Задача A. Гострота та кислинка

Опис задачі

- Дано гостроту a_i та кислинку s_i для n видів приправ.
- Для всіх можливих пар приправ потрібно визначити, чи можна отримати цільові гостроту x та кислинку y, змішавши їх у відповідній пропорції.

Обмеження

- $n \leqslant 50$
- $0 \leqslant a_i, s_i, x, y \leqslant 50$
- $(a_i, s_i) \neq (x, y)$
- Усі вхідні дані — цілі числа.

Розв'язок

- Нанесіть значення гостроти та кислинки на двовимірну площину. Координати смаку, які можна отримати, змішуючи приправи i та j, лежать строго на відрізку, що з'єднує точки (a_i, s_i) та (a_j, s_j).
- Чи лежить цільова точка (x, y) на цьому відрізку, можна визначити за $O(1)$ часу, використовуючи векторні операції, такі як скалярний добуток і векторний добуток.
- Або, якщо існує коректна суміш, то є пропорція змішування $p : q$, де p і q — додатні цілі числа, для яких $p + q \leqslant 50$. Тому задачу також можна розв'язати повним перебором усіх можливих пропорцій.

Задача B. Зламана клавіша

Розв'язок (підхід з використанням динамічного програмування)

- Нехай $\mathrm{dp}(i, c)$ — це мінімальна кількість виконань **Операції 1** так, що i-й символ S уже дописано, а символ c наразі не міститься в X.
- Наївний перехід займає $O(|S|\sigma^2)$ часу, де $\sigma = 26$ — розмір алфавіту.
- Обмеживши переходи лише заміною найостаннішого доданого символу на інший символ, можна зменшити складність переходу до $O(|S|\sigma)$ часу.

Розв'язок ($O(|S|)$ підхід)

- Немає потреби виконувати **Операцію 1** взагалі до першої появи '`z`' у S.
- Після цього щоразу, коли ми один раз виконуємо **Операцію 1**, ми можемо продовжувати виконувати **Операцію 2** доти, доки кількість різних символів у поточному підрядку не перевищує 25. Ця жадібна стратегія є оптимальною.
- Загальна часова складність: $O(|S|)$.

Задача C. Підрахунок трикутників

Властивості триангуляції

- Якщо всі n точок лежать на одній прямій, кількість трикутників дорівнює 0.
- Інакше, коли є n точок загалом і m точок лежать на межі їх опуклої оболонки, будь-яка триангуляція цих точок дає рівно $2n - m - 2$ трикутники.
- $\rightarrow$ Отже, щоб максимізувати кількість трикутників, слід мінімізувати кількість точок m, що лежать на межі нової опуклої оболонки.

Випадок $k \geqslant 2$

- Коли $k \geqslant 2$, ми завжди можемо досягти $m = 3$ на новій опуклій оболонці. Це робиться шляхом розміщення двох із нових точок достатньо далеко, щоб утворити величезний трикутник, який містить усі інші $N + k - 3$ точок.
- $\rightarrow$ Отже, суму відповідей для всіх $k = 2, \ldots, K$ можна обчислити за $O(1)$ часу за формулою $2(N + k) - 3 - 2 = 2N + 2k - 5$.

Випадок $k = 1$

- Коли $k = 1$, позначимо точки на опуклій оболонці початкових N точок як $1, 2, \ldots, m$ у порядку проти годинникової стрілки.
- Додавши рівно одну точку, чи можемо ми побудувати нову опуклу оболонку так, щоб лише неперервний відрізок початкових точок оболонки $i, i+1, \ldots, j$ (арифметика індексів за модулем m) залишився на межі нової оболонки?
- Цю перевірку здійсненності можна виконати, перевіривши перетин прямої, що проходить через точки i та $i - 1$, і прямої, що проходить через точки j та $j + 1$.
- Ми можемо знайти мінімальне значення $j - i$ за допомогою підходу двох вказівників (або бінарного пошуку).

Задача D. Проведи інспекцію

Розв'язання

- Обчисліть XOR-суму всіх ребер кімнат (відрізків, які з'являються непарну кількість разів у вхідних прямокутниках).
  - Для всіх горизонтальних ребер ефективно обчислювати префіксні суми зліва для кожної координати y. Та сама логіка застосовується і до вертикальних ребер.
- Побудова неорієнтованого графа з решти граничних відрізків розбиває межу на кілька неперетинних циклів. Рівно один із цих циклів відповідає зовнішній межі будинку, а решта — дивним просторам. Отже, ми можемо знайти дивні простори, просто виключивши цикл з найбільшою площею.
- Загальний алгоритм можна реалізувати за час $O(n \log n)$.
- Серед альтернативних підходів — використання формули Ейлера для многогранників, щоб порахувати кількість областей, і трасування граничних циклів (наприклад, за правилом лівої руки) для обчислення площ.

