Ідеї розв'язків задач другого туру

28 березня 2011 р.

1 "Точки" (Даніїл Нейтер).

Наївний розв'язок Підрахувати шукану суму за означенням:перебрати всі пари індексів i та j та просумувати доданки вигляду $(x_i - x_j)^2 + (y_i - y_j)^2$. Часова складність алгоритму $O(N^2)$. Такий розв'язок набирає 50 балів.

Ефективний розв'язок Якщо перегрупувати доданки, можна розбити шукану суму на дві:

$$\sum_{1 \le j < i \le N} ((x_i - x_j)^2 + (y_i - y_j)^2) = \sum_{1 \le j < i \le N} (x_i - x_j)^2 + \sum_{1 \le j < i \le N} (y_i - y_j)^2$$

Таким чином ми можемо окремо розв'язувати задачу для іксів і для ігріків, а потім просто просумувати результат.

Розглянемо підрахунок суми по іксах:

$$\sum_{1 \le i < j \le N} (x_i - x_j)^2 = \sum_{1 \le i < j \le N} (x_i^2 - 2x_i x_j + x_j^2) = \sum_{1 \le i \le N} ((i-1)x_i^2 - 2x_i \sum_{1 \le j < i} x_j + \sum_{1 \le j < i} x_j^2)$$

Отримана формули дозволяє обчислити сумму реккурентно за один прохід: по мірі підрахунку суми по і значення

$$sumx = \sum_{1 \le j < i} x_j \text{ та } sumx2 = \sum_{1 \le j < i} x_j^2$$

підраховуються в окремих змінних. Використовуючи підраховані значення цих сум черговий доданок суми обчислюється за константний час. Після цього залишається додати x_i та x_i^2 до значень змінних $sumx$ та $sumx2$ відповідно і перейти до обчислення наступного доданку.

Часова складність алгоритму $O(N)$.

Особливості реалізації Обмеження задачі вимагають для представлення $sumx$, $sumx2$ та сумарної відповіді використовувати 64-бітні типи даних (у C/C++ це `long long`, в Pascal - `int64`).

2 "Миші та сир" (Роман Різванов).

Позначення $MouseX_i$ -– абсциса миші з номером i. $CheeseX_j$ -– абсциса шматку сиру з номером j. $D(i, j)$ -– відстань від i-ої миші до j-ого шматку сиру.

Припущення Будемо вважати, що $N > 0$ і $M > 0$, бо якщо N або M дорівнює нулю, то відповідь до задачі -– це N. Також припустимо що $Y_0 = Y_1 = 0$, тобто всі миші і шматки сиру знаходяться на одній горизонтальній прямій.

Ефективний розв'зкок

Лема 1. *Кількість найближчих шматків сиру для кожної миші не більше ніж 2.*

Доведення. Існує всього 2 напрямки куди миша може побігти: вліво або вправо. Як тільки вона натрапить на сир, то цей сир буде найближчім у відповідному напрямку. □

