ЗАВДАННЯ XXVI МІЖНАРОДНОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ РОЗВ'ЯЗАННЯ

Гуржій Андрій Миколайович,
віце-президент НАПН України, доктор технічних наук, професор, академік НАПН України.
Бондаренко Віталій Вікторович,
асистент факультету кібернетики Київського Національного університету ім. Тараса Шевченка.

ЗАВДАННЯ ПЕРШОГО ТУРУ

1. Колія

У Тайвані є велика залізнична лінія, що з'єднує західне і східне узбережжя острова. Лінія складається із m блоків. Послідовні блоки пронумеровані 0,..., m–1, починаючи із західного кінця. Кожен блок має на півночі колію з дозволеним рухом на захід, на півдні колію з дозволеним рухом на схід, і, можливо, станцію між ними.

Є три типи блоків. Блок типу C містить станцію, на яку ви маєте заїжджати з північної колії, і виїжджати по південній колії, блок типу D містить станцію, на яку ви маєте заїжджати з південної колії, і виїжджати по північній колії, блок типу *пустий* не містить станції. Наприклад, на рис. 1 блоки 0, 4 та 6 мають тип пустий, блоки 1, 2 та 3 мають тип C, та блок 5 має тип D. Колії сусідніх блоків поєднано *з 'єднувачами*, позначеними на рис. 1 як затемнені прямокутники.

Колійна система має n станцій, пронумерованих від 0 до n–1. Ми припускаємо, що *ми можемо дістатись довільної станції з довіль-ної іншог станції* слідуючи коліям. Наприклад, ми можемо дістатися зі станції 0 до станції 2, почавши з блоку 2, потім слідуючи через блоки 3 та 4 по південній колії, потім через блок 5, проїхавши через станцію 1, потім проїхавши через блок 4 по північній колії, та, нарешті, прибувши на станцію 2 у блоці 3.

Оскільки може бути кілька маршрутів, відстань від однієї станції до іншої визначається як *мінімальна* кількість з'єднувачів, через які проходить маршрут. Наприклад, найкоротший маршрут від станції 0 до станції 2 проходить через блоки 2-3-4-5-4-3 та проходить через 5 з'єднувачів, тобто відстань дорівнює 5.

Колійною системою керує комп'ютер. На жаль, після збою живлення комп'ютер більше не знає, де знаходяться станції і в блоках якого типу вони розташовані. Єдине, що відомо комп'ютеру, це номер блока для станції 0, який завжди має тип C. На щастя, комп'ютер може запитувати відстань від довільної станції до довільної іншої станції. Наприклад, комп'ютер може запитати: «чому дорівнює відстань від станції 0 до станції 2?», та отримає відповідь 5.

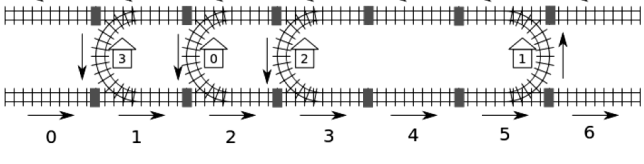

Рис. 1

Задача. Ви маєте реалізувати функцію findLocation, що визначить для кожної станції номер блока і тип блока.

findLocation(n, first, location, stype)

- **n**: кількість станцій.
- **first**: номер блока станції 0.
- **location**: масив довжини n; ви маєте записати номер блока станції i до location [i].
- **stype**: масив довжини n; ви маєте записати тип блока станції i до stype[i]: 1 для типу C та 2 для типу D.

Ви можете викликати функцію getDistance, що потрібно дня знаходження розташування і типу станцій. **getDistance**(i, j) повертає відстань від станції i до станції j. getDistance(i, i) поверне 0. getDistance(i, j) поверне –1, якщо i або j знаходиться за межами діапазону $0 \le i, j \le n-1$.

