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

Гуржій А.М., Бондаренко В.В.

ЗАВДАННЯ ПЕРШОГО ТУРУ

1. ТРОПІЧНИЙ САД

Ботанік Сомхед часто організує екскурсії груп студентів у один з найбільших тропічних садів Таїланду. Цей сад складається з N фонтанів, пронумерованих числами від 0 до (N–1), і M стежок. Кожна стежка з'єднує пару різних фонтанів, і по ній можна ходити в обох напрямках. При цьому різні стежки з'єднують різні пари. Від кожного фонтана веде хоча б одна стежка. Вздовж кожної зі стежок знаходяться красиві колекції рослин, які хотів би побачити Сомхед. Кожна група може розпочати свою екскурсію від довільного з фонтанів.

Сомхед любить тропічні рослини. Тому від кожного з фонтанів він і його група студентів підуть по найбільш красивій стежці, яка веде від цього фонтана, *за одним виключенням*: якщо самою красивою є та стежка, по якій вони тільки що пройшли. У цьому випадку, якщо є інші стежки, що йдуть від цього фонтана, то Сомхед і його група підуть по другій за красою стежці. Зрозуміло, якщо іншої стежки немає, вони повернуться по тій самій стежці, по якій і прийшли, використовуючи цю стежку другий раз поспіль. Так як Сомхед — професійний ботанік, то серед стежок немає двох однаково красивих для нього.

Студенти не дуже цікавляться рослинами. Однак їм хотілося б пообідати в шикарному ресторані, котрий знаходиться біля фонтана з номером P. Сомхед знає, що його студенти зголоднюють після того, як пройдуть рівно K стежок, де K може бути різним для різних груп студентів. Сомхеда цікавить, скільки різних маршрутів він може обрати для кожної з груп студентів, якщо:

- кожна група може починати прогулянку від довільного фонтана;
- наступна стежка завжди обирається описаним вище способом;
- кожна група має прийти до фонтана з номером P, пройшовши рівно по K стежках.

Необхідно зауважити, що на своєму шляху група студентів може проходити поруч з фонтаном із номером P, йдучи по своєму маршруту, проте маршрут обов'язково має завершуватись у фонтана з номером P.

Завдання

За наданою інформацією про фонтани і стежки необхідно знайти кількість різних маршрутів для кожної з Q груп студентів, тобто Q значень K.

Необхідно написати процедуру count_routes(N,M, P,R,Q,G), яка має такі параметри:

- N — кількість фонтанів. Фонтани нумеруються від 0 до (N–1);
- M — кількість стежок. Стежки нумеруються від 0 до (M–1). Стежки задано в порядку зменшення краси: для $0 \le i < M-1$, стежка з номером i більш красивіша, ніж стежка з номером i+1;
- P — номер фонтана, біля якого знаходиться шикарний ресторан;
- R — двовимірний масив, у якому описано стежки. Для $0 \le i < M$, стежка з номером i з'єднує фонтани з номерами $R[i][0]$ і $R[i][1]$. Варто зауважити, що кожна стежка з'єднує різні фонтани, і ніякі дві стежки не з'єднують одну й ту саму пару фонтанів;
- Q — кількість груп студентів;
- G — одновимірний масив цілих чисел, що містять значення K. Для $0 \le i < Q$ в елементі масиву $G[i]$ задано кількість стежок K, по яких має пройти i-а група.

Для $0 \le i < Q$ процедура має знайти кількість можливих маршрутів з кількістю стежок, рівною $G[i]$, по яких може пройти i-а група, щоб потрапити до фонтана з номером P. Для кожної групи з номером i процедура має викликати процедуру answer(X), щоб повідомити, що число маршрутів дорівнює X. Відповіді необхідно видавати у тому ж порядку, у якому задано групи. Якщо припустимих маршрутів не існує, процедура має викликати процедуру answer(0).

Приклади

Приклад 1

Розглянемо приклад, показаний на рис. 1, де N=6, M=6, P=0, Q=1, G[0]=3 і

$$R = \begin{matrix} 1 & 2 \\ 0 & 1 \\ 0 & 3 \\ 3 & 4 \\ 4 & 5 \\ 1 & 5 \end{matrix}$$

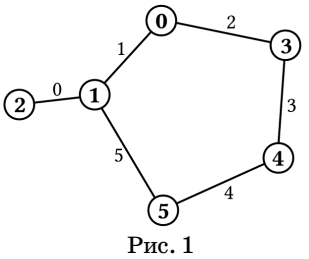

Рис. 1

Необхідно звернути увагу, що стежки перелічено в порядку спадання краси. Це значить, що стежка з номером 0 є самою красивою, стежка з номером 1 є другою за красою і так далі.