Задача E. Смарагдовий магазин

Розв'язок

- Розглянемо граф, де предмети — це вершини, а люди — це ребра. Якщо знайти відповідь для кожної зв'язної компоненти, то можна перемножити ці відповіді та помножити на відповідний мультиноміальний коефіцієнт, щоб отримати загальну відповідь.
- Для того щоб у зв'язної компоненти існував коректний розв'язок, вона має бути або деревом, або псевдодеревом (компонента рівно з одним циклом).
- Знайдемо відповідь для кожного випадку.
- Нехай орієнтоване ребро проведено від A_i до B_i.

Випадок дерева

- Степінь виходу єдиної непроданої вершини в компоненті має бути 0 (тобто вона не може з'являтися в A).
- Зафіксуємо непродану вершину і вважатимемо її коренем. Для кожної вершини i нехай S_i — це таке значення:
  - Кількість нащадків вершини i, яких можна досягти з i.
- Тоді відповідь дорівнює $N!$, помноженому на добуток обернених до S_i.
- S_i можна обчислити за допомогою динамічного програмування (DP) на дереві або rerooting tree DP (DP на дереві в усіх напрямках), щоб опрацювати всі можливі корені.

Випадок псевдодерева

- Якщо подивитися на циклічну частину, існує лише 2 можливі способи покупки, тому зафіксуємо орієнтацію циклу і розглянемо їх.
- Подібно до випадку дерева, добуток обернених до S_i безпосередньо дає відповідь.
- Зауважте, що ми розглядаємо сам цикл як корінь. Потрібно бути обережним, коли всі ребра в циклі спрямовані в один бік.
- Реалізація є досить складною.

Задача F. Функції та рядки

Розв'язок (1)

- Розглянемо кількість потрібних операцій для кожного підрядка.
- Якщо рядок можна перетворити на "01", кількість операцій дорівнює $|T| - 2 +$ (кількість перестановок).
- Якщо рядок можна перетворити на "0" або "1", кількість операцій дорівнює $|T| - 1 +$ (кількість перестановок).
- Отже, достатньо визначити:
  - Чи можна кожен підрядок перетворити на "0", "1" або "01";
  - Мінімальну кількість перестановок, потрібну для такого перетворення.

Розв'язок (2)

- Розглянемо заміни:
  - **Операція 2:** Вибрати "00" і замінити його на "1".
  - **Операція 3:** Вибрати "11" і замінити його на "0".
- Можна відобразити символи в елементи $\mathbb{Z}_3$ (цілі числа за модулем 3) так:
  - **Операція 2:** Вибрати "11" і замінити його на "2".
  - **Операція 3:** Вибрати "22" і замінити його на "1".
- Це відповідає заміні двох сусідніх однакових значень на їхню суму за модулем 3, що є інваріантом (сума елементів масиву за модулем 3 не змінюється).
- Використовуючи динамічне програмування або подібні техніки, можна порахувати кількість підрядків, які можна успішно перетворити на "0", "1" або "01".

Розв'язок (3)

- Перестановки (Операція 1) потрібні лише тоді, коли в рядку символи '0' і '1' чергуються (за винятком самих цільових рядків "0", "1" і "01").
- Якщо рядок не складається з чергування символів, перестановки не потрібні:
  - Загалом, якщо вибрати "00", що стоїть поруч із '1' (або "11" поруч із '0'), наприклад $\ldots 0001 \cdots \to \ldots 0011 \ldots$, то можна виконати наступну Операцію 2 або 3 (або завершити процес).
  - Хоча може здатися, що ми можемо дійти до стану "10", єдиними станами безпосередньо перед "10" є "01", "000" і "111". Останні два ("000" і "111") можна перетворити на "01" рівно за одну операцію.
- Якщо інвертувати символи на непарних позиціях і застосувати run-length compression, загальну кількість перестановок можна легко обчислити.

