II (районний) етап Всеукраїнської учнівської олімпіади з інформатики

Київ, 2018/19 н. р.

Максимальна оцінка за кожну з чотирьох задач — 100 балів.
Для всіх задач обмеження на час — 1 секунда / тест; обмеження на пам'ять — 256 МБ.

Матеріали олімпіади буде оприлюднено на сайті **kievoi.ippo.kubg.edu.ua**, а також на **soi.org.ua**.
Автор задач — Данило Мисак.

---

1. Укриття (назва програми: `shelters.cpp` / `shelters.pas` / `shelters.*`)

Уявіть, що вас, як людину з умінням мислити логічно й гарними навичками програмування, найняли покращувати благоустрій міста. Першим і одним з найвідповідальніших ваших завдань є забезпечення максимальної безпеки його мешканців.

У місті проживає p осіб і вже є n укриттів, що здатні вмістити по m людей кожне. Допоможіть з'ясувати, яку найменшу кількість таких самих укриттів необхідно добудувати, щоб у разі потреби усі мешканці могли водночас сховатися від небезпеки.

Вхідні дані
У єдиному рядку вхідного файлу задано додатні цілі числа p, n і m, кожне з яких не перевищує 180.

Вихідні дані
У вихідний файл виведіть кількість укриттів, які необхідно добудувати для повної безпеки міста. Якщо місця на всіх його мешканців вистачає і так, виведіть нуль.

Приклади

Вхідний файл `shelters.in`	Вихідний файл `shelters.out`
52 3 7	5
35 7 5	0

Коментарі до прикладів
У першому прикладі сімох укриттів не вистачить, щоб прихистити всіх жителів міста: $7 \times 7 < 52$. А от восьми укриттів буде вже достатньо — залишається до існуючих трьох добудувати ще п'ять. У другому прикладі місця в укриттях на всіх мешканців вистачає відразу.

2. Стовпчики (назва програми: `poles.cpp` / `poles.pas` / `poles.*`)

Дуже важливою у міському побуті є й культурна складова. На центральній галявині нового парку, який невдовзі буде відкрито в центрі міста, планують розмістити коло зі стовпів різної висоти, що умовно зображають його населення. У центрі галявини установлять стовпчик, який символізуватиме рівність і баланс: його висота за задумом скульпторів має дорівнювати середньому арифметичному висот усіх стовпчиків навколо. Знаючи висоти всіх стовпів, що установлять у парку, допоможіть вибрати з них стовпчик, який можна було б поставити у центрі.

Вхідні дані

У першому рядку вхідного файлу вказано кількість стовпчиків n; ця кількість є натуральним числом, що лежить у межах від 3 до 10^5 включно. У наступному рядку задано висоти n стовпчиків: кожна висота — натуральне число, менше за 10^9.

Вихідні дані

У вихідний файл виведіть висоту стовпчика, яка дорівнює середньому арифметичному усіх інших висот із вхідного файлу. Вхідні дані гарантують, що така висота дійсно існує і притому єдина.

Приклад

Вхідний файл **poles.in**	Вихідний файл **poles.out**
6 2 5 4 2 4 7	4

Коментар до прикладу

Якщо прибрати з набору будь-який один із двох стовпчиків заввишки 4, середнє арифметичне висот решти стовпів якраз дорівнюватиме $(2 + 2 + 4 + 5 + 7)/5 = 4$.

3. Паркування (назва програми: **parking.cpp** / **parking.pas** / **parking.***)

Поки що ваше місто складається з єдиної прямої вулиці, уздовж якої розташовано n будинків. Нещодавно перед містянами постала проблема з паркуванням, і було вирішено наземні стоянки переобладнати під велосипедні, а всі стоянки для машин перенести під землю — побудувати вздовж вулиці по одному підземному паркуванню для кожного будинку. З міркувань безпеки та зручності будівництва визначено, якими мають бути точні відстані між кожними двома сусідніми паркуваннями. Але самі паркування необхідно при цьому розташувати так, щоб сума відстаней між кожним із будинків та його відповідним паркуванням була найменшою можливою.

Паркування можна розміщати як безпосередньо під будинком, якому воно належить, так і в довільному іншому місці вздовж вулиці. Вулиця є достатньо довгою, тож ті чи інші паркування за потреби можна розміщати як завгодно далеко від будинків.

Ваше завдання — за інформацією про розташування будинків і відстанями між кожними двома сусідніми паркуваннями визначити найменшу можливу сумарну відстань від будинків до відповідних паркувань.

