- Остання цифра числа P дорівнює 9, але P має цифри, крім першої, що менші за 9. Якщо K — перестановка цифр числа P, застосовуємо міркування з попереднього пункту (у них той факт, що остання цифра числа була меншою за 9, ніяк не використовувався). Інакше $K = P + 1$. Тоді

$$O_K = O_{P+1} = S_{P+1} + G'_{P+1} - E'_{P+1} - 1 \leqslant (S_P - 8) + (G'_P + 1) - (E'_P - 1) - 1 < \\ < (S_P + G'_P - E'_P - 1) + 1 = O_P + 1 = O'_P + 1 = O'_K.$$

  Дістали суперечність.

- Усі цифри числа P, крім першої, дорівнюють 9. Якщо K — перестановка цифр числа P, застосовуємо міркування з двох попередніх пунктів. Інакше $K = P + 1$.

$$O_K = O_{P+1} = S_P + G'_P - E'_P = (S_P + G'_P - E'_P - 1) + 1 = O_P + 1 = O'_P + 1 = O'_K.$$

  Знов дістали суперечність.

Лему доведено. ■

Наслідок. Щоб перейти від числа 10^{n-1} до числа 10^n, потрібно здійснити $10n - 1$ операцій, а щоб дійти від числа 1 до числа 10^{n-1}, доведеться провести $(5n - 1)(n - 1)$ операцій.

Доведення. Перейти від числа 10^{n-1} до 10^n можна тільки через число $M = 10^n - 1$. За лемою на це знадобиться $O_M = S_M + G'_M - E'_M - 1 = 9n + (n - 1) - 0 - 1 = 10n - 2$ операцій у випадку $n > 1$ або, очевидно, $8 = 10n - 2$ операцій у випадку $n = 1$. У сумі з останньою операцією маємо $(10n - 2) + 1 = 10n - 1$.

Таким чином, щоб перейти від 1 до 10^{n-1}, потрібно

$$\sum_{k=1}^{n-1}(10k - 1) = 10\sum_{k=1}^{n-1} k - (n - 1) = 5n(n - 1) - (n - 1) = (5n - 1)(n - 1)$$

операцій. ■

Відповідь для кожного заданого числа N_i можна підрахувати, скориставшись доведеними лемою та її наслідком.

2.3. Декартівщина (Роман Єдемський)

Умова

Держава Іксівщина складається з N_x міст, деякі пари яких сполучено дорогами з двостороннім рухом. Кожна дорога має свою довжину. Усього міжміських доріг у державі M_x, причому відомо, що з кожного міста Іксівщини можна доїхати по дорогах до кожного іншого міста цієї держави. Міста Іксівщини занумеровано натуральними числами від 1 до N_x.

Держава Ігреківщина складається з N_y міст, деякі пари яких також сполучено дорогами з двостороннім рухом. Кожна дорога має свою довжину. Усього міжміських доріг у державі M_y, причому відомо, що з кожного міста Ігреківщини можна доїхати по дорогах до кожного іншого міста цієї держави. Міста Ігреківщини занумеровано натуральними числами від 1 до N_y.

Держава Декартівщина складається з $N = N_x \cdot N_y$ міст: кожному місту Декартівщини у взаємно однозначну відповідність можна поставити пару міст-побратимів (x, y), де x — місто Іксівщини, а y — місто Ігреківщини. Деякі пари міст Декартівщини також сполучено дорогами з двостороннім рухом. Доріг у державі рівно $M = N_x \cdot M_y + N_y \cdot M_x$. При цьому дорога між містами (x_1, y_1) та (x_2, y_2) існує лише в одному з таких двох випадків:

- Якщо $x_1 = x_2$, а між містами y_1 та y_2 Ігреківщини прокладено дорогу. При цьому довжина дороги між містами (x, y_1) та (x, y_2) Декартівщини дорівнює довжині дороги між містами y_1 та y_2 Ігреківщини.
- Якщо $y_1 = y_2$, а між містами x_1 та x_2 Іксівщини прокладено дорогу. При цьому довжина дороги між містами (x_1, y) та (x_2, y) Декартівщини дорівнює довжині дороги між містами x_1 та x_2 Іксівщини.