Задача G. Величезний плейлист

Розв'язок

- Нехай $t[i] + x[i]$ секунд — це момент, коли людина i зрадіє, якщо ми запускаємо плейлист з початку пісні 1.
- Якщо ми починаємо відтворення плейлиста зі зсувом на y секунд (де y — початкова позиція відносно пісні 1), то момент, коли людина i зрадіє, дорівнює:
  - Якщо $y \leqslant x[i]$: $t[i] + x[i] - y$ секунд
  - Якщо $y > x[i]$: $t[i] + x[i] - y + \sum L$
- Якщо відсортувати M людей за зростанням $x[i]$ і зафіксувати пісню, з якої починається відтворення (це визначає значення y), то людей можна розділити на дві групи за наведеними вище випадками в деякій межі. Після цього мінімальний час, необхідний, щоб задовольнити всіх для кожної початкової пісні, можна обчислити за допомогою префіксних/суфіксних максимумів (накопичуваних максимумів) або дерева відрізків.

Задача H. Скільки способів упакувати?

Розв'язок

- Припустимо, що $A[1] \leqslant A[2] \leqslant \ldots \leqslant A[N]$. Визначимо, чи є $S = \{1, 2, \ldots, N\}$ хорошою для заданого w.
- Будь-яка кулька, важча за $w - A[1]$, повинна утворювати власну групу, що складається лише з цієї однієї кульки.
- Якщо сума ваг усіх кульок із вагою не більшою за $w - A[1]$ не перевищує w, то вони повинні бути згруповані разом в одну групу.
- А що, якщо це не так?
- Якщо сума ваг кульок із вагою не більшою за $w - A[1]$ строго більша за w:
  - Для кожного i такого, що $A[i] \leqslant w - A[1]$, існує коректне розбиття, у якому кулька 1 і кулька i належать до однієї групи.
  - Однак неможливо помістити всі кульки з вагою не більшою за $w - A[1]$ в одну групу.
  - $\rightarrow$ Це означає, що існує принаймні два різні коректні розбиття!
- Якщо припустити, що $A[1] \leqslant A[2] \leqslant \ldots \leqslant A[N]$, то необхідна і достатня умова того, що $S = \{1, 2, \ldots, N\}$ є хорошою відносно w, така:
  1. $A[N] \leqslant w$
  2. Сума ваг кульок у S, вага яких не перевищує $w - A[1]$, не перевищує w.
- Підрахунок можна виконати, відсортувавши кульки за спаданням ваг і використовуючи динамічне програмування.

Задача I. В одному рядку

Упорядкування блоків

Для будь-якого фіксованого неперервного блоку вершин зберігайте три значення:

$$\text{info}(B) = (z_B, o_B, c_B),$$

де z_B — кількість нулів у блоці, o_B — кількість одиниць у блоці, а c_B — кількість інверсій, що повністю лежать усередині цього блоку.

Якщо два блоки A і B розташувати в такому порядку, то кількість інверсій між ними дорівнює $o_A z_B$. Отже, розташувати A перед B не гірше, ніж B перед A, тоді і тільки тоді, коли

$$o_A z_B \leqslant o_B z_A.$$

Тому блоки слід сортувати за зростанням відношення $\frac{o}{z}$. Ділити не потрібно: порівнюйте два блоки за допомогою перехресного множення. Визначимо

$$A \prec B \quad \Longleftrightarrow \quad o_A z_B < o_B z_A.$$

Рівність можна розв'язувати довільно, оскільки обидва порядки дають однаковий внесок між блоками.

Якщо блок A має бути безпосередньо перед блоком B, ми замінюємо їх одним блоком

$$\text{merge}(A, B) = (z_A + z_B,\ o_A + o_B,\ c_A + c_B + o_A z_B).$$

Розв'язання одного дерева

Розглянемо одне кореневе дерево. Ми перетворимо його на список блоків. Пізніше ці блоки можна буде відсортувати за $\prec$, і їх конкатенація дасть оптимальний допустимий порядок для цього дерева.

Обробляйте вершини знизу вгору. Для кожної вершини u підтримуйте пріоритетну чергу блоків, що належать уже обробленим піддеревам дітей u, впорядковану за $\prec$. Щоб обробити u:

1. Злити всі пріоритетні черги дітей u у пріоритетну чергу u.
2. Створити поточний блок C, що складається лише з u:
$$C = (1, 0, 0) \text{ якщо } v_u = 0, \qquad C = (0, 1, 0) \text{ якщо } v_u = 1.$$
3. Поки найменший блок B у пріоритетній черзі задовольняє $B \prec C$, злити його в C:
$$C \leftarrow \text{merge}(C, B).$$
Потім видалити B з пріоритетної черги.
4. Додати фінальний блок C у пріоритетну чергу u.

