Задача 1A. Сім'я

Автор задачі: **Меліса Галіба**
Задачу підготував: **Данило Квіт**
Розбір написав: **Костянтин Денисов**

Кількість братів Ксенії і є кількістю братів в сім'ї, при цьому кількість братів Федора на один менше за цю кількість. Тобто братів у сім'ї є $\max(b_1, b_2)$. Аналогічні міркування можна застосувати щодо кількості сестер.

Тому загалом є $\max(b_1, b_2)$ братів і $\max(s_1, s_2)$ сестер.

Задача 1B. A+B=C

Автор задачі: **Костянтин Денисов**
Задачу підготував: **Данило Квіт**
Розбір написав: **Данило Квіт**

Неважко зрозуміти, що вираз $a_1 + b_1 = c$ можна перетворити у вираз $a_2 + b_2 = c$ тільки якщо сума кількості сірників, що використовуються у a_1 та b_1, дорівнює кількості сірників, що використовуються у a_2 та b_2. Тому достатньо перебрати всі можливі комбінації a_2 та b_2 (яких є $10 \cdot 10$) та перевірити, чи хоча б якась з них одночасно задовольняє 2 умови:

1. вираз $a_2 + b_2 = c$ є правильним;
2. сума кількості сірників, що використовуються у a_1 та b_1, дорівнює кількості сірників, що використовуються у a_2 та b_2.

Кількості сірників, що використовуються для кожного числа при цьому можна зберегти у пам'яті як масив.

Задача 1C. Площа торта

Автор задачі: **Микола Арзубов**
Задачу підготував: **Данило Квіт**
Розбір написав: **Костянтин Денисов**

Нехай a та b — це сторони прямокутника, які ми шукаємо. Очевидно, що виконується рівність $a \cdot b = P + Q$. Це означає, що a і b є дільниками числа $x := P + Q$. Усі дільники числа x можна знайти за $O(\sqrt{x})$ операцій.

Розглянемо кожен дільник d числа x. Для нього перевіряємо, чи підходить прямокутник зі сторонами d та $\frac{x}{d}$. Тобто, якщо двоє людей почнуть відрізати від такого прямокутника найбільші можливі квадрати, ми можемо обчислити площі цих квадратів і порівняти їх із P та Q.

Щоб обчислювати площі швидко, зауважимо, що прямолінійне віднімання меншої сторони c від більшої d може бути дуже повільним. Наприклад, якщо $c = 1$ і $d = 10^{12}$, знадобиться 10^{12} операцій.

Однак можна скористатися ефективнішим підходом. Помітимо, що менша сторона c змінюється лише тоді, коли більша сторона d зменшується до $d \mod c$. За один такий перехід виконується $\lfloor \frac{d}{c} \rfloor$ операцій. Цей підхід аналогічний алгоритму Евкліда для знаходження найбільшого спільного дільника (НСД) двох чисел, який працює за $O(\log \min(a, b))$.

Таким чином, загальна складність алгоритму становить $O(\sqrt{P+Q} \cdot \log(P+Q))$. На практиці алгоритм працює ще швидше, оскільки кількість дільників числа значно менша за $\sqrt{P+Q}$.

Задача 1D. Цифрова гра

Автор задачі: **Костянтин Денисов**
Задачу підготував: **Данило Квіт**
Розбір написав: **Костянтин Денисов**

Твердження. Позначимо $\text{cnt}(l, r)$ — кількість різних цифр на відрізку $[l, r]$ рядка s. Якщо існує відрізок $[l, r]$ для якого виконується нерівність $\text{cnt}(l, r) \cdot 2 \leqslant r - l + 1$, то гарантовано у грі виграє другий гравець.

Доведення.

Доведемо це твердження методом математичної індукції за довжиною відрізка $[l, r]$.

База індукції. Розглянемо випадок, коли $2 \leqslant r - l + 1 \leqslant 3$. У цьому випадку всі цифри на відрізку $s[l..r]$ однакові (тобто $s_l = s_{l+1}$), а отже, другий гравець завжди перемагає.

Індуктивний перехід. Припустимо, що твердження справедливе для всіх відрізків довжини менше ніж $r - l + 1$. Доведемо його для відрізка $[l, r]$, де $\text{cnt}(l, r) \cdot 2 \leqslant r - l + 1$.

Розглянемо величину

$$d = r - l + 1 - 2 \cdot \text{cnt}(l, r).$$

Тепер обмежимо гру лише цим відрізком і припустимо, що перший гравець може робити ходи поза відрізком, але це не вплине на нашу стратегію.

Як змінюється d за один хід?

