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

Бондаренко В.В., Галковський Т.О., Коротков А.С., Нейтер Д.Ю., Ягіяєв Ш.І.

ЗАВДАННЯ ПЕРШОГО ТУРУ

1. Брехуни

У ток-шоу бере участь N знайомих між собою осіб, серед яких можуть бути такі, що завжди говорять неправду, а решта завжди говорять правду (принаймні одна особа). Наприкінці програми ведучий вирішив визначити, хто з учасників до якої групи належить. Для цього він задав запитання: «Скільки серед вас тих, хто завжди говорить правду?». Кожен з учасників дав відповідь: число від 0 до N. Після цього ведучий може відібрати певних осіб, задати їм те ж саме запитання, та, отримавши відповідь, гарантовано визначити, хто з учасників ток-шоу говорить правду, а хто бреше. Учасники відповідають на друге запитання лише щодо вибраних ведучим людей.

Завдання. Напишіть програму LIARS, яка за кількістю учасників ток-шоу та їхніми відповідями на перше запитання знайде мінімальну кількість осіб, яку необхідно обрати ведучому для другого етапу опитування.

Вхідні дані. Перший рядок вхідного файлу LIARS.DAT містить одне ціле число N ($1 \le N \le 1000$) — кількість учасників ток-шоу. Наступний рядок містить N цілих чисел від 0 до N — відповіді кожного з учасників на перше запитання.

Вихідні дані. Єдиний рядок вихідного файлу LIARS. SOL має містити ціле число — шукану мінімальну кількість учасників, яку ведучому необхідно обрати для повторного опитування. У випадку, якщо ведучий має можливість виявити брехунів та тих, хто завжди каже правду, вже після першого етапу опитування, потрібно вивести число 0.

Приклад вхідних та вихідних даних

LIARS.DAT	LIARS.SOL
4 3 1 3 3	2

LIARS.DAT	LIARS.SOL
3 1 2 3	0

Рекомендації щодо розв'язування

Хто може потенційно казати правду? Припустимо, що серед учасників ток-шоу рівно k людей, що завжди говорять правду. Тоді всі вони дадуть у відповідь число k, а решта, як брехуни, дадуть відповідь, відмінну від k. Отже, буде одержано рівно k відповідей k. Отже, якщо розбити учасників на групи за даною ними відповіддю, то кожна така група або вся каже правду, або вся бреше, причому необхідною умовою для їх правдивості є рівність кількості людей у групі даній цією групою відповіддю.

Якщо остання умова виконується, то група потенційно може бути правдивою. Підрахуємо кількість таких груп. Якщо така група одна, то, зрозуміло, вона і є правдивою. Якщо таких груп не менше за дві, то потенційно кожна із них може бути правдивою, причому за результатами першого опитування виявити, яка саме з них каже правду, неможливо, бо за припущення правдивості будь-якої з них не виникає суперечностей з умовою (усі відповіді на перше запитання будуть коректними, яка з груп не була б правдивою).

Вибір учасників для другого опитування. Для другого опитування достатньо вибрати по одному учаснику з кожної групи, *для якої дана ними відповідь рівна кількості людей, що дали таку відповідь*. Тоді серед цих учасників буде рівно один, що завжди каже правду (правдива лише одна з груп), і він у другому опитуванні дасть відповідь 1, а ніхто інший відповідь 1 не дасть, що і дозволить виявити його, а за ним і всю його групу. *Відповіддю буде кількість вище вказаних груп, якщо таких груп не менше за 2, та 0, якщо така група одна*.

Доведення необхідності такого вибору. Якщо з деякої потенційно правдивої групи не взяти учасників на друге опитування взагалі, то якщо ця група виявиться правдивою, то на друге опитування потраплять самі брехуни, і вони можуть дати будь-які ненульові відповіді. Якщо ж на друге опитування потрапить хоч один, що говорить правду, то ті, що говорять правду, дадуть ненульову відповідь, і, якщо брехуни також дадуть ненульові відповіді, то одержимо два різних несуперечливих сценарії розвитку другого опитування, і для довільного обраного критерію визначення тих, що кажуть правду, в одному випадку він спрацює, а в іншому ні. Тому в такому разі таке друге опитування не дозволить однозначно визначити тих, що говорять правду. Отже, вибір хоча б одного учасника з кожної потенційно правдивої групи є необхідною умовою визначення тих, що говорять правду, за другим голосуванням. Як було зазначено вище, вибір рівно одного учасника з кожної з таких груп є достатнім для визначення тих, що говорять правду, за другим голосуванням. Отже, вибір рівно одного учасника з кожної такої групи і є оптимальним розв'язком поставленої задачі. Ще раз зазначимо, що за наявності лише однієї потенційно правдивої групи проводити друге опитування не потрібно.

2. Гірський туризм

Клуб з активного туризму на планеті Олімпія вирішив запропонувати клієнтам маршрут вздовж мальовничого гірського хребта. Хребет достатньо довгий і його важко пройти відразу, тому у клубі шукають найпривабливіший з маршрутів обмеженої довжини. Згідно з результатами соціального дослідження туристи полюбляють проходити місцями, які вищі за інші на якомога більшому проміжку, завдяки ширшому огляду та ейфорії від відчуття висоти.