Чому це правильно? Припустімо, що найменший доступний блок дитини — це B, і $B \prec C$. Якби не було обмежень дерева, B слід було б розташувати перед C. Однак кожна вершина в B є нащадком u, а C містить u, тому B не може стояти перед C. Серед усіх блоків, які мають бути розташовані після C, вибір найменшого наступним є оптимальним за аргументом про обмін сусідніх елементів вище. Отже, B примусово має стояти безпосередньо після C, і ці два блоки можна злити. Ми повторюємо це, доки жоден із решти блоків дітей не хоче стояти перед C.

Після обробки кореня його пріоритетна черга містить усі фінальні блоки дерева у відсортованому порядку. Якщо ці блоки — $B_1, B_2, \ldots, B_s$, то мінімальна кількість інверсій усередині цього дерева дорівнює

$$\sum_{i=1}^{s} c_{B_i} + \sum_{1 \leqslant i < j \leqslant s} o_{B_i} z_{B_j}.$$

Другу суму можна обчислити, проходячи блоки зліва направо та підтримуючи префіксну суму одиниць.

Об'єднання двох дерев

Для дерева Райана один раз обчисліть його фінальний відсортований список блоків. Нехай це буде

$$R_1, R_2, \ldots, R_s.$$

Також попередньо обчисліть:

- оптимальну кількість інверсій усередині дерева Райана, позначену як base_R;
- префіксні суми нулів і одиниць у списку R.

Для кожного дерева друга після розшифрування його міток обчисліть його фінальний відсортований список блоків

$$G_1, G_2, \ldots, G_t$$

та його внутрішню оптимальну кількість інверсій base_G так само.

Між двома деревами немає обмежень на предків. Тому після того, як обидва дерева зведено до відсортованих блоків, оптимальний спільний порядок — це просто відсортоване злиття двох списків блоків за тим самим порядком $\prec$.

Нам не потрібно фактично зливати два списки. Для блоку G з дерева друга нехай $pos(G)$ — це кількість блоків Райана, які строго менші за G. Тоді:

- блоки Райана перед G дають внесок

$$z_G \cdot \left(\text{кількість одиниць у } R_1, \ldots, R_{pos(G)}\right);$$

- блоки Райана після G дають внесок

$$o_G \cdot \left(\text{кількість нулів у } R_{pos(G)+1}, \ldots, R_s\right).$$

Оскільки блоки Райана відсортовані, $pos(G)$ знаходиться бінарним пошуком. Підсумувавши це для всіх блоків дерева друга, отримаємо перехресні інверсії $\text{cross}(R, G)$, а відповідь для цього друга дорівнює

$$\text{base}_R + \text{base}_G + \text{cross}(R, G).$$

Складність

Для дерева з m вершинами пріоритетні черги можна зливати за методом small-to-large. Кожен блок переміщується $O(\log m)$ разів, а кожна операція над пріоритетною чергою коштує $O(\log m)$, тому ця реалізація працює за

$$O(m \log^2 m)$$

часу.

Для кожного запиту після зведення дерева друга кожен із його фінальних блоків виконує один бінарний пошук у списку блоків Райана. Отже, сумарна додаткова вартість для всіх запитів становить

$$O\left(\left(\sum m_k\right) \log n\right).$$

Задача J. Японське мистецтво

- Для будь-яких двох точок існує рівно два можливі варіанти ламаної, залежно від їх взаємного розташування.
- Призначте кожному кольору булеву змінну, де один із двох варіантів відповідає True, а інший — False.
  - Наприклад, якщо одна точка розташована внизу ліворуч від іншої, то два можливі варіанти ламаної — ⌊ і ⌉.
- Перевірте перетини між усіма парами можливих ламаних і зведіть задачу до 2-SAT.
  - Наприклад, якщо одночасний вибір ламаних x і y неможливий, бо вони перетинаються, це можна подати як умову 2-SAT $\neg(x \wedge y) \equiv \neg x \vee \neg y$.
- Оскільки існує $O(n^2)$ пар ламаних, загальна кількість диз'юнктів також дорівнює $O(n^2)$. Отже, задачу можна розв'язати за час $O(n^2)$.

Задача K. Знай свій префікс

- Якщо довжина A менша за M, можна наївно симулювати операції (це трапляється щонайбільше $O(\log M)$ разів).
- Коли довжина A перевищує M, можна відкинути все після перших M елементів. Тоді операцію можна переформулювати так:
  - **Операція:** Замінити послідовність A на $(A[1], A[2], \ldots, A[x], A[1], A[2], \ldots, A[M-x])$.