Підзадачі. У всіх підзадачах (табл. 1) кількість блоків m не перевищує 1000000. У деяких підзадачах обмежено кількість викликів getDistance. Це обмеження відрізняється для підзадач. Ваша програма отримає 'wrong answer', якщо вона перевищить це обмеження.

Деталі реалізації. Ви маєте відіслати тілвки один файл, що має ім'я rail.c, rail cpp або rail.pas. Цей файл реалізує findLocation, як описано вище.

Рекомендації щодо розв'язання. По-перше, можна помітити, що відстані є симетричними: за маршрутом від A до B можна відбиттям отримати маршрут від B до A. Отже, обмеження в $n(n-1)/2$ запитів не є проблемою, адже ми можемо дізнатись запитами всі відстані. Це може бути корисним під час розв'язання перших трьох підзадач, але перейдемо до найскладнішої підзадачі.

Якщо ми знаємо позицію і тип станції, є ще одна станція, яку ми можемо відразу знайти: найближча

Таблиця 1

Підзадача	Балів	n	Викликів getDistance	Примітка
1	8	$1 \le n \le 100$	Не обмежено	Всі станції за винятком 0 розташовано у блоках типу D
2	22	$1 \le n \le 100$	Не обмежено	Всі станції на схід від станції 0 розташовано у блоках типу D, та всі станції на захід від станції 0 розташовано у блоках типу C
3	26	$1 \le n \le 5\,000$	$n(n-1)/2$	Немає інших обмежень
4	44	$1 \le n \le 5\,000$	$3(n-1)$	Немає інших обмежень

до неї. Вона буде мати протилежний тип і буде досяжною прямим маршрутом. Нехай станція X буде найближчою до станції 0. Інші станції можна розбити на 3 групи:

1) $d(X,Y) < d(X,0)$: ці станції безпосереднво досяжні зі станції X і маютв тип C, ми можемо їх визначити точно;

2) $d(0, X)+d(X, Y)=d(0, Y)$, але не типу 1: ці є досяжними від станції 0 через станцію X, отже вони лежатв ліворуч від станції 0;

3) усі інші станції лежать праворуч від станції X.

Розглянемо тепер станції праворуч від X, у порядку зростання відстані від 0. У цвому випадку станції типу D будутв розглядатися у правильному порядку, а станції типу C будуть траплятись station у деякий момент, після того, як знайдено станцію типу D, через яку вони досяжні. Нехай Y буде найправішою знайденою станцією типу D, і розглянемо відстані до нової станції Z. Нехай $z=d(0, Z)-d(0, Y)-d(Y, Z)$. Якщо Z має тип C, то має бути станція типу D на відстані $z/2$ зліва від Y. З іншого боку, якщо Z має тип D (й лежить праворуч від Y), тоді має бути станція типу C на відстані $z/2$ ліворуч від Y. У першому випадку ми вже мали знайти відповідну станцію, отже, ми завжди зможемо визначити позицію та тип Z.

Станції зліва від станції 0 можна обробляти аналогічно, використовуючи станцію X як базову станцію замість станції 1.

Кожна станція Z, крім 0 і X потребує не більше трьох запитів: $d(0, Z)$, $d(X, Z)$ та $d(Y, Z)$, де Y може бути різним для кожного Z. Це дає $3(n-2)+1$ запитів, чого достатньо для розв'язання всіх підзадач.

2. Стіна

Жиан-Жиа будує стіну, зіставляючи разом цеглини одного розміру. Стіна складається з n стовпчиків цеглин, які занумеровані від 0 до $n-1$ зліва направо. Стовпчики можуть мати різну висоту. Висотою стовпчика є кількість цеглин у ньому.

