**II (районний) етап Всеукраїнської учнівської олімпіади з інформатики  
Київ, 20 грудня 2015 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 **[soi.org.ua](http://soi.org.ua/)**.  
Автор задач — Данило Мисак.

**1. Митько та подібні трикутники** (назва програми: **similar.cpp** / **similar.pas**)

Якось на уроці геометрії Митько дізнався, що два трикутники є подібними тоді й лише тоді, коли три сторони одного з них є пропорційними трьом сторонам іншого. Допоможіть Митькові з домашнім завданням: визначте, чи є два заданих трикутники подібними.

**Вхідні дані**

У вхідному файлі вказано шість натуральних чисел: перші три — довжини сторін першого трикутника, наступні три — довжини сторін другого трикутника. Усі числа менші за тисячу. Відомо, що з відрізків заданих довжин дійсно можна скласти трикутники.

**Вихідні дані**

У вихідний файл виведіть число 1, якщо задані трикутники подібні; в іншому разі виведіть 0.

**Приклади**

|  |  |
| --- | --- |
| Вхідний файл **similar.in** | Вихідний файл **similar.out** |
| 2 3 4 4 6 8 | 1 |
| 3 5 3 50 30 30 | 1 |
| 11 3 9 4 3 5 | 0 |

**Пояснення до прикладів**

У першому прикладі трикутники подібні, бо їхні сторони пропорційні: .

У другому прикладі сторони також пропорційні, хоч і не в такому порядку, в якому задані у вхідному файлі: .

У третьому прикладі трикутники не подібні, адже, незалежно від порядку, їхні сторони не є пропорційними.

**2. Митько та дивовижний острів** (назва програми: **island.cpp** / **island.pas**)

Якось на уроці географії Митько почув про незвичайний острів, що має форму круга: посередині острова височіє скеля, а населення живе у хижах уздовж периметра острова і через прямовисність скелі може пересуватися від хижі до хижі також виключно по периметрі. Для зручності вважатимемо, що периметр острова розбито на кілька однакових частин, які умовно назвемо секторами, і з однієї такої частини в сусідню можна перейти рівно за хвилину. У деяких секторах розташовано по хижі (але не більш ніж одна хижа в секторі). Визначте, за який час можна подолати відстань між парою найвіддаленіших хиж на острові.

**Вхідні дані**

У першому рядку вхідного файлу вказано два натуральних числа та — кількість секторів та хиж на острові відповідно. Відомо, що . Сектори занумеровано числами від до у тому порядку, в якому вони йдуть на острові (при цьому сектори з номерами та замикають коло і також є сусідніми). У другому рядку в порядку зростання вказано номери секторів, у яких є хижі.

**Вихідні дані**

У вихідний файл виведіть єдине число — відстань між двома найвіддаленішими хижами острова, тобто час у хвилинах, за який можна дійти від однієї з цих хиж до іншої.

**Приклади**

|  |  |
| --- | --- |
| Вхідний файл **island.in** | Вихідний файл **island.out** |
| 100 4  3 7 19 20 | 17 |
| 22 4  3 7 19 20 | 10 |

**Пояснення до прикладів**

У першому прикладі найвіддаленішими є перша та остання хижі, тож відповідь дорівнює .

У другому прикладі перша та остання хижі вже не є найвіддаленішими, адже між ними можна пройти за хвилин (таким чином: сектор — сектор — сектор — сектор — сектор — сектор ). Найвіддаленішими натомість є хижі в секторах і : вибравши оптимальний напрямок руху, дійти від однієї з них до іншої можна лише за хвилин.

**3. Митько та арифметичні прогресії** (назва програми: **progress.cpp** / **progress.pas**)

Якось на уроці алгебри Митько довідався, що арифметичною прогресією називають послідовність чисел, у якій різниця між кожними двома сусідніми членами однакова. Щоб учні краще засвоїли матеріал, учитель взяв деякі дві арифметичні прогресії, кожна з яких складається з натуральних чисел, перемішав між собою всі чисел (вони виявилися попарно різними) і виписав утворену послідовність на дошці. Допоможіть Митьку виконати вчителеве завдання: відновити з заданого набору чисел дві початкові арифметичні прогресії. Вхідні дані гарантують, що зробити це є рівно один спосіб.