- Довжина $r-l+1$ може зменшитися на 1 (або не змінитися, якщо хід зроблено поза відрізком).
- $\text{cnt}(l, r)$ або залишається незмінною, або зменшується на 1 (якщо видалено цифру, яка зустрічалася лише раз).

Отже, d у найгіршому випадку зменшується на 1 за хід.

Тепер розглянемо три випадки залежно від значення d **після ходу першого гравця**:

1. $d \geqslant 1$**.** Після будь-якого ходу другого гравця $d \geqslant 0$;
2. $d = -1$**.** У цьому випадку на відрізку обов'язково існує цифра, яка зустрічається рівно один раз (інакше $d \geqslant 0$). Другий гравець може видалити цю цифру, і d стане рівним 0;
3. $d = 0$**.** Якщо є цифра, яка зустрічається лише раз, другий гравець також видалить її і перейде до виграшного стану. Інакше усі цифри зустрічаються рівно 2 рази. Тоді розглянемо підвідрізок $[l + 1, r]$. На цьому підвідрізку величина d дорівнює -1, і обов'язково існує цифра, що зустрічається один раз. Другий гравець видаляє її та переходить у виграшний стан.

В усіх випадках кількість цифр на відрізку зменшується, а припущення індукції дозволяє завершити гру на користь другого гравця. □

Рядок складається лише з цифр, тому $\text{cnt}(l, r) \leqslant 10$. Тоді, якщо $|s| \geqslant 20$, то з нашого твердження маємо, що гарантовано виграє другий. Тепер треба визначити, хто виграє для $|s| < 20$.

Для коротших рядків ($|s| < 20$) можна скористатися рекурсією з мемоізацією:

- Кожен стан гри описується бітовою маскою $mask$, де $mask_i = 1$, якщо i-ту цифру ще не видалено, і $mask_i = 0$, якщо видалено;
- Усього можливих станів — $2^{|s|}$;
- Для кожного стану перебираємо можливі ходи і рекурсивно визначаємо переможця.

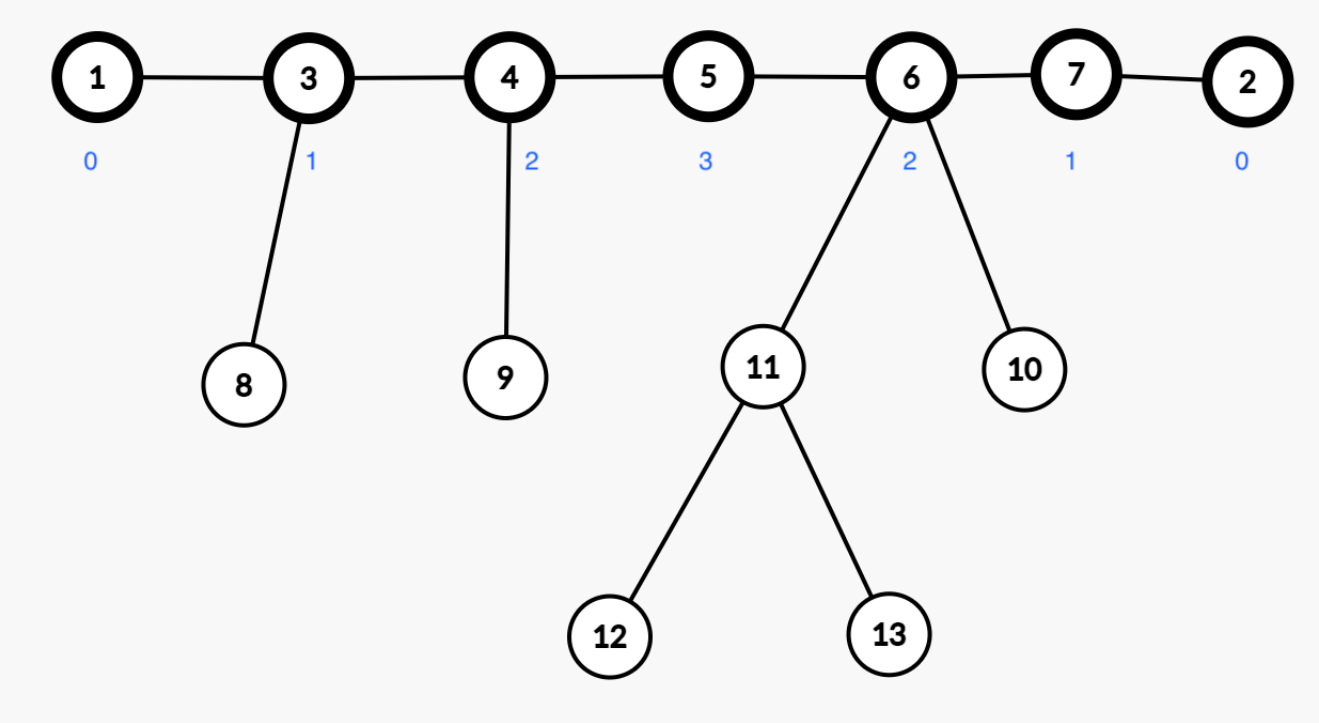
Складність алгоритму $O(2^{|s|} \cdot |s|)$.