Для спрощення задачі хребет розділили на одометрові відрізки та визначили середню висоту над рівнем моря кожного з них. Числове значення привабливості кожного такого відрізка хребта дорівнює кількості послідовних відрізків ліворуч і праворуч, починаючи з безпосередніх його сусідів, які мають висоту строго меншу ніж він сам. Сам відрізок до цієї суми не входить. Привабливість маршруту обчислюється як сума привабливостей одометрових відрізків хребта, що до нього входять. Довжина маршруту має бути не більша за T метрів. Напрямок маршруту значення не має, оскільки не змінює його привабливості. Маршрут може починатися з будь-якого відрізку хребта. Маршрут не може містити розривів, тобто до маршруту можна включати лише послідовні відрізки хребта.

Завдання. Напишіть програму RIDGE, що за інформацією про висоту над рівнем моря кожного одометрового відрізку гірського хребта обчислить привабливість найбільш привабливого маршруту довжини не більше ніж T метрів.

Вхідні дані. Перший рядок вхідного файлу RIDGE.DAT містить два цілих числа N — довжина всього хребта у метрах та T ($1 \le T \le N \le 100\,000$) — обмеження на довжину маршруту. Другий рядок файлу містить N цілих чисел від 1 до 10^6 — висоти послідовних одометрових відрізків.

Вихідні дані. Єдиний рядок вихідного файлу RIDGE.SOL має містити одне ціле число — числове значення привабливості найпривабливішого маршруту вздовж гірського хребта довжини не більше ніж T.

Приклад вхідних та вихідних даних

RIDGE.DAT	RIDGE.SOL
10 5 1 2 3 4 5 4 3 2 1 5	18

На рисунку 1 зображено хребет, що відповідає вхідним даним, розбитий на 10 відрізків. Числа зверху відповідають висоті кожного з відрізків, а числа знизу — привабливості відповідного відрізку.

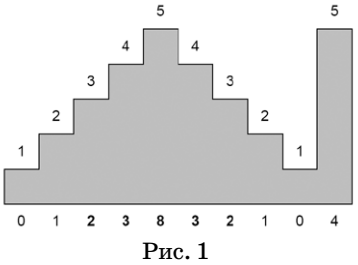

Рис. 1

Рекомендації щодо розв'язування

Формалізація задачі і загальна ідея розв'язку. Нехай N — кількість одометрових відрізків, на які розбито хребет. T — максимальна допустима довжина шуканого маршруту. H_i — висота i-го відрізка (починаючи з 1). Нехай p_i — привабливість i-го відрізка. Також нехай L_i — індекс останнього відрізка такого, що $L_i < i$ і $H_{L_i} \ge H_i$ (якщо такого відрізка не існує, то $L_i = 0$). Аналогічно R_i — індекс першого відрізка такого, що $R_i > i$ і $H_{R_i} > Hi$ (якщо такого відрізка не існує, то $R_i = N+1$). Тоді $p_i = R_i - L_i - 2$ і $S_i = p_i + p_{i+1} + \cdots + p_{i+T-1}$ задача полягає у знаходженні $\max\{S_i\}$ (очевидно, оскільки всі p_i невід'ємні, то завжди оптимально буде вибирати маршрут довжини саме T).

Розв'язок задачі складається з двох етапів:

1. Знайти значення L_i і R_i.

2. Знайти $\max\{p_i + p_{i+1} + \ldots + p_{i+T-1}\}$.

Можливі варіанти реалізації першого етапу. Очевидна реалізація першого етапу алгоритму має часову складність за $O(N^2)$, що є недостатньо ефективним для розв'язку задачі. Ефективні реалізації мають часову складність $O(N)$.

Крім того, існують реалізації зі складність $O(N \log N)$, але вони є менш ефективними і потребують складних структур даних (дерева відрізків, бінарні дерева, або RMQ + бінарний пошук).

Range Minimum Query — структура, яка дозволяє відповідати на запити максимумів на довільних відрізах за час $O(1)$ за умови виконання попередньої підготовки за час $O(N \log N)$ і потребує $O(N \log N)$ пам'яті. Існує також реалізація RMQ, яка потребує $O(N)$ пам'яті і часу на підготовку, але вона є занадто складною для написання під час туру.

Очевидний алгоритм. Для кожного i від 1 до N знаходимо L_i за такою схемою:

1. Покладемо $k = i-1$.

2. Доки є правильним, що $k > 0$ і $H_k < H_i$, присвоюємо $k = k-1$.

3. $L_i = k$.

R_i знаходяться аналогічно.

Часова складність такого алгоритму $O(N^2)$, просторова — $O(N)$.

Ефективний алгоритм 1. Ключовим для роботи даного алгоритму є той факт, що якщо $H_k < H_i$, то $H_j \le H_k < H_i$ для всіх j від L_k+1 до k (за означенням L_k). Тому у реалізації очевидного алгоритму від k замість $k-1$, можемо перейти на L_k. Назвемо це перестрибуванням з k на L_k. Алгоритм набуде такого вигляду:

Для кожного i від 1 до N знаходимо L_i за наступною схемою:

1. Покладемо $k = i-1$.

2. Доки є правильним, що $k > 0$ і $H_k < H_i$, присвоюємо $k = L_k$.

