Задача A. Дерево та особлива вершина

Переформулюємо задачу: необхідно додати мінімальну кількість ребер до заданого дерева так, щоб в ньому не існувало мосту. Визначимо коренем дерева деяку вершину степені не менше двох (це неможливо зробити лише у випадку $n = 2$, який можна розглянути окремо). Випишемо всі листи у порядку Ейлерового обходу дерева (почнемо обхід з кореня).

Нехай виписані вершини утворили список $l_1, l_2, \ldots, l_k$. Відповідь не може бути меншою за $\lceil \frac{k}{2} \rceil$, бо інакше існував би лист степінь якого рівна 1 після додавання ребер до дерева, що в свою чергу означало б, що інцидентне йому ребро є мостом.

Стверджується, що після додавання ребер між парами вершин виду $(l_i, l_{i+\lfloor \frac{k}{2} \rfloor})$ для $1 \leqslant i \leqslant \lceil \frac{k}{2} \rceil$ у графі не залишиться мостів.

Доведення

Припустимо у графі залишився деякий міст. У порядку Ейлерового обходу кореневого дерева йому відповідає деякий підвідрізок листів $l_L, l_{L+1}, \ldots, l_R$, причому $L > 1$ або(та) $R < k$. У такому випадку не існує новоутвореного ребра, яке сполучає деякі l_u та l_v такі, що одне зі значень u та v належить відрізку $[L; R]$, а інше ні.

Покажемо, що таке ребро існує і отримаємо протиріччя. ¯_(ツ)_/¯

1. Якщо $R - L < \lfloor \frac{k}{2} \rfloor$ то хоча б одне з ребер $(l_L, l_{L+\lfloor \frac{k}{2} \rfloor}), (l_{R-\lfloor \frac{k}{2} \rfloor}, l_R)$ задовільняє описану вище умову.
2. Якщо $R - L \geqslant \lfloor \frac{k}{2} \rfloor$ то хоча б одне з ребер $(l_1, l_{1+\lfloor \frac{k}{2} \rfloor}), (l_{k-\lfloor \frac{k}{2} \rfloor}, l_k)$ задовільняє описану вище умову.

Складність: $O(n)$ часу та $O(n)$ пам'яті.

Задача B. Лотерея

Визначимо неорієнтований граф G на $n + 1$ вершинах $p_0, p_1, \ldots, p_n$, де підказка що характеризується числами l_j, r_j та коштує w_j монет відповідатиме ребру між вершинами $p_{l_j - 1}$ та p_{r_j} вагою w_j. Помітимо, що множина ребер такого графу відповідає множині підказок, причому за допомогою такої множини підказок можна однозначно зрозуміти кількість золота у кожній зі скринь тоді і тільки тоді, коли деяка підмножина цієї множини ребер є кістяковим деревом графу G.

Доведення цього факту наступне. Співставимо вершинам $p_0, p_1, \ldots, p_n$ деякі «префіксні суми» $pref_0, pref_1, \ldots, pref_n$, де $pref_i = \sum_{j=1}^{i} a_j$, a_j — кількість золота у скрині з номером j. Помітимо, що до купівлі підказок вам відомо лише значення $pref_0 = 0$, всі інші значення $pref$ невідомі. Підказка що характеризується числами l_j, r_j дає змогу створити лінійну залежність між значеннями $pref_{l_j - 1}$ та $pref_{r_j}$. Ви знатимете всі значення a тоді і тільки тоді, коли знатимете всі значення $pref$. З цього очевидно, що повинна існувати певна лінійна залежність між $pref_0$ та $pref_i$ для будь-якого $1 \leqslant i \leqslant n$. Іншими словами, у графі G повинен існувати шлях між вершинами p_0 та p_i для будь-якого $1 \leqslant i \leqslant n$ $\iff$ граф G повинен бути зв'язним.

Якщо граф G не є зв'язним то відповідь рівна -1.

