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

Бондаренко В.В.

ЗАВДАННЯ ПЕРШОГО ТУРУ

1. Триоміно

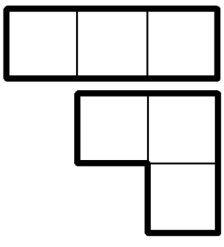

Рис. 1

Фігуркою триоміно називають зв'язну фігуру, що складається з трьох квадратів. Існує лише два суттєво різних типи таких фігурок — вони зображені на рис. 1. Усі інші відрізняються лише поворотами. Прямокутне поле розмірами M рядків і N стовпчиків вважається заповненим фігурками триоміно, коли:

1. Кожна фігурка знаходиться у межах поля і накриває рівно три клітинки.

2. Фігурки не накладаються.

3. Порожніх (не покритих фігуркою) клітинок не більше ніж дві.

Заповнене фігурками поле можна подати у вигляді прямокутної таблиці. Значення 0 позначають не заповнену клітинку. Однакові натуральні значення позначають, що клітинки належать одній фігурці, різні — різним.

Завдання. Напишіть програму **triomino**, яка, проаналізувавши вміст кількох числових двовимірних таблиць, визначить для кожної з них, чи задає вона правильно заповнене фігурками триоміно прямокутне поле.

Вхідні дані. Перший рядок вхідного файлу **triomino.dat** містить єдине ціле число T ($2 \le T \le 10$) — кількість прямокутних таблиць. Далі йдуть T блоків такої структури. Перший рядок блоку містить два цілих числа M і N ($1 \le M \le 200$, $1 \le N \le 200$) — кількість рядків і кількість стовпчиків відповідної таблиці. Далі йдуть M рядків по N цілих чисел у кожному. Значення цих чисел від 0 до $[M \times N/3]$ включно. Розмір вхідного файлу не перевищуватиме 512 Кб.

Вихідні дані. Вихідний файл **triomino.sol** повинен містити T рядків, у кожному — слово **YES** або слово **NO** (великими латинськими літерами) на позначення того, чи задає відповідна таблиця правильно заповнене фігурками триоміно поле.

Оцінювання. Щонайменше у 30% тестів не буде фігурок 2-го типу — будуть або «прямі» фігурки вигляду 1×3 чи 3×1, або взагалі не фігурки триоміно. Щонайменше у 40% тестів обидва значення M і N кратні 3.

Приклад вхідних і вихідних даних

triomino.dat	triomino.sol
2 4 8 1 2 2 2 6 7 7 9 1 4 4 3 6 7 8 9 1 0 4 3 6 8 8 9 10 10 10 3 0 5 5 5 2 6 1 1 2 2 1 1 0 1 0 2 0 1	YES NO

Рисунок до першого блоку:

1	2	2	2	6	7	7	9
1	4	4	3	6	7	8	9
1		4	3	6	8	8	9
10	10	10	3		5	5	5

Рекомендації щодо розв'язування

Розглянемо два різні правильні й ефективні способи, кожен з яких працює за час, пропорційний розміру вхідних даних. Спосіб (Б) очевидніший (більш стандартний), але обсяг коду виявляється меншим у способі (А).

Спосіб (А). Легко переконатися, що масив відповідає правильному покриттю фігурками триоміно тоді й тільки тоді, коли виконуються такі дві вимоги:

1. Значень «0» є рівно $(N*M)\text{mod}3$ штук, а кожного зі значень від «1» до $(M*N)\text{div}3$ — рівно по три штуки.

2. Для кожного зі значень від «1» до $(M*N)\text{div}3$, зайняті ним клітинки утворюють фігурку триоміно.

Твердження 1. *Три різні клітинки, «координати» (пари індексів) яких подані в парах* $(A.i, A.k)$, $(B.i, B.k)$ *та* $(C.i, C.k)$, *утворюють фігурку триоміно тоді й тільки тоді, коли* $(|A.i-B.i|+|A.k-B.k|)+(|A.i-C.i|+|A.k-C.k|)+(|B.i-C.i|+|B.k-C.k|)=4$.

Доведення. Для кожної з двох можливих фігурок триоміно дійсно маємо $1+1+2=4$.