3. $L_i = k$.

R_i знаходяться аналогічно.

Коректність алгоритму випливає з наведених вище міркувань.

Доведемо, що часова складність алгоритму $O(N)$. Назвемо пропуском k ситуацію, коли на другому кроці відбувається перехід від k до L_k. Доведемо таку лему:

Лема 1. *Кожне k протягом роботи алгоритму може бути пропущено не більше одного разу. Після пропуску позиція k більше на другому кроці розглядатися не буде.*

Доведення. Припустимо, що один раз вже стався пропуск k під час пошуку L_{i_0}. Тоді $H_k < H_{i_0}$. І припустимо, що k повторно розглядається при пошуку L_i,

$i>i_0$. Тоді перед розглядання k ми повинні перетнути i_0. Є два варіанти:

- i_0 було пропущено. Але $H_k<H_{i_0}$, з чого випливає $L_{i_0}\neq k$. Протиріччя.
- i_0 було перестрибнуто під час пропуску i_1 ($i_0<i_1<i$). Тоді $H_k<H_{i_0}<H_{i_1}$. Але $H_{i_1}\le H_{L_{i_1}}\le H_{L_{L_{i_1}}}\le \ldots \le H_k$. Протиріччя.

В обох випадках ми прийшли до протиріччя. Отже початкове припущення було неправильне, і після пропуску позиція k більше розглядатись не буде.

Тепер проаналізуємо час виконання алгоритму. Кроки 1 і 3 виконуються один раз для кожного i від 1 до N. Крок 2 за доведеною вище лемою виконується для кожного k від 1 до N не більше одного разу. Крім того, одноразове виконання кожного разу потребує O(1) часу. Таким чином, сумарна часова складність алгоритму O(N). Просторова складність, очевидно, теж складає O(N).

Ефективний алгоритм 2. Є альтернативний алгоритм, що дозволяє визначити L_i і R_i за O(N). У процесі роботи цей алгоритм використовує стек S, у якому зберігаються індекси k елементів, для яких у поточний момент вже відомо L_k, але ще не відомо R_k. Позначимо вершину стеку $Top(S)$. Алгоритм має такий вигляд:

1. На початку роботи стек S порожній.

2. Для кожного i від 1 до N виконуємо наступні дії:

a) Доки S не порожній і $H_{Top(S)}<H_i$, виштовхуємо Top(S) зі стеку. При цьому присвоюємо $R_{Top(S)}=i$.

b) Якщо стек порожній, $L_i=0$, інакше $L_i=Top(S)$.

c) Якщо стек непорожній і $H_{Top(S)}=H_i$, виштовхуємо $Top(S)$ зі стеку. При цьому присвоюємо $R_{Top(S)}=i$.

3. Для всіх k, що залишились у стеку після завершення другого кроку, присвоюємо $R_k=N+1$.

Кожне число протягом роботи алгоритму попадає у стек рівно один раз, тому в сумі алгоритм виконується за O(N) операцій.

Висоти елементів, які знаходяться в стеку, є строго спадною послідовністю. Число буде виштовхнуто зі стеку тільки тоді, коли буде знайдено елемент із більшою або рівною висотою. З цього випливає коректність алгоритму.

Можливі варіанти реалізації другого етапу

Очевидний алгоритм. Для кожного i від 1 до $N-T+1$, підраховуємо S_i прямим сумуванням доданків і вибираємо максимум отриманих сум.

Часова складність O(NT).

Ефективний алгоритм. Використаємо метод ковзання. Спочатку знайдемо S_1 за означенням. Далі для знаходження S_{i+1} використаємо тотожність $S_{i+1}=S_i+p_{i+T}-p_i$.

Очевидно, часова складність такого алгоритму буде O(N).

Особливості реалізації. Під час реалізації треба враховувати, що S_i можуть не вміщуватися у 32-бітні типи даних, тому для підрахунку сум треба використовувати 64-бітні типи.

Крім того, роботу програми можна прискорити, якщо одразу весь зміст вхідного файлу у тимчасовий буфер та виділити з нього числа вручну. Але це не є обов'язковим для ефективного розв'язку.

3. Танглеграм

На площині розташовано два повних бінарних дерева глибини N. Їх 2×2^N листків розміщено на двох паралельних прямих і пронумеровано зліва направо. Листки з однаковими номерами у першому та другому деревах знаходяться один напроти одного. Між листками дерев задано відповідність — кожен із листків одного дерева має рівно один відповідний йому листок у іншому дереві і навпаки. На рисунку 2 такі відповідності задані прямими відрізками між листками. Таку конструкцію — два дерева разом із відповідностями між листками — називають танглеграмом. Танглеграми, наприклад, використовуються біологами під час дослідження взаємозв'язку генів різних видів рослин.

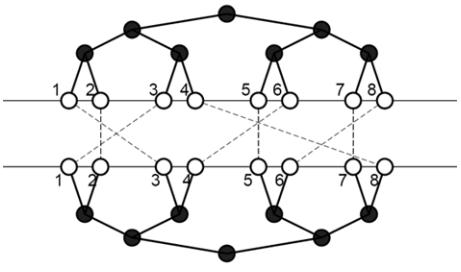

Рис. 2

Танглеграм важко досліджувати, якщо багато з відрізків (відповідностей) перетинаються. Для того щоб зменшити кількість перетинів дозволено здійснювати єдину операцію: в першому (верхньому) дереві можна міняти місцями два піддерева довільної вершини, як зображено на рис. 3.