Є всього два допустимих маршрути, що проходять по трьох стежках: 1→2→1→0, і 5→4→3→0.

Перший маршрут починається біля фонтана з номером 1. Найкрасивіша стежка звідси веде до фонтана з номером 2. Біля фонтана з номером 2 у групи немає вибору, вони мають повернутися, використовуючи ту ж стежку. Тепер, знову знаходячись біля фонтана з номером 1, група не може використовувати стежку з номером 0, і замість цього обирає стежку з номером 1. Ця стежка приводить їх до фонтана з номером P=0.

Отже, потрібно викликати процедуру answer(2).

Приклад 2

Розглянемо приклад, показаний на рис. 2, де N=5, M=5, P=2, Q=2, G[0]=3, G[1]=1, і

$$R = \begin{matrix} 1 & 0 \\ 1 & 2 \\ 3 & 2 \\ 1 & 3 \\ 4 & 2 \end{matrix}$$

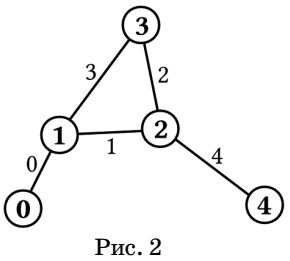

Рис. 2

Для першої групи є тільки один маршрут, який приводить до фонтана з номером 2, проходячи по трьох стежках: 1→0→1→2.

Для другої групи є два маршрути, які приводять до фонтана з номером 2, проходячи по одній стежці: 3→2, і 4→2.

Отже, правильна реалізація процедури **count_routes** має спочатку викликати **answer(1)**, щоб повідомити відповідь для першої групи, після чого викликати **answer(2)**, щоб повідомити відповідь для другої групи.

Підзадачі

Підзадача 1 (49 балів)

- $2 \le N \le 1\,000$;
- $1 \le M \le 10\,000$;
- $Q = 1$;
- Кожен елемент G є цілим числом від 1 до 100 включно.

Підзадача 3 (31 бал)

- $2 \le N \le 150\,000$;
- $1 \le M \le 150\,000$;
- $1 \le Q \le 2\,000$;
- Кожен елемент G є цілим числом від 1 до 1 000 000 000 включно.

Підзадача 2 (20 балів)

- $2 \le N \le 150\,000$;
- $1 \le M \le 150\,000$;
- $Q = 1$;
- Коженелемент G є цілим числом від 1 до 1 000 000 000 включно.

Рекомендації щодо розв'язування

У цій задачі нам потрібно підрахувати кількість маршрутів, що могли б привести кожну групу до вказаного фонтана P, використавши задану кількість кроків K.

Зауважте, що кожен маршрут повністю визначається своїм початковим фонтаном. Отже, щоб підрахувати кількість можливих маршрутів, нам достатньо перевірити для кожного фонтана, чи приведе розпочатий там маршрут до фонтана P за рівно K кроків. Оскільки нам треба перевірити всі N початкових фонтанів для кожної з Q груп, потрібно використати ефективний алгоритм для перевірки, чи буде група на перехресті P після рівно K кроків, який буде обговорюватись далі.

Побудова графу

Цю задачу можна розглядати як задачу на графі. Природним підходом буде побудувати граф G, що містить таку інформацію: для кожного фонтана, куди може рухатись група. Оскільки вони можуть обрати тільки одну з двох стежок, будемо використовувати дві вершини для подання фонтана. А саме, нехай для i-го фонтана v_i подає цей фонтан, коли наступною обраною стежкою буде найбільш красива стежка, та v_i' подає той самий фонтан, але коли наступною обраною стежкою має бути друга за красою стежка, що йде від нього (або найбільш красива стежка, якщо іншої немає). Іншими словами, v_i представляє i-й фонтан, коли остання обрана стежка не є найбільш красивою стежкою, що йде від цього фонтана, та v_i' представляє цей фонтан, коли остання обрана стежка є найкрасивішою, що йде від нього.

Для кожної вершини ми додамо до графу ребро, що виходить з цієї вершини і відповідає найкрасивішій або другій за красою стежці, відповідно до зазначених вище умов. G буде містити $2N$ вершин і рівно по одному ребру, що виходить з кожної з них.

Побудова графа G займе час $O(M+N)$, спочатку на створення $2N$ вершин, а потім на прохід по масиву R, що задає стежки і додавання відповідних ребер до G відповідно до зазначених.