Оскільки A, B і C — різні клітинки, то кожна з узятих у дужки сум двох модулів (вони називаються *манхетенськими відстанями* між клітинками) ціла додатна. Число 4 можна розкласти в суму трьох цілих додатних лише як $1+1+2$ (або $1+2+1$, або $2+1+1$). З іншого боку, манхетенська відстань дорівнює 1 тоді й тільки тоді, коли клітинки мають спільну сторону. Зразу дві одиниці для пар (A, B), (A, C) та (B,C) означають, що якась із клітинок має спільні сторони з обома іншими (а ті дві між собою — не мають). А це і є всі можливі фігурки триоміно, й лише вони.

Зрозуміло, перед застосуванням розглянутого критерію (для кожного зі значень від «1» до $(M*N)\text{div}3$) треба переконатися, що виконується перша вимога (щодо кількостей), а також сформувати самі трійки «координат». Це можна зробити, наприклад, так. Створимо масиви **num: array[0.. (200*200)div3] of integer** та **C: array[1.. (200*200)div3] of array [1..3] of record i, j: integer end.** В елементах **num** зберігатимемо кіль-

кості відповідних значень. Кожен елемент $C[v]$ буде масивом, що міститиме перелік «координат» різних клітинок зі значенням v. Масиви **num** та **C** за бажання можна формувати одразу, читаючи вхідні дані (прочитали з файлу v, збільшили num[v], якщо воно не перевищило 3, то дописали поточні «координати» до $C[v]$). Але тоді не забуваємо, що у вхідному файлі кілька тестових блоків, тому навіть якщо вже знаємо, що поточна відповідь «N0», дані блоку треба дочитати до кінця.

Спосіб (Б). Задача виділення фігурок у клітчатому полі (за допомогою, наприклад, пошуку в ширину або в глибину) досить стандартна, вказівки щодо її розв'язання можна знайти у багатьох джерелах. Вважаючи задачу виділення однієї фігурки відомою, дану задачу можна розв'язати так:

1. Завести допоміжний масив [1. . (200*200)div3] **of boolean,** смисл якого — «чи відбувався вже пошук фігурок, утворених даним числом».

2. Прочитати вхідні дані блоку у двовимірний масив.

3. Переглядати елементи двовимірного масиву рядок за рядком, і щоразу, знайшовши додатне число:

- якщо фігурку, утворену таким самим значенням, уже шукали — відповідь на поточний блок «N0», перевірку можна обривати;
- виділити фігурку, утворену поточним числом, замінюючи входження цього числа, наприклад, на протилежне (mass[i][j]:=−mass[i][j]), і рахуючи кількість клітинок у фігурці. Якщо кількість ≠3 — відповідь на поточний блок «N0», перевірку можна припинити.

4. Якщо перевірка не була припинена, перевірити будь-яку одну з властивостей: або «кількість нулів у масиві від 0 до 2», або «для всіх значень від 1 до (*M***N*)div3 фігурка була виділена» — це дасть остаточну відповідь «YES» або «N0» для поточного блоку.

Технічне зауваження. Вхідні дані досить великі за розміром, тому слід вибрати ефективний спосіб читання. **Read** мови Паскаль, **scanf** мови C та оператор >> класу **ifstream** мови C++ достатньо ефективні. А поєднання оператора >> об'єкта **cin** з f**reopen** — ні.

2. Мутація

Вчені планети Олімпія впритул наблизилися до відкриття характерного геному для виду олімпійських амеб, а саме: вони знайшли *послідовність генів*, про яку відомо, що вона містить рівно *один* зайвий ген. На поточному етапі вченим необхідно знайти всі гени, які можуть виявитися зайвими у цій послідовності. Для цього вчені використовують геном організму, який гарантовано є представником цього виду.

Послідовність генів, знайдена вченими, і геном організму можуть бути представлені у вигляді рядка, складеного з маленьких літер англійського алфавіту. Кожна літера відповідає окремому гену. Відомо, що організм A належить до певного виду X, якщо з рядка, що є *геномом* організму A, можна викреслити певні символи й отримати рядок, що є характерним геномом для виду X.

Завдання. Напишіть програму **mutation**, що за заданою послідовністю генів, що знайшли вчені, і геному представника виду, знайде індекси всіх генів, які можуть бути зайвими в послідовності генів, знайденій вченими.

Вхідні дані. Вхідний файл **mutation.dat** містить два рядки. Перший рядок є знайденою вченими послідовністю генів. Другий рядок — геном представника виду. Обидва рядки непорожні, складаються з маленьких літер латинського алфавіту, і довжина кожного з них не перевищує 40 000 символів.

