Ідеї розв'язку задач першого туру

28 березня 2011 р.

> Якщо програма працює не на 100%, за неї будуть якісь бали?
>
> з питань учасників

1 "Розбір задачі Plusxor" (Даніїл Нейтер).

> Там ограничение 2 в 64 -1 для а и б? просто для стандартного типа надо 2 в 63 -1!
>
> з питань учасників

Спочатку розв'яжемо задачу в припущенні, що шукана пара чисел X, Y існує. Будемо позначати двійкові цифри в записі чисел наступним чином: $X = (x_{63}x_{62}\dots x_1x_0)_2$, $Y = (y_{63}y_{62}\dots y_1y_0)_2$, $A = (a_{63}a_{62}\dots a_1a_0)_2$, $B = (b_{63}b_{62}\dots b_1b_0)_2$.

З означення побітового "виключного або" випливає, що $b_i = x_i + y_i (mod\ 2)$ (1). З іншого боку $a_i = x_i + y_i + c_i - 1 (mod 2)$ (2), де c_i позначає перенос з i-го двійкового розряду, який обчислюється за такою формулою: $c_i = [\frac{x_i+y_i+c_i-1}{2}]$, де через [] позначається операція взяття цілої частини. Для зручності позначимо $c_{-1} = 0$.

З (1) і (2) випливає $a_i + b_i (mod\ 2) = [(x_i + y_i)\ mod\ 2 + (x_i + y_i + c_{i-1})\ mod\ 2]\ mod\ 2 = (2x_i + 2y_i + c_{i-1})\ mod\ 2 = [(2x_i)\ mod\ 2 + (2y_i)\ mod\ 2 + c_{i-1}\ mod\ 2]\ mod\ 2 = 0 + 0 + c_{i-1}$. Отже, $c_i = a_{i+1} + b_{i+1} (mod\ 2)$. Звідси маємо $C = (A\ xor\ B)\ shr\ 1$, де через shr позначений бітовий зсув вправо.

Враховуючи, що $b_i = 0$ тоді і тільки тоді, коли $x_i = y_i$, а $c_i = 0$ тільки тоді, коли $x_i + y_i < 2$, складемо таблицю істинності для визначення x_i та y_i через b_i та c_i:

b_i	c_i	x_i	y_i	примітки
0	0	0	0	
1	0	0	1	необхідно, що $c_{i-1} = 0$
0	1	1	1	
1	1	0	1	необхідно, що $c_{i-1} = 1$

У випадку, коли $b_i = 1$, вибираємо саме $x_i = 0$, $y_i = 1$ для забезпечення умови мінімальності X.

Виходячи з таблиці, можна записати бітові вирази для знаходження X та Y через нормальну диз'юнктну форму:

$$X = C\ and\ not\ B$$

$$Y = B\ or\ C$$

де and, or, not позначають побітове "І", побітове "АБО" та побітове "НІ" відповідно.

Отже, якщо розв'язок задачі існує, його можна знайти за наведеними вище формулами. Для перевірки існування розв'язку знайдемо X та Y за цими формулами та первіримо виконання умов $A = X + Y$, $B = X\ xor\ Y$. Якщо вони виконуються – розв'язок знайдено, якщо ні – розв'язку для заданої пари чисел A, B не існує.

Отриманий алгоритм дозволяє розв'язати задачу за константний час, тобто його часова складність $O(1)$.

1.1 Інші підходи.

1.1.1 Альтернативний підхід.

Можна помітити, що біти, встановлені в X, встановлені і в Y. Тоді $Y = X + B$. Враховуючи, що $X + Y = X + X + B = A$, отримуємо наступні формули для обчислення X та Y:

$$X = (A - B)/2$$

$$Y = X + B$$

1.1.2 Наївний підхід.

Перебирати всі числа X за зростанням від 0 до $\frac{A}{2}$. Якщо $X\ xor\ (A - X) = B$, то пара $X,\ A - X$ є розв'язком задачі.