Розв'язок з часом $O(M+NKQ)$

Простим способом перевірити, куди прийде група після K кроків, є симуляція їх шляху, для кожного фонтана як початкового. Оскільки вони завжди обирають найбільш красиві стежки для першого кроку, відповідні початкові вершини G будуть $v_0,\ldots,v_{N-1}$.

Щоб просимулювати їх прогулянку, просто почнемо у деякій вершині v_s, потім підемо по єдиному ребру, що з неї виходить і повторимо цей процес K разів. Оскільки вершинами, що відповідають фонтану P є v_p і v_p', маршрут закінчується біля цього фонтана тоді і тільки тоді, коли після K кроків ми зупиняємося в одній з цих вершин. Отже, щоб знайти кількість можливих шляхів, ми симулюємо прогулянки для всіх можливих початкових вершин v_i, і рахуємо кількість початкових вершин, що приводять до вершини v_p або v_p' після K кроків.

Зрозуміло, що цей процес потребує час $O(K)$ для кожної початкової вершини. Оскільки є N можливих початкових вершин і Q запитів, цей алгоритм потребує час $O(M+NKQ)$, включаючи час на побудову графа. Цього часу достатньо, щоб повністю розв'язати підзадачу 1.

Розв'язок з часом $O(M+NQ\log K)$

Зі збільшенням K у підзадачі 2 нам потрібно знати кращий метод симуляції порівняно з алгоритмом, наведеним у попередньому розділі. Зауважимо, що ребра в G задають 1 крок подорожі. Щоб проводити симуляцію швидше, використаємо підхід композиції перестановок.

На початку обчислимо результат 2^k крокової подорожі з кожної вершини G, де k=0, 1, 2,..., використовуючи техніку, схожу на техніку послідовних квадратів. Нехай $T_{v,2^k}$ задає вершину, до якої ми приходимо після подорожі з v за 2^k кроків. Тепер для k=0, 1, 2,..., ми

можемо просто обчислити $T_{v,2^k}$: якщо k=0, то відповідну вершину вказано в G; інакше потрібно побудувати 2 маршрути довжини 2^{k-1}, використовуючи формулу

$$T_{v,2^k} = T_{T_{v,2^{k-2}},2^{k-1}}.$$

Іншими словами, подорож з 2^k кроків для v це те саме, як подорож з 2^{k-1} кроків для v, після чого ще 2^{k-1} кроків з отриманої вершини.

Потім, зауважте, що кожне значення K ми можемо розкласти на суму різних не від'ємних степенів двійки. Нехай, $K=2^{k_1}+2^{k_2}+\ldots+2^{k_l}$ де $k_1<k_2<\ldots<k_l$ для деякого додатного цілого числа l. Тоді результат подорожі на k кроків з v можна буде знайти композицією подорожей на $2^{k_1},2^{k_2},\ldots,2^{k_l}$ кроків, які ми попередньо обчислили. Використовуючи цю техніку, ми можемо знайти кінцеву вершину для кожної початкової вершини за час O(log$K$).

Оскільки $K<2^{30}$, нам потрібно обчислити $T_{v,2^k}$ лише для k=0, 1, 2,..., 29.

Цей алгоритм потребує додатково O(Nlog$K$) час на обчислення значень $T_{v,2^k}$, оскільки кожне з них може бути обчислене за константний час. Потім, ми можемо знайти кінцеву вершину для кожного з маршрутів за час O(log$K$). Отже, загальний час виконання буде O(M+NQlog$K$), чого достатньо для повного розв'язання підзадачі 2.

Розв'язок з часом O(M+NQ)

Розглянемо більш загальне питання визначення, чи закінчується маршрут з початком у вершині s та довжиною K у вершині t. Згадаємо, що кожна вершина в G має рівно одне ребро, що виходить з неї. Отже, з довільної початкової вершини, просто рухаючись по ребрах, ми в якийсь момент попадемо у цикл. Таким чином, почавши з s, буде виконуватись одна з умов:

- ми ніколи не потрапимо у t;
- ми потрапляємо до t тільки раз після деякої кількості кроків F. У цьому випадку t є досяжною, але не в циклі;
- ми потрапляємо до t перший раз після F кроків, і після цього кожні C кроків. У цьому випадку t є досяжною і вона знаходиться у циклі довжини C.

Для розв'язання задачі s може змінюватись залежно від початкової вершини, а саме вона може бути довільною вершиною v_i для i=0, 1,..., N−1. Однак t може бути тільки вершиною v_p або v_p'. Оскільки t майже не змінюється, простіше перевірити, чи лежить t у циклі і чи можливо досягти t з s.