    Операція: Замінити A на $(A[1], A[2], \ldots, A[x], A[1], A[2], \ldots, A[M-x])$.
- Префікс коротшої послідовності між $(A[1], \ldots, A[x])$ та $(A[1], \ldots, A[M-x])$ збігається з префіксом довшої послідовності.
- Отже, нам потрібно знати лише перші $\max(x, M-x)$ елементів A після попередньої операції.
- Аналізуючи запити у зворотному порядку, можна визначити, скільки елементів префікса потрібно на кожному кроці, а потім побудувати послідовність від початку.
  - Деталі реалізації: наприклад, заповнення масиву на місці, використання `std::deque` тощо.

Задача L. Залишити лише корінь

1 Формулювання задачі та рекурентні співвідношення

Нехай $T = (V, E)$ — дерево, коренем якого є вершина 1. Дозволено повторно видаляти лист. Якщо видалений лист u є нащадком кореня (вершини 1), нічого не відбувається. Інакше нехай p — батько u, а gp — батько p. Видалення u спричиняє приєднання K нових копій піддерева з коренем у p (без урахування u) до gp.

Наша мета — знайти мінімальну кількість операцій, потрібних, щоб звести дерево лише до вершини 1.

1.1 Ефективна вага піддерева

Нехай $E(u)$ позначає **ефективну вагу** піддерева з коренем у вершині u. Ця вага відображає мультиплікативний вплив піддерева, коли обробляється його батько.

- Якщо u — лист, його видалення не запускає жодного копіювання його братів. Тому базовий внесок такий:
$$E(u) = 1$$
- Якщо u — внутрішня вершина з множиною дітей $\mathcal{C}(u) = \{v_1, v_2, \ldots, v_d\}$, то видалення листів її дітей рекурсивно запускає копіювання. Зокрема, видалення компонента v_i множить решту структури на $K + 1$ копій. За індукцією ефективна вага u задовольняє:
$$E(u) = 1 + \sum_{v \in \mathcal{C}(u)} (K+1)^{E(v)-1}$$

1.2 Вартість обробки піддерева

Нехай $dp[u]$ — мінімальна кількість операцій, щоб повністю видалити піддерево з коренем у u, за умови що батька u не видалено. Нехай $R[u]$ позначає накопичену вартість обробки всіх нащадків u до моменту, коли залишається лише u.

- Для листа u:
$$R[u] = 0$$
- Для внутрішньої вершини u:
$$R[u] = \sum_{v \in \mathcal{C}(u)} (R[v] + h(E(v) - 1)) \pmod{998244353}$$
де $h(x)$ — вартість згортання ефективної ваги x, визначена як:
$$h(x) = \frac{(K+1)^x - 1}{K}$$

Загальна вартість обробки піддерева з коренем у u (коли його батько не змінюється) дорівнює:

$$dp[u] = h(E(u)) + R[u] \pmod{998244353}$$

Для кореня (вершини 1) його дітей можна видаляти незалежно, не викликаючи копіювання на рівні батька. Тому загальна мінімальна кількість операцій:

$$\text{Total Cost} = \sum_{u \in \mathcal{C}(1)} dp[u] \pmod{998244353}$$

2 Керування степеневими баштами за допомогою ланцюга функції Ейлера

Визначення $E(u)$ містить степеневі башти виду $(K+1)^{(K+1)^{\cdots}}$. Ці значення зростають надзвичайно швидко і не можуть бути збережені у стандартних машинних типах. Потрібно обчислювати їх за модулем $M_0 = 998244353$.

2.1 Ланцюг модулів

Згідно з **узагальненою теоремою Ейлера**, для будь-яких цілих a, x та модуля m:

$$a^x \equiv a^{(x \bmod \phi(m))+\phi(m)} \pmod{m} \quad \text{для } x \geqslant \phi(m)$$

Щоб обчислити $E(u) \pmod{M_0}$, потрібно обчислити показник $E(v) - 1$ за модулем $\phi(M_0)$. Це вимагає обчислення показника *цього* виразу за модулем $\phi(\phi(M_0))$, і так далі. Визначимо спадний ланцюг модулів:

$$M_0 = 998244353$$

$$M_{i+1} = \phi(M_i) \quad \text{для } i \geqslant 0$$

Оскільки $\phi(m)$ є парним для всіх $m > 2$, значення M_i щонайменше вдвічі зменшується принаймні кожні два кроки. Отже, ланцюг дуже швидко завершується на 1. Довжина ланцюга дорівнює $L = 31$.