Визначимо два матроїди M_1 та M_2 на множині ребер графу G. M_1 — матроїд, де множина ребер незалежна тоді і тільки тоді, коли для будь-якого i такого, що $1 \leqslant i \leqslant m$, виконується наступне: кількість ребер цієї множини що співставлені підказкам продавця з номером i не перевищує $c_i - k_i$. M_2 — матроїд, де множина ребер незалежна тоді і тільки тоді, коли граф побудований на ребрах з G що **не** належать цій множині є зв'язним. Якщо вважати, що ви **не** купите у продавців лише ті підказки, що співставлені певній множині ребер яка є незалежною у матроїдах M_1 та M_2, то куплені підказки задовільнятимуть наступним критеріям:

1. Для кожного i такого, що $1 \leqslant i \leqslant m$, ви купите не менше k_i підказок у продавця з номером i.
2. Множина підказок які ви купите дозволить вам дізнатися кількість золота у кожній зі скринь.

Знайдемо перетин матроїдів M_1 та M_2 максимальної сумарної ваги. Якщо його розмір менше ніж $\sum_{i=1}^{m}(c_i - k_i)$ то відповідь рівна -1, а інакше відповідь — сума ваг ребер що **не** ввійшли до перетину.

Складність: $O((\sum c)^4 + (\sum c)^2 \cdot n)$ часу та $O((\sum c)^2 + (\sum c) \cdot n)$ пам'яті.

Задача C. Множина

Спробуємо спочатку розв'язати задачу у випадку, коли є тільки запити першого типу «+ x w». Згадаємо, як можна підтримувати базис максимального розміру при додаванні нового числа x.

```
const int max_b = 30;

bool get_bit(int mask, int pos) {
    return (mask >> pos) & 1;
}

struct state {
    int val[max_b];

    void add(int x) {
        for (int i = 0; i < max_b; ++i) {
            if (get_bit(x, i)) {
                if (val[i]) {
                    x ^= val[i];
                } else {
                    val[i] = x;
                    break;
                }
            }
        }
    }
};
```

У цьому коді $val[i]$ позначає елемент базису, який відповідає за біт з номером i. Тому при додаванні нового елементу x необхідно розглянути усі його одиничні біти від молодшого до старшого, та для кожного з них зробити наступне:

1. якщо за черговий біт i не відповідає жоден елемент поточного базису, необхідно виставити $val[i] = x$. Це означає, що розмір базису збільшується на один, та тепер за біт з номером i відповідає число x;
2. якщо за черговий біт i вже відповідає значення $val[i]$, необхідно «занулити» відповідний біт числа x.

Подивимось, як можна модифікувати цей код для додавання взважених предметів. На лекції було доведено, що при додаванні нового елементу необхідно розглянути два випадки:

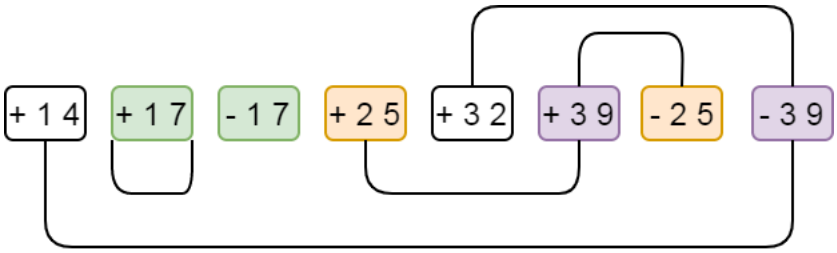
1. додавання нового елементу не утворює циклів з усіма раніше доданими елементами. У такому випадку необхідно просто додати новий елемент до відповіді;
2. додавання нового елементу утворює певний цикл з раніше доданими елементами. У такому випадку необхідно видалити елемент з мінімальною вагою на утвореному циклі.

```
struct state {
    int val[max_b], w[max_b];

    void add(int x, int cost) {
        for (int i = 0; i < max_b; ++i) {
            if (get_bit(x, i)) {
                if (val[i]) {
```

```
                    if (w[i] < cost) {
                        swap(val[i], x);
                        swap(w[i], cost);
                    }
                    x ^= val[i];
                } else {
                    val[i] = x;
                    w[i] = cost;
                    break;
                }
            }
        }
    }

    long long get_weight() const {
        return accumulate(w, w + max_b, 0LL);
    }
};
```

Відмінність від попереднього коду полягає в тому, що разом $val[i]$ зберігається також вага $w[i]$ елементу, що відповідає за біт з номером i. При розгляданні чергового одиничного біту i числа x необхідно також трохи модифікувати попередній код. Якщо за біт i відповідає деяке число $val[i]$, це означає, що елемент $val[i]$ знаходиться на одному циклі з числом x. Тому у випадку, коли вага $w[i]$ менше за вагу нового предмету, необхідно просто обміняти ці елементи місцями.