Часова складність алгоритму $O(A)$. Такий розв'язок набирає 50 балів.

1.2 Особливості реалізації.

Обмеження задачі потребує використання 64-бітних беззнакових типів даних (у C/C++ це unsigned long long, в Pascal - uint64).

2 "Подарунок" (Роман Єдемський).

> А разбойники могут потребовать 0 золотых и 0 серебряных монет за дорогу?
>
> з питань учасників

2.1 Формалізація умови задачі.

Перекладемо умову задачі мовою теорії графів. Дано граф який складається з N вершин та M ребер в якому між двома вершинами може існувати більше одного ребра. Для кожного ребра відомі деякі параметри $Silver_i$ та $Gold_i$ (в умові - мінімальні обмеження на кількість срібних та золотих монет).

Означення 2.1. Нехай задано деяку пару невід'ємних чисел (A, B). Граф який складається з вихідних вершин, а також набору ребер для яких виконується $Silver_i \leq A \wedge Gold_i \leq B$ будемо позначати $R(A, B)$.

Означення 2.2. Розв'язком задачі будемо називати пару чисел (A, B) для якої $R(A, B)$ – зв'язний.

Означення 2.3. Будемо вважати розв'язок (A, B) оптимальним, якщо величина $Cost(A, B) = A \cdot S + B \cdot G$ є мінімальною можливою.

Означення 2.4. Позначимо $Silver = Silver_i | i \in (1, M)$.

Означення 2.5. Позначимо $Gold = Gold_i | i \in (1, M)$.

2.2 Очевидний алгоритм.

Нехай пара чисел (A, B) є розв'язком задачі. Якщо $A \notin Silver$, то ми можемо зменшити A не змінивши зв'язність графу. Аналогічно можна зробити, якщо $B \notin Gold$. Отже, оптимальний розв'язок слід шукати лише серед пар чисел (A, B), у яких $A \in Silver \wedge B \in Gold$.

Звідси можна отримати алгоритм із часовою складністю $O((N+M) \cdot M^2)$: переберемо усі такі пари (A, B) і для кожної перевіримо, чи є ця пара розв'язком, тобто чи є граф $R(A, B)$ зв'язним. Якщо є – порівняємо $Cost(A, B)$ з поточною відповіддю і змінимо її при не обхідності.

2.3 Покращення очевидного алгоритму за допомогою бінарного пошуку.

Нехай пара чисел (A, B) є розв'язком задачі. Тоді пара чисел $(A, B + 1)$ також буде розв'язком задачі, оскільки ребра у граф $R(A, B)$ можуть тільки додатися.

Аналогічно, якщо пара (A, B) не є розв'язком задачі, то і пара $(A, B - 1)$ також не є розв'язком задачі.

Розглянемо такий алгоритм:

- Переберемо усі можливі значення A з множини $Silver$.
- Для кожного фіксованого A зробимо бінарний пошук по B.
- Якщо для даного B граф зв'язний, то і для всіх більших B він також буде зв'язним, отже ми можемо зменшити праву границю пошуку.
- Якщо для даного B граф незв'язний, то і для всіх менших B він також буде незв'язним, отже ми можемо збільшити ліву границю пошуку.

Ми отримали алгоритм з часовою складністю $O((N + M) \cdot M + \log M)$.

2.4 Ефективний алгоритм.

Зафіксуємо якесь $A \in Silver$. Розглянемо усі можливі ребра i, для яких $Silver_i \leq A$. Побудуємо на усіх таких ребрах граф $H(A)$ з вагами ребер $Gold_i$. Нам треба знайти таке найменше B, для якого прибравши з цього графа усі ребра з вагами, більшими за B, граф все ще залишиться зв'язним. Якщо граф зв'язний, то в нього існує каркасне дерево, причому усі його ребра мають вагу не більшу за B.

