**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**](http://www.kievoi.ippo.kubg.edu.ua), а також на [**soi.org.ua**](http://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 осіб і вже є укриттів, що здатні вмістити по 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 , і , кожне з яких не перевищує .

**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: . 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 ; ця кількість є натуральним числом, що ле­жить у межах від до включно. У наступному рядку задано висоти стовпчиків: кожна висота — на­ту­ральне число, мен­ше за .

**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 , середнє арифметичне висот решти стовпів якраз дорівнюватиме .

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

Поки що ваше місто складається з єдиної прямої вулиці, уздовж якої розташовано 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 — кількість будинків (і паркувань, які необ­хід­но по­будува­ти). Відомо, що .

У другому рядку в порядку зростання перераховано 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 , і жодні два з цих чисел не збіга­ють­ся.

У наступному рядку вказано натуральне число: перше з них задає відстань між лівим (най­захід­нішим) паркуванням та сусіднім до нього справа паркуванням ; друге число задає відстань між паркуванням та сусіднім до нього справа паркуванням і т. д. Усі числа в цьому рядку не перевищують .

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 .

**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 , , і . Тоді сумарна відстань до будинків складатиме . 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 та кількість но­вих доріг . Кількість площ не перевищує , і між кожною парою площ буде прокладено не більше ніж одну до­рогу. Усі площі занумеровано натуральними числами від до .

Наступні 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  —  — , іншу за маршрутом  —  — , третю — між пло­щами і .

**Ідеї розв’язання**

**1. Укриття**

Нехай позначає кількість містян, що в разі небезпеки не матимуть укриття. Тоді відповідь — округлена у більший бік частка від ділення на . Її можна отримати, наприклад, за формулою , 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 числа дорівнює , то середнє арифметичне на­бору з чисел, що складається із чисел з початкового набору та самого числа , також дорівнює . 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 точок на пло­щині, абсцисою кожної з яких є деякий будинок, а ординатою — відповід­не паркування. Також проведемо пряму . Відповіддю для даного розташування паркувань є сума від­ста­ней по вертикалі (або — альтернативно — по горизонталі) від кожної з точок до прямої.

Якщо розмістити всі паркування дуже низько (що відповідає у термінах початкової задачі розміщен­ню далеко на заході), то всі позначені точок опиняться під прямою . 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 точок під прямою і точок над нею, то після зсуву на сума відстаней зміниться на . 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 , і розглянемо мо­мент , коли на прямій опиниться «середня» точка набору — точка з індексом у впорядкованому ма­сиві (тут ми вважаємо, що індексація в масиві починається з нуля, а через позначаємо цілу частину числа ). 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 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 ; шукаємо 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 ; 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](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%BE%D0%B2%D0%BE%D1%97_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B8) ефективно (не вдаючися до сортування), час роботи алго­ритму вийде лінійним від кількості будинків та паркувань. Але й розв’язок, що знаходить правильну відпо­відь за 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](https://uk.wikipedia.org/wiki/%D0%95%D0%B9%D0%BB%D0%B5%D1%80%D1%96%D0%B2_%D0%BB%D0%B0%D0%BD%D1%86%D1%8E%D0%B3) і його можна покрити однією вулицею, замкнутою у цикл. Інакше від­повід­дю буде кількість вершин непарного степеня, поділена на . Справ­ді: з одного боку, кожна вулиця до­дає у граф не більше ніж дві непарних вершини. З іншого боку, непарних вершин можна довільним чи­ном об’єднати по дві та сполучити між собою кожну з утворених пар вершин додатковим умовним реб­ром (навіть якщо деякі або всі відповідні пари вершин уже було сполучено). Отриманий таким чи­ном граф або [мультиграф](https://uk.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%B3%D1%80%D0%B0%D1%84) вийде ейлеровим і міститиме ейлерів цикл. Коли з цього циклу вилучимо дода­них рані­ше умов­них ребер, він розпадеться якраз на 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](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D1%83_%D0%B3%D0%BB%D0%B8%D0%B1%D0%B8%D0%BD%D1%83). Час роботи алгоритму є лінійним від кіль­кості ребер.