Вихідні дані. Перший рядок вихідного файлу **mutation.sol** має одне ціле число — кількість генів, кожен з яких, можливо, є зайвим у знайденій ученими послідовності генів. У другому рядку виведіть *індекси* всіх таких генів у порядку зростання. Гарантовано, що існує хоча б один зайвий ген.

Оцінювання. У 50% тестів довжини рядків у вхідних даних не перевищують 2000.

Приклад вхідних та вихідних даних

mutation.dat	mutation.sol
adca	2
abcdaba	2 3

З послідовність ***adca*** потрібно видалати або ген ***d***, або ген ***c***. Отже, у першому випадку ми отримаємо послідовність ***aca***, яку можна отримати з геному, наприклад, так: ***abcdaba***, а у другому випадку — послідовність ***ada***, яку можна отримати, наприклад, так: ***abedaba***.

Рекомендації щодо розв'язування

Після формалізації отримаємо таку постановку задачі.

Дано два рядки A та B довжиною до 40 000. Необхідно знайти всі такі символи рядка B, що після видалення одного з них залишок B був би підпослідовністю рядка A.

Є відома задача про перевірку того, що один рядок є підпослідовністю іншого. Тобто нехай ми перевіряємо, що B є підпослідовністю A. Розв'язати таку задачу можна за допомогою жадібного алгоритму: перебираємо символи рядка A, починаючи з початку, і якщо зустрічаємо символ, рівний першій літері рядка B, то видаляємо цю літеру з нього. Якщо після закінчення виконання цього алгоритму рядок B залишився порожнім, то початковий рядок B був підпослідовністю рядка A. Складність такого алгоритму є лінійною відносно суми довжин двох рядків.

Інтерпретуємо цей жадібний алгоритм як алгоритм динамічного програмування: для кожного префікса рядка $A[1\ldots i]$ підрахуємо $P[i]$ — найбільшу довжину префікса рядка B, що $B[1\ldots P[i]]$ є підпослідовністю префікса $A[1\ldots i]$. Нехай $P[0]=0$, тоді для $i>0$: $P[i]=P[i-1]$, якщо $P[i-1]=|B|$ або $A[i]\neq B[P[i-1]+1]$ та $P[i]=P[i-1]+1$ у випадку $A[i]=B[P[i-1]+1]$ та $P[i-1]<|B|$. Якщо $P[|A|]=|B|$, то B є підпослідовністю A.

Адаптуємо цю ідею на випадок, коли нам треба видалити одну літеру з рядка B, а саме: підрахуємо $P[i]$

й аналогічну їй величину $S[i]$ — найбільшу довжину суфікса рядка B, за якого він є підпослідовністю суфікса $A[i\ldots|A|]$. Припустимо, що ми видалили деякий символ x з рядка B і отримали рядок $C(x)=B[1\ldots x-1]+B[x+1\ldots|B|]$. Нескладно зрозуміти, що $C(x)$ є підпослідовністю A тоді і тільки тоді, коли знайдеться префікс $A[1\ldots i]$, що $B[1\ldots x-1]$ є підпослідовністю $A[1\ldots i]$, а $B[x+1\ldots|B|]$ є підпослідовністю $A[i+1\ldots|A|]$. Тобто для деякого індексу i виконується $P[i]\geqslant x-1$ та $S[i+1]\geqslant|B|-x$, а з властивостей монотонності величин P, S можна стверджувати, що існує також індекс $0\leqslant j\leqslant|A|$ такий, що $P[j]=x-1$ та $S[j+1]\geqslant|B|-x$. Вважаємо, що $S[|A|+1]=0$.

Звідси маємо такий алгоритм:

1. Побудуємо величини P і S.

2. Перебираємо індекс j $(0\leqslant j\leqslant|A|)$: якщо $P[j]+S[j+1]\geqslant|B|-1$ та $P[j]+1\leqslant|A|$: індекс $P[j]+1$ додаємо до набору шуканих індексів.

3. База даних

На підприємстві працює N співробітників. Між ними існує M зв'язків «керівник — підлеглий». У співробітника може бути декілька керівників і підлеглих. Не існує послідовності зв'язків «керівник —підлеглий», яка починається і закінчується одним і тим самим співробітником.