Не важко переконатись, що зворотнє твердження також вірне: якщо у графі $H(A)$ існує каркасне дерево, усі ребра якого не перевищують якогось B', то $B \leq B'$. Отже, нам треба знайти каркасне дерево, у якого максимальне ребро є найменшим можливим.

Розглянемо такий алгоритм:

- Відсортуємо ребра за зростанням $Silver_i$
- Будемо йти по цьому масиву зліва направо та на i-му кроці додавати i-те ребро до графа. Вага ребра рівна $Gold_i$.
- Якщо при цьому утворився цикл, то знайдемо максимальне ребро у ньому і видалимо його.
- Після додавання кожного ребра перевіримо, чи зв'язний граф і якщо так – знайдемо у ньому максимальне ребро. Порівняємо вагу цього ребра з відповіддю і змінимо її, якщо треба.

Доведення коректності алгоритму.

Перед початком роботи алгоритму ребер в графі немає і у ньому N компонент зв'язності. При додаванні чергового ребра у нас або зливаються 2 компоненти між собою, або ребро додається у якусь одну компоненту і з неї ж видаляється одне ребро. Неважко переконатись, що після кожного кроку алгоритму граф буде являти собою ліс.

Твердження 2.1. Якщо для якогось ребра (a, b) існує шлях з вершини a до вершини b, який не містить це ребро, причому у цьому шляху усі ребра мають не більшу вагу, ніж ребро (a, b), то існує мінімальне каркасне дерево, яке не містить цього ребра.

Доведення. Розглянемо будь-яке мінімальне каркасне дерево. Якщо йому не належить ребро (a, b) – твердження доведено. Припустимо, що воно містить це ребро. Після його видалення граф розпадеться на 2 дерева. Оскільки з a у b існує шлях, який оминає це ребро, то існує ребро крім (a, b), яке сполучає ці 2 компоненти, причому це ребро має вагу не більшу, ніж у (a, b). Отже, при додаванні цього ребра вага мінімального каркасного дерева не збільшується, отже нове дерево також буде мінімальним каркасним. □

Твердження 2.2. Кожна компонента графа є мінімальним каркасним деревом для графа з вершин тієї компоненти.

Доведення. Перед початком роботи алгоритму кожна компонента складається з однієї вершини, і, очевидно, є мінімальним каркасним деревом графа з однією вершино. Нехай після кроку K твердження 2 справедливе. Доведемо, що після додавання одного ребра воно також залишиться справедливим. Можливі 2 випадки:

1. Ребро сполучає 2 різні компоненти зв'язності. Оскільки до цього вони не були сполучені, то це ребро обов'язково належить мінімальному каркасному дереву з цих двох компонент. Видаливши його ми отримуємо 2 компоненти, у яких вже побудовані мінімальні каркасні дерева. Очевидно, що при цьому отримане дерево також буде мінімальним каркасним.
2. Ребро додається до одної компоненти. Розглянемо граф G, який складається з вершин і ребер цієї компоненти до додавання нового ребра. Усі ребра, які не належать цьому графу можна викинути з розгляду, оскільки для кожного з них існує мінімальне каркасне дерево, якому вони не належать, що буде вірним і після додавання ребра e (за твердженням 1 для кожного з цих ребер існує шлях по дереву з одного кінця до іншого). При додаванні нового ребра в графі утвориться цикл, і щоб мінімізувати вагу дерева, потрібно з цього циклу видалити максимальне ребро. Неважко переконатись, що отримане дерево є мінімальним каркасним.

□

Отримано алгоритм, часова складність якого $O(N \cdot M + M \cdot \log M)$.

3 "Розбір задачі Турист" Ілля Порубльов).

Швидкість V позначає
максимальну кількість клітин,
що може пройти турист за
одиницю часу?

з питань учасників

З двох відповідей, які просять знайти, легше шукати другу (коли турист може прибути до моменту 0 в точку з будь-якою координатою). Першу відповідь можна знайти повторним застосуванням того ж алгоритму, відкинувши зі вхідних даних події, на які неможливо встигнути пішки з $(x = 0, t = 0)$. (Отже, алгоритм доцільно організувати як підпрограму.) Зосередимося на пошуку другої відповіді.