Жиан-Жиа будує стіну так. На початку цеглин у стовпчиках немає. Потім Жиан-Жиа виконує k фаз *додавання* або *вилучення* цеглин. Процес побудови закінчено, коли закінчено всі k фаз. На кожній фазі Жиан-Жиа дається діапазон послідовних стовпчиків з цеглинами та висота h, він виконує таку процедуру:

- у фазі *додавання*, Жиан-Жиа додає цеглини до тих стовпчиків у заданому діапазоні, у яких менше ніж h цеглин, так щоб у них стало рівно h цеглин. Він нічого не робить зі стовпчиками, у яких h або більше цеглин;
- у фазі *вилучення*, Жиан-Жиа вилучає цеглини з тих стовпчиків у заданому діапазоні, що мають більше ніж h цеглин, так щоб стало рівно h цеглин. Він нічого не робить зі стовпчиками, у яких h або менше цеглин.

Ваша задача — визначити кінцеву форму стіни.

Приклад. Припустимо, що є 10 стовпчиків з цеглою і 6 фаз побудови. Усі відрізки у таблиці 2 включають свої кінці. Діаграму стіни після кожної з фаз показано на рис. 2.

Таблиця 2

Фаза	Тип	Діапазон	Висота
0	Додати	Стовпчики 1 до 8	4
1	Вилучити	Стовпчики 4 до 9	1
2	Вилучити	Стовпчики 3 до 6	5
3	Додати	Стовпчики 0 до 5	3
4	Додати	Стовпчик 2	5
5	Вилучити	Стовпчики 6 до 7	0

Оскільки всі стовпчики на початку пусті, після фази 0 кожен стовпчик від 1 до 8 буде містити 4 цеглини. Стовпчики 0 і 9 залишаться пустими. У фазі 1 видаляються цеглини зі стовпчиків з 4 до 8, щоб в них залишилось по 1 цеглині, а стовпчик 9 залишається пустим. Стовпчики з 0 до 3, що знаходяться за межами діапазону, залишаються незміненими. Фаза 2 не вносить змін, оскільки стовпчики з 3 до 6 не мають більше 5 цеглин. Після фази 3 кількість цеглин у стовпчиках 0, 4 та 5 збільшується до 3. 5 цеглин опиняється у стовпчику 2 після фази 4. Фаза 5 видаляє всі цеглини зі стовпчиків 6 та 7.

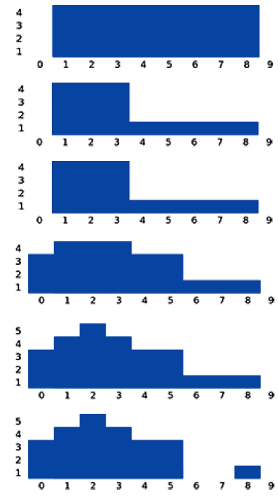

Рис. 2

Задача. Маючи опис фаз будівництва, обчисліть кількість цеглин у кожному із стовпчиків після закінчення всіх фаз. Ви маєте реалізувати наступну функцію build Wall.

buildWall(n, k, op, left, right, height, finalHeight)

- **n**: кількість стовпчиків у стіні;
- **k**: кількість фаз;
- **op**: масив довжини k; op[i] є типом фази i: 1 для фази додавання і 2 для фази вилучення, для $0 \le i \le k-1$;
- **left та right**: масиви довжини k; діапазон стовпчиків у фазі i починається зі стовпчика left[i] і закінчується стовпчиком right[i] (включаючи крайні стовпчики left[i] і right[i]), для $0 \le i \le k-1$. Завжди буде виконуватись left[i]≤right[i];
- **height**: масив довжини k; height[i] задає висоту для фази i, для $0 \le i \le k-1$;
- **final Height**: масив довжини n; ви маєте повернути ваш результат, помістивши фінальну кількість цеглин у стовпчику i у finalHeight[i], для $0 \le i \le n-1$.

Підзадачі (табл. 3). Для всіх підзадач параметер, що задає висоту, у всіх фазах буде невід'ємним цілим, що менше або дорівнює 100 000.