Задача 1E. OldPost — NewPost

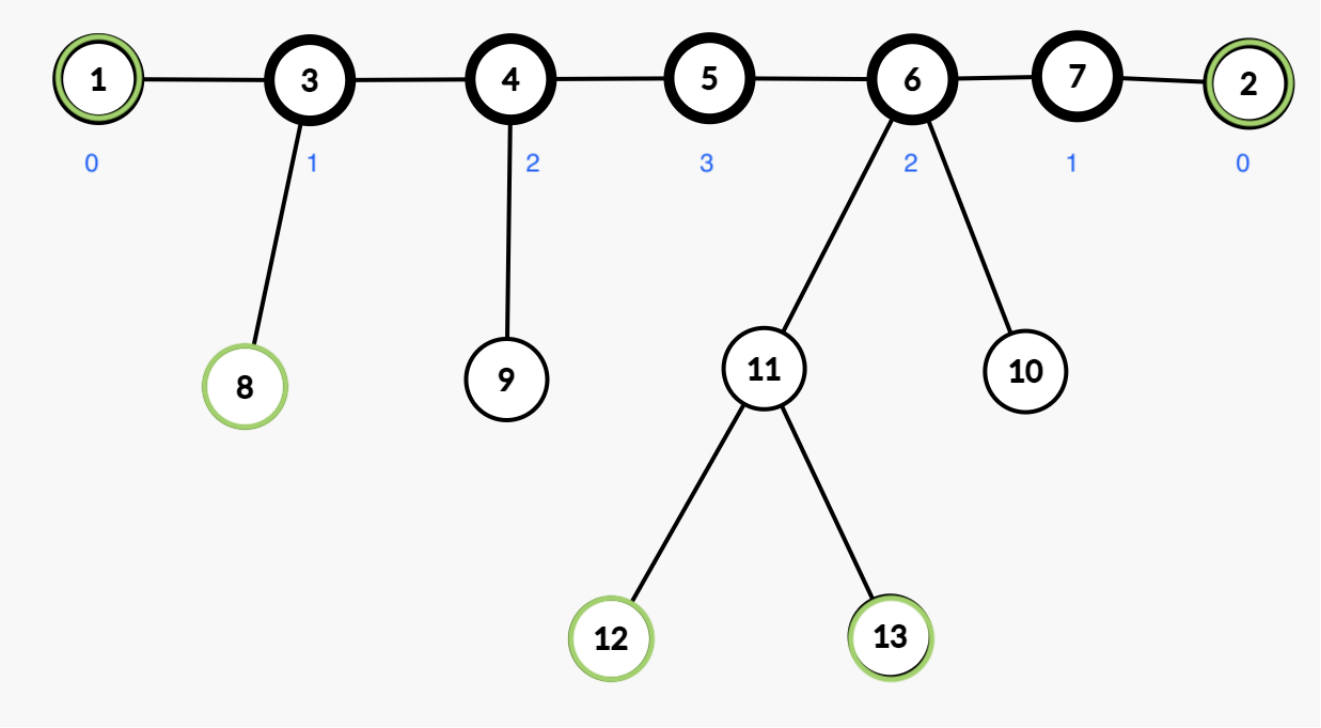
Автор задачі: **Костянтин Денисов**
Задачу підготував: **Данило Квіт**
Розбір написав: **Костянтин Денисов**

Формально кажучи, в задачі треба було знайти два діаметри на дереві, що перетинаються по найменшій кількості вершин.

Розберемося, як можна просто працювати з діаметрами на дереві. Не порушуючи загальності, припустимо, що шлях між вершинами 1 та 2 — діаметр. Зобразимо наше дерево так, ніби воно "підвішене"за цей шлях. Як тоді можна представити інші діаметри?



Для кожної вершини v діаметра подивимось на піддерева сусідніх вершин, що не належать діаметру. Випишемо, яка максимальна можлива відстань від вершини v до вершин в цих піддеревах. Якщо $v_1, v_2, \ldots, v_l$ — вершини діаметра в порядку обходу, то максимальна відстань в цих піддеревах для вершини v_i буде $\min(i-1, n-i)$. Позначимо зеленим вершини в піддеревах, на яких відповідний максимум досягається.