Ми навчилися додавати взважені елементи до базису, підтримуючи максимальну сумарну вагу доданих елементів. Тепер подивимось, як можна опрацьовувати операції видалення. Спробуємо отримати множину усіх елементів, що були наявні в множині в кожен момент часу. Для кожної операції видалення «`- x w`» знайдемо відповідну їй операцію додавання та скажемо, що елемент (x, w) був наявним у множині у певний проміжок часу $[l; r)$, де l — номер запиту додавання «`+ x w`», а r — видалення (див. малюнок). Якщо елемент так і не був видалений, то можна вважати, що $r = n + 1$.



Побудуємо на інтервалі часу від 1 до n дерево відрізків. Для кожної операції додавання `+ x w` додамо елемент (x, w) в $O(\log n)$ вершин дерева відрізків, що відповідають розбиттю відповідного інтервалу $[l; r)$. Тепер для того, щоб дізнатися відповідь після i-го запиту, необхідно дізнатися базис з максимальною вагою серед предметів, що знаходяться на шляху від кореня дерева відрізків, до листа з номером i. Саме тому можна просто зробити звичайний обхід `dfs` дерева відрізків, підтримуючи глобальний базис доданих елементів. При обробці чергової вершини v необхідно спочатку запам'ятати поточний стан глобального базису (скопіювати поточне значення `state`), додати до глобального базису усі елементи, записані в вершині v, рекурсивно обійти ліве та праве піддерево, та повернути поточний стан базису до запам'ятованого.

Складність: $O(n \log^2 n)$ часу та $O(n \log n)$ пам'яті.

Задача D. Гра на матроїді

Помітимо що для відомого кольорового матроїду M легко визначити кольори елементів з X. Зробити це можна перевіривши для кожної пари елементів чи їх кольори рівні (чи множина побудована з цих елементів є залежною) та побудувавши компоненти зв'язності на такому графі-відношенні.

Для кожного кольору визначимо mn та mx — мінімальну та максимальну вагу предмету такого кольору відповідно. Помітимо що відповідь буде рівною $\sum_c mx(c)$ - $(\sum_{t \in T} mx(t) - mn(t))$, де T — множина кольорів, що елемент такого кольору додасть до множини S другий гравець. З очевидних жадібних міркувань: якщо відсортувати множину кольорів C по незростанню значення $mx(c) - mn(c)$ то T буде множиною всіх кольорів на парних позиціях в C.

Складність: $O(2^n + n^2 + m \cdot n \cdot \log(n))$ часу та $O(2^n + n^2)$ пам'яті.

Задача E. Складні задачі

На перший погляд цю задачу можна розв'язати за допомогою решета Ератосфена: наприклад, можна згенерувати відповіді для $n = k \cdot 10^8$, а для інших n використати блочне решето Ератосфена та значення пораховані заздалегіть. Саме для мотивації учасників реалізувати таке рішення, у задачі використано обмеження $1 \leqslant n \leqslant 0.75 \cdot 10^{10}$.

Рішення

Відомо, що для будь-якого простого $p > 3$ виконується $p = (6 \cdot n - 1)$ або $p = (6 \cdot n + 1)$ для деякого цілого n. Тоді $(p^2 - 1) = 36 \cdot n^2 \pm 12 \cdot n \equiv 12 \cdot n \cdot (n \pm 1) \pmod{24}$. Оскільки рівно одне з значень n та $(n \pm 1)$ парне, то при $p > 3$, $(p^2 - 1) \equiv 0 \pmod{24}$, а тому існує лише дві складні задачі — з номерами 2 та 3. Відповідь рівна $[n \geqslant 2] + [n \geqslant 3]$.

Складність: $O(1)$ часу та $O(1)$ пам'яті.

Задача F. Відгадай матроїд

Нескладно помітити, що всі базиси M є множинами однакового розміру. Нехай базиси M мають розмір k. Знайти значення k можна додаючи до початково порожньої множини предмети послідовно так, щоб ця множина залишалась незалежною (аналогічно алгоритму знаходження базису).

Також помітимо, що множина всіх базисів M однозначно задає I. Щоб дізнатись множину всіх базисів задамо питання для всіх підмножин $A \subset X$ таких, що $|A| = k$: ті з них які є незалежними і будуть базисами M.