Щоб розв'язати цю задачу, ми створимо граф G^T, який є таким самим як і граф G, але з розвернутими ребрами. Після цього виконаємо пошук у глибину на графі, розпочавши у t. Протягом пошуку будемо записувати відстані від t до кожної з досяжних вершин. Це число буде відстанню від s до t в G; тобто кількість кроків F, що приводять нас від s до t перший раз. Одночасно, якщо ми потравляємо у якусь вершину, з якої ребро веде до t, то ми отримуємо розмір циклу C, який є відстанню від t до цієї вершини плюс 1.

Отже, перевірити, чи закінчується в t маршрут у G, що починається в s і має довжину K, можна так:

- Якщо s є не досяжною в G^T,то такий маршрут в G не може закінчуватись в t.
- Якщо ми досягаємо s в G^T за F кроків, але не в циклі, то цей маршрут в G закінчується в t тоді і тільки тоді коли K=F.
- Якщо ми досягаємо s в G^T за F кроків, і t знаходиться у циклі розміру C, то цей маршрут в G закінчується в t тоді і тільки тоді, коли K=F+nC для якогось не від'ємного цілого n.

Для нашої задачі, маршрут довжини K з початком біля i-го фонтана буде закінчуватись біля фонтана P тоді і тільки тоді, якщо маршрут в G, що починається в v_i і має довжину K, закінчується в v_p або v_p'. Зауважте, що протягом реалізації нам не потрібно будувати граф G, достатньо побудувати відразу граф G^T. Також зручно створити на початку масив для зберігання F_s для кожної вершини s. Ми проініціалізуємо F_s та C як нескінченність, і будемо модифікувати їх протягом пошуку в глибину. Виконаємо два пошуки, починаючи в v_p і v_p' відповідно.

Оскільки пошук у глибину потребує час O(M+N) і кожен запит перевіряється за константний час, цей алгоритм потребує загалом часу O(M+NQ), чого достатньо для отримання всіх балів по задачі.

2. ПЕРЕГОНИ

Поряд з IOI в Паттаї проходять *міжнародні олімпійські перегони* (МОП) 2011. Стороні, що приймає, потрібно знайти найбільш придатну для перегонів трасу.

У регіоні Паттая-Чонбурі знаходиться N міст, з'єднаних мережею із N−1 магістралей. Кожна магістраль — двостороння, з'єднує два різних міста, і для неї відомо довжину в кілометрах — ціле число. Відомо, що між кожною парою міст існує *рівно один* можливий шлях, що з'єднує ці міста. Отже, для довільної пари міст існує рівно одна послідовність різних магістралей, по яких можна проїхати з одного міста в інше, не відвідуючи ніяке місто двічі.

За вимогами МОП траса має бути шляхом сумарної довжини *рівно* K кілометрів, що починається і закінчується в різних містах. Зрозуміло, що ніяка магістраль і, тому, ніяке місто не можуть бути використані двічі під час вибору траси, інакше можливі зіткнення. Щоб мінімізувати вплив перегонів на дорожний рух у регіоні, необхідно обрати для траси шлях з найменшою можливою кількістю магістралей.

Завдання

Написати процедуру best_path(N,K,H,L), якій передаються такі параметри:

- N — кількість міст. Міста нумеруються від 0 до N−1;
- K — потрібна довжина траси;
- H — двомірний масив, що описує магістралі. Для 0≤i<N−1 магістраль з номером i з'єднує міста з номерами H[i][0] і H[i][1];
- L — одномірний масив, що містить довжини магістралей. Для 0≤i<N−1 довжина магістралі з номером i дорівнює L[i].

Гарантується, що всі значення в масиві H лежать у межах від 0 до N−1 включно, і описані в цьому масиві магістралі з'єднують всі міста, як описано вище. Також гарантується, що всі значення в масиві L — цілі числа у межах від 0 до 1 000 000 включно.

Ваша процедура має повертати *мінімальну можливу кількість магістралей* на допустимій трасі,

що має довжину, рівну K. Якщо такої траси не існує, ваша процедура має повернути значення −1.

Приклади

Приклад 1

Розглянемо приклад, зображений на рис. 3, де N=4, K=3,

$$H=\begin{matrix}0&1\\1&2\\1&3\end{matrix}\qquad L=\begin{matrix}1\\2\\4\end{matrix}$$

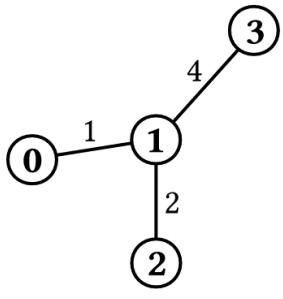

Рис. 3

Траса може починатись у місті з номером 0, проходити через місто з номером 1 і закінчуватись у місті з номером 2. Довжина траси буде дорівнювати (1+2)=3 км, як і потрібно, вона складається з двох магістралей. Це найкраща можлива траса; тому процедура best_path(N,K,H,L) має повернути значення 2.