Твердження 1. Розділимо наші вершини на три частини (у випадку діаметра парної довжини на дві частини). Групи:

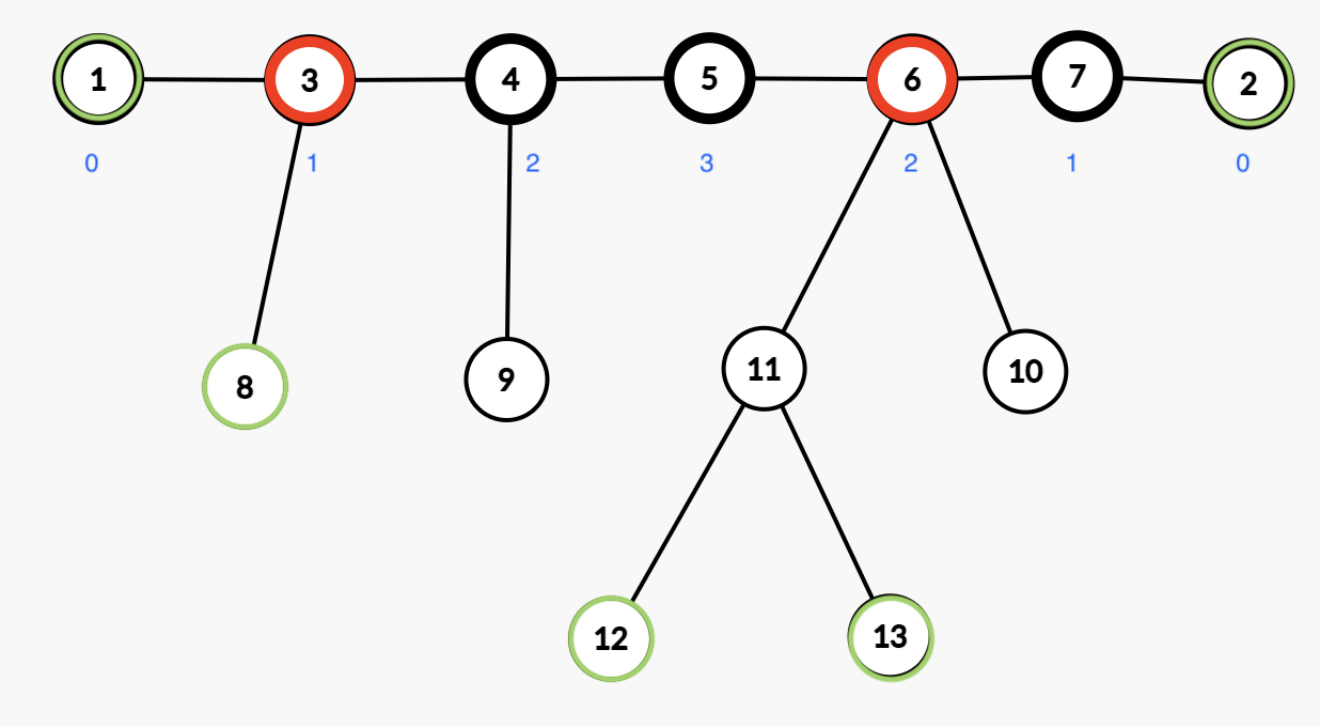
1. Вершини, що знаходяться у піддеревах вершин v_1, v_2, ..., $v_{\lfloor \frac{l}{2} \rfloor}$;
2. Вершини, що знаходяться у піддеревах вершин $v_{\lfloor \frac{l+1}{2} \rfloor+1}$, $v_{\lfloor \frac{l+1}{2} \rfloor+2}$, ..., v_l;
3. Вершини, що знаходяться у піддеревах вершини $v_{\lfloor \frac{l+1}{2} \rfloor}$ (лише у випадку непарного l).

Для будь-якого діаметра (нехай a та b його кінцеві вершини) виконуються наступні твердження:

- a, b — зелені вершини;
- хоча б одне з наступних тверджень:
  - a — у першій групі, b — у другій групі (або навпаки);
  - a — у першій або другій групі, b — у третій (або навпаки);
  - a, b — у третій групі, але у різних піддеревах відносно вершини $v_{\lfloor \frac{l+1}{2} \rfloor}$;