Складність: $O(n + C_n^k)$ питань.

Задача G. Остовні дерева

Побудуємо граф з $n \cdot k$ вершин та $m \cdot k$ ребер, де підграф на вершинах $[k \cdot n + 1; (k + 1) \cdot n]$, $k \geqslant 1$ є копією підграфу на вершинах $[1; n]$ (кожне ребро початкового графу продубльовано k разів, по одному разу для кожної копії). Також визначимо для кожного ребра колір таким чином, щоб два ребра мали однаковий колір тоді і тільки тоді, коли вони є копіями одного і того самого початкового ребра.

Визначимо два матроїди M_1 та M_2 на множині ребер новоутвореного графу. M_1 — матроїд, де множина ребер незалежна тоді і тільки тоді, коли граф побудований на ребрах цієї множини не має циклів. M_2 — матроїд, де множина ребер незалежна тоді і тільки тоді, коли кольори ребер цієї множини різні.

Знайдемо перетин матроїдів M_1 та M_2. Якщо його розмір менше ніж $(n-1) \cdot k$ то відповіді не існує, а інакше i-а група ребер відповіді — множина ребер перетину що належать i-ій копії початкового графу.

Для ефективної реалізації знаходження перетину матроїдів журі використало наступну техніку: щоб ефективно визначати чи після видалення ребра з перетину та додавання нового до перетину у графі не утвориться циклів побудуємо компоненти зв'язності для всіх графів, що можуть утворитися після видалення одного з ребер графу.

Складність: $O(m^3 \cdot k + m^2 \cdot n \cdot k)$ часу та $O(m^2 \cdot k)$ пам'яті.

Задача H. Гра з одним переможцем

Лема 1. Множина ребер формує набір реберно-простих циклів, що не перетинаються тоді і тільки тоді, коли степені всіх вершин графу побудованого на цих ребрах є парними.

Лема 2. У грі «Нім» другий гравець перемагає тоді і тільки тоді, коли `xor` розмірів купок дорівнює нулю.

Для кожного ребра визначимо бітову маску з $n+60$ елементів: перша частина маски довжиною n міститиме рівно 2 одиничних біти у позиціях u_i та v_i, а друга частина маски довжиною 60 міститиме бітове представлення числа a_i. Використовуючи описані леми та побудовані бітові маски бачимо, що множина S_1 буде «виграшною» для першого гравця тоді і тільки тоді, коли відповідні бітові маски є лінійно незалежними у полі $\mathbb{Z}_2$.

Використовуючи алгоритм Радо-Едмондса будуватимемо відповідь додаючи ребра у порядку незростання ваги так, щоб описана множина бітових масок залишалась лінійно незалежною у полі $\mathbb{Z}_2$.

Складність: $O(m \cdot \log(m) + m \cdot (n + \log(a))^2/W)$ часу та $O(m + (n + \log(a))^2)$ пам'яті, де W — довжина машинного слова.

Задача I. Матроїд?

Перевіримо правильність кожної аксіоми незалежно:

1. Просто подивимось, чи виконується рівність $c_0 = 1$. Якщо це не так, одразу виведемо «`No 1`». Це можна зробити за $O(1)$.

2. Переберемо підмножину B та перевіримо, що друга аксіома виконується для усіх $A \subset B$ таких, що A відрізняється від B рівно одним предметом (іншими словами, відповідна маска A відрізняється від відповідної маски B рівно в одній позиції). Це можна зробити за $O(2^n n)$.

   Доведемо, що якщо ми не знайшли жодного протиріччя, то друга аксіома виконується для кожної пари $A \subset B$. Припустимо, що це не так, тобто існує пара $A \subset B$, для якої $c_a = 0$ та $c_b = 1$. Розглянемо одну таку пару з мінімальною різницею $|B| - |A|$. Помітимо, що маска a відрізняєьтся від маски b хоча б у двох позиціях, оскільки в протилежному випадку на попередньому кроці ми би вивели `No 2`. Тоді ми можемо завжди знайти таку множину C, що $A \subset C \subset B$ та $|A| + 1 = |C|$. Множина C буде незалежною, оскільки в протилежному випадку $|B| - |C| < |B| - |A|$. Але оскільки $A \subset C$, та A відрізняється від C рівно в одній позиції, то ми повинні були вивести «`No 2`» на попередньому кроці алгоритму.