Вхідні дані

У першому рядку вхідного файлу задано натуральне число n — кількість будинків (і паркувань, які необхідно побудувати). Відомо, що $2 \leqslant n \leqslant 10^5$.

У другому рядку в порядку зростання перераховано n цілих чисел, що задають розташування будинків: число нуль (якщо воно є серед чисел) відповідає точці відліку; від'ємні числа (якщо такі є) відповідають будинкам на захід від цієї точки; додатні числа (якщо є) відповідають будинкам на схід від неї; абсолютна величина числа задає відстань від будинку до точки відліку. Усі числа, що задають розташування будинків, за абсолютним значенням не перевищують 10^9, і жодні два з цих чисел не збігаються.

У наступному рядку вказано $n - 1$ натуральне число: перше з них задає відстань між лівим (найзахіднішим) паркуванням A та сусіднім до нього справа паркуванням B; друге число задає відстань між паркуванням B та сусіднім до нього справа паркуванням C і т. д. Усі числа в цьому рядку не перевищують 10^4.

Перше (найзахідніше) паркування належатиме першому (найзахіднішому) будинку, друге паркування — другому будинку і т. д.

Вихідні дані

У вихідний файл виведіть ціле число — найменшу можливу суму відстаней від кожного будинку до його паркування. Вхідні дані гарантують, що ця сума не перевищить 10^9.

Приклад

Вхідний файл **parking.in**	Вихідний файл **parking.out**
4 -2 1 8 13 5 10 4	7

Коментар до прикладу

Паркування можна побудувати, наприклад, у позиціях -5, 0, 10 і 14. Тоді сумарна відстань до будинків складатиме $|-5-(-2)|+|0-1|+|10-8|+|14-13|=7$. Будь-яке інше розташування паркувань дасть або таку саму, або більшу сумарну відстань.

4. Вулиці (назва програми: **streets.cpp** / **streets.pas** / **streets.***)

У планах розбудови міста — відкрити кілька нових площ і між деяками парами цих площ прокласти прямі дороги з двостороннім рухом (можливо, з використанням мостів над іншими дорогами).

Ваша задача — розбити нові дороги на якомога меншу кількість вулиць так, щоб кожна дорога належала рівно одній вулиці. Вулицею може бути довільна послідовність доріг, у якій дороги не повторюються і кожна наступна дорога починається з тієї площі, де завершилася попередня. Зверніть увагу, що вулицям дозволено перетинати себе та замикатися у коло.

Вхідні дані

У першому рядку вхідного файлу вказано два натуральних числа — кількість нових площ n та кількість нових доріг m. Кількість площ не перевищує 800, і між кожною парою площ буде прокладено не більше ніж одну дорогу. Усі площі занумеровано натуральними числами від 1 до n.

Наступні m рядків задають дороги: у кожному рядку вказано по два числа — номери площ, які сполучає відповідна дорога, причому спочатку йде менший з двох номерів, а потім більший. Жодні два рядки не задають одну й ту саму дорогу. Крім того, з кожної площі виходить принаймні одна дорога.

Вихідні дані

У вихідний файл виведіть найменшу кількість вулиць, на які можна розбити нові дорогі.

Приклад

Вхідний файл **streets.in**	Вихідний файл **streets.out**
7 5 1 2 1 3 1 5 6 7 1 4	3

Коментар до прикладу

Одну з вулиць можна прокласти за маршрутом $2 — 1 — 3$, іншу за маршрутом $4 — 1 — 5$, третю — між площами 6 і 7.

Ідеї розв'язання

1. Укриття

Нехай $s = \max\{p - nm, 0\}$ позначає кількість містян, що в разі небезпеки не матимуть укриття. Тоді відповідь — округлена у більший бік частка від ділення s на m. Її можна отримати, наприклад, за формулою $\left[\frac{s+m-1}{m}\right]$, де квадратні дужки позначають найбільше ціле число, що не перевищує дане. У мові Pascal цю формулу найпростіше записати так: `(s+m-1) div m`.

2. Стовпчики

Якщо середнє арифметичне деякого набору з $n-1$ числа дорівнює m, то середнє арифметичне набору з n чисел, що складається із чисел з початкового набору та самого числа m, також дорівнює m. Отже, достатньо знайти середнє арифметичне усіх заданих у вхідному файлі чисел і вивести знайдене значення як відповідь.

Для підрахунку суми чисел (яка знадобиться при обчисленні середнього арифметичного) слід використати тип даних, що здатен вмістити число потрібної величини: наприклад, `int64` у мові Pascal та `long long` у C++.