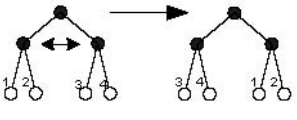

Рис. 3

Завдання. Напишіть програму TANGLE, що за інформацією про відповідності між листками двох дерев визначить найменшу можливу кількість перетинів відрізків у графічному представленні танглеграма, що може бути досягнута шляхом виконання операцій обміну суміжних піддерев першого дерева. Якщо в одній точці перетинаються більше двох відрізків, то під кількістю перетинів треба розуміти кількість попарних перетинів відрізків. Наприклад, на першому рисунку відповідності 4–8, 5–5 та 6–4 утворюють три перетини.

Вхідні дані. У першому рядку вхідного файлу TANGLE.DAT знаходиться натуральне число N ($1\le N\le19$) — глибина дерев. Другий рядок містить 2^N різних цілих чисел від 1 до 2^N, кожне i-те з яких задає листок другого дерева, який зв'язаний відрізком із i-им листком першого дерева.

Вихідні дані. Єдиний рядок вихідного файлу TANGLE. SOL має містити найменшу можливу кількість перетинів відрізків (відповідностей) у танглеграмі, що може бути досягнена шляхом обмінів піддерев у першому дереві.

Приклад вхідних та вихідних даних

TANGLE.DAT	TANGLE.SOL
3 3 2 1 8 5 4 7 6	6

Спочатку, як видно з рис. 2, є 9 перетинів відрізків. Для досягнення мінімальної кількості перетинів потрібно обміняти у верхньому дереві місцями піддерева (1)↔(2), (5)↔(6), (7)↔(8). Уданому випадку піддерева складаються з одного листка.

Рекомендації щодо розв'язання

Вхідними даними задачі є перестановка B чисел від 1 до 2^N, що задає відповідності між листками дерев вигляду: $1–B_1$, $2–B_2$ . . . $2^N–B_{2^N}$. Тобто початковий порядок листків першого дерева задається ідентичною перестановкою $A=(1, 2, \ldots 2^N)$. Операції обміну сусідніх піддерев першого дерева призведуть до змін перестановки A. Отже, стан дерев у будь-який час, а також початковий і результуючий стан системи можна описати парою перестановок (A, B), що задають відповідності між листками $A_1–B_1$, $A_2–B_2$,…, $A_{2^N}–B_{2^N}$.

Перші 2^{N-1} елементів перестановки A є листками лівого піддерева, відповідно, останні 2^{N-1} елементів перестановки A є листками правого піддерева. Будемо їх позначати A_l та A_r, аналогічно визначивши перестановки B_l та B_r. Підзадачами (A, B) будемо називати задачі (A_l, B_l) та (A_r, B_r).

Кількість перетинів відрізків-відповідностей задачі (A, B) будемо позначати $f(A, B)$.

Назвемо A' оптимальною перестановкою A, якщо $\forall C : f(A', B)\le f(C, B)$, де C — перестановка A. Тобто A' є шуканою перестановкою, що забезпечує мінімальну кількість перетинів відрізків.

Лема 2 (Про локальну оптимальність). *Нехай A' — оптимальна перестановка A. Тоді A'_l та A'_r є оптимальними перестановками для підзадач* (A_l,B_l) та (A_r, B_r).

Доведення. Очевидно від супротивного.

Лема 3 (Про розділення підзадач). $f(A, B)=f(A_l, B_l)+f(A_r, B_r)+g(A_l, B_l, A_r, B_r)$, *де* $g(A_l, B_l, A_r, B_r)$ *— кількість перетинів лише між відповідностями, що починаються у різних піддеревах*.

Доведення. $f(A, B)$ підраховує перетини між відрізками (відповідностями), (а) обидва відрізки виходять з листів лівого піддерева, (б) обидва відрізки виходять з листів правого піддерева, (в) один з відрізків виходить з листа лівого піддерева, а інший — з листа правого піддерева.

Оскільки перераховані класи перетинів, очевидно, не перетинаються, і крім того покривають всі можливі варіанти. Лема доведена.

Отже, з лем отримуємо алгоритм знаходження оптимальної перестановки: для кореня дерева порахувати $g(A_l, B_l, A_r, B_r)$ та $g(A_r, B_r, A_l, B_l)$. Вибравши порядок, що забезпечує мінімальну кількість перетинів, обміняємо, якщо необхідно, піддерева, і для кожного піддерева повторимо алгоритм.

Обчислення $g(A_l, B_l, A_r, B_r)$ можна провести кількома методами. Найпростіший: для кожної пари відрізків, де один відрізок виходить з лівого піддерева, а інший виходить з правого, перевіряємо, чи перетинаються. Очевидно, складність такої процедури $O(m^2)$, де m — кількість листів у лівому та правому піддереві.