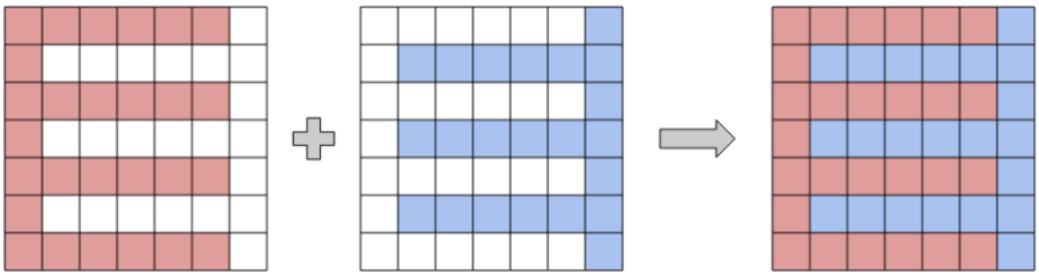
3. Будемо перебирати усі підмножини (маски) A в порядку від найбільшої за розміром до найменшої. Для фіксованої маски a побудуємо за $O(n)$ маску add, у якій одиничні біти записані в таких позиціях i, що можна додати до множини A без втрати незалежності. Тоді для усіх більших за розміром масок b повинна виконуватись умова add `AND` $b \neq 0$. Можна помітити, що ця умова еквівалентна умові «b не є підмаскою $\overline{add}$», де $\overline{mask}$ позначає заміну кожного біту маски на протилежне значення.

   Тому необхідно навчитися швидко виконувати наступні операції: «додати в множину маску $mask$», «перевірити, чи присутня в множині деяка підмаска маски $mask$». Будемо підтримувати масив $f[mask]$, де $f[mask]$ позначає відповідь на запитання другого типу для поточної множини масок. При додаванні чергової маски $mask$, необхідно виставити значення $f[m] = 1$, де $mask \subset m$. Це можна зробити рекурсивном шляхом за допомогою звичайного dfs, якщо розглядати тільки такі m, що відрізняються від $mask$ тільки в одному біті. Оскільки для кожної маски m процедура dfs буде викликане лише один раз, загальна складність перевірки третьої аксіоми — $O(2^n n)$.

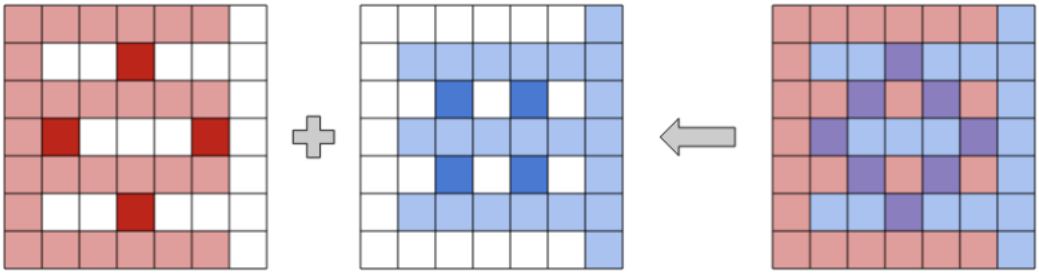
Складність: $O(2^n n)$ часу та $O(2^n)$ пам'яті.

Задача J. Перетин мартоїдів

Побудуємо відповідь конструктивно. Початково визначимо мартоїди так, як показано на рисунку.



Помітимо, що при записі одиниці до будь-якої не крайньої клітинки будь-якого з цих мартоїдів вони залишатимуться коректними. Тому потрібно лише записати одиниці у клітинки цих мартоїдів, відповідні елементи матриці C яких є одиницями. Приклад показано на рисунку.



Складність: $O(n \cdot m)$ часу та $O(n \cdot m)$ пам'яті.

Задача K. Унікальні суми

Нехай mn та mx — мінімальне та максимальне значення суми на непорожньому підвідрізку масиву a. Тоді досягається будь-яке значення з відрізку $[mn; mx]$ (відповідь рівна $mx - mn + 1$).