Приклад 2

Розглянемо приклад, зображений на рис. 4, де N=3, K=3,

$$H=\begin{matrix}0&1\\1&2\end{matrix}\qquad L=\begin{matrix}1\\1\end{matrix}$$

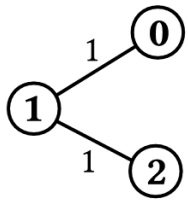

Рис. 4

Тут допустимої траси не існує. У цьому прикладі процедура best_path(N,K,H,L) має повернути значення −1.

Приклад 3

Розглянемо приклад, зображений на рис. 5, де N=11, K=12,

$$H=\begin{matrix}0&1\\0&2\\2&3\\3&4\\4&5\\0&6\\6&7\\6&8\\8&9\\8&10\end{matrix}\qquad L=\begin{matrix}3\\4\\5\\4\\6\\3\\2\\5\\6\\7\end{matrix}$$

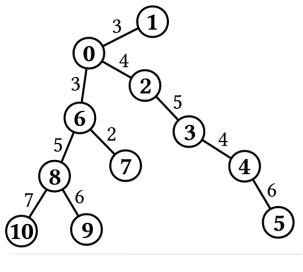

Рис. 5

Одна з можливих трас складається з 3 магістралей: вона йде з міста з номером 6 через міста з номерами 0 і 2 в місто з номером 3. Інша траса починається в місті з номером 10 і йде через місто з номером 8 у місто з номером 6. Обидві ці траси мають довжину 12 кілометрів, як і вимагається. Друга з них оптимальна, так як не існує придатної траси з одної магістралі. Отже, процедура best_path(N,K,H,L) має повернути значення **2**.

Підзадачі

Підзадача 1 (9 балів)

$1 \le N \le 100$;
$1 \le K \le 100$.

Мережа магістралей є лінією: для $0 \le i < N-1$, магістраль з номером i з'єднує міста з номерами i і $i+1$.

Підзадача 2 (12 балів)

$1 \le N \le 1\,000$;
$1 \le K \le 1\,000\,000$.

Підзадача 3 (22 бали)

$1 \le N \le 200\,000$;
$1 \le K \le 100$.

Підзадача 4 (57 балів)

$1 \le N \le 200\,000$;
$1 \le K \le 1\,000\,000$.

Рекомендації щодо розв'язування

Маючи дерево T з N вершинами, у цій задачі потрібно знайти шлях P^* довжини K з мінімальною кількістю ребер. Виглядає як звичайна задача на динамічне програмування, але коли K велике, потрібно використовувати інший підхід.

Зразковий розв'язок цієї задачі використовує підхід «розділяй і володарюй».

Розглянемо вершину графа u. Можливі 2 випадки: коли вузол u належить потрібному шляху P^* або коли вузол u не належить.

У другому випадку ми видаляємо вузол u з дерева, чим розбиваємо його на менші дерева. Після цього можна рекурсивно аналізувати кожне з дерев для пошуку розв'язку.

Маючи на увазі цей загальний підхід, потрібно відповісти на такі запитання:

- Як знайти найкращий шлях, що містить вузол u?
- Як вибрати u, щоб досягнути кращого часу виконання програми?

Зауважте, що друге запитання є дуже важливим, оскільки якщо ми можемо гарантувати, що розміри отриманих дерев є малими, ми можемо обмежити кількість рівнів рекурсії.

Пошук розв'язку, що містить вузол u

Розглянемо випадок, коли P^* містить вузол u. Простим випадком є випадок, коли нам потрібно з'ясувати, чи існує шлях довжини рівно K, що містить u.

Якщо u є одним із кінцевих вузлів P^*, потрібний шлях можна знайти, використавши пошук у глибину.

Однак, якщо u знаходиться всередині P^*, то дві суміжні до u вершини x і y також мають бути у P^*. Отже, нам потрібно знайти x і y.

Розглянемо вершину w, суміжну до u. За допомогою пошуку в глибину ми можемо знайти множину L_w довжин всіх шляхів, що починаються u й містять ребро $(u;w)$.