Швидше можна реалізувати обчислення g так: відсортуємо кінці відрізків, що починаються у лівому піддереві (i, j), де $i\in A_l$, $j\in B$. Результат — список L, що складається з j таких відрізків. Оберемо деякий відрізок, що починається у правому піддереві (i, j), де $i\in A_r$, $j\in B$. Помітимо, що кількість перетинів цього відрізку із відрізками, що починаються у лівому піддереві, буде кількість чисел у L більших за j. Складність пошуку елементу у відсортованому списку розміру m є $O(\log m)$, а складність сортування $O(m\log m)$.

Неважко також помітити, що обчислення g можна проводити відразу для всіх піддерев глибини i (так легше оцінити загальну складність алгоритму). Отже, для кожного з N рівнів піддерев виконуються алгоритми складності $O(m^2)$ чи $O(m\log m)$. Відповідно загальна складність розв'язків є $O(m^2\log m)$ чи $O(m\log^2 m)$, де m — кількість листів дерева, тобто $m=2^N$, де N — глибина дерева.

ЗАВДАННЯ ДРУГОГО ТУРУ

1. Еволюція

Під час досліджень, присвячених появі життя на планеті Олімпія, ученими було зроблено декілька сенсаційних відкриттів.

1. Усі живі організми планети походять від однієї бактерії Brtozoria Programulis.

2. Еволюція проходила крок за кроком (за припущенням вчених — під час змін клімату на планеті).

3. На кожному кроці еволюції з кожного виду утворювалися рівно два підвиди, а попередній вид зникав.

4. Якщо вважати появу бактерії Brtozoria Programulis першим кроком еволюції, то нині існуючі живі організми знаходяться на N-му кроці.

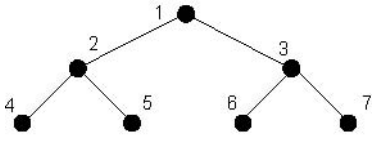

Рис. 4

Щоб не вигадувати назви під час досліджень, учені пронумерували всі види організмів, що будь-коли існували на планеті. Для цього вони намалювали дерево еволюції із коренем Brtozoria Programulis, яка отримала номер 1. Далі нумерували види кожного кроку еволюції зліва направо. Таким чином, безпосередні підвиди Bzooa Programulis отримали номери 2 та 3. Наступними були занумеровані види третього кроку еволюції — підвиди виду 2 отримали номери 4 та 5, а виду 3 — номери 6 та 7, і т. д.

Завдання. Напишіть програму EVO, яка за номерами двох видів обчислить номер виду їх найближчого спільного предка у дереві еволюції.

Вхідні дані. Перший рядок вхідного файлу EVO.DAT містить ціле число N ($1\le N\le 100$) — кількість етапів еволюції, що відбулися на планеті Олімпія до теперішнього часу. Другий та третій рядки

файлу містять по одному натуральному числу, що представляють номери видів, для яких потрібно знайти номер їх найближчого спільного предка.

Вихідні дані. Єдиний рядок вихідного файлу EVO.SOL має містити натуральне число — номер найближчого предка для двох видів.

Приклад вхідних та вихідних даних

EVO.DAT	EVO.SOL
4 15 12	3

EVO.DAT	EVO.SOL
18 233016 233008	14563

Рекомендації щодо розв'язування

Опис алгоритму. Як зрозуміло з рисунку, прямими нащадками i-го виду, є види $2i$ і $2i+1$. Відповідно, безпосереднім предком виду i є вид з номером $[i/2]$.

Звідси випливає основна ідея розв'язку. Нехай треба знайти останнього спільного предка видів a і b. Доки $a \neq b$, ділимо більше з пари чисел на 2 з остачею. Отримане в результаті число є номером виду останнього спільного предка.

Пояснимо, чому такий розв'язок є коректним.

Твердження. Номер виду, який знаходиться на k-му кроці еволюції, записується за допомогою k-бітного числа, старший біт якого 1.

Доведення. Доводиться очевидним чином за принципом індукції по κ.

Очевидним наслідком наведеного вище твердження є факт, що якщо а є предком b, то $a<b$.

Нехай c — останній спільний предок a і b. Тоді у кожний момент виконання алгоритму є 3 варіанти:

- $a=b$. Тоді очевидно, алгоритм зупиниться, вивівши як відповідь a. Очевидно, що $c=a=b$;
- a і b знаходяться на різних кроках еволюції. Нехай для, визначеності, b знаходиться на пізнішому кроці. Тоді з доведеного вище твердження випливає, що $a<b$. Маємо $c \leq a<b$, тому $c \leq [b/2]$ і ми можемо перейти до розгляду пари вершин a і $[b/2]$;
- a і b знаходяться на одному кроці еволюції, але $a \neq b$. Тоді зрозуміло, що $c<a$ і $c<b$, звідки маємо $c \leq [b/2]$ і ми можемо перейти до розгляду пари вершин a і $[b/2]$.

Таким чином, алгоритм є коректним.

Особливості реалізації. Основна особливість реалізації полягає в тому, що номери видів можуть досягати 2^N-1. А при $N=100$ це більше, ніж здатні представити 32-бітні і 64-бітні типи. Тому для проходження всіх тестів, необхідно реалізовувати дії з довгими числами: а саме порівняння і цілочисельне ділення на 2. Обидві ці операції легко реалізуються з часовою складністю $O(N)$. Оскільки на кожному кроці алгоритму одне з пари чисел a і b зменшується вдвічі, алгоритм гарантовано завершиться за $2N-2$ кроків. Таким чином, загальна часова складність алгоритму складає $O(N^2)$.