Доступ до *корпоративної бази даних* регулюється системою прав. У кожен момент часу для кожного співробітника однозначно відомо: має він права на доступ до бази даних, чи ні. Певний час тому, коли базу даних було встановлено у підприємстві, ніхто не мав прав на доступ до неї. У процесі роботи з базою даних права на доступ змінювались за допомогою операцій виду.

1. Адміністратор надає права співробітнику X.

2. Адміністратор позбавляє прав співробітника X.

3. Співробітник X починає ділитися правами з усіма безпосередніми підлеглими.

4. Співробітник X починає ділитися правами з усіма безпосередніми підлеглими. Потім для кожного безпосереднього підлеглого співробітника X виконується операція 4.

5. Співробітник X припиняє ділитися своїми правами з усіма безпосередніми підлеглими.

Співробітник X має права на доступ до корпоративної бази даних, якщо виконується хоча б одна з таких умов.

1. Останньою операцією Адміністратора відносно співробітника X було надання йому прав.

2. Хоча б один з безпосередніх керівників, що має права на поточний момент, ділиться з ним правами на доступ.

Зверніть увагу. Якщо співробітник поділився правами доступу зі своїми підлеглими, а потім їх втратив, він все рівно продовжує ділитись правами зі своїми підлеглими. Однак вони не зможуть цим скористатися для одержання прав, поки цей співробітник знову не отримає прав на доступ (див. другий приклад з умови).

Завдання. Напишіть програму **database**, яка за заданою послідовністю операцій із зміни прав на доступ до корпоративної бази даних для кожного співробітника визначатиме, чи буде він мати права на доступ після виконання цієї послідовності операцій.

Для заданої послідовності операцій виконуються такі обмеження. Для кожного співробітника жодна з операцій 1 і 2 не зустрічається двічі поспіль. Не зустрічаються операції 3 та 4, коли у відповідний момент часу співробітник не має прав на доступ. Не зустрічаються операції 5, коли у відповідний момент часу співробітник не ділиться правами зі своїми підлеглими.

Вхідні дані. Перший рядок вхідного файлу **database.dat** містить два цілих числа: N $(1\leq N\leq 10\,000)$ — кількість співробітників підприємства, і M $(1\leq M\leq 50\,000)$ — кількість зв'язків «керівник — підлеглий». Наступні M рядків містять по два натуральних числа X і Y $(1\leq X, Y\leq N)$, визначаючи, що X є безпосереднім керівником Y. Наступний рядок містить число K $(1\leq K\leq 20\,000)$ — кількість операцій зміни прав. Наступні K рядків містять по два натуральних числа T і X $(1\leq T\leq 5, 1\leq X\leq N)$ — відповідно тип операції і номер співробітника, якого вона стосується.

Вихідні дані. Єдиний рядок вихідного файлу **database.sol** має містити N цілих чисел, розділених пропусками. i-те число рівне 1 або 0, залежно від того, матиме i-й співробітник права доступу після виконання заданих команд (1), чи ні (0).

Оцінювання. Принаймні у 20% тестів присутні лише операції 1–3. Принаймні у 35% тестів присутні лише операції 1–4. Принаймні в 40% тестів $N\leq 500$, $M\leq 500$, $K\leq 1000$.

Приклади вхідних і вихідних даних

database.dat	database.sol
5 5 1 2 1 3 2 4 3 4 3 5 4 1 1 4 1 1 2 5 1	1 1 0 1 0
2 1 1 2 4 1 1 3 1 2 1 1 1	1 1

Пояснення до прикладу 1

1: 1-й співробітник отримує права на доступ від адміністратора.

2: 1-й співробітник починає ділитися правами на доступ із 2-м і 3-м співробітниками. У свою чергу 2-й ділиться з 4-м, а 3-й ділиться з 4-м і 5-м співробітниками.

3: 2-й співробітник отримує права доступу від адміністратора.

4: 1-й співробітник припиняє ділитися правами на доступ із 2-м і 3-м співробітниками. У результаті 3-й співробітник втрачає права на доступ, а 2-й співробітник — ні, оскільки він має права на доступ від адміністратора. 4-й співробітник також не втрачає права на доступ, оскільки 2-й співробітник має права на доступ і ділиться з ним правами. 5-й співробітник втрачає права, тому що 3-й співробітник хоч і ділиться правами, на поточний момент не має прав.

Пояснення до прикладу 2

1: 1-й співробітник отримує права на доступ від адміністратора.