Отже, щоб знайти x і y нам потрібно знати дві вершини x і y, такі що існує пара $l_x \in L_x$ і $l_y \in L_y$ для яких $l_x + l_y = K$. Це можна зробити пошуком у глибину від u через кожне ребро $(u;w)$ для всіх суміжних вузлів w, використовуючи для запису результатів масива $A[0; \ldots; K]$ розміру $K+1$.

Час виконання цього кроку є $O(N)$.

Пошук потрібного вузла

Нашою метою є знаходження такого вузла u, що після його видалення всі дерева, що залишилися, є достатньо малими. У цьому випадку ми повинні знайти вузол u, такий що кожне з отриманих дерев

буде містити не більше $N/2$ вузлів. Будемо називати такий вузол u центральним *вузлом*.

Спочатку треба впевнитись, що такий вузол існує.

Беремо кандидатом довільний вузол v. Позначимо $T'=T\setminus\{v\}$ ліс, що отримується видаленням v з T. Для кожного суміжного до v вузла w, позначимо T_w дерево, що містить w в T'. Якщо кожне дерево $T_w \in T'$ містить не більше $N/2$ вузли, то пошук можна завершувати і v є потрібним центральним вузлом.

Інакше існує одне дерево T_w, що містить більше $N/2$ вузли (зрозуміло, що може бути тільки одне дерево, що порушує наш критерій). У цьому випадку ми обираємо w нашим новим кандидатом і повторюємо процес.

У якийсь момент цей процес зупиниться на якомусь кандидаті, і це буде потрібний центральний вузол. Це слідує з того, що після аналізу вузла v ми ніколи не повернемось назад до v; і оскільки є всього N вузлів, процес буде повторюватись не більше N разів.

Впевнившись, що центральний вузол існує, треба знайти спосіб його знаходження. Можна реалізувати описаний вище алгоритм, але він занадто повільний. Нижче описано 2 процедури, які знаходять центральний вузол за час $O(N\log N)$ і $O(N)$.

Підхід знизу-вверх

Ми можемо знайти вузол u знизу-вверх. Будемо підтримувати чергу з пріоритетами Q для всіх «оброблених» піддерев, використовуючи їх розмір як вагу.

Для кожного вузла будемо підтримувати його статус, яким може бути *новий* або *оброблений*. На початку всі вузли є новими. Кожен вузол також має вагу. На початку кожен вузол має вагу 1.

Занесемо в чергу Q усі листові вузли. Зауважимо, що кожен вузол у Q це вузол, у якого оброблено всі суміжні вузли, крім одного. Для кожного вузла $v\in Q$, позначимо $p(v)$ єдиного нового сусіда v.

Поки в Q є вузли, візьмемо вузол v з найменшою вагою. Змінимо статус v на «оброблено» і збільшимо вагу $p(v)$ на вагу v. Якщо всі сусіди $p(v)$, крім одного, оброблені, додамо v у чергу Q.

Останній вузол, який додано до Q, буде шуканим центральним вузлом.

Пошук вглибину з підрахунками

Використовуючи пошук у глибину й охайні підрахунки, ми можемо знайти центральний вузол за час $O(N)$.

Візьмемо довільний вузол r, щоб розпочати пошук у глибину. У цій процедурі будемо розглядати T як дерево з коренем в r і визначеним відношенням батько-син між суміжними вузлами.

Під час пошуку в глибину для кожного вузла v будемо підраховувати кількість його нащадків $D(v)$.

Маючи цю інформацію, ми можемо з'ясувати, чи є кандидат u центральним вузлом. Для кожного вузла w, суміжного до u, якщо w є одним із синів u, розмір отриманого після видалення u дерева, що містить w, є $D(w)+1$. Якщо w є батьком u, розмір дерева, що містить w, після видалення u буде

$$n-1-\sum_{v\in Ch(u)}(D(v)+1),$$

де $Ch(u)$ є множиною синів u. Якщо розмір кожного отриманого дерева є не більше $N/2$, u є шуканим центральним вузлом. Час, необхідний для перевірки u, пропорційний степені u. Отже, ми можемо перевірити всі вузли за час $O(N)$.

Час виконання

Нехай $\tau(N)$ буде найгіршим часом виконання, коли дерево містить N вузлів. Можемо записати у таку рекурентність:

$$\tau(N)=A(N)+cN+\sum_i \tau(N_i),$$

де $A(N)$ є часом для знаходження u, N_i є розміром i-го нового дерева і c є деякою константою.

Оскільки ми знаємо, що $N_i \le N/2$, буде не більше $O(\log N)$ рівнів рекурсії.