Міста різних держав між собою дорогами не сполучені.

Завдання. Дана задача складається з двох підзадач. В обох підзадачах усю інформацію про сполучення дорогами задано у вхідних даних.

- У першій підзадачі потрібно визначити довжину найкоротшого шляху дорогами Декартівщини з міста $(1, 1)$ у місто (N_x, N_y).
- У другій підзадачі деякі дороги Декартівщини потрібно закрити. Ваша задача — визначити, дороги якої найменшої сумарної довжини можна залишити в Декартівщині, щоб із будь-якого її міста досі можна було потрапити в будь-яке інше.

Напишіть програму `cartesius`, що розв'язує ці підзадачі.

Вхідні дані. Перший рядок вхідного файла `cartesius.dat` містить номер підзадачі, яку потрібно розв'язати (1 або 2).

Другий рядок містить натуральні числа N_x та M_x ($1 \leqslant N_x \leqslant 5 \cdot 10^4$, $1 \leqslant M_x \leqslant 5 \cdot 10^4$) — кількість міст і доріг в Іксівщині. У наступних M_x рядках описано дороги Іксівщини: в кожному рядку по три числа, де перші два задають номери різних міст, сполучених дорогою, а третє є довжиною відповідної дороги (натуральне число, що не перевищує 10^7).

У наступному рядку вхідного файла вказано натуральні числа N_y та M_y ($1 \leqslant N_y \leqslant 5 \cdot 10^4$, $1 \leqslant M_y \leqslant 5 \cdot 10^4$) — кількість міст і доріг в Ігреківщині. Наступні M_y рядків містять опис доріг Ігреківщини; формат даних і обмеження збігаються з описаними вище.

Вихідні дані. Вихідний файл `cartesius.sol` повинен містити єдине ціле число — відповідь на питання підзадачі.

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 12 балів: номер підзадачі — 1, жодне з чисел N_x, M_x, N_y, M_y не перевищує 200.
- 28 балів: номер підзадачі — 1, додаткових обмежень немає.
- 12 балів: номер підзадачі — 2, жодне з чисел N_x, M_x, N_y, M_y не перевищує 200.
- 48 балів: номер підзадачі — 2, додаткових обмежень немає.

Приклади вхідних та вихідних даних.

cartesius.dat	cartesius.sol
1 3 2 2 1 15 3 1 14 3 2 2 1 15 3 2 15	44
2 3 2 2 1 15 3 1 14 3 2 2 1 15 3 2 15	117

Розв'язання

Для ефективного розв'язування задачі буде зручно візуалізувати графи доріг трьох країн: розташуймо міста Іксівщини в цілих точках з проміжку $[1, N_x]$ на осі абсцис, міста Ігреківщини — в цілих точках з проміжку $[1, N_y]$ на осі ординат, а міста Декартівщини — у вузлах цілочисельної решітки $[1, N_x] \times [1, N_y]$. Таким чином, місто x Іксівщини матиме координати $(x, 0)$, місто y Ігреківщини матиме координати $(0, y)$, а місто (x, y) Декартівщини — координати (x, y) на декартовій площині. Будь-якому ребру Іксівщини відповідає набір з N_y горизонтальних ребер Декартівщини, що сполучають вершини відповідних стовпців. Аналогічно й будь-якому ребру Ігреківщини відповідає набір з N_x вертикальних ребер Декартівщини, що сполучають вершини відповідних рядків.

Підзадача 1. Для того щоб отримати 12 балів за цю підзадачу, можна безпосередньо побудувати в програмі граф, описаний в умові, та запустити на ньому алгоритм Дейкстри.

