Ідеї розв'язків задач першого туру

23 березня 2010 р.

1 "Многочлен" (Андрій Коротков).

Ефективний розв'язок. Будь-який многочлен степені k, розкривши дужки та звівши подібні доданки, можна привести до вигляду $a_0x^k + a_1x^{k-1} + \cdots + a_{n-1}x + a_n$, де a_i – коефіцієнти. Шукана сума рівна значенню многочлена при $x = 1$, що можна підставити у початковий вигляд многочленна, оскільки перетворення рівносильні.

Відповідь буде рівна $(1 + 2 + 3 + \cdots + n^2)^n = (\frac{n^2(n^2+1)}{2})^n$. Складність алгоритму – $O(n)$ або $O(n^2)$, якщо суму обчислювати циклом.

При правильній реалізації розв'язок набирає 100% балів.

Ручне розкриття дужок. Більш-менш зручно, на думку автора, це можна зробити при $n \leq 4$. Через велику кількість доданків при більших значеннях n це практично важко.

При правильному обчисленні вказаних значень можна одержати 15% балів.

Безпосереднє перемноження многочленів. При перемноженні многочленів степенів a та b результуючий многочлен буде мати степінь $a + b$. При обчисленні в циклі це займе $n^4 + 2n^4 + 3n^4 + \cdots + (n-1)n^4 = \frac{n^5(n+1)}{2} = O(n^6)$ операцій. Автору не відомо, чи існують більш оптимальні способи обчислення, що використовують звичайний метод множення многочленів. Слід лише зазначити, що піднесення до степеню логарифмічним алгоритмом не дасть покращення асимптотики, приклад наведено для степені двійки 2^m: $n^4 + 2^2n^4 + 4^2n^4 + 2^{2k-2}n^4 = \frac{2^{2k}-1}{2^2-1}n^4 = \frac{n^2-1}{3}n^4 = O(n^6)$. Результуючий многочлен буде мати степінь $n(n^2-1)$, що менше за n^3. Затрати по пам'яті будуть $O(n^3)$.

При правильній реалізації і здачі без додаткових оптимізацій набирає 35% балів.

Генерація масиву констант. Описаний вище розв'язок можна використати для генерації масиву констант.

При правильному знаходженні можливих констант, з урахування обмеженої тривалості олімпіади, розв'язок набирає 65% балів.

Зауваження. Існують більш швидкі методи множення многочленів, наприклад, такі як метод Карацуби та метод Фур'є. Останній дає змогу перемножити два многочлени степеня n за кількість операцій $O(n \log n)$.

2 "Мутація" (Ярослав Твердохліб).

Формалізація задачі і загальна ідея розв'язку. Нехай A – базова послідовність геному піддослідного організму, B – базова послідовність геному цільового організму. N – довжина геномів обох організмів. P_i – кількість генів, які будуть змінені на i-му етапі.

Ключовою фразою в умові є «Вчені можуть обирати, які саме гени змінювати». Це означає, що нам не важливі самі геноми піддослідного і цільового організмів, а важлива лише кількість генів, у яких вони різняться.

Розв'язок задачі складається з двох етапів:

1. Знайти кількість генів, у яких різняться геноми піддослідного і цільового організмів (назвемо її K).

2. Знайти найменшу кількість етапів, за яку можна отримати з геному що збігається з початковим, геном що відрізняється від нього у K генах.

Можливі варіанти реалізації першого етапу. Очевидна реалізація першого етапу алгоритму має часову складність O(N), що є недостатньо ефективним для розв'язку задачі. Також існує більш швидкий алгоритм, який має часову складність O(LCM($|A|, |B|$)), де LCM — найменше спільне кратне. Оптимальний алгоритм має часову складність O($|A| + |B|$). Тут $|A|$ — довжина рядка A, $|B|$ — довжина рядка B.

Очевидний алгоритм. Тут і надалі будемо вважати, що геноми представляють собою рядки довжини N, які складаються з символів 0 та 1. Перший геном позначимо S, а другий — F.

Очевидно, що $S_i = A_{i \ mod \ |A|}$. Тут S_i позначає символ рядку S з номером i, якщо нумерація починається з нуля; mod — звичайна операція знаходження остачі від ділення. Аналогічно, $F_i = B_{i \ mod \ |B|}$. Таким чином, можна послідовно знаходити S_i та F_i, і якщо вони різняться, то збільшувати K на 1.

Часова складність такого алгоритму O(N), просторова — O(1).

Трохи швидший алгоритм. Повністю повторимо попередній алгоритм за виключенням того, що згенеруємо S та F не до кінця, а до позиції LCM($|A|, |B|$). Нехай знайдена кількість різних символів рівна T. Далі рядки S та F почнуть повторюватись, отже $K = T \cdot N/\text{LCM}(|A|, |B|)$.