2: 1-й співробітник починає ділитися правами на доступ із 2-м співробітником. 2-й співробітник отримує права, тому що 1-й має права і ділиться правами з ним.

3: Адміністратор позбавляє прав 1-го співробітника. 2-й співробітник також втрачає права, оскільки 1-й хоч ділиться з ним правами, на даний момент їх не має.

4: 1-й співробітник отримує права на доступ від адміністратора. 2-й співробітник отримує права, оскільки 1-й співробітник має права на поточний момент часу і ділиться з ним правами.

Рекомендації щодо розв'язування

Введемо позначення: $\Gamma=(V, R)$ — граф, що задає ієрархію на підприємстві, V — множина вершин (співробітників), R — множина ребер (начальник, підлеглий). $last_0[t][x]$ — найбільший час виконання команди tx. $last[t][x]$ — найбільший момент часу, коли для вершини x була виконана операція t. $ans[x]$ — чи матиме права x після виконання заданої послідовності операцій.

$$last[t][x]=\begin{cases} last_0[t][x], & \text{якщо } t\neq 4;\\ \min(last_0[t][x], \min_{(y,x)\in R}(last[t][y])), & \text{якщо } t=4. \end{cases}$$

Топологічно відсортуємо вершини графа. Обчислимо відповідь для вершини графу, починаючи з топологічно старших. У такому разі, під час розгляду довільної вершини x всі її начальники будуть розглянуті попередньо. Умови володіння правами для x будуть виглядати наступним чином:

- $last[1][x] > last[2][x]$;
- $ans[y]$ AND $max(last[3][y], last[4][y]) > last[5][y]$, $(y, x)\in R$. Складність алгоритму $O(N+M+K)$.

4. Автомагістраль

Команда Логарифмічної області країни Олімпія збирається на олімпіаду, що відбудеться в далекому місті Експоненціальську. Їхати команда буде власним автобусом. Автомагістраль, що з'єднує Логарифмічну область з Експоненціальськом, складається з N послідовних фрагментів. Всередині кожного фрагмента можна їхати або по безплатній дорозі, витрачаючи a_i секунд, або по платній, витрачаючи c_i олімпійських центів і b_i секунд. Між цими фрагментами є транспортні розв'язки, через які можна з'їхати з однієї дороги на іншу. Такі перебудови вимагають q_i секунд (без різниці, з платної дороги на безплатну чи з безплатної на платну). При продовженні руху по тій самій дорозі аналогічних затримок не виникає. Спочатку можна виїхати хоч на безплатну дорогу, хоч на платну. Завершити шлях теж можна по будь-якій із двох доріг останнього фрагмента. Тому, розв'язки є лише між 1-м і 2-м фрагментами, між 2-м і 3-м, ..., між $(N-1)$-м і N-м.

У поїздці на олімпіаду дуже важливо встигнути вчасно, тому команду цікавить, за яку найменшу вартість можна дістатися з Логарифмічної області до Експоненціальська, витративши сумарно не більш як T секунд. Під час повернення з олімпіади час не настільки критичний, і перед командою була поставлена вимога сплатити на зворотньому шляху не більш як S олімпійських центів дорожніх зборів. Усі витрати часу і дорожні збори однакові в разі руху в обох напрямках.

Завдання. Напишіть програму **highway**, яка за даними про безплатні і платні дороги автомагістралі знаходитиме:

- найменшу ціну, за яку можна дістатися на олімпіаду за час, не більший T секунд;
- найменший час, за який можна повернутися з олімпіади, сплативши не більш ніж S центів дорожніх зборів.

Вхідні дані. Перший рядок вхідного файлу **highway.dat** містить три цілих числа N ($2\le N\le 40$) — кількість фрагментів магістралі, T ($0\le T\le 10^{16}$) і S ($0\le S\le 10^{16}$) — обмеження на сумарний час при пошуку першої відповіді й обмеження на сумарну вартість при пошуку другої. Другий рядок містить три цілі числа a_1, b_1 та c_1 — час руху по безплатній і платній дорогах 1-го фрагмента, і ціну проїзду по платній. Кожен з наступних $N-1$ рядків містить по чотири цілі числа q_i, a_i, b_i та c_i — спочатку час на перебудову між дорогами, потім час руху по безплатній і платній дорогах цього фрагмента, потім ціну проїзду по платній. Зауважимо, що на шляху на олімпіаду q_i — час, необхідний на перебудову з $(i-1)$-го фрагмента на i-й, а на зворотньому шляху q_i — час, необхідний на перебудову з i-го фрагмента на $(i-1)$-й (за умови, що автобус з'їжджає з платної дороги на безплатну або навпаки). Усі значення a_i, b_i та c_i ($1\le i\le N$) у межах від 1 до 10^{15}. Усі значення q_i ($2\le i\le N$) у межах від 0 до 10^9.