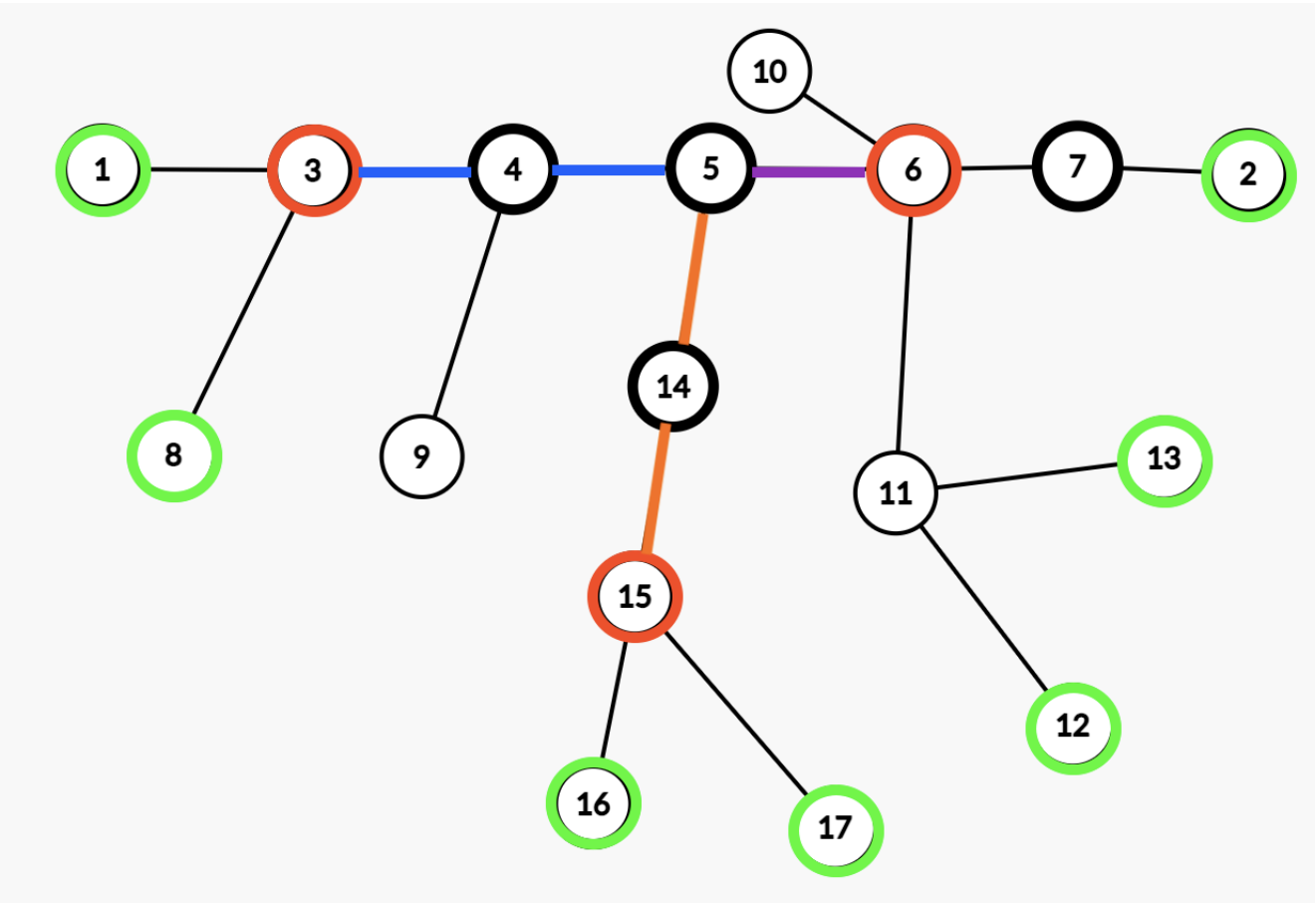
Твердження 2. Якщо є дві зелені вершини a, b третьої групи, що знаходяться в різних піддеревах відносно вершини $v_{\lfloor \frac{l+1}{2} \rfloor}$, то можна зробити перетин діаметрів рівний 1, взявши діаметри 1 — 2 та a — b.

Твердження 3. Нехай немає зелених вершин третьої групи. Для вершин першої групи знайдемо вершину **діаметра** v_{right} першої групи, що в її піддереві є зелена вершина a, та v_{right} якомога ближче до середини діаметра. Аналогічно треба знайти вершину діаметра v_{left} другої групи, що в її піддереві є зелена вершина b та v_{left} якомога ближче до середини діаметра. Тоді вигідно взяти такі діаметри 1 — 2 та a — b, бо в цьому випадку діаметри завжди будуть перетинатися по вершинах $v_{right}, v_{right+1}, \ldots, v_{left}$.



На малюнку червоним зображено вершини v_{right} та v_{left}.

Твердження 4. Нехай є зелена вершина a третьої групи й немає таких двох вершин з твердження 2. Тоді всі такі зелені вершини третьої групи розташовані в одному піддереві відносно вершини $v_{\lfloor \frac{l+1}{2} \rfloor}$. У цьому випадку нам, крім знаходження v_{right} та v_{left}, треба знайти v_{down}, вершину, що в її піддереві є дві зелені вершини в різних піддеревах відносно неї (або $v_{down} = a$, якщо такої не існує), що якомога ближче до вершини $v_{\lfloor \frac{l+1}{2} \rfloor}$.