Часова складність такого алгоритму O(LCM($|A|, |B|$)), просторова– O(1).

Ефективний алгоритм Для визначеності припустимо, що $|A| \geq |B|$ і N =LCM($|A|, |B|$). Розглянемо символ B_i. При багаторазовому повторенні рядку B символ B_i буде опинятись на позиціях F_0, $F_{|B|}$, $F_{2|B|}$, і т. д. у рядку F. Усього таких позицій буде рівно $N/|B| = (|A| \cdot |B|)/(\text{GCD}(|A|, |B|) \cdot |B|) = |A|/\text{GCD}(|A|, |B|)$, де GCD — найбільший спільний дільник. Нехай R_i — набір усіх індексів, на яких опиниться елемент B_i у рядку F. L_i — набір відповідних їм індексів елементів A, які відповідають індексам з R_i.

Лема 1. *Усі елементи з R_i мають однакову остачу від ділення на G =GCD($|A|, |B|$).*

Доведення. Очевидно, що вони мають однакову остачу від ділення на $|B|$. Але $|B|$ ділиться націло на G. Кожний елемент набору R може бути записаний у вигляді $i + tB$, де t — ціле. Звідси випливає, що усі елементи R мають таку саму остачу від ділення на G, як і i. □

Лема 2. *Якщо $x \equiv y \ (mod \ G)$, то і $x \ mod \ |A| \equiv y \ mod \ |A| \ (mod \ G)$.*

Доведення. $a \ mod \ b$ означає віднімання від a b доти, доки a не стане менше b. Запишемо $x \equiv y \ (mod \ G)$ і будемо віднімати від лівої частини $|A|$ доти, доки вона не стане меншою за $|A|$. Повторимо те саме з правою частиною. Очевидно, що така операція не змінить рівність, тому що $|A|$ ділиться націло на G. Тоді рівність набуде вигляду $x \ mod \ |A| \equiv y \ mod \ |A| \ (mod \ G)$. □

Лема 3. *Набір L_i містить $|A|/GCD(|A|,|B|)$ різних елементів, причому усі елементи мають однакову остачу від ділення на $GCD(|A|,|B|)$.*

Доведення. Ми довели, що усі елементи з L_i мають однакові остачі від ділення на G. Залишилося довести, що усі вони різні. Якщо якісь два елементи з L рівні, то існують такі a_1 і a_2 з R, для яких виконується $a_1 \equiv a_2 \ (mod\ A)$ та у цей же час $a_1 \equiv a_2 \ (mod\ B)$. З цього випливає, що $a1 \equiv a2 \ (mod\ \mathrm{LCM}(A,B))$, але це неможливо, тому що $N = \mathrm{LCM}(A,B)$, $a_1 \neq a_2$, $a_1 < N$ та $a2 < N$. □

Отже, для пошуку K нам треба B_0 порівняти з усіма $A_x, x \in L_0$, B_1 порівняти з усіма $A_x, x \in L_1$ і т. д. Інакше кажучи, $K = \sum_0^{|B|-1} C_i$, де C_i рівне кількості одиниць серед $A_x, x \in L_i$, якщо B_i рівне 0 і кількості нулів серед $A_x, x \in L_i$, якщо B_i рівне 1. Очевидно, що кількість нулів рівна $|A|/G$ мінус кількість одиниць. Знайти кількість одиниць у кожній з таких множин можна за допомогою одного лінійного проходу по A із запам'ятовуванням кількості одиниць для усіх остач індексів від ділення на G. Таким чином ми отримали алгоритм зі складністю роботи $\mathrm{O}(|A| + |B|)$.

Можливі варіанти реалізації другого етапу. Перед початком експерименту маємо рядок, який відрізняється від S у 0 символах. Після першого етапу ми можемо змінити довільні P_1 символів, отримавши при цьому рядок, який відрізняється від S у P_1 символах. Подивимось, що буде на наступному етапі. Ми можемо вибрати P_2 символів, які не було змінено на першому кроці, тоді загалом за 2 етапи буде змінено $P_1 + P_2$ символів. Замість того щоб вибрати P_2 ще не змінених символів, можна вибрати і змінити $P_2 - 1$ не змінений символ, тоді загалом буде змінено $P_1 + P_2 - 2$ символи. Таким чином, після двох етапів ми можемо отримати рядок, що відрізняється від S у $|P_2 - P_1|$ символах або $|P_2 - P_1| + 2$ символах і так далі до $P_2 + P_1$ символів. Будемо казати, що після кроку з номером 2 ми можемо отримати рядок, який відрізняється від S у будь-якій кількості символів, що лежить на проміжку від $|P_1 - P_2|$ до $P_1 + P_2$ і має таку саму парність, як і кінці цього проміжку. Якщо $P_1 + P_2$ перевищує N, то замість цього виберемо найбільше число, яке не перевищує N і має таку саму парність, як і P_1