**Вхідні дані**

У першому рядку вхідного файлу вказано натуральне число — кількість членів кожної з двох арифметичних прогресій, . У другому рядку записано різних натуральних чисел, менших за , — перемішані елементи обох прогресій.

**Вихідні дані**

У перший рядок вихідного файлу виведіть усі члени першої арифметичної прогресії в порядку зростання, а в другий рядок — усі члени другої арифметичної прогресії в порядку зростання. Прогресії виведіть у такому порядку, щоб перше число в першому рядку було меншим за перше число в другому рядку.

**Приклад**

|  |  |
| --- | --- |
| Вхідний файл **progress.in** | Вихідний файл **progress.out** |
| 4 7 9 23 3 16 15 11 2 | 2 9 16 23 3 7 11 15 |

**Пояснення до прикладу**

Виведені у вихідний файл послідовності є арифметичними прогресіями, адже і .

**4. Митько та міжпланетна подорож** (назва програми: **journey.cpp** / **journey.pas**)

Якось після важкого дня у школі з уроками астрономії, фізики та економіки у голові Митька все перемішалося, і хлопцю наснився дивний сон. У віддаленому майбутньому люди заселяють планет, між яким­и пересуваються за допомогою телепортації. Для зручності планети занумеровано числами від до . Процес телепортації обслуговують різних компаній, і вони конкурують між собою. Тому телепортуватися можна не між будь-якою парою планет, а лише між тими, які обслуговує одна й та сама компанія. На щастя, одну й ту саму планету може обслуговувати відразу кілька різних компаній. До того ж відомо, що з кожної планети можна переміститися на будь-яку іншу якщо й не за одну, то принаймні за декілька послідовних телепортацій. З’ясуйте, за яку найменшу кількість послідовних телепортацій можна переміститися з планети на планету .

**Вхідні дані**

У першому рядку вхідного файлу записано два натуральних числа та — кількість планет та компаній відповідно; , , а добуток цих двох чисел не перевищує мільйона. Кожен з наступних рядків містить по цифр, *не розділених пробілом*, та задає інформацію про відповідну планету (у першому з цих рядків — інформація про планету , в останньому — про планету ): якщо цифра на позиції в рядку є одиницею, то компанія під номером обслуговує дану планету; якщо ж ця цифра нуль, то не обслуговує. Кожна компанія обслуговує хоча б дві планети.

**Вихідні дані**

У вихідний файл виведіть єдине число — найменшу кількість послідовних телепортацій, необхідних, щоб з планети дістатися на планету .

**Приклад**

|  |  |
| --- | --- |
| Вхідний файл **journey.in** | Вихідний файл **journey.out** |
| 4 2 01 01 11 10 | 2 |

**Пояснення до прикладу**

Першу планету обслуговує тільки друга компанія, тому з неї можна потрапити на другу і третю планети, але не на четверту. Зате за дві телепортації — з транзитом через третю планету — з першої на четверту потрапити вже можна.

**Ідеї розв’язання**

**1. Митько та подібні трикутники**

Щоб зрозуміти, чи є трикутники подібними, можна перепробувати всі можливі варіанти порядку, в якому сторони можуть виявитися пропорційними (якщо зафіксувати порядок сторін одного з трикутників, для іншого є варіантів розстановки). А можна зробити інакше: просто відсортувати сторони кожного з трикутників. Тоді найменшій стороні першого трикутника відповідатиме найменша сторона другого трикутника; середній стороні відповідатиме середня; найбільшій — найбільша.

Перевірку пропорційності ні в якому разі не можна здійснювати за допомогою оператора цілочисельного ділення (як-от div у Pascal’і), адже цей оператор бере лише цілу частину від ділення одного числа на інше та ігнорує остачу. Не варто користуватися і звичайним діленням у дійсних числах: при обчисленні та поданні таких чисел комп’ютер може втратити точність, а відмінність хоч би й в одну мільйонну означатиме, що два числа, які насправді є рівними, комп’ютер рівними не ви́знає. Натомість можна скористатися рівноцінним порівнянням добутків, адже — це те саме, що . Щоправда, такі добутки можуть вийти за межі двобайтової змінної (як integer у Pascal’і). Це треба врахувати й скористатися відповідним типом даних.