Об'єднаємо мишей і шматки сиру в один відсортований за абсцисою список. Так як в умові сказано, що списки мишей і шматків сиру відсортовані, то це можна зробити зі складністю $O(N)$. Тоді за один прохід зліва направо можна для кожної миші знайти найближчій шматок сиру зліва. Для цього будемо запам'ятовувати останній розглянутий шматок сиру, він і є найближчій для миші яку ми розглядаємо. Аналогічно шукаються найближчі справа. Для кожної миші виберемо з 2-ох напрямків, той який дає коротший шлях, або обидва якщо вони однакової довжини. Якщо на обраному напрямку між сиром і мишею є інша миша, то ми цей напрямок вилучаємо, так як інша миша швидше з'їсть той сир. Тепер всі напрямки мишей безпосередньо ведуть к сиру, і до кожного шматку сиру веде не більше одного напрямка з кожної сторони.
Будемо опрацьовувати мишей зліва направо. Якщо чергова миша може рухатись вліво, і сир зліва не обрала жодна з попередніх мишей, або цей сир обрала миша з такою самою відстанню до сиру, то ця миша рухаючись вліво встигне поїсти сир не заважаючи іншим мишам. Так як к сиру веде не більше одного напрямку, то цей вибір не вплине на наступних мишей, а тому і не може в подальшому погіршити відповідь. В інших випадках рухаючись вліво ми не зможемо покращіти відповідь, тому якщо миша має можливість рухатись вправо то це єдина можливість для цієї миши відвідати сир, тому вона обирає правий напрямок. Якщо миша не може рухатись вправо, то ніяким чином покращіти відповідь вона не зможе. Так як кожен вибір рухатись вліво ніяким чином не впливає на наступні, і кожен такий вибір не збільшує відповіді, то цей вибір є оптимальним для відповідної миші. Правий напрямок обирається тільки якщо рухаючись вліво ми погіршимо відповідь, цей вибір може в подальшому збільшити відповідь на одиницю, але так як ми заощадили одиницю не рухаючись вліво, то цей вибір не гірше чим вибір рухатись вліво (але рухатись вліво може бути гірше чим рухатись вправо). Тому наведена стратегія дає оптимальну відповідь.
Наведений алгоритм має складність $O(N)$.

Зведення до одновимірного випадка

Лема 2. *Y_0 і Y_1 не впливають на результат.*

Доведення. Єдине на що впливають Y_0 і Y_1 це відстані D(i, j), але згідно до розв'язку нам не потрібно їх обчислювати, а потрібно вміти перевіряти чи $D(i,j) < D(i,k)$ для деяких j та k, так як відстані не від'ємні то цю перевірку можна переписати як $D^2(i,j) < D^2(i,k) \Leftrightarrow$
$\Leftrightarrow (MouseX_i - CheeseX_j)^2 + (Y_0 - Y_1)^2 < (MouseX_i - CheeseX_k)^2 + (Y_0 - Y_1)^2 \Leftrightarrow$
$\Leftrightarrow (MouseX_i - CheeseX_j)^2 < (MouseX_i - CheeseX_k)^2 \Leftrightarrow$
$\Leftrightarrow |MouseX_i - CheeseX_j| < |MouseX_i - CheeseX_k|$ □

Таким чином, не втрачаючи загальності можна перенести (зберігаючи абсциси) всіх мишей і шматки сиру на пряму $Y = 0$.

3 "Мутація" (Ярослав Твердохліб).

1. Очевидний алгоритм. Переберемо усі можливі множини символів, які будуть викидатись. Для кожної такої множини знайдемо рядок (геном), який залишиться після викидання цієї множини. Для цього рядка знайдемо його вартість (ризик захворювання) та додамо до неї вартість викидання набору. Складність такого алгоритму $O(2^K N)$.

2. Попередній підрахунок. Умова задачі наштовхує на перебір усіх множин символів, що видаляються. Тому множник 2^K з оцінки складности прибрати не вийде. Отже, залишається лише зменшувати коефіцієнт при ньому. Для цього треба навчитись для кожної множини символів швидко шукати вартість рядку, що залишиться після її видалення. Нехай M – набір символів, що викидаються, а S - початковий рядок.

Розглянемо символи початкового рядка, що стоять на позиціях l і r ($l < r$). Який вигляд повинен мати M, щоб l і r стали поруч після викидання? Виходячи з природніх міркувань, можемо поставити 2 умови:

1. $\forall i \in (l, r) : \ S_i \in M$
2. $S_l \notin M \wedge S_r \notin M$

З другої умови маємо, що для фіксованого l кандидатів на r не може бути більше, ніж K, оскільки між l та r не може бути символів S_r. Таким чином, кількість пар потенційних сусідів не перевищує NK. Ми можемо перебрати їх усіх за $O(NK)$, йдучи вказівником на l з права на ліво і підтримуючи для кожного символу номер його останньої зустрічі.