2. Торговець

Торговець володіє трьома видами товарів: алмази, яблука та шовк. Для кожного товару відомо вартість у золотих монетах за одиницю ваги та його кількість у торговця.

У країні, де живе торговець, є N міст, які пронумеровані від 1 до N. Рідне місто торговця має номер 1, а столиця — номер N. Щоб дістатися столиці, де торговець може продати товар, йому потрібно проїхати певним маршрутом через інші міста. Між деякими парами міст існують дороги, проїзд по яких коштує певної кількості золотих. У кожному місті стягується податок за провезення кожного з видів товару, заданий у відсотках від вартості провезеного через місто товару. Відомо, що виїхавши з будь-якого міста, торговець не може до нього повернутися. Будь-які два міста з'єднані не більше ніж однією дорогою.

Задача торговця отримати найбільший прибуток — різницю отриманих у столиці коштів за проданий товар та витрат за подорож до столиці. Він не зобов'язаний брати із собою весь свій товар. Торговець завжди має достатньо золотих для виплати податків, та не може розрахуватися товаром, який він везе до столиці. Усі дороги ведуть лише в одному напрямку.

Завдання. Напишіть програму SALESMAN, що за інформацією про кількість одиниць ваги різних видів товару у торговця, ціни на ці товари у столиці, податки у містах, дороги між містами та вартість проїзду по цих дорогах встановить максимальний прибуток, що може отримати торговець від реалізації товару.

Вхідні дані. Перший рядок вхідного файлу SALESMAN.DAT містить два цілих числа N ($2 \leq N \leq 500$) та M ($M \geq 1$) — кількість міст та доріг між ними. Другий рядок містить три цілих невід'ємних числа, що відповідають кількостям одиниць ваги алмазів, яблук та шовку, що належать торговцю. Третій рядок містить три цілих невід'ємних числа — вартість одиниці ваги алмазів, яблук та шовку відповідно. Наступні рядки з 4-го по $N+1$ містять по три цілих числа від 0 до 100 включно, що відповідають відсоткам від вартості алмазів, яблук та шовку, що стягується відповідно у містах від 2 до $N-1$ як податок. У списку міст не враховані рідне місто торговця 1 та столиця N, як такі, що не стягують податок. Наступні M рядків містять по три цілих невід'ємних числа, перші два з яких від 1 до N задають пару міст, між якими прокладено дорогу, а третє — вартість проїзду цією дорогою. Дороги ведуть в напрямку від міста, яке вказано першим, до того, яке вказано другим. Кількості одиниць ваги кожного з видів товару у торговця, вартості товарів та ціни проїзду по дорогах не перевищують 100.

Вихідні дані. Єдиний рядок вихідного файлу SALESMAN. SOL має містити одне число — точне значення знайденого максимального прибутку від поїздки до столиці. Відповідь завжди повинна містити рівно два знаки після крапки. У випадках, коли торговець не може отримати прибутку чи дістатися столиці існуючими дорогами, потрібно вивести 0.00

Приклади вхідних та вихідних даних

SALESMAN.DAT
4 4 10 5 20 100 5 12 15 40 25 90 20 10 1 2 5 1 3 10 3 4 10 2 4 15

SALESMAN.SOL
1025.00

Рекомендації щодо розв'язування

Може скластися враження, що завжди вигідно брати всі товари, але це не так.

Твердження. *Який би торговець не вибрав шлях, кожен із видів товару потрібно брати або повністю, або не брати взагалі (для отримання максимального прибутку).*

Доведення. Нехай обрано шлях, що проходить вершинами:

$$1, v_1, v_2, \ldots v_{k-1}, v_k, N.$$

Позначимо через C сумарну вартість проїзду ребрами від v_1 до v_2, від v_2 до v_3,..., від v_{k-1} до v_k; через c_i — кількість одиниць ваги i-го виду товару в торгівця; через p_i — вартість однієї одиниці ваги i-го товару у столиці; через z_i — суму по всіх містах $v_2, \ldots, v_{k-1}$ податків за i-й вид товару, виражену у відсотках. Тоді торговець одержить прибуток $[((100-z_1)/100)c_1p_1+((100-z_2)/100)c_2p_2+((100-z_3)/100)c_3p_3-C]$. Звідси видно, що якщо $z_i<100$, то прибуток за i-й вид товару буде додатнім, якщо $z_i>100$, то від'ємним, а якщо $z_i=100$, то нульовим. Якщо прибуток по товару від'ємний, то брати його із собою невигідно незалежно від кількості, якщо нульовий, то значення не має (будемо вважати, що товар не беремо), якщо додатній, то чим більше взяти товару, тим більшим буде добуток і прибуток за товар. Отже, твердження доведено.

Загальна ідея розв'язку. Переберемо всі варіанти беремо/не беремо по кожному з видів товару — всього $2^3=8$ варіантів. Коли обрано, які товари буде везти торговець, то: а) однозначно визначено, скільки коштів він одержить за продаж товару у столиці; б) однозначно визначено, скільки податків у золотих заплатить торговець під час відвідування кожного міста.