На малюнку червоним зображено вершини v_{right}, v_{down} та v_{left}. В цьому випадку ми не можемо зробити перетин краще ніж

$$\min(d(v_{right}, v_{\lfloor \frac{l+1}{2} \rfloor}), d(v_{left}, v_{\lfloor \frac{l+1}{2} \rfloor}), d(v_{down}, v_{\lfloor \frac{l+1}{2} \rfloor})),$$

де $d(u, v)$ — кількість вершин на шляху між вершинами u та v.

Наприклад, якщо взяти діаметри $1 - 2$ та $2 - a$, то перетин буде

$$v_{\lfloor \frac{l+1}{2} \rfloor}, v_{\lfloor \frac{l+1}{2} \rfloor + 1}, \ldots, v_{left}.$$

Тоді розв'язання задачі має наступний вигляд:

1. Знаходимо будь-який діаметр. Це можна зробити загальновідомим алгоритмом з двома dfs-ами просто знайшов найдальшу вершину v від вершини 1, а потім аналогічно знайти найдальшу вершину u від v. Можна показати, що $v - u$ — діаметр;
2. Розглядаємо випадки в залежності в яке твердження попадаємо.

Складність алгоритму $O(n)$.

Задача 2A. Фільми

Автор задачі: **Костянтин Денисов**
Задачу підготував: **Павло Цікалишин**
Розбір написав: **Костянтин Денисов**

Фільми розподіляються по сторінках рівномірно:

- Перші $\lfloor n/k \rfloor$ сторінок містять рівно k фільмів.
- Остання сторінка може містити або k фільмів (якщо n ділиться на k), або меншу кількість $n \bmod k$.

Отже, кількість фільмів на останній сторінці:

$$\texttt{last\_page\_count} = k, \texttt{якщо} \quad n \quad \texttt{кратне} \quad k;$$

$$\texttt{last\_page\_count} = n \mod k, \texttt{інакше}.$$

Тоді час перегляду:

$$\texttt{time} = \texttt{last\_page\_count} \cdot c$$

Задача 2B. Катамарани

Автор задачі: **Микола Арзубов**
Задачу підготував: **Павло Цікалишин**
Розбір написав: **Костянтин Денисов**

Поділимо всі ваги на 20. Тоді у нас є ваги 1, 2, 3, 4, 5, а максимальна вага, яку витримує катамаран, — 5.

Для зручного зберігання кількості людей із кожною вагою можна використовувати масив $cnt[1..5]$.

Для розв'язання задачі доцільно використовувати жадібний підхід: спершу розміщувати людей із більшою вагою, намагаючись максимально ефективно заповнювати катамарани.

1. Для кожної людини з вагою 5 потрібно виділити окремий катамаран. Після цього залишаються люди з вагами 1, 2, 3, 4.

2. Людину з вагою 4 можна об'єднати з людиною вагою 1, якщо такі є. Виділяємо катамарани для груп $4 + 1$, поки є обидва типи людей. Якщо людей вагою 1 не вистачає, групуємо лише 4. Після цього залишаються люди з вагами 1, 2, 3.

3. Людину з вагою 3 можна об'єднати або з людиною вагою 2, або з двома людьми вагою 1. Вигідніше спершу об'єднувати з 2, оскільки з двох одиничок можна утворити 2, але навпаки — ні. Тому формуємо групи $3 + 2$, а коли людей вагою 2 не залишиться, групуємо $3 + 1 + 1$ (або менше одиничок, якщо їх не вистачає). Після цього залишаються люди з вагами 1 та 2.

4. Далі компонування відбувається у такій пріоритетності:

   - $2 + 2 + 1$ (оскільки дозволяє ефективно використовувати вагу);
   - $2 + 2$ (якщо одиничок немає);
   - $2 + 1 + 1 + 1$ (щоб заповнити катамаран повністю);
   - $2 + 1 + 1$, $2 + 1$, 2 (у міру зменшення можливих комбінацій).

5. Коли залишаються лише люди з вагою 1, їх групуємо по п'ятеро в один катамаран. Якщо кількість не ділиться на 5, додаємо ще один катамаран для решти.

Такий підхід гарантує мінімальну кількість катамаранів для перевезення всієї групи.
Складність алгоритму: $O(n)$ (через зчитування).