Зафіксуємо деяку пару потенційних сусідів, які стоять на позиціях l і r. Насправді, для нас не важливо на яких саме позиціях стоять ці символи у початковому рядку, якщо ми знаємо що це за символи та які символи стоять між ними, оскільки нам потрібно лише вміти для кожного M знаходити вартість рядку що залишився, а не сам рядок. Нехай $M' = \{S_i \mid i \in (l, r)\}$. Тоді трійка (S_l, S_r, M') дозволяє однозначно визначити, чи стануть символи на позиціях l та r сусідами, що в свою чергу дає нам знати, чи треба додавати вартість їх сусідства до сумарної вартості рядка при її підрахунку.

Звідси отримуємо алгоритм:

Переберемо усі пари кандидатів на l і r, яких не більше за NK. Для кожної пари знайдемо M'. Заведемо масив, у якому для усіх трійок (a, b, P) будемо зберігати сумарну вартість сусідства для усіх пар l і r, таких що:

- $l < r$
- $S_l = a \wedge S_r = b$
- $M' = P$

Усе це можна зробити за $O(NK)$ часу. Далі переберемо усі набори символів, що викидаються. Для кожного фіксованого набору M переберемо усі трійки (a, b, P), та перевіримо, чи виконуються для них такі умови:

- $a \notin M \wedge b \notin M$
- $P \subset M$

Для трійок, для яких вони виконуються, знайдемо сумарну вартість, додамо до неї вартість викидання набору M та якщо усе це не перевищує T – збільшимо відповідь на 1. Складність такого алгоритму $O(4^K K^2 + NK)$.

3. Швидкий перебір підмасок. Викладений вище алгоритм за даних обмежеть працює не набагато швидше, ніж очевидний перебір, але він має одну суттєву перевагу – при експотенційному множнику в оцінці складності не присутня довжина рядку. Не зважаючи на це, експонента зросла від 2^K до 4^K, що без сумніву є дуже повільним. Якщо подивитись, звідки береться це 4^K, ми побачимо, що для кожного набору M ми перебираємо усі можливі P і вже потім перевіряємо, чи є вони його підмножинами. Таким чином, ми переглядаємо багато зайвих кандидатів на P, що дуже сильно впливає на час роботи програми.

Розглянемо усі пари (M, P). Не важко переконатись, що для кожного символа існують лише 3 варіанти:

- символ належить і M, і P
- символ належить M, але не належить P
- символ не належить ні M, ні P

Отже, усього таких пар 3^K. Якщо ми зможемо для фіксованого M перебрати лише такі P, які є його підмножинами, то сумарна складність алгоритму зменшиться до $O(3^K K^2 + NK)$.

Зазвичай при переборі множини задаються бітовими масками, які зберігаються у вигляді цілих чисел. Останні K бітів числа однозначно визначають множину: якщо біт з номером i рівний одиниці, то i-й символ присутній у множині, інакше він відсутній. Зараз буде описано алгоритм, який за даною бітовою маскою перебирає усі її підмаски у спадному порядку (при порівнянні бітових масок порівнюються числа, що їх задають).

Нехай нам дано бітову маску $mask$ і ми вже перебрали усі її підмаски, які більші або рівні за $mask'$, причому $mask' \neq 0$. Знайдемо наступну у порядку спадання підмаску. Поглянемо на останню групу нульових бітів у числі $mask'$, перед якими стоїть одиничний біт. Щоб отримати наступну у порядку спадання підмаску, нам потрібно обнулити цей одиничний біт, а усі біти після нього, які є одиничними у $mask$ зробити одиничними і у $mask'$. Не важко переконатись, що операція $mask' = (mask' - 1)\&mask$ як раз це і зробить. Тут під операцією & розуміється побітове 'і'.

Як вже було написано вище, це дає нам змогу покращити складність алгоритму до $O(3^K K^2 + NK)$.

4. Формула включення-виключення. Нам вдалося зменшити експоненту в оцінці складності, тепер спробуємо зменшити множник при 3^K. Цей множник з'являється через те, що окрім множини M та її підмножини P ми перебираємо ще 2 символи, які не повинні бути присутніми в M.