Отже, задача зводиться до мінімізації витрат на подорож (оскільки загальний прибуток = гроші за продаж товарів у столиці – витрати на подорож; і перший доданок фіксований). У термінах графу необхідно знайти мінімальний маршрут із вершини 1 у вершину N, якщо відомі вартості ребер та вартості вершин. Розглянемо можливі реалізації цієї підзадачі.

Алгоритм Дейкстри. Розглянемо ціну ребра як суму його безпосередньої вартості та вартості вершини, у яку воно входить: якщо буде відбуватися прохід по ребру $(a; b)$, то кандидатом на мінімальну вартість маршруту до вершини b буде $f[a]+c(a; b)+p[b]$, де через $f[x]$ позначено мінімальну вартість маршруту до вершини x, $c(x; y)$ — вартість ребра з вершин x до вершини y; $p[x]$ — вартість вершини x. У кожному маршруті кожне ребро і кожна вершина будуть враховані у вартість рівно один раз. Вибір на кожному кроці сірої вершини з найменшою вартістю буде коректним: так само, як у чистому алгоритмі Дейкстри, легко показати від супротивного, що для такої вершини знайдено маршрут найменшої вартості. Оцінка складності підзадачі $O(N^2)$. Сумарна оцінка для задачі $O(8N^2)$.

Топологічне сортування з динамікою. Орієнтуємо ребра у протилежному напрямку, та зробимо топологічне сортування. Вершина N буде найстаршою. Позначимо через $t[i]$ i-ту в топологічному порядку вершину ($t[N]=N$); через $f[i]$ максимальний прибуток, який можна одержати, мандруючи із вершини 1 до вершини i, та вважаючи, що товари продаються у i-й вершині. Початкові умови:

$f[1]$ = вартості взятих із собою товарів у столиці;

$f[i]=0$ для $i=2...N$.

Обчислення f[i]

Пройдемо циклом по i від 1 до N, на i-му кроці обчислюючи $f[t[i]]$. $f[t[i]]=\max\{f[adj[t[i], j]]-c[t[i], adj[t[i], j]]-p[t[i]] \mid j=1..num[i]\}$, якщо $t[i]\neq 1$. Тут через $adj[x, y]$ позначено суміжну вершину до x, що стоїть у списку суміжних вершин під номером y; через $c[x, y]$ — вартість ребра із вершини x у вершину y; $p[i]$— вартість вершини i; $num[i]$ — кількість суміжних з i-ю вершин. Формат формули близький до її запису у коді програми.

Оскільки вершина топологічно старша за всіх своїх потомків, то при обчисленні функції для вершини функції для потомків уже будуть обчислені, так що динаміка коректна.

Оцінка складності $O(N+M)$, що у найгіршому випадку дає $O(N^2)$. Сумарна оцінка для задачі $O(8(N+M))$.

Можливі евристики та неефективні розв'язки

1. Повний перебір усіх шляхів із визначенням відсотків податку по кожному з видів товару та вартості проїзду по дорогах. Оцінка складності $O(N!)$, хоча при розріджених графах та відтинах завідомо неоптимальних шляхів швидкість буде набагато більшою порівняно із вказаною.

2. Брати всі товари (евристика).

3. Для пошуку маршруту найменшої вартості використовувати алгоритм Флойда-Уоршала. Оцінка його складності $O(N^3)$. Сумарна оцінка на задачу $O(8N^3)$.

4. Проводиться динаміка без топологічного сортування (*евристика*).

3. Кола

На площині задано N різних кіл. Два кола перетинаються, якщо мають хоча б одну спільну точку.

Завдання. Напишіть програму CIRCLES, що за координатами центрів кіл та їх радіусами знайде пару кіл, що перетинаються.

Вхідні дані. У першому рядку вхідного файлу CIRCLES. DAT міститься ціле число N ($1\le N\le 10\,000$). У кожному з наступних N рядків міститься три натуральних числа X, Y, R менших за 10 000, що задають координати центру кола (X, Y) та його радіус R.

Вихідні дані. Єдиний рядок вихідного файлу CIRCLES. SOL має містити пару номерів кіл, що перетинаються, або єдине число 0, якщо жодні два кола не перетинаються. Кола нумеруються відповідно до порядку у вхідному файлі, починаючи з 1 до N. Якщо є декілька пар кіл, що перетинаються, виведіть будь-

яку з них. Елементи пари можуть бути виведені в довільному порядку.

Приклади вхідних та вихідних даних

CIRCLES.DAT	CIRCLES.SOL
5 5 10 4 6 20 3 10 15 3 12 8 2 13 13 1	5 3

Рекомендації щодо розв'язування

Загальна ідея розв'язку. Задача легко розв'язується перебором всіх пар кіл і перевіркою, чи перетинаються кола кожної пари. Нескладно переконатися, що кількість таких пар n^2, де n — кількість кіл. Очевидно, що перебирати всі пари є неоптимально. Наприклад, кола, що розташовані по різні кінці площини, і між ними знаходиться велика кількість інших кіл. Як розрізнити такі пари кіл: такі, що знаходяться «далеко»одне від одного та такі, що знаходяться поруч, і між ними немає інших кіл?