Щоб отримати решту 28 балів за цю підзадачу, слід зауважити таку властивість (через $d(a, b)$ позначено найкоротшу відстань від міста a до міста b):

$$d((1, 1), (N_x, N_y)) = d(1, N_x) + d(1, N_y).$$

Це нескладно довести, взявши до уваги, що будь-який шлях з міста (x_1, y_1) в місто (x_2, y_2) Декартівщини розпадається на два шляхи: між містами x_1 та x_2 Іксівщини та між містами y_1 та y_2 Ігреківщини; водночас будь-яку пару шляхів між містами x_1 та x_2 Іксівщини та між містами y_1 та y_2 Ігреківщини можна поєднати у шлях з міста (x_1, y_1) в місто (x_2, y_2) Декартівщини відповідної сумарної довжини.

Підзадача 2. Для того щоб отримати 12 балів за цю підзадачу, можна, як і раніше, безпосередньо побудувати в програмі граф та запустити на ньому алгоритм побудови мінімального кістякового дерева.

Щоб дібрати останні 48 балів, треба промоделювати алгоритм Крускала на графі Декартівщини, але додаючи ребра не по одному, а групами: в одній групі додаємо всі ребра, що сполучають деякі два рядки або деякі два стовпці вершин. Ребра з однієї групи мають однакову довжину, що й дозволяє додавати їх водночас.

1. Відсортуємо спільно ребра першого та другого графа за довжиною.
2. Заведемо дві окремі структури disjoint-set («система множин, що не перетинаються») для множини вершин графа Іксівщини та множини вершин графа Ігреківщини.
3. Перебираємо ребра в порядку від найкоротших до найдовших:
   - Якщо ребро належить графу Іксівщини та сполучає вершини з різних множин (компонент зв'язності):
     - об'єднуємо відповідні множини у disjoint-set Іксівщини — додаємо групу горизонтальних ребер у Декартівщині;
     - додаємо до відповіді довжину ребра, помножену на поточну кількість множин у disjoint-set Ігреківщини.
   - Якщо ребро належить графу Ігреківщини та сполучає вершини з різних множин (компонент зв'язності):

- об’єднуємо відповідні множини у disjoint-set Ігреківщини — додаємо групу вертикальних ребер у Декартівщині;
- додаємо до відповіді довжину ребра, помножену на поточну кількість множин у disjoint-set Іксівщини.

- Якщо ребро сполучає вершини з однієї компоненти зв’язності, ігноруємо його.

Зауважимо, що після додавання чергової групи ми фактично «склеюємо» (тобто ототожнюємо) два рядки або два стовпці, які сполучали ребра з даної групи.

2.4. Археологи (Ярослав Твердохліб)

Умова

Учені-археологи планети Олімпії нещодавно знайшли руїни старовинного міста, яке належало невідомій раніше цивілізації. Все, що збереглося, — це N башт та M стін, кожна з яких сполучає деяку пару башт між собою. У спрощеній моделі башти можна представити точками на площині, а стіни — відрізками, що їх сполучають. Природно, дві стіни не можуть перетинатись усередині, хоча можуть виходити з однієї і тієї ж башти. Для вивчення міста було задіяно K археологів, які розташувались на території міста. У спрощеній моделі їхні положення можна зобразити точками на площині. Жоден з археологів не перебуває на башті або на стіні. На території міста жодного мобільного чи радіозв'язку немає, тому спілкуватись між собою археологи можуть лише при зустрічі. Вважаємо, що два археологи можуть зустрітись одне з одним, якщо вони можуть дійти від своїх початкових позицій до однієї і тієї самої точки площини, при цьому не перетинаючи стін та башт. Від однієї точки до іншої археолог може пересуватись по довільній кривій.

Завдання. Напишіть програму `archaeology`, що за даними про розташування башт, стін та археологів знайде кількість пар археологів, які можуть зустрітися один з іншим.