Вихідні дані. Єдиний рядок вихідного файлу **highway.sol** має містити два цілих числа — вартість найдешевшого серед способів дістатися на олімпіаду за час, не більший T секунд, і тривалість найшвидшого серед способів повернутися з олімпіади, сплативши не більш ніж S центів дорожніх зборів. Якщо жодного відповідного способу не існує, слід виводити -1, як одне з чисел.

Оцінювання. Щонайменше у 30% тестів виконується додаткове обмеження $2\le N\le 17$. Щонайменше у 40% тестів виконується додаткове обмеження вигляду: T, S, усі q_i, a_i, b_i та c_i не перевищують 10^5.

Приклад вхідних і вихідних даних

highway.dat	highway.sol
5 2012 2012 10000 17 10000 4 1000 17 1000 3 100 17 100 2 10 17 10 1 1 17 1	10000 10051

Пояснення. Найдешевший спосіб за час, не більший 2012 — проїхати 1-й фрагмент по платній дорозі, далі по безплатній: сумарна вартість 10000+0+0+0+0=10000 за час 17+4(перебудова)+1000+100+10+1=1132. Найшвидший спосіб повернутися із сумою зборів не більше 2012 — 5-й і 4-й фрагменти по безплатній, 3-й і 2-й по платній, 1-й по безплатній: час 1+10+2(перебудова)+17+17+4(перебудова)+10000=10051 при сумі зборів 0+1000+100+0+0=1100.

Рекомендації щодо розв'язування

Спосіб з *гарантовано* допустимими часом роботи і обсягом пам'яті використовує «meet in the middle», «2 вказівники» і модифікації злиття і має асимптотичну складність $O(2^{N/2})$ часу, $O(2^{N/2})$ пам'яті. Значно ефективнішого правильного розв'язку, мабуть, не існує, бо задача NP-повна (як узагальнення NP-повної задачі «є сукупність елементів, вибрати деякі з них так, щоб сума вибраних була якомога ближчою знизу до S»).

Враховуючи спосіб оцінювання даної олімпіади, непоганих результатів можна досягти також і перебірними (які не гарантують, що програма завжди вкладатиметься в обмеження по часу) і/або евристичними (які не гарантують правильності) методами — можна перепробувати кілька переборів і/або евристик і вибрати з них варіант, який приносить максимальні бали.

Оптимальний розв'язок — підтримка сукупності не домінованих пар. Побудуємо для фрагментів магістралі по дві сукупності пар (t, p) тобто (час; вартість) — одна сукупність для шляхів (від початку), що закінчуються на безплатній дорозі даного фрагмента, інша — на платній. На попередніх фрагментах дороги можна міняти.

Якщо для $(a.t, a.p)$ та $(b.t, b.p)$ виконується $(a.t \leqslant b.t)$ and $(a.p \leqslant b.p)$, будемо називати пару $(b.t, b.p)$ *домінованою* — її можна викинути, бо пара $(a.t, a.p)$ гарантовано не гірша.

Твердження 2. *Сукупність не домінованих пар можна впорядкувати $a_1.p < a_2.p < \ldots < a_L.p$; при цьому одночасно виконуватиметься $a_1.t > a_2.t > \ldots > a_L.t$.*

Доведення. Впорядкувати $a_1.p \leqslant \ldots \leqslant a_L.p$ можна будь-яку сукупність. Різних пар $(a.t, a.p)$ та $(b.t, b.p)$ з $a.p = b.p$ не буде, бо при $a.t < b.t$ пара b буде домінованою, при $a.t > b.t$ — пара a, а при $a.t = b.t$ немає смислу зберігати дві копії однакової пари. Далі, при $a.p < b.p$ не може бути $a.t \leqslant b.t$, інакше пара a була б домінованою.