Задача 2C. Діагональні числа

Автор задачі: **Йосиф Ентін**
Задачу підготував: **Павло Цікалишин**
Розбір написав: **Костянтин Денисов**

Оптимальний порядок перестановки кубиків можна описати наступним чином:

1. Спочатку правильно складаємо останній стовпчик за $n + 2$ операції:
   (a) Перемістимо кубик 1 з першого стовпчика на другий;
   (b) Перемістимо кубик n з останнього стовпчика на перший;
   (c) Далі вже без перешкод за n операцій можна буде скласти останній стовпчик, просто переміщаючи кубики 1, 2, ..., n.
2. Далі будемо продовжувати складати стовпчики починаючи з кінця, але тепер на складання стовпчика i витратимо $i + 1$ операцію, а не $i + 2$, як для останнього стовпчика;
3. Для складання стовпчика i будемо діяти таким чином:
   (a) Перемістимо єдиний кубик i зі стовпчика i на стовпчик $i + 1$ (в попередньому випадку не було такого стовпчика);
   (b) Далі за i операцій можна буде скласти стовпчик, переміщаючи кубики 1, 2, ..., i.
4. Після правильного складання стовпчиків з 2 по n перший стовпчик автоматично буде складений, тому додаткові операції на нього не знадобляться.

Тоді сумарно витратимо $(n + 2) + n + (n - 1) + \ldots + 3 = n + 2 + \frac{n \cdot (n+1)}{2} - 3 = \frac{n^2+3 \cdot n-2}{2}$ операцій.

Задача 2D. Непарні рядки

Автор задачі: **Костянтин Денисов**
Задачу підготував: **Павло Цікалишин**
Розбір написав: **Костянтин Денисов**

Нехай ми розставили одинички в перших i стовпчиках. Яку інформацію нам варто зберегти, щоб ефективно працювати з таблицею і надалі розставляти наступні стовпці? Всю матрицю зберігати це досить багато інформації. Можна помітити, що єдине, що нам потрібно знати про перші i стовпчиків — це скільки поточно є рядків з непарною кількістю 1. Тоді якщо маємо cnt рядків з непарною кількістю одиниць, то матимемо $n - cnt$ з парною кількістю. Далі коли будемо ставити новий стовпчик, то будемо намагатися перераховувати цю кількість "непарних"рядків (це ті рядки в яких непарна кількість одиничок).

Тому виникає ідея писати динаміку $dp[i][cnt] = true/false$. Стан $dp[i][cnt] = true$ тоді та тільки тоді, коли можна поставити одинички у перших i стовпчиках, так що кількість непарних рядків у них дорівнює cnt. База динаміки $dp[0][0] = true$.

Як робити переходи? Нехай ми маємо стан $dp[i][cnt] = true$ та у $i+1$ стовпчику в нас x одиничок. Розглянемо два випадки:

1. $x + cnt \leqslant n$.

   В цьому випадку максимальна кількість одиничок після того, як ми встановимо $i+1$ стовпчик буде $x + cnt$, оскільки ми можемо поставити всі наші x в "парні"рядки. Тобто в цьому випадку **перетин** поточних непарних рядків з новими одиничками буде порожнім.

   А який максимальний перетин може бути? Виявляється, $\min(x, cnt)$. Нескладно зрозуміти, що ми можемо зробити всі перетини з відрізка $[0, \min(x, cnt)]$. Тоді можливі кількості непарних рядків після встановлення стовпчика $i + 1$ мають вигляд $x + cnt - 2 \cdot y$, де y — ціле число з проміжку $[0, \min(x, cnt)]$.

2. $x + cnt > n$.

   В цьому випадку ми вже не можемо зробити порожній перетин між поточними "непарними"рядками та одиничками у новому стовпчику. Який тоді можливий мінімальний перетин? Нескладно зрозуміти, що $x + cnt - n$. Коли ми розставимо $n - cnt$ одиничок у $i + 1$ стовпчику у "парні"рядки, то всі інші прийдеться розставляти у непарні.

   Максимальний перетин залишається $\min(x, cnt)$. Тоді можливі кількості непарних рядків після встановлення стовпчика $i + 1$ мають вигляд $x + cnt - 2 \cdot y$, де y — ціле число з проміжку $[x + cnt - n, \min(x, cnt)]$.

Помітимо, що наші переходи зі стану $dp[i][cnt]$ відбувається у якийсь відрізок станів $[l, r]$ у $dp[i+1]$, але по парності (тобто переходимо у $l, l + 2, l + 4, \ldots$).

В лоб робити такі переходи займає $O(n^2 \cdot m)$ операцій, тому є потреба у оптимізаціях. А саме помітимо, що для фіксовано i всі стани, де $dp[i][cnt] = true$ утворюють неперервний відрізок $[l, r]$ по парності (тобто $dp[i][l] = true, dp[i][l + 2] = true, \ldots$).