Математичний доказ скінченності ланцюга модулів

Нехай m — парне ціле число. Функція Ейлера $\phi(m)$ рахує кількість чисел $1 \leqslant k \leqslant m$, взаємно простих із m. Оскільки m парне, будь-яке взаємно просте k має бути непарним. Отже, $\phi(m) \leqslant \frac{m}{2}$. Для будь-якого непарного m значення $\phi(m)$ є парним. Тому після щонайбільше одного кроку модуль стає парним, а далі значення зменшується щонайменше вдвічі кожні дві ітерації:

$$\phi(\phi(m)) \leqslant \frac{m}{2}$$

Це гарантує логарифмічне спадання, обмежуючи довжину ланцюга $L \leqslant 2\log_2(M_0) \approx 60$. Для 998244353 він завершується рівно за 31 крок.

3 Система переходів і відстеження “великих” значень

Оскільки теорема Ейлера працює по-різному залежно від того, чи менший показник x за $\phi(m)$, потрібно відстежувати, чи є $E(u)$ великим. Визначимо поріг $T_{\text{large}} = 10^9 + 7$. Оскільки $T_{\text{large}} > M_i$ для всіх $i \geqslant 0$, будь-яке значення $E(u) \geqslant T_{\text{large}}$ гарантовано більше за $\phi(M_i)$ для будь-якого модуля в нашому ланцюгу.

3.1 Подання стану динамічного програмування

Для кожної вершини u зберігаємо:

1. `large_val[u]`: булевий прапорець, що вказує, чи $E(u) \geqslant T_{\text{large}}$.
2. `exact_E[u]`: точне значення $E(u)$, якщо `large_val[u]` хибне.
3. `E[u][i]`: значення $E(u) \pmod{M_i}$ для $0 \leqslant i \leqslant L$.

3.2 Переходи станів

Для вершини u з дітьми $\mathcal{C}(u)$:

1. Оцінка точного значення: Щоб уникнути переповнення цілого типу, обчислюємо:

$$\text{exact\_E}[u] = 1 + \sum_{v \in \mathcal{C}(u)} (K+1)^{\text{exact\_E}[v]-1}$$

Якщо в будь-який момент під час цього підсумовування $\text{exact\_E}[u] \geqslant T_{\text{large}}$ або будь-яка дитина має `large_val[v] == true`, встановлюємо `large_val[u] = true`.

2. Модульне піднесення до степеня: Для кожного індексу модуля $i \in [0, L]$:

$$E[u][i] = \left(1 + \sum_{v \in \mathcal{C}(u)} (K+1)^{\exp(v,i)}\right) \pmod{M_i}$$

де показник $\exp(v, i)$ визначається масштабом v:

- Якщо `large_val[v]` хибне і $\text{exact\_E}[v] - 1 < M_{i+1}$:

$$\exp(v, i) = \text{exact\_E}[v] - 1$$

- Інакше (показник великий відносно модуля $\phi(M_i) = M_{i+1}$):

$$\exp(v, i) = ((E[v][i+1] - 1) \bmod M_{i+1}) + M_{i+1}$$

4 Аналіз складності

4.1 Часова складність

- **Побудова ланцюга модулів:** знаходження $\phi(M_i)$ займає $O(\sqrt{M_i})$ часу. Оскільки M_i зменшується експоненційно, цей крок займає $O(\sqrt{M_0}) \approx 31594$ операцій, що є незначним.
- **Топологічне впорядкування:** обхід BFS займає $O(N)$ часу.
- **Переходи динамічного програмування:** для кожної вершини u ми ітеруємося по її дітях $\mathcal{C}(u)$ і обчислюємо степені для L модулів. Використовуючи бінарне піднесення до степеня, обчислення $(K+1)^{\exp} \pmod{M_i}$ займає $O(\log M_i)$ часу. Отже, сумарна вартість переходу для вершини u дорівнює $O(|\mathcal{C}(u)| \cdot L \cdot \log M_0)$. Підсумовуючи по всіх вершинах V:

$$\sum_{u \in V} O(|\mathcal{C}(u)| \cdot L \cdot \log M_0) = O(N \cdot L \log M_0)$$

4.2 Просторова складність

Список суміжності займає $O(N)$ пам'яті. Масиви станів `E[N][L]` потребують зберігання 31 цілого числа на вершину. Отже, загальна просторова складність становить $O(N \cdot L)$.