Лема 4. *Нехай після $h-1$ етапів ми можемо отримати рядок, який відрізняється у будь-якій кількості символів, що лежить проміжку від l до r і має таку саму парність, як і кінці (які є одної парності), тоді після наступного етапу ми зможемо отримати рядок, який відрізняється від S у будь-якій кількості символів, яка лежить на проміжку від $|l_0 - P_h|$ до $r + P_h$ і матиме таку саму парність, як і кінці проміжку. Тут l_0 — число з проміжку від l до r, яке має таку саму парність, як і кінці і таке, що $|l_0 - P_h|$ — мінімальне. Якщо $r + P_h$ перевищує N, то замість цього виберемо найбільше число, яке не перевищує N і має таку саму парність, як і $r + P_h$.*

Доведення. За припущенням усі можливі кількості символів, у яких рядки відрізняються після $h-1$ кроку мають однакову парність. Розглянемо якийсь рядок, який відрізняється від S у Q символах, $l \leq Q \leq r$. З нього ми можемо отримати рядок, що відрізняється від S у $|Q - P_h|$ символах, наклавши змінювані P_h символів на якомога більшу кількість з Q символів, у яких рядки відрізнялись. Аналогічно ми можемо отримати рядок, який відрізняється від S у будь-якій кількості символів, яка лежить на проміжку від $|Q - P_h|$ до $Q + P_h$ символів. Якщо замість Q брати послідовно усі можливі кількості, отримані після $h-1$ кроку, застосовуючи ті самі міркування, можна отримати будь-яку кількість, яка лежить на проміжку від $|l_0 - P_h|$ до $r + P_h$. □

Таким чином, знаючи кількість символів, у яких рядки відрізняються після h кроків (зберігаючи цю кількість у вигляді інтервалу), можна отримати кількість символів на кроці

$h+1$. Перший раз коли K потрапить у потрібний інтервал і матиме парність таку саму, як і кінець інтервалу і буде потрібним етапом.

3 "Музей" (Даніїл Нейтер).

Формалізація задачі і загальна ідея розв'язку. З умови задачі слідує, що музей являє собою зв'язний граф $G = (V, E)$ з одним циклом. Позначимо множину вершин графа (залів музею) через $V = \{v_1, v_2, \ldots, v_N\}$, а множину ребер (коридорів) – через $E = \{\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_N, v_N\}\}$. Єдиний цикл в графі G позначимо через $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$.

Також позначимо через $G\backslash\{v\}$ граф, що утворюється з графа G вилученням вершини v і всіх суміжних з нею ребер.

План патрулювання задається перестановкою $a_1, a_2, \ldots, a_n$ чисел від 1 до n такою, що або $a_i = i$, або $\{i, a_i\} \in E$.

Відомо, що кожну перестановку можна розкласти у множину циклів. Розглянемо 3 можливих варіанти циклів:

1. $a_i = i$. Охоронець стоїть на місці.
2. $a_i = j, a_j = i$. Тоді необхідно, щоб $\{i, j\} \in E$. Охоронці залів i та j міняються місцями.
3. $a_{i_1} = i_2, a_{i_2} = i_3, \ldots, a_{i_{k-1}} = i_k, a_{i_k} = i_1$. Тоді необхідно, щоб у графі G також існував цикл $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$. Охоронці переходять вздовж циклу.

В графі G за умовою рівно один цикл. Значить або цей цикл не входить в план патрулювання, або входить в одному з 2 можливих напрямків обходу.

Тоді всі можливі плани патрулювання можна розбити на наступні категорії:

- Цикл $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ входить в план патрулювання. Тоді $G\backslash\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ — «ліс» (ациклічний, але не обов'язково зв'язний граф).
- Цикл $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ не входить в план патрулювання. Тоді розглянемо $\{v_{i_1}, v_{i_2}\}$ – одне з ребер циклу. Знову можливо 2 варіанти:
  - (i_1, i_2) входить в план патрулювання. Тоді $G\backslash\{v_{i_1}, v_{i_2}\}$ — «ліс».
  - (i_1, i_2) не входить в план патрулювання. Тоді вилучемо з графа ребро $\{v_{i_1}, v_{i_2}\}$ і отримаємо дерево.

В кожному з випадків нам достатньо вміти розв'язувати задачу для «лісу», для чого достатньо вміти знаходити розв'язок для дерева (розв'язок для лісу — добуток розв'язків для всіх його дерев).

Отже загальна схема розв'язку має наступний вигляд:

1. Знайти у початковому графі цикл.
2. Для кожного з описаних вище варіантів звести задачу до «лісу».
3. Розв'язати задачу для «лісу» в кожному з випадків.
4. Просумувати відповіді для всіх випадків (враховуючи, що цикл можна орієнтувати 2 способами).