Вхідні дані. У першому рядку файла `archaeology.dat` містяться три натуральних числа N, M і K ($1 \leqslant N \leqslant 5 \cdot 10^4$, $1 \leqslant M \leqslant 10^5$, $1 \leqslant K \leqslant 5 \cdot 10^4$) — кількість башт, стін та археологів відповідно. Наступні N рядків містять по два цілих числа, кожне з яких не перевищує за абсолютною величиною 10^6 — координати башт. Жодні дві башти не розташовані в одній точці. Наступні M рядків містять по два різних цілих числа — номери башт, які сполучені черговою стіною. Башти нумеруються числами від 1 до N. Гарантується, що дві стіни не будуть мати ніяких інших спільних точок, окрім своїх кінців. Також гарантується, що жодна башта не лежить на жодній стіні, окрім випадку, коли башта є кінцем стіни. Наступні K рядків містять по два цілих числа, кожне з яких не перевищує за модулем 10^6, — координати археологів. Жоден археолог не перебуває на стіні чи на башті.

Вихідні дані. Вихідний файл `archaeology.sol` повинен містити єдине число — кількість пар археологів, які можуть зустрітися один з іншим.

Оцінювання. Набір тестів складається з 6 блоків, для яких додатково виконуються такі умови:

- 15 балів: $2 \leqslant N \leqslant 4$, $1 \leqslant K \leqslant 10$.
- 15 балів: $3 \leqslant N \leqslant 1000$, $1 \leqslant K \leqslant 1000$, кожна башта є кінцем не більше ніж двох стін.
- 20 балів: усі стіни паралельні осям координат, усі координати не перевищують 500 за абсолютною величиною.
- 15 балів: $2 \leqslant N \cdot K \leqslant 10^6$.
- 15 балів: усі стіни паралельні до осей координат.
- 20 балів: немає додаткових обмежень.

Приклад вхідних та вихідних даних.

archaeology.dat	archaeology.sol
4 5 7 0 0 10 0 10 10 0 10 1 2 2 3 3 4 4 1 4 2 1 1 2 2 9 9 8 8 -1 -1 -5 -5 100 100	5

Розв'язання

Існує декілька підходів до розв'язання цієї задачі.

Розв'язання з виділенням граней планарного графа. Башти та стіни утворюють на площині планарний граф. *Гранями* планарного графа будемо називати максимальні за включенням зв'язні області площини, на які вона розбивається даним графом. Один з підходів полягає у виділенні всіх граней та визначення для кожної точки-археолога того, до якої саме грані вона належить. Далі треба просумувати величини $C_i \cdot (C_i - 1)/2$ по всіх гранях, де C_i — кількість археологів у грані i.

Цей підхід можна реалізувати за $O(NK)$ або за $O((N + K) \cdot \log(N + K))$ з використанням збалансованих бінарних дерев. У розборі детально буде описано інший підхід, який має таку ж складність виконання, але є простішим для реалізації.

Розв'язання за допомогою методу скануючої прямої. Спочатку розв'яжемо задачу для випадку, коли всі точки у вхідному файлі мають різні абсциси. Як і у згаданому вище розв'язанні, будемо розбивати площину на зв'язні області і підраховувати кількості точок-археологів у кожній з них, але при цьому не будуватимемо явно границі цих областей.