Якщо ми використаємо алгоритм з часом $O(N)$ для пошуку u, кожен рівень буде виконано за час $O(N)$ і загальний час виконання буде $O(N\log N)$. Якщо ми використаємо більш повільний алгоритм з часом виконання $O(N\log N)$, загальний час виконання буде $O(N\log^2 N)$.

Зауваження

Існують інші евристики для пошуку u, які не завжди працюють. Наприклад:

- у методі розділяй і володарюй обирається вузол із найбільшою степенню;
- у методі розділяй і володарюй обирається вузол, що мінімізує максимальну відстань до всіх вузлів.

3. РИСОСХОВИЩЕ

У сільській місцевості знаходиться довга пряма дорога, яка відома як Рисовий Шлях. Вздовж цієї дороги розташовано R рисових полів. Кожне поле має цілочисельну координату від 1 до L включно. Рисові поля задаються в порядку неспадання їхніх координат. Формально, позначимо для $0\le i<R$ координату рисового поля з номером i як $X[i]$. Гарантується, що $1\le X[0]\le\ldots\le X[R-1]\le L$.

Зазначимо, що *кілька рисових полів можуть мати однакову координату*.

Планується побудувати одне *рисосховище*, у яке потрібно завезти з полів якомога більше рису. Як і рисові поля, *рисосховище повинно мати цілочисельну координату* від 1 до L включно. Рисосховище дозволяється будувати в довільній цілочисельній координаті, у тому числі й у координаті, у якій уже є одне або більше рисових полів.

З кожного поля протягом кожного сезону збирають врожай, який вміщається *рівно в 1 вантажівку*. Щоб доставити урожай у рисосховище, необхідно найняти водія вантажівки. Вартість роботи водія з перевезення вантажу на одиницю відстані складає 1 бат. Іншими словами, вартість транспортування рису від заданого поля до рисосховища дорівнює модулю різниці їхніх координат.

На жаль, бюджет на поточний сезон обмежено: неможна витратити більше B бат на транспортування. Необхідно побудувати рисосховище у такому місці, щоб можна було завезти в нього якомога більше рису.

Завдання

Написати процедуру besthub(R,L,X,B), якій передаються такі параметри:

- R — кількість рисових полів. Поля пронумеровано від 0 до (R–1);
- L — максимальна координата;

- X — одномірний масив цілих чисел, відсортованих у порядку неспадання. Для кожного i ($0 \le i < R$) рисове поле з номером i має координату $X[i]$;
- B — бюджет.

Ваша процедура має знаходити оптимальне розташування рисосховища і повертати максимальну кількість вантажівок рису, які можуть бути перевезені у рисосховище, не перевищуючи заданий бюджет.

Необхідно звернути увагу на те, що вартість транспортування рису може бути дуже великою. Бюджет задається 64-бітним цілим числом, і тому рекомендується використовувати для обчислень 64-бітні цілі числа. У мовах C/C++ використовуйте тип longlong; у мові Паскаль використовуйте тип Int64.

Приклад

Розглянемо приклад, уякому R=5, L=20, B=6, і

X= 1, 2, 10, 12, 14

Рис. 6

У цьому прикладі існує кілька оптимальних розташувань рисосховища. Можна помістити його у довільній точці з цілими координатами від 10 до 14 включно. На рисунку вище показано одне із цих оптимальних розташувань. У цьому випадку до рисосховища можна перевезти рис з полів у точках з координатами 10, 12 і 14. Для кожного з цих оптимальних розташувань загальна вартість транспортування буде не більше 6 бат. Очевидно, що жодне з можливих розташувань рисосховища не дозволить зібрати рис більше ніж з трьох полів, отже, цей розв'язок є оптимальним і процедура besthub має повертати число 3.

Підзадачі

Підзадача 1 (17 балів)

- $1 \le R \le 100$
- $1 \le L \le 100$
- $0 \le B \le 10\,000$
- Ніякі два рисових поля не мають однакову координату (*тільки для цієї підзадачі!*).

Підзадача 2 (25 балів)

- $1 \le R \le 500$
- $1 \le L \le 10\,000$
- $0 \le B \le 1\,000\,000$

Підзадача 3 (26 балів)

- $1 \le R \le 5\,000$
- $1 \le L \le 1\,000\,000$
- $0 \le B \le 2\,000\,000\,000$

Підзадача 4 (32 бали)

- $1 \le R \le 100\,000$
- $1 \le L \le 1\,000\,000\,000$
- $0 \le B \le 2\,000\,000\,000\,000\,000$