3.1 Проста динаміка

Відсортуємо події за часом. Поставимо серію підзадач «Яку максимальну кількість подій $R(i)$ можна відвідати, так, щоб останньою була подія i?» (де i — номер події після сортування за неспаданням t). Легко отримати рівняння ДП (динамічного програмування)

$$R(i) = \max_{k\,:\,\left\{\begin{array}{c} 1 \leqslant k < i \\ |x_i - x_k| \leqslant (t_i - t_k)\cdot V \end{array}\right.} \{R(k)\} + 1.$$

Перебираємо у зовнішньому циклі значення i, для кожного i перебираємо всі менші значення k, пропускаючи ті, з яких неможливо встигнути на подію i, а серед тих, з яких можна встигнути, шукаємо максимальне $R(k)$. Знайшовши максимум, додаємо 1 — відвідання самої події i. Для 1-ої події, а також для всіх тих, на які не можна встигнути з жодної попередньої, $R(i) = 1$. Остаточна відповідь — $\max R(i)$ (по всім i).

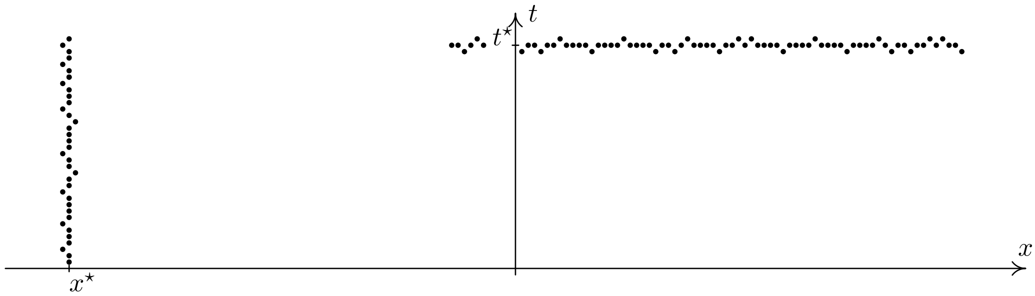
Цей алгоритм має складність $O(N^2)$, що при $N \sim 100\,000$ надто багато.

3.1.1 Евристичні оптимізації даного ДП

У щойно розглянутому ДП перебираються всі $k < i$. Це можна оптимізувати, зберігаючи результати розв'язаних підзадач у масиві пар «номер події, значення R для цієї події», впорядкованому за R. Ідучи від більших R до менших, можна гарантувати, що перша ж знайдена подія (така, що з неї можна встигнути на дану) дасть шуканий максимум.

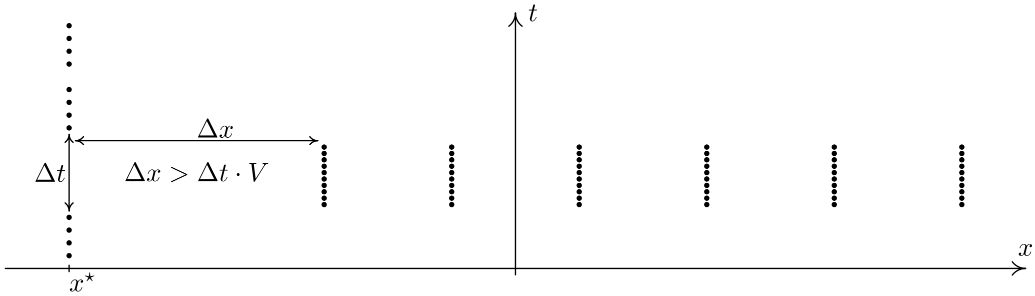
Застосування цієї ідеї для більшості вхідних даних дає оптимізацію і тому збільшує бали за реалізацію вищенаведеного ДП. Але як розвинути її у повноцінний (завжди правильний і завжди ефективний) розв'язок — незрозуміло.