Відсортуємо всі точки (башти й археологів) за зростанням абсциси. Будемо пересувати вертикальну пряму у напрямі збільшення абсциси. Перетин цієї вертикальної прямої з ребрами планарного графа — це множина вертикальних відрізків. Кожен відрізок належить певній зв'язній області площини, причому деякі відрізки можуть належати одній і тій самій області, попри те що вони не розташовуються поряд на вертикальній прямій. Будемо підтримувати цю множину відрізків у бінарному дереві. Наприклад, можна використовувати `std::set` у C++ або власноруч написане декартове дерево. При використанні `std::set` потрібно написати компаратор, який залежатиме від глобальної змінної, що задає поточне розташування скануючої прямої. Важливо помітити, що при пересуванні вертикальної скануючої прямої вперед відносний порядок вертикальних відрізків, які зберігаються у дереві, не буде змінюватись доти, доки ми не натрапимо на якусь вершину графа. Розглянемо детальніше наші дії при натраплянні на точки кожного з двох типів:

- Якщо скануюча пряма натрапила на точку-археолога, треба у дереві швидко знайти відрізок, якому належить дана точка, та додати одиницю до кількості точок у відповідній йому області. Для цього потрібно використовувати ту властивість, що відрізки у дереві відсортовані за ординатою.
- Якщо скануюча пряма натрапила на вершину графа, треба модифікувати множину відрізків, яка зберігається у дереві:
    - На рис. 1 зображено випадок, коли ми натрапили на точку, з якої всі ребра виходять у точки з меншою абсцисою. У такому випадку ми повинні об'єднати між собою відрізки, що відповідають областям A і D. Це можна зробити за допомогою структури даних disjoint-set («система множин, що не перетинаються»). Області B та C у нашому випадку закінчуються, тому відповідні їм відрізки ми повинні видалити з дерева.
    - На рис. 2 зображено випадок, коли у поточної точки є як ребра, які виходять у точки з меншою абсцисою, так і ребра, які виходять у точки з більшою абсци-

сою. У цьому випадку області A та C залишаються незмінними, область B закінчується (відповідний відрізок видаляється), а області D та E починаються (ми додаємо відповідні їм відрізки).

- У випадку, коли в поточної точки є лише ребра, які виходять у точки з більшою абсцисою, ми повинні роздвоїти відрізок, на якому лежала поточна вершина. Обидва нових відрізки будуть належати одній і тій самій області площини.

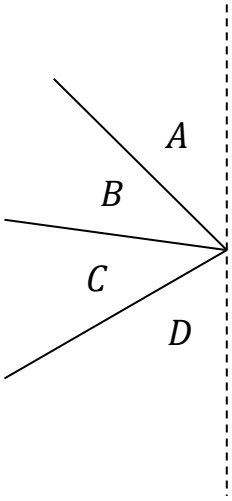

Рис. 1

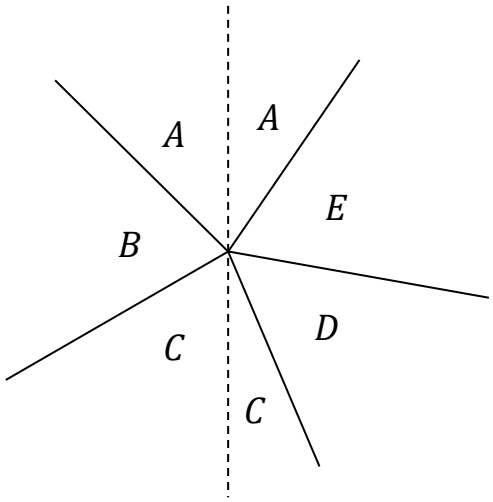

Рис. 2

Отже, з допомогою поданого алгоритму можна розв'язати задачу за $O((N+K)\cdot\log(N+K))$ часу, покладаючись на те, що всі точки мають різні абсциси. У випадку, якщо це не так, можна, наприклад, повернути всю площину навколо точки $(0,0)$ на довільний кут, відношення якого до π є ірраціональним (підійде, зокрема, будь-яке ціле число радіанів). Після повороту за годинниковою стрілкою точка (x,y) перейде в точку

$$(x\cdot\cos(\alpha)+y\cdot\sin(\alpha),\ -x\cdot\sin(\alpha)+y\cdot\cos(\alpha)).$$

Задачу можна було розв'язувати і повністю в цілих числах, але від учасників це не вимагалося.