3. Паркування

Розглянемо координатну площину, на осі абсцис якої розташовано будинки відповідно до того, як вони розміщені на вулиці міста, а по осі ординат — паркування. Паркування зафіксуємо лише з точністю до відстаней, щоб мати змогу рухати їх усі разом угору або вниз. Для будь-якого можливого розташування паркувань розглянемо n точок на площині, абсцисою кожної з яких є деякий будинок, а ординатою — відповідне паркування. Також проведемо пряму $x = y$. Відповіддю для даного розташування паркувань є сума відстаней по вертикалі (або — альтернативно — по горизонталі) від кожної з n точок до прямої.

Якщо розмістити всі паркування дуже низько (що відповідає у термінах початкової задачі розміщенню далеко на заході), то всі позначені n точок опиняться під прямою $x = y$. Далі, коли плавно рухати паркування вище (тобто «на схід»), точки почнуть наближатися до прямої, поступово через неї перейдуть і зрештою опиняться над нею. Поки точок під прямою більше, ніж точок над нею, під час руху вгору сумарна відстань буде зменшуватися: якщо є k точок під прямою і l точок над нею, то після зсуву на d сума відстаней зміниться на $l \times d - k \times d < 0$. Аналогічно, коли точок над прямою стане більше, ніж під нею, сумарна відстань почне збільшуватися. Якщо ж точок знизу й згори однакова кількість, сумарна відстань не змінюється.

Упорядкуємо всі точки за моментом часу, коли вони опиняться на прямій $x = y$, і розглянемо момент t, коли на прямій опиниться «середня» точка набору — точка з індексом $[n/2]$ у впорядкованому масиві (тут ми вважаємо, що індексація в масиві починається з нуля, а через $[n/2]$ позначаємо цілу частину числа $n/2$). При будь-якому зсуві паркувань униз відносно даного розташування кількість точок під прямою стане більшою, ніж над нею, а при будь-якому зсуві вгору кількість точок над прямою стане не меншою, ніж кількість точок під нею. Тому в момент t досягається найменша можлива сумарна відстань від точок до прямої (хоча, можливо, є й інші моменти часу, у які досягається та сама найменша відстань).

На практиці реалізація описаного алгоритму виглядатиме таким чином: фіксуємо певне розташування паркувань — наприклад, таке, у якому найзахідніше паркування має координату 0; шукаємо n різниць (з урахуванням знака) між координатами будинків та відповідних паркувань; знаходимо медіану отриманого масиву, тобто таку різницю, що у відсортованому масиві різниць стояла б на позиції $[n/2]$; зміщуємо паркування таким чином, щоб будинок і паркування, які відповідали різниці-медіані, сумістилися в одній точці; рахуємо на основі цього зсуву сумарну відстань між будинками й паркуваннями та виводимо її як відповідь.

Якщо реалізувати пошук медіани масиву ефективно (не вдаючися до сортування), час роботи алгоритму вийде лінійним від кількості будинків та паркувань. Але й розв'язок, що знаходить правильну відповідь за $O(n \log n)$ часу, має набирати повний бал.

4. Вулиці

Розглянемо граф, вершинами якого є площі, а ребрами — вулиці між ними. Не втрачаючи загальності, будемо вважати, що граф зв'язний. Інакше порахуємо відповідь окремо для кожної його компоненти зв'язності, а отримані значення просто складемо.

Якщо всі вершини зв'язного графа мають парний степінь (тобто з кожної вершини виходить парна кількість ребер), то граф є ейлеровим і його можна покрити однією вулицею, замкнутою у цикл. Інакше відповіддю буде кількість вершин непарного степеня, поділена на 2. Справді: з одного боку, кожна вулиця додає у граф не більше ніж дві непарних вершини. З іншого боку, $2k$ непарних вершин можна довільним чином об'єднати по дві та сполучити між собою кожну з k утворених пар вершин додатковим умовним ребром (навіть якщо деякі або всі відповідні пари вершин уже було сполучено). Отриманий таким чином граф або мультиграф вийде ейлеровим і міститиме ейлерів цикл. Коли з цього циклу вилучимо k доданих раніше умовних ребер, він розпадеться якраз на k окремих шляхів-вулиць.

Зауважимо, що кількість вершин непарного степеня у довільному графі неодмінно є парною, адже сума степенів усіх вершин графа дорівнює подвоєній кількості його ребер, тобто парному числу.

Виділення компонент зв'язності графа паралельно з підрахунком кількості вершин непарного степеня найзручніше реалізувати за допомогою пошуку в глибину. Час роботи алгоритму є лінійним від кількості ребер.