«Чесний» варіант (пам'ятати розв'язки усіх раніше розглянутих підзадач) працюватиме надто довго на вхідних даних, подібних до таких:



(багато подій мають координати $x \approx x^\star$ у різні моменти часу, і багато подій відбуваються у час $t \approx t^\star$ у різних точках; $t^\star$ пізніший за моменти більшості подій у околі $x^\star$; швидкість V мала, і на більшість подій часу $t \approx t^\star$ або не можна встигнути взагалі ні з якої ранішої події, або можна встигнути лише з деяких подій для яких R зовсім мале). Тоді при аналізі більшості подій вигляду $t \approx t^\star$ доведеться перебирати практично всі події у околі координати $x^\star$, нічого не знаходячи. Тобто, отримуємо ту саму складність $O(N^2)$.

«Нечесний» варіант — підтримувати перелік лише кращих результатів розв'язаних підзадач: наприклад, 500 кращих на даний момент результатів зберігаються, а ті, які не ввійшли до п'ятисот кращих — забуваються. Ця евристика прискорює програму від $O(N^2)$ до $O(500N) \approx O(N)$. Але вона не є правильним алгоритмом. Нехай зберігають не 500 кращих оцінок, а лише 5; розглянемо вхідні дані:

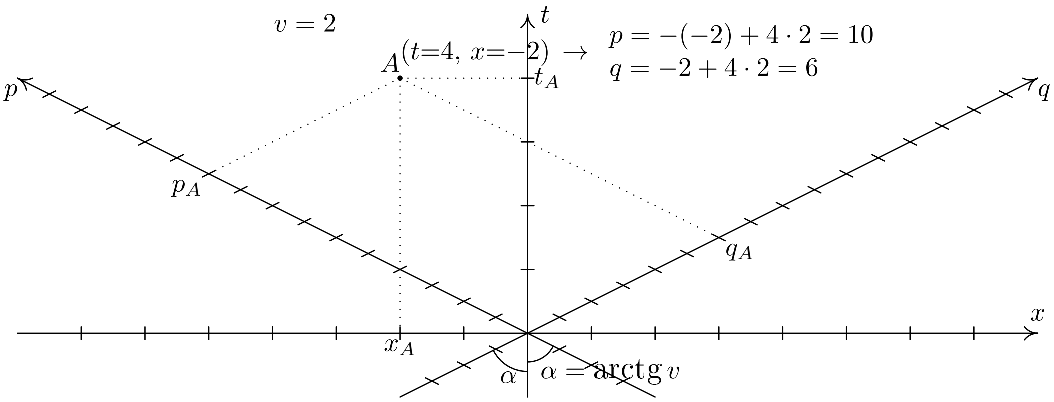


Оскільки є 6 вертикальних «стовпчиків» з 10 подій кожен, інформація про найраніші 4 події при $x \approx x^\star$ буде забута. Але потім (при більших t) старі 4 плюс нові 8 виявляються більшими за 10. А дізнатись про це неможливо, бо інформація про старі 4 вже забута.

Зрозуміло, що коли запам'ятовувати 500 кращих результатів, конкретно цей приклад опрацьовується правильно. Але можна зробити вхідні дані з кількома тисячами аналогічних «стовпчиків». А збільшувати кількість кращих задач (які пам'ятаємо) суттєво понад 500 ризиковано, бо тоді втратиться оптимізація за часом роботи.

Ми детально розглянули дані евристичні модифікації ДП, бо, з одного боку, вони не є правильним ефективним алгоритмом; з іншого — ***якби*** тести були випадкові (незалежно рівномірно розподілені), реалізація будь-якої з цих евристик набирала *б* 80–95% балів.