Проведемо деяку пряму, паралельну Ox. Вона перетне деякі кола. Помітимо точки перетину прямої та кіл мітками. Очевидно, що підозрілими є лише такі пари кіл, що мітки яких є сусідами на такій прямій: між ними немає іншої мітки.

Припустимо, що кола не перетинаються, а пряма «замітає» площину, паралельно до Ox. Тоді мітки, що ми розставляємо на прямій будуть «рухатися», «плавати» по прямій. Але можна довести (Лема 1), що порядок міток на цій прямій не буде змінюватися: можливими змінами є лише поява нових міток (по 2 на кожне коло) та зникнення старих. Моментами таких подій будуть точки дотику замітаючої прямої до кола.

Отже, нові пари сусідніх кіл (по відношенню до замітаючої прямої) будуть з'являтися лише в моменти дотику прямої до кіл. На кожну подію дотику певного кола вперше утвориться дві нових пари сусідніх кіл, а на подію дотику до певного кола вдруге, утвориться лише одна пара сусідніх кіл.

Теоретичне обгрунтування, чому ми можемо зупинятися лише в точках дотику замітаючої прямої до кіл, та чому необхідно перевіряти лише пари сусідів, які утворюються при доданні/виключенні наведено у наступній частині.

Зауважимо, що перевірка перетину кіл може виконуватися з використанням цілочисельної арифметики, а дробові числа використовуються лише для сортування міток на замітаючій прямій. Точність обчислення в цьому випадку, з огляду на наведені обмеження задачі, має бути не менше за 10^{-5}, що забезпечується стандартними типами запропонованих компіляторів.

Обгрунтування

Означення. *Структурою перетину P* в момент часу t назвемо впорядковану множину точок, таку що: для кожного кола C_i, яке перетинається замітючою прямою в момент часу t, у множині P знаходиться рівно дві, можливо, співпадаючі точки $(l_i(t), r_i(t))$, що є точками перетину замітаючої прямої та кола. Нехай на замітаючий прямій визначений певний напрям. На структурі перетину визначимо природний порядок: $\forall a, b \in P\ a \prec_P b$, якщо a слідує раніше за b на замітаючій прямій.

Лема 4 (Про кола, що перетинаються). *Нехай для довільних моментів 'часу t_1, t_2, кола C_i, C_j перетинаються замітаючою прямою в точках $l_i(t)$, $r_i(t)$ та $l_j(t), r_j(t)$ відповідно. Якщо $r_j(t_1) \prec P_{t_1} l_i(t_1) \wedge l_i(t_2) \prec P_{t_2} r_j(t_2)$ тоді кола C_i та C_j перетинаються.*

Доведення. За теоремою про середні значення функцій отримуємо, що $\exists t \in [t_1,t_2] : r_j(t) = l_i(t)$. Отже, кола мають спільну точку. Що і треба було довести.

Важливим наслідком леми 4 є факт, що за відсутності кіл, що перетинаються, порядок точок перетину у структурі перетинів не буде змінюватися. Цей факт необхідний для підтримання структури перетинів, при переміщенні замітаючої прямої. Також наслідком з леми 4 є властивість, що для двох кіл, що перетинаються існує такий момент часу t, коли ці кола є сусідами в структурі перетину. Отже, для того щоб впевнитися, що жодна пара кіл не перетинається, нам необхідно перевіряти всі сусідні пари в структурі перетину.

Схема алгоритму. Опишемо схему алгоритму для визначення наявності кіл, що перетинаються серед заданих. Нехай замітаюча пряма рухається площиною. Протягом свого руху пряма перетинає деякі кола. Для деякого кола C_i моментом часу, коли пряма вперше дотикається до нього є S_i, і відповідно останнім дотиком замітаючої прямої є момент часу f_i. З визначення структури перетину, ця структура в точці S_i буде поповнюватися двома точками $(l_i(t), r_i(t))$. У разі внесення нових точок до структури перетину, перевіряємо сусідні з ними точки інших кіл на перетин між цими парами кіл. Аналогічно, в момент часу f_i, зі структури зникнуть дві точки $(l_i(t), r_i(t))$. У структурі перетину утвориться нова пара сусідів, які треба також перевірити на перетин.

Оцінимо складність алгоритму. Першим кроком алгоритму сортується всі S_i, f_i — моменти зупинки замітаючої прямої, тобто O($n\log n$), де n — кількість кіл. Кожна зупинка може збільшити кількість елементів у структурі перетину на 2 елементи, але структура сусідства не буде більшою за $2n$ елементів. Якщо операції пошуку та вставки в структурі перетину реалізовані за O ($\log m$), де m — кількість елементів у структурі, то загальна складність алгоритму O($n\log n$).

Реалізувати структуру перетину можна збалансованими деревами, які забезпечать необхідну складність операцій вставки та пошуку.

Предикат перетину двох кіл. Коло — геометричне місце точок, рівновіддалених від центру. Два кола перетинаються, якщо мають хоча б одну спільну точку. Нагадаємо, що два кола можуть знаходитися в трьох принципово різних положеннях: перетинатися, бути вкладеними одне в одне та не перетинатися. Перевірити взаєморозташування двох кіл можливо за допомогою (а) розв'язати систему рівнянь, що складається з рівнянь обох кіл; (б) використовуючи значення радіусів кіл, та відстані між їх центрами.