Будуватимемо послідовності не домінованих пар, впорядковані $p\nearrow$, $t\searrow$. Для 1-го фрагмента ці послідовності тривіальні — єдина пара $\langle(a_1,0)\rangle$ для безплатної дороги і єдина $\langle(b_1,c_1)\rangle$ для платної. Надалі, є впорядковані послідовності не домінованих пар для $(s-1)$-го фрагмента (позначимо їх *lastfree* та *lastpaid*), будемо будувати послідовності не домінованих пар для s-го (позначимо *nextfree* і *nextpaid*).

Аналогічно злиттю, будемо рухатися індексом i по *lastfree*, індексом j по *lastpaid*, щоразу порівнювати поточні пари *lastfree*[i] та *lastpaid*[j], і вибирати ту з них, де менше p. Тільки, на відміну від злиття:

1. Кандидатом на додавання у результат є не просто вибрана пара, а пара, побудована за такими правилами:

a) При побудові *nextfree* поле p копіюється з вибраної пари, а при побудові *nextpaid* до нього ще додається вартість проїзду по платній дорозі поточного фрагмента c_s.

b) Поле t збільшується: на час проїзду по самому фрагменту (a_s для *nextfree* або b_s для *nextpaid*); якщо відбулася зміна дороги (платна→безплатна або навпаки), то додатково ще на q_s.

2. У разі рівності *lastfree*[i].p = *lastpaid*[j].p, вибираємо пару з меншим (з урахуванням всіх поправок п. 1b) часом.

3. Дописуємо побудовану пару до результату, тільки коли або результат ще порожній, або t побудованої пари строго менше за t останньої наразі пари результату.

Вибір меншого p і п. 2 гарантують, що домінована пара не потрапить у результат раніше пари, яка її домінує. А п. 3 — що не потрапить пізніше.

Зберігання лише не домінованих пар, *як правило,* суттєво зменшує розміри сукупностей. Але можливі вхідні дані (наприклад, всі $q_i = 1$, всі $b_i = 3$, решту визначити як $a_i = c_i = 7 \cdot 2^i$), для яких розміри кожної із сукупностей не домінованих пар для i-го фрагменту будуть 2^{i-1}, що при $i \approx N \approx 42$ забагато. Тому, *крім* вже розглянутого етапу, треба користуватися також ідеєю meet in the middle. Під час розробки тестів планувалося, щоб ефективна побудова не домінованих пар (без meet in the middle) набирала 60–70% балів.

Оптимальний розв'язок — meet in the middle (з «2-ма вказівниками»). Сукупності не домінованих пар можна будувати не для всіх фрагментів зразу, а двічі по $N/2$. Візьмемо $N_1 := N \operatorname{div} 2$, $N_2 := N - N_1$, і побудуємо сукупності не домінованих пар для N_1 фрагментів, починаючи від початку («ліва половина») і N_2 фрагментів, починаючи від кінця («права»). Тоді розміри кожної з послідовностей не перевищуватимуть $2^{20} \approx 10^6$. Остаточні відповіді задачі будемо шукати, поєднуючи ці послідовності для «половинок». Причому, для поєднань дуже зручно, що послідовності задані в порядку $p\nearrow$, $t\searrow$.

Твердження 3. *«Ліва половинка» з характеристиками (left[i].t, left[i].p) може бути частиною оптимального маршруту (вигляду* «min *час при вартості* $\leqslant S$») *лише при поєднанні з «правою половинкою», яка має значення right[j].p, найближче знизу до S — left[i].p.*

Доведення. «Праву половинку» з більшим значенням p брати не можна, бо шлях буде не допустимим (вартість $> S$). А взявши «половинку» зі значенням $right[k].p < right[j].p$, отримаємо (завдяки $p\nearrow$, $t\searrow$) $right[k].t > right[j].t$, тоді як час треба мінімізувати.

Отже, у разі перебору i в порядку спадання, кожне відповідне за твердженням 3 значення j буде або тим самим, що для попереднього i, або більшим ($left[i].p$ зменшилося $\Rightarrow S$ — $left[i].p$ збільшилося). Тобто, застосовна ідея «два вказівники»: організувавши цикл пошуку j всередині циклу перебору i, можна продовжувати пошук j з того місця, де спинилися при попередньому i, й усе проходження двома вказівниками працюватиме за $O(|left|+|right|)$.

Усього проходів 2-ма вказівниками треба зробити $2\times2\times2 = 8$ штук: min вартість для часу $\leqslant T$ чи min час для вартості $\leqslant S$; якою (безплатною чи платною) дорогою закінчується ліва половина; якою права половина. Якщо одна з доріг безплатна, а інша платна, треба безпосередньо в даному проході врахувати $q[N_1]$.