3.2 Ефективний розв'язок (пошук монотонної підпослідовності)



Введемо косокутню систему координат (p, q) так, щоб подія у момент t_A в точці з координатою x_A мала координати $p_A = -x_A + t_A \cdot V$, $q_A = x_A + t_A \cdot V$. Згадана в тексті задачі умова $|x_k\breve{\ }x_j| \leqslant |t_k\breve{\ }t_j| \cdot V$ при $t_k > t_j$ набуває вигляду

$$\left\{ \begin{array}{lcl} x_k - x_j & \leqslant & (t_k - t_j) \cdot V, \\ x_j - x_k & \leqslant & (t_k - t_j) \cdot V \end{array} \right. \Leftrightarrow \left\{ \begin{array}{rcr} -x_j + t_j \cdot V & \leqslant & -x_k + t_k \cdot V, \\ x_j + t_j \cdot V & \leqslant & x_k + t_k \cdot V \end{array} \right. \Leftrightarrow \left\{ \begin{array}{l} p_j \leqslant p_k, \\ q_j \leqslant q_k, \end{array} \right.$$

тобто $(p_j \leqslant p_k)$ `and` $(q_j \leqslant q_k)$. Отже, після однієї події можна встигнути на іншу тоді й тільки тоді, коли обидві координати p та q наступної події не менші, ніж у попередньої.

*Відсортуємо всі події, як пари (p, q), лексикографічно (пара (p_1, q_1) має йти раніше по порядку, ніж пара (p_2, q_2), тоді й тільки тоді, коли $(p_1 < p_2)$ **or** $((p_1 = p_2)$ **and** $(q_1 < q_2)))$.* Легко бачити, що умова «$(p_A \leqslant p_B)$ `and` $(q_A \leqslant q_B)$» рівносильна одночасному виконанню двох умов «подію A при сортуванні розмістили раніше за подію B» та «$q_A \leqslant q_B$».

Отже, послідовність подій, таких, що після A_1 можна встигнути на A_2, після A_2 можна встигнути на A_3, $\ldots$, після A_{k-1} можна встигнути на A_k, характеризується тим, що усі події A_1, A_2, $\ldots$, A_k розміщені після лексикографічного сортування саме в такому порядку між собою (не обов'язково підряд, між ними можуть бути якісь інші події), і $q_{A_1} \leqslant q_{A_2} \leqslant \ldots \leqslant q_{A_k}$. Тобто, *після вищезгаданого лексикографічного сортування достатньо знайти довжину найдовшої монотонно неспадної (за q) підпослідовності.*

Алгоритм, що дозволяє знайти найдовшу монотонно неспадну підпослідовність за час $O(n \log n)$, описаний у багатьох джерелах, зокрема — А. Шень, «Программирование: теоремы и задачи», задача 1.3.4; `ru.wikipedia.org`, стаття «Задача поиска наибольшей увеличивающейся подпоследовательности»; Порублёв—Ставровский, «Алгоритмы и программы. Решение олимпиадных задач», розд. 13.2.2. Додамо лише, що при використанні мови C++ можна не писати свій бінарний пошук, а застосувати бібліотечну функцію `upper_bound` (мова йде про функцію, що застосовується, наприклад, до відсортованого `vector`-а, а не про метод контейнера `set` або подібного).

Складність даного розв'язку — $O(n \log n)$, причому є два етапи, кожен з яких має таку складність: спочатку сортування згідно $(p_1 < p_2)$ `or` $((p_1 = p_2)$ `and` $(q_1 < q_2))$, потім пошук найдовшої монотонно неспадної підпослідовності.

На думку автора задачі, геометрична її трактовка провокує ідею розв'язувати через замітання (воно ж «вимітання», «метод сканирующей прямой», «sweeping»), але навряд чи цей метод може дати оптимальний розв'язок. Хоча б тому, що і при горизонтальному руху вертикальної прямої, і при вертикальному руху горизонтальної прямої незрозуміло як добитися, щоб кількість точок подій ні при яких вхідних даних не сягала порядку N^2. Замітання після введення косокутньої системи координат може дати ефективний розв'язок, але він складніший за наведений.