Доведення

Побудуємо послідовність підвідрізків $p_1, p_2, \ldots, p_k$ таку, що $val(p_1) = mn$, $val(p_k) = mx$ та $|p_i.l - p_{i+1}.l| + |p_i.r - p_{i+1}.r| = 1$ для всіх $1 \leqslant i < k$, де $val(p) = \sum_{i=p.l}^{p.r} a_i$. Нехай p_1 — підвідрізок з a на якому досягається мінімальна сума, деяке p_m $(1 \leqslant m \leqslant k)$ — підвідрізок $[1; n]$, а p_k — підвідрізок з a на якому досягається максимальна сума. Побудувати всю послідовність p можна наступним чином: для $1 \leqslant i < m$ p_{i+1} є розширенням p_i, а для $m \leqslant i < k$ p_{i+1} є звуженням p_i. Помітимо, що для послідовності $v_i = val(p_i)$ виконується $v_1 = mn$, $v_k = mx$ та $|v_i - v_{i+1}| \leqslant 1$ для всіх $1 \leqslant i < k$.

З останнього легко бачити факт що доводиться.

Складність: $O(n)$ часу та $O(n)$ пам'яті.

Задача L. Якщо вам стало нудно

Нехай $n = R - L + 1$. Для початку опишемо рішення зі складністю $O(n^3)$, що використовує динамічне програмування. ДП матиме $O(n^2)$ станів:

- $L(x, r)$: на дисплеї відображається число x, а шукане число знаходиться на відрізку $[x + 1; r]$,
- $R(l, x)$: на дисплеї відображається число x, а шукане число знаходиться на відрізку $[l; x - 1]$.

Прискоримо таке рішення. Помітимо, що відповідь ніколи не перевищує $M = 50$ (за 50 операцій завжди можна відгадати число x, використовуючи бінарний пошук спочатку по першій, другій, ..., п'ятій цифрах). Визначимо нові стани ДП, їх кількість буде $O(n \cdot M)$:

- $range\_left(c, x) = \min\{l : R(l, x) \leqslant c\}$,
- $range\_right(c, x) = \max\{r : L(x, r) \leqslant c\}$.

Нехай $dist(a, b)$ — кількість цифр, яку необхідно змінити, щоб перетворити число a в b.

Порахуємо значення станів $range\_right(c, \star)$ для заданого c, вважаючи що для всіх менших c ці значення вже пораховано. Помітимо, що $range\_right(c, x) \geqslant y$ тоді, коли існує деяке m таке, що $range\_left(c - dist(x, m) - 1, m) \leqslant x + 1$ та $range\_right(c - dist(x, m) - 1, m) \geqslant y$. Тут m — оптимальне значення що відображатиметься на дисплеї на при наступній операції другого типу.

При переході від m до x використаємо наступний трюк: змінимо деякі цифри m на «?» і вважатимемо, що «?» еквівалентен виконанню однієї операції першого типу; змінимо деякі цифри x також на «?». Тепер перехід можна виконувати тоді, коли $m = x$. Наприклад $m \to x$: $31337 \to 3??3? \to 35932$. Значення з «?» називатимемо «макетом».

Підрахуємо значення $range\_right(c, x)$ у порядку збільшення значення x. Будемо підтримувати масив D розміру 11^5, який співставляє кожному можливому макету певне ціле значення (початково рівне $-\infty$).

Щоб порахувати значення для деякого x необхідно наступне:

- Для кожного можливого значення $d \in [1; 5]$ і кожного можливого $m \in [1; 10^5)$ такого, що $range\_left(c - d - 1, m) = x + 1$ розглянемо всі C_5^d способів змінити d цифр m щоб утворити макет. Для кожного макету w виконаємо $D(w) := max(D(w), range\_right(c - d - 1, m))$.
- Неважко побачити, що тепер $range\_right(c, x)$ дорівнює максимальному значенню $D(w)$, де макет w підходить під x.

Простий спосіб реалізувати це такий: посортуємо пари (d, m) по значенню $range\_left(c-d-1, m)$ і виконаємо описані вище дії для кожного x. Така реалізація дозволяє підрахувати значення ДП для кожного c за $O(5 \cdot 10^5 \cdot \log(5 \cdot 10^5) + 10^5 \cdot 2^5 + 11^5)$. Значення $range\_left(c, \star)$ можна порахувати аналогічно. Як вже було згадано вище, достатньо порахувати значення ДП лише для $c \leqslant M = 50$.