Деталі реалізації. Ви маєте відіслати тільки один файл, що має ім'я wall.c, wall.cpp або wall.pas. Цей файл реалізує buildWall, як описано вище.

Рекомендації щодо розв'язання. Це достатньо стандартна задача на використання дерева відрізків. Кожен вузол містить інтервал, у якому знаходяться значення його дітей. Коли застосовується нова інструкція, це робиться зверху донизу, відповідні обмежен-

Таблиця 3

Підзадача	Балів	n	k	Примітка
1	8	$1 \le n \le 10\,000$	$1 \le k \le 5\,000$	Немає інших обмежень
2	24	$1 \le n \le 100\,000$	$1 \le k \le 500\,000$	Всі фази додавання передують всім фазам вилучення
3	29	$1 \le n \le 100\,000$	$1 \le k \le 500\,000$	Немає інших обмежень
4	39	$1 \le n \le 2\,000\,000$	$1 \le k \le 500\,000$	Немає інших обмежень

ня протягуються рекурсивно в глибину дерева. Щоб визначити значення кожного елемента у стіні, необхідно обійти все дерево, застосовуючи обмежуючі інтервали на шляху вниз.

Цікавою особливістю цієї задачі є те, що всі дії задано наперед, а знайти потрібно тільки фінальну конфігурацію. Можливо, існує інший розв'язок, що обробляє інструкції не в порядку їх надходження.

3. Гра

Жиан-Жиа — це маленький хлопчик, який полюбляє грати в ігри. Коли його щось запитують, він воліє почати грати, а не відповісти прямо. Жиан-Жиа зустрів свою подругу Мей-Ю і розповів їй про мережу перельотів у Тайвані. У Тайвані є n міст (пронумерованих 0,..., n–1), деякі з них з'єднано перельотами. Кожен переліт з'єднує два міста та може виконуватисв у двох напрямках.

Мей-Ю запитала Жиан-Жиа, чи можна потрапити з одного міста до іншого літаком (прямо чи ні). Жиан-Жиа не хоче розкривати відповідь, замість цього він пропонує зіграти у гру. Мей-Ю може задавати запитання вигляду «Чи з'єднані міста x та y **прямим** перельотом?», а Жиан-Жиа буде негайно на такі запитання відповідати. Мей-Ю буде запитувати про кожну пару міст тільки один раз, задавши всього $r=n(n-1)/2$ запитань. Мей-Ю виграє гру, якщо, отримавши відповіді на перші i запитань для деякого $i<r$, вона зможе зробити висновок, чи є мережа зв'язною, тобто чи можна подорожувати літаком між довільною парою міст (прямо чи через інші міста). У іншому випадку, тобто якщо їй потрібно всі r запитань, переможцем буде Жиан-Жиа.

Щоб зробити гру більш цікавою для Жиан-Жиа, друзі домовились забути про справжню мережу перельотів у Тайвані, і вигадувати мережу в процесі гри, обираючи відповідь, базуючись на попередніх запитаннях Мей-Ю. Ваша задача — допомогти Жиан-Жиа виграти гру, вирішуючи, як він має відповідати на запитання.

Приклад. Пояснимо правила гри на трьох прикладах. Кожен приклад має $n=4$ міст та $r=6$ раундів запитання-відповідь.

У першому прикладі (таблиця 4) Жиан-Жиа *програє*, оскільки після раунда 4 Мей-Ю знає напевно, що можна подорожувати літаком між довілвною парою міст, незалежно від того, як Жиан-Жиа буде відповідати на запитання 5 і 6.

У наступному прикладі (таблиця 5) Мей-Ю може довести після раунда 3, що, незалежно від відповідей Жиан-Жиа на запитання 4, 5 або 6, *немає* можливості подорожувати літаком між містами 0 та 1. Отже, Жиан-Жиа знову програє.