Рекомендації щодо розв'язування

Ключовим спостереженням у цій задачі є те, що для довільних K рисових полів, розташованих у координатах $r_0 \le r_1 \le \ldots \le r_{K-1}$ вартість транспортування з усіх цих K полів мінімізується, якщо розташувати рисосховище на медіані. Наприклад, коли K=1, сховище має бути у r_0, а коли K=2, є оптимальним його розташування між r_0 і r_1. У цій задачі для простоти будемо розташовувати сховище у точці $r_{[K/2]}$. Слідуючи цьому спостереженню, позначимо розв'язок послідовністю $S \subseteq \langle r_0, \ldots, r_{R-1} \rangle$ і нехай $|S|$ позначає довжину S, що є значенням нашого розв'язку (кількість рисових полів з яких рис буде транспортуватись до сховища). Вартість S є

$$\mathrm{cost}(S) = \sum_{r_j \in S} \left| r_j - h(S) \right|,$$

де $h(S)$ є $\lfloor |S|/2 \rfloor$-им елементом S.

Розв'язок з часом $O(R^3)$

Базуючись на описаному вище, можна розв'язувати задачу з використанням алгоритму «вгадай і перевір». Будемо перевіряти всі можливі довжини S (з діапазону від 1 до R). Далі побачимо, що в довільному оптимальному розв'язку S^* використані поля розташовано неперервно одне за одним, тобто S^* є обов'язково $(r_s, r_{s+1}, \ldots, r_t)$ для деякого $0 \le s \le t \le R-1$. Отже, є всього $R-K+1$ розв'язків довжини K. Для кожного вибору S ми обчислюємо $h(S)$ і вартість транспортування за час $O(|S|)$ і перевіряємо, чи знаходимось ми в межах бюджету B. Це призводить до алгоритму з часом $O(R^2)$, чого достатньо для розв'язання підзадачі 2.

Розв'язок з часом $O(R^2)$

Щоб покращити час до $O(R^2)$, ми пришвидшимо обчислення cost(S). Помітимо, що ми маємо справу тільки з послідовними рисовими полями. Отже, для кожного S, вартість cost(S) може бути обчислена за $O(1)$ після попереднього обчислення префіксних сум. А саме, нехай $T[i]$ буде сумою координат ліворуч від рисового поля i, тобто

$$T[0] = 0 \text{ і } T[i] = \sum_{j=0}^{i-1} X[j].$$

Потім, якщо $S = \langle r_s, \ldots, r_t \rangle$, cost($S$) визначається як

$$(p-s)r_p - (T[p] - T[s]) + (T[t+1] - T[p+1]) - (t-p)r_p,$$

де $p = \lfloor (s+t)/2 \rfloor$.

Цього алгоритму з часом виконання $O(R^2)$ достатньо для розв'язання підзадачі 3.

Розв'язок з часом $O(R \log R)$

Застосування бінарного пошуку довжини замість лінійного пошуку покращує час до $O(R \log R)$, цього достатньо, щоб розв'язати всі підзадачі.

Розв'язок з часом $O(R)$

Ми замінимо бінарний пошук варіантом лінійного пошуку, який ретельно зроблений, щоб використати результати, які отримуються кожен раз, коли ми аналізуємо комбінації рисових полів. Припустимо, що ми додаємо рисові поля одне за одним. На ітерації i, ми додаємо r_i і знаходимо (а) S_i^* кращий розв'язок, що використовує тільки якісь з перших i рисових полів (тобто, $S_i^* \subseteq \langle r_0, \ldots, r_i - 1 \rangle$), і (б) S_i, кращий розв'язок, що використовує тільки якісь з перших i рисових полів і *включає* поле $r_i - 1$. Це можна підрахувати індуктивно так. Як базовий випадок, коли i=0, як S_i, так і S_i^* є просто $\langle r_0 \rangle$ та cost=0, що знаходиться у рамках бюджету $B \ge 0$. Індуктивно, припустимо, що нам відомо S_i і S_i^*. Тепер нехай S_{i+1} буде S_i, до якого додали r_i позначене $S_i \cdot r_i$, якщо вартість cost($S_i \cdot r_i$) не перевищує B, або у протилежному випадку це найдовший суфікс $S_i \cdot r_i$, який коштує не більше B. Далі, S^*_{i+1} буде кращим з S_i^* та S_{i+1}. Щоб це реалізувати, представимо кожен S_i своєю початковою точкою s та кінцевою точкою t. На кожній ітерації будемо збільшувати t та, можливо, s, але s завжди не перебільшує t. Оскільки cost($\langle r_s, \ldots, r_t \rangle$) обчислюється за $O(1)$, загальний час виконання алгоритму буде $O(R)$, чого достатньо для розв'язання всіх підзадач.