ЗАДАЧІ XXVII ВСЕУКРАЇНСЬКОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ РОЗВ'ЯЗУВАННЯ

Бондаренко Віталій Вікторович,
асистент факультету кібернетики Київського Національного університету ім. Тараса Шевченка.
Мисак Данило Петрович,
керівник гуртка СШ №52 м. Києва,
Ягіяєв Шаміль Ігорович,
менеджер проектів компанії «Арісент Україна».

Завдання першого туру

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 ($2\le N\le 10^5$), що задає розмір плитки шоколаду, тобто кількість у ній квадратиків. Усіх кондитерів занумеровано числами від 1 до N. Другий і 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: $2\le N\le 3$;
- 20 балів: $3<N\le 6$;
- 30 балів: $6<N\le 1000$;
- 30 балів: $1000<N\le 10^5$.

Приклад вхідних і 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. Певним 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, $1<i<N$, 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.

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. У 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. У 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 ($1\le L_i\le 10^9$) 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: $1 \le N \le 100$;
- 20 балів: $100 < N \le 3000$;
- 30 балів: $3000 < N \le 10^4$;
- 30 балів: $10^4 < N \le 10^5$.

Приклади вхідних і 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: $F_1 = F_2 = 1$, $F_k = F_{k-1} + F_{k-2}$, $k \ge 3$.

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. Отже, 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.

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 і 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 shoppi*n*g, що за даними про розмір ділянки й 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 ($2 \le N \le 5 \cdot 10^4$) — 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: $1 \le N \le 10$;
- 10 балів: $10 < N \le 30$;
- 10 балів: $30 < N \le 70$;
- 30 балів: $70 < N \le 1000$;
- 40 балів: $1000 < N \le 5 \cdot 10^4$.

Приклади вхідних і 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 ($x_1 \le i \le x_2$). 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).

- Заводимо **стек**, що в кожен момент часу міститиме число 0 та деяку підпослідовність індексів масиву обмежень однієї з орієнтацій.
- Кладемо в порожній стек число 0.
- 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.

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 немає із собою ручки і 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 із 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 і 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 у 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, у 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 і 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, і 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 і 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, у якій лежатиме 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 і 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 деякій вершині 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 піднімає-

мося та продовжуємо аналогічні дії. Коли станеться так, що у вершини, куди ми піднялися, більше немає нульових сусідів, у ній 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 (у 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 у 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, у 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 неї 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, у 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, у якій записано на одиницю менше число у тому стані, у якому перебувала поточна вершина раніше (зауважимо, що може вийти так, що ми піднімемося 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 й 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, у 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: danmvsak@gmail.com.

(Далі буде)