Розглянемо такий (неправильний) алгоритм:

Для кожної пари кандидатів на сусідство l і r будемо зберігати лише множину M' символів, що знаходяться між ними у S. Заведемо масив v, у якому для кожної множини P будемо зберігати сумарну вартість усіх пар l і r, для яких M' співпадає з P. При фіксованому M знайдему суму по всім його підмножинам у v і вважатимемо, що це і є вартістю отриманого рядку.

Як вже було сказано, цей алгоритм не є вірним. Кожну пару (l, r) ми враховуємо у деяких M, у яких за попереднім алгоритмом ми не повинні були їх враховувати, а саме:

- Ми додали вартість (l, r) до усіх множин, які містять l, отже нам потрібно відняти цю вартість від $v[M \cup \{S_l\}]$.
- Аналогічно, віднімемо цю вартість від $v[M \cup \{S_r\}]$.
- Оскільки тепер ми відняли цю вартість двічі від усіх множин, що містять обидва кінці, віднімемо її від $v[M \cup \{S_l, S_r\}]$.

Тепер після виконання описаного вище алгоритму сумарна вартість отриманого рядку буде рахуватись правильно, отже і алгоритм буде працювати правильно. Складність становить $O(3^K + NK)$.

Підрахунок сумми по всім підмаскам усіх масок у режимі offline. Описаний вище алгоритм при кожному новому M перебирає усі його підмножини, хоча масив v не змінюється з моменту заповнення. Якщо нам вдасться шляхом деяких маніпуляцій перетворити масив v так, щоб для кожної множини M в ньому одразу зберігалась сума значень зі старого v по всім її підмаскам, то знаходити вартість рядку ми зможемо за $O(1)$.

Позначимо через $mask_k$ бітову маску, яка утворена з останніх (молодших) k біт маски $mask$.

Розглянемо такий ітеративний алгоритм, який складається з K ітерацій. Перед початком (після нульової ітерації) масив v побудований за попереднім пунктом розв'язку. Після ітерації з номером k для кожної бітової маски $mask$ у комірці $v[mask]$ буде зберігатись сума усіх значень з початкового v по усім маскам $mask'$, для яких виконується:

- $mask'_k \subset mask_k$
- Перші (старші) $K - k$ біт маски $mask'$ співпадають з першими $K - k$ бітами маски $mask$.

На ітерації з номером k будемо послідовно у порядку зростання для усіх масок знаходити $v[mask]$. Невжако переконатись, що якщо біт з номером k (якщо вважати що біти нумеруються з одиниці) рівний 0, то $v[mask]$ не зміниться з попередньої ітерації. Якщо ж цей біт рівний 1, то до $v[mask]$ потрібно додати $v[mask \setminus \{k\}]$.

Після виконання K ітерацій цього алгоритму у кожній комарці нового масиву v буде сума по всім підмаскам маски, що задається номером цієї комірки, з початкового v. Розв'язок набуває складності $O(2^K K + NK)$.

5. Остаточний алгоритм. Підсумуємо викладений вище алгоритм.

1. Переберемо усі пари потенційних сусідів з рядка S (параграф 2).
2. За цими парами, використовуючи формулу включення-виключення, побудуємо масив v (параграф 4).
3. Застосуємо ітеративний алгоритм, після виконання якого у масиві v будуть суми з початкового v по всім підмаскам кожної маски (параграф 5).
4. Переберемо множину символів M, які будуть викинуті, та за допомогою масиву v знайдемо вартість рядку, який залишиться після її викидання, а також вартість самого викидання множини M.
5. Якщо ця вартість не перевищує T – збільшимо відповідь на 1.

6. Зауваження При виконання пунктів 2-3 алгоритму з попереднього параграфу в масиві v може статись переповнення типу int у C++ або longint у Pascal, але оскільки в умові гарантується, що вартість кожного з отриманих рядків є меншою за 2^{31} – на загальний результат це не вплине.