Бажано *не* переписувати код 8 разів, а оформити як підпрограму і багатократно викликати. Наприклад, підпрограм може бути дві: min вартість для часу $\leqslant T$ і min час для вартості $\leqslant S$; яка дорога з якою перевіряються, варто задати параметрами. Аналогічно, варто зробити однією процедурою (з різними параметрами) побудову сукупностей не домінованих пар *nextfree* і *nextpaid*.

Інші способи формування не домінованих переліків. Замість модифікацій злиття, можна формувати кожен перелік не домінованих пар так: включити до переліку пари, відповідні абсолютно всім 2^{Ni} шляхам; відсортувати за неспаданням p, а за рівних p за неспаданням t. Після цього, за один прохід, легко вибрати лише не доміновані пари, порівнюючи кожну поточну пару з останньою вибраною.

Такий підхід простіший для написання, але менш ефективний, бо сортування переліку довжиною $2^{N/2}$ потребує $O(2^{N/2}\log(2^{N/2})) = O(2^{N/2}\cdot N)$ дій, а рекомендований алгоритм формує правильно впорядкований перелік за $O(2^{N/2})$.

Ще один спосіб побудови не домінованих пар — виконання додаткових дій з map-ом (STL мови C++); див., зокрема, вказівки до задачі «Choosing a camera» студентської олімпіади SEERC-2011 **(codeforces.ru/blog/entry/2880).** Цей спосіб теж потребує $O(2^{N/2}\cdot N)$ дій.

Під час розробки тестів планувалося, щоб дані способи (за використання meet in the middle) набирали 75–80% балів.

Розв'язання динамічним програмуванням при S, T до 10^5–10^6. Якщо S і T досить малі, щоб можна було підтримувати кілька масивів від 0 до max(S, T), задачу можна вирішити, розв'язавши серії підзадач «*Яку мінімальну суму $S(i, r, t)$ доведеться сплатити, щоб дістатися від початку до кінця i-го фрагменту, витративши не більш ніж t центів, причому r=0 означає, що останній i-й фрагмент їхали по безплатній дорозі, r=1 — по платній*», й аналогічну серію $T(i, r, s)$.

При $S, T \sim 10^{16}$, цей спосіб практично неможливий. Але він гарантовано проходить 40% тестів, де значення до 10^5. В інших тестах можна всі значення відповідної природи поділити (із заокругленням до цілого), щоб поділені S і T були в межах до, наприклад, $4\cdot10^6/N$ (приблизно максимальне значення, за якого програма ще поміщається в обмеження по часу і пам'яті). Сума заокруглених може відрізнятися від заокруглення суми, тому такий спосіб не завжди даватиме правильну відповідь. Але є ймовірність, що якась частина відповідей інших тестів теж буде правильною. Причому, якщо значення ділити саме з заокругленням (до найближчого), ця ймовірність вища, ніж коли ділити цілочисельно (без остачі).

Гілки і межі, інші перебірні методи. Бажаючі можуть знайти інформацію про ці методи за ключовими словами «метод гілок та меж», він же «метод розгалужень та обмежень», «метод ветвей и границ», «branch and bound», а також «backtracking». Можна пробувати й інші перебірні або перебірно-евристичні методи («метод отжига», генетичні алгоритми, і ще багато інших).

Під час розробки тестів ставилася мета, щоб очевидні варіанти гілок і меж набирали 60–70% балів. Але поведінка гілок і меж сильно й малопрогнозовано залежить від дрібних змін, тому можливі значні відхилення від оголошених відсотків. Ще раз нагадаємо, що при використаному лояльному способі перевірки можна було пробувати змінювати оціночні функції та інші деталі перебору й вибирати варіант, що набере найбільше балів.

Більш ефективними виявилися переборні підходи, які починаються із сортування за певною евристичною властивістю, яка визначає, припущення щодо якого фрагменту варто робити на зовнішньому рівні рекурсії, щодо якого (не обов'язково сусіднього) — на наступному, і т. д. Інше евристичне порівняння визначає, у якому порядку робити рекурсивні виклики: спочатку розглянути, що даний фрагмент їдуть по платній дорозі, а потім, що по безплатній, чи навпаки.

(Далі буде)

* * *