Пошук циклу в початковому графі. Цикл у початковому графі можна знайти за допомогою пошуку в глибину. Почнемо пошук з вершини v_1. Для кожної проглянутої пошуком вершини будемо запам'ятовувати par_i — номер батька i-ї вершини в дереві пошуку. Тоді алгоритм має наступний вигляд:

1. $v = 1$ – номер поточної вершини.
2. Якщо в поточної вершини не залишилося непроглянутих ребер, то перейти до вершини par_v і повернутися на крок 2.
3. Нехай $\{v, u\}$ – наступне ще непроглянуте ребро. Тоді:
   - u — ще не проглянута вершина. Тоді покладемо $par_u = v, v = u$ і перейдемо до кроку 2.
   - $u = par_v$. Тоді пропускаємо поточне ребро і переходимо до кроку 2.
   - u — проглянута і $u \neq par_v$. Тоді, оскільки граф не орієнтований, u буде предком v у дереві пошуку, а отже ми знайшли цикл: $v, par_v, par_{par_v}, \ldots, par_{par_{\ldots par_v}} = u$. Кінець алгоритму.

Час виконання алгоритму складає $O(N)$.

Розв'язок задачі для дерева. Як вже сказано вище, відповідь для «лісу» – добуток відповідей для кожного з дерев, що утворюють цей «ліс». Тому достатньо розглянути розв'язок лише для дерева.

Оскільки в дереві немає циклів, кожна вершина може входити в план патрулювання або як нерухома ($a_i = i$), або в циклі з 2 вершин, що проходить по ребру дерева.

Для розв'язання задачі для дерева використаємо метод динамічного програмування. Для цього спочатку підвісимо дерево за довільну вершину. Позначимо множину номерів номерів дітей i-ї вершини через $chld_i = \{c_i^{(1)}, c_i^{(2)}, \ldots, c_i^{(k_i)}\}$. Нехай $f(i)$ — відповідь для піддерева, що починається з i-тої вершини; $g(i)$ — відповідь для піддерева, що починається з i-тої вершини за умови, що (i, par_i) входить в план патрулювання. Тоді з очевидних міркувань отримуємо наступні рекурентні співвідношення:

$$f(i) = g(i) = 1, \text{ якщо } v_i \text{ — «лист».} \tag{1}$$

$$g(i) = \prod_{j \in chld_i} f(j) \tag{2}$$

$$f(i) = g(i) + \sum_{k \in chld_i} g(k) \prod_{\substack{j \in chld_i \\ j \neq k}} f(j) \tag{3}$$

Відповіддю для дерева буде $f(r)$, де r — номер кореня дерева.

Співвідношення (1)-(3) можна безпосередньо реалізувати в програмі. Тоді часова складність розв'язку буде $O(N^2)$ у гіршому випадку.

Покажемо, як покращити цю оцінку. Використаємо метод динамічного програмування вдруге – при підрахунку $f(i)$ і $g(i)$:

$$f^{(0)}(i) = g^{(0)}(i) = 1 \tag{4}$$

$$g^{(j)}(i) = g^{(j-1)}(i) f(c_i^{(j)}) \tag{5}$$

$$f^{(j)}(i) = g^{(j-1)}(i) g(c_i^{(j)}) + f^{(j-1)}(i) f(c_i^{(j)}) \tag{6}$$

$$f(i) = f^{(k_i)}(i) \tag{7}$$

$$g(i) = g^{(k_i)}(i) \tag{8}$$

При підрахунку $f(i)$ і $g(i)$ за співвідношеннями (4)-(8) часова складність розв'язку буде $O(N)$.

Технічні особоливості реалізації

1. Оскільки за умовою необхідно вивести остачу від ділення шуканої кількості планів на P, то під час розрахунків необхідно після кожної операції брати остачу від ділення на P.

2. У випадку, якщо в розв'язку використовується рекурсія, необхідно задати налаштування компілятора для збільшення розміру стеку:

   - В Turbo Delphi Explorer це можна зробити за допомогою директиви "`{$MAXSTACKSIZE <STACK_SIZE>}`", де `<STACK_SIZE>` – необхідний розмір стеку в байтах.
   - В Free Pascal – "`{$M STACK_SIZE}`"
   - В Microsoft Visual Studio 2008 Express Edition – "`#pragma comment(linker,"/STACK:<STACK_SIZE>")`"
   - В Dev-C++ таких директив немає. Проте при акуратній реалізації достатньо розміру стеку, що виділяється цим компілятором за замовчуванням.

   Альтернативою є реалізація розв'язку без використання рекурсії. Це можливо, але призводить до трохи складнішого коду.