Тоді виникає ідея зберігати масиви $l[i]$ та $r[i]$. Коли переходимо до $i + 1$ рядка з i-ого, то проходимося по всім станам cnt з відрізка $[l[i], r[i]]$ по парності. Для кожного cnt рахуємо відрізок $[a, b]$ в яких переходимо і визначаємо $l[i + 1], r[i + 1]$ як об'єднання всіх таких відрізків $[a, b]$ по всім cnt.

Тоді $r[m]$ буде максимальною кількістю непарних рядків. Це вже працює за $O(n \cdot m)$, що досить швидко.

Як тоді відновлювати? Можемо іти з кінця і за допомогою наших $l[i], r[i]$ відновлювати cnt в кожному стовпчику. Ми знаємо cnt останнього рядка — $r[m]$. Дізнаємося cnt для передостаннього і далі послідовно будемо рахувати $cnt[i]$ через $cnt[i + 1]$.

Дізнаватися $cnt[i]$ через $cnt[i + 1]$ досить просто. Проходимося по всім станам по парності з відрізка $[l[i], r[i]]$. Нехай зараз розглядаємо стан cur_cnt. Знаходимо відрізок в який можемо перейти з нього $[a, b]$. Якщо $cnt[i + 1] \in [a, b]$, то нам підходить cur_cnt в якості $cnt[i]$.

Так побудувавши масив cnt, вже нескладно відновити вихідну таблицю.

Складність алгоритму: $O(n \cdot m)$.

Задача 2E. Три запити

Автор задачі: **Костянтин Денисов**
Задачу підготував: **Павло Цікалишин**
Розбір написав: **Костянтин Денисов**

Будемо розв'язувати нашу задачу в офлайні. Тобто зчитаємо наші запити наперед і будемо відповідати на них в зручному нам порядку.

Спершу зробимо стиск можливих значень масиву a. Тобто відсортуємо масив b унікальних можливих значень елементів масиву a за час всіх запитів та змінимо всі значення b_i на i. Таким чином отримаємо відносно невеликі значення масиву a: $0 \leqslant a_i < n + q$.

Будемо обробляти запити по блоках розміром C. Основна ідея полягає в тому, що ми розділяємо всі значення у межах поточного блоку на дві групи:

- **Особливі значення** — це значення чисел нашого масиву, які змінювалися хоча б один раз у поточному блоці. Мається на увазі, що y — особливе що змінювалося w_y під час блоку або в якийсь момент був запит третього типу з $t = y$, або був запит третього, що виконувалось $a_x = y$ до запиту.
- **Неособливі значення** — це всі інші значення.

Кількість особливих значень у кожному блоці обмежена не перевищує $C \cdot 2$, що дозволяє опрацьовувати їх **окремо** для кожного запиту. А неособливих значень не так мало, але їх перевага в тому, що для цих значень нічого не змінюється під час блоку, що дозволить їх зручно обробити.

Для кожного особливого значення x:

1. Знаходимо всі позиції, де воно знаходилося або могло знаходитися після оновлень у поточному блоці.
2. Будемо проходитися по таких позиціях у порядку зростання.
3. Одночасно з цим другим вказівником перебираємо запити, відсортовані за r.
4. Для поточного запиту перебираємо позиції, які менше або рівні за r, з кінця поки не знайдемо позицію $last$, де стоїть x (поточне особливе значення) (щоб дізнаватися яке значення стоїть в тій позиції на момент запиту, то можемо для позицій, де були запити третього типу підрахувати для всіх моментів часу у блоку їх значення, це займає C^2 додаткового часу та пам'яті).
5. Якщо $w[x] = 1$ та $last \geqslant l$, то враховуємо його у підрахунку для поточного запиту.

Оскільки особливих значень не більше ніж $\cdot 2$, тому цей підхід ефективний.
Оновлення неособливих значень відбувається наступним чином:

1. Проходимо масивом зліва направо, оновлюючи останню позицію кожного числа. При цьому ігноруємо особливі значення.
2. Якщо $w[x] = 1$, додаємо $+1$ у поточну позицію та -1 у попередню, де на поточному префіксі це число було (або нічого не робимо, якщо це перше входження такого числа).
3. Другим вказівником будемо проходитися по запитах, які відсортовані за r.
4. Тоді дізнатися кількість унікальних неособливих чисел на відрізку $[l, r]$ враховуючи, що ми обробили лише перші r це те саме, що знайти суму на відрізку $[l, r]$;
5. Через те, що оновлень сумарно буде $O(n \cdot \frac{q}{C})$, а самих запитів лише $O(q)$, використовуємо sqrt-декомпозицію для оновлення точкових значень за $O(1)$ і знаходження суми на відрізку за $O(\sqrt{n})$.

Оберемо $C = \sqrt{q}$ і тоді рішення працює за $O(n \cdot \sqrt{q})$.