У наступному прикладі Мей-Ю не може визначити, чи можна подорожувати літаками між довільною парою міст поки не буде отримано відповіді на всі шість запитань, отже, Жиан-Жиа *виграє* цю гру. А саме, оскільки Жиан-Жиа відповів *так* на останнє запитання (у таблиці 6), то можна подорожувати літаком між довільною парою міст. Однак, якби Жиан-Жиа відповів на останнє запитання *ні*, це було б неможливо.

Таблиця 4

Раунд	Запитання	Відповідь
1	0, 1	Так
2	3, 0	Так
3	1, 2	Ні
4	0, 2	Так
—	—	—
5	3, 1	Ні
6	2, 3	Ні

Таблиця 5

Раунд	Запитання	Відповідь
1	0, 3	Ні
2	2, 0	Ні
3	0, 1	Ні
—	—	—
4	1, 2	Так
5	1, 3	Так
6	2, 3	Так

Задача. Напишіть програму, що допоможе Жиан-Жиа перемогти у грі. Зауважте, що ні Мей-Ю, ні Жиан-Жиа не знають стратегію один одного. Мей-Ю може запитувати про пари міст у довільному порядку, а Жиан-Жиа має негайно відповідати, не знаючи наступних запитань. Ви маєте реалізувати такі дві функції:

Таблиця 6

Раунд	Запитання	Відповідь
1	0, 3	Ні
2	1, 0	Так
3	0, 2	Ні
4	3, 1	Так
5	1, 2	Ні
6	2, 3	Так

- initialize(n) — Ми спочатку визвемо вашу функцію initialize. Параметр n — це кількість міст;
- hasEdge(u, v) — Потім ми визвемо hasEdge r — $n(n-1)/2$ раз. Ці виклики подають запитання Мей-Ю, у порядку, як вона їх задає. Ви маєте відповідати, чи існує прямий переліт між містами u та v. Значення, що повертається, має бути 1, якщо прямий переліт існує, та 0 у іншому випадку.

Підзадачі. Кожна підзадача складається з кількох ігор. Ви отримаєте бали за підзадачу (табл. 7), тільки якщо ваша програма переможе у всіх іграх за Жиан-Жиа.

Таблиця 7

Підзадача	Бали	n
1	15	$n=4$
2	27	$4\le n\le 80$
3	58	$4\le n\le 1500$

Деталі реалиіації. Ви маєте відіслати тільки один файл, що має ім'я game.c, game.cpp або game.pas. Цей файл реалізує підпрограми, що описані вище.

Рекомендації щодо розв'язання. Нехай Мей-Ю вже знає, що деякі міста з'єднані. Якщо при цьому між містами є перельоти, про які вона ще не запитувала, вона може залишити один з них на потім, оскільки відповідь не впливає на те, чи є країна з'єднаною. З цього випливає, що для виграшу Жиан-Жиа потрібно завжди відповідати «ні», якщо його запитують про переліт між двома компонентами, Мей-Ю не відомо, чи зв'язані вони, крім випадку, коли це останній переліт між цими компонентами.

Що буде, якщо він відповість «так» на запитання про останній переліт між цими двома компонентами? У цьому випадку він виграє. Поки залишається принаймні два компоненти, є неперевірені ребра між ними, отже, Мей-Ю не знає, чи з'єднані вони. Усі ребра всередині компонента відомі, отже, кількість компонентів стане відомою тільки після останнього запитання.

Яка складність цього алгоритму? Нам потрібно зберігати кількість ребер між кожною парою компонентів, що потребує пам'яті обсягом $O(N^2)$. Більшість операцій буде просто зменшувати один з цих лічільників. Також буде $N-1$ операцій злиття компонентів, кожен з яких потребує лінійний час на злиття лічільників й оновлення таблиці відповідності між вершинами й компонентами. Отже, весь алгоритм потребує час $O(N^2)$. Це оптимальний алгоритм, оскільки Мей-Ю буде задавати $O(N^2)$ запитань.

(Далі буде)