Отже, один з можливих способів розв’язати задачу на повний бал такий: вивести одиницю, якщо пара з найменшої та середньої сторони першого трикутника пропорційна парі з найменшої та середньої сторони другого трикутника, а пара з середньої та найбільшої сторони першого трикутника пропорційна парі з середньої та найбільшої сторони другого трикутника; інакше вивести нуль.

**2. Митько та дивовижний острів**

Відстань між хижами в секторах та (де ) обчислюється за формулою , тобто дорівнює меншій з двох відстаней: у випадку, якщо йти в порядку збільшення номера сектора, та у випадку, якщо йти в порядку зменшення номера сектора. Ідейно найпростіший спосіб розв’язати задачу — перебрати всі пари хиж, порахувати відстань між кожною та вибрати з усіх підрахованих величин найбільшу. [Час виконання програми](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D1%8E%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D1%96%D1%81%D1%82%D1%8C#.D0.9F.D1.80.D0.B8.D0.BA.D0.BB.D0.B0.D0.B4.D0.B8_.D0.B0.D1.81.D0.B8.D0.BC.D0.BF.D1.82.D0.BE.D1.82.D0.B8.D1.87.D0.BD.D0.B8.D1.85_.D1.81.D0.BA.D0.BB.D0.B0.D0.B4.D0.BD.D0.BE.D1.81.D1.82.D0.B5.D0.B9) в такому випадку буде квадратичним від кількості хиж, що дозволить набрати лише половину балів за задачу. А от більш оптимальний алгоритм, що заробить повний бал, можна побудувати, спираючись на спостереження, яке наводимо нижче.

Нехай на колі зафіксовано деякий набір точок (хиж). Найвіддаленішою від точки буде та з точок набору, що лежить найближче до точки кола, діаметрально протилежної до . Якщо ми пересуватимемо точку , скажімо, за годинниковою стрілкою, то й діаметрально протилежна до неї точка рухатиметься за годинниковою стрілкою, а значить, у порядку руху за годинниковою стрілкою змінюватиметься і найвіддаленіша від хижа.

Отже, алгоритм буде таким: зчитуючи розташування кожної наступної хижі, визначаємо найвіддаленішу від неї (вже зчитану раніше) хижу. Для цього беремо хижу, яка виявилася найвіддаленішою від попередньої зчитаної хижі, та рухаємось у порядку збільшення номера хижі, допоки відстань до поточної хижі збільшується. Зокрема, якщо відстань не збільшилася вже при першому порівнянні, то найвіддаленішою від даної хижі є та сама хижа, яка була найвіддаленішою від попередньої; якщо, навпаки, відстань збільшувалася увесь час, аж поки ми не натрапили на ту саму хижу, яку зчитували (а від неї до неї самої відстань, очевидно, нульова), то найвіддаленішою від даної є попередня зчитана хижа. Таким чином, здійснивши в процесі зчитування даних загалом не більше ніж один повний прохід по колу у пошуках найвіддаленіших хиж, ми визначимо відповідь — це максимальна зі знайдених для кожної хижі найбільших відстаней.

Є й інші алгоритми, що працюють, як і даний, лінійний від кількості хиж час, проте наведений алгоритм є одним із найпростіших у реалізації.

Насамкінець додамо, що алгоритм, який спирається на ідею [двійкового пошуку](https://uk.wikipedia.org/wiki/%D0%94%D0%B2%D1%96%D0%B9%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BF%D0%BE%D1%88%D1%83%D0%BA) найвіддаленішої хижі, хоч і має час виконання , що гірше за лінійний, тим не менше набирає повний бал. Стільки ж потенційно дозволяють заробити і решта алгоритмів із часом виконання .

**3. Митько та арифметичні прогресії**

Один з можливих способів розв’язати задачу такий. [Відсортуємо](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F#.D0.92.D1.96.D0.B4.D0.BE.D0.BC.D1.96_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B8_.D1.81.D0.BE.D1.80.D1.82.D1.83.D0.B2.D0.B0.D0.BD.D0.BD.D1.8F) всі чисел у порядку від наймен­шого до найбільшого. Серед трьох найменших елементів відсортованої послідовності принаймні два належатимуть до однієї й тієї ж прогресії, причому будуть двома найменшими її членами (а якщо всі три найменші числа належать одній і тій самій прогресії, то найменшими двома її членами є, очевидно, два перших числа). Тепер, послідовно розглядаючи гіпотези про те, що двома найменшими членами однієї з прогресій є перший і другий; перший і третій; другий і третій елементи відсортованої послідовності, встановимо, котра з цих гіпотез є правильною. Для перевірки можемо скористатися таким підходом: перебиратимемо в порядку збільшення всі чисел; якщо чергове число, яке ми розглядаємо, є таким, що підходить до першої прогресії (а, беручи припущення гіпотези, ми вже знаємо і перший член цієї прогресії, і її різницю, і кількість елементів), то долучаємо це число до першої прогресії, інакше — до другої. Якщо обидві побудовані послідовності дійсно є арифметичними прогресіями (з елементами в кожній), то маємо відповідь; інакше переходимо до наступної гіпотези. Описаний процес перевірки можна втілити за один лінійний прохід послідовності.

Оскільки алгоритм складається з сортування і кількох лінійних проходів масиву на елементів, його складність можна оцінити як . Це дозволяє заробити повний бал.

Утім, алгоритм можна оптимізувати і до лінійного. Для пошуку трьох найменших елементів замість сортування використаємо, наприклад, три послідовних лінійних проходи. А для перевірки гіпотези перебиратимемо числа не в порядку збільшення, а в довільному. Це не завадить відібрати з набору ті й лише ті числа, що належать до першої прогресії. Щоб установити, чи решта чисел утворюють другу прогресію, знайдемо шляхом двох лінійних проходжень два найменших числа, що не потрапили до першої прогресії: вони якраз і мають становити два перших члени другої прогресії. Залишається ще раз пройтися по масиву і перевірити, чи решта чисел у ньому «узгоджуються» зі знайденими першими членами потенційної прогресії. При цьому, звичайно, не слід забувати, що в обох прогресіях повинно бути рівно по членів. Нарешті, щоб уникнути сортування при виведенні відповіді, можна конструювати прогресії безпосередньо з їхніх арифметичних властивостей (а не виводити у вихідний файл упорядковані елементи масиву).

**4. Митько та міжпланетна подорож**

Це — одна з типових задач, які можна розв’язати за допомогою [пошуку в ширину](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D1%83_%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D1%83): між двома планетами існує перехід тоді й лише тоді, коли їх обслуговує хоча б одна спільна компанія. Однак у даному випадку планет може бути кілька сотень тисяч, і кожну або майже кожну їх пару може бути сполучено. Тому при реалізації звичайного пошуку в ширину доведеться зіткнутися якщо й не з проблемою виділення па­м’яті, то принаймні з тим, що програма не вкладатиметься в обмеження на час на великих вхідних даних.

Ключовим спостереженням, що дозволить вирішити дану проблему, буде таке: на довільному кроці виконання алгоритму немає сенсу розглядати сполучення по тих компаніях (по тих стовпцях вхідних даних), сполучення по яких ми вже один раз розглядали раніше. При цьому, розглядаючи планету з [черги](http://uk.wikipedia.org/wiki/%D0%A7%D0%B5%D1%80%D0%B3%D0%B0_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)) пошуку, суміжні до неї планети знаходимо за допомогою дворівневого циклу, де в зовнішньому циклі перебираємо компанії, що обслуговують дану планету, а для кожної компанії, яку ще є сенс розглядати, у вкладеному циклі перебираємо інші планети, які ця компанія обслуговує. Складність виконання алгоритму — .

Для кращого розуміння радимо переглянути [прокоментований код](http://soi.org.ua/_files/938c3427f8/programs-2015-author.zip) авторського розв’язання задачі.

Окремо пояснимо, як можна ефективно зберігати вхідні дані, враховуючи, що конкретні обмеження на кількість планет і компаній нам не відомі, а знаємо лише добуток цих кількостей. Нам достатньо завести один одновимірний масив, розмірність якого відповідає максимальному значенню добутку кількостей планет і компаній. Тоді значення, що стоїть на перетині -го рядка та -го стовпця у вхідному файлі, якщо домовитись, що нумерацію рядків і стовпців ведемо з нуля, а не з одиниці, слід зберігати в комірці з індексом (нумерацію комірок теж ведемо з нуля — стандартно для C++). Це дозволить заповнити вхідними даними без жодного перекриття всі комірки масиву з нульової до -ї включно.