

Задача А. 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:

$\text{last_page_count} = k$, якщо n кратне k ;

$\text{last_page_count} = n \bmod k$, інакше.

Тоді час перегляду:

$\text{time} = \text{last_page_count} \cdot c$

Задача В. 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 одиниць можна утворити 2, але навпаки — ні. Тому формуємо групи $3 + 2$, а коли людей вагою 2 не залишиться, групуємо $3 + 1 + 1$ (або менше одиниць, якщо їх не вистачає). Після цього залишаються люди з вагами 1 та 2.
4. Далі компоновання відбувається у такій пріоритетності:
 - $2 + 2 + 1$ (оскільки дозволяє ефективно використовувати вагу);
 - $2 + 2$ (якщо одиниць 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 С. Діагональні числа

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

Оптимальний порядок перестановки кубиків можна описати наступним чином:

- Спочатку правильно складаємо останній стовпчик за $n + 2$ операції:
 - Перемістимо кубик 1 з першого стовпчика на другий;
 - Перемістимо кубик n з останнього стовпчика на перший;
 - Далі вже без перешкод за n операцій можна буде скласти останній стовпчик, просто переміщаючи кубики $1, 2, \dots, n$.
- 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;
- Для складання стовпчика i будемо діяти таким чином:
 - Перемістимо єдиний кубик i зі стовпчика i на стовпчик $i + 1$ (в попередньому випадку не було такого стовпчика);
 - Далі за i операцій можна буде скласти стовпчик, переміщаючи кубики $1, 2, \dots, i$.
- Після правильного складання стовпчиків з 2 по n перший стовпчик автоматично буде складений, тому додаткові операції на нього не знадобляться.

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

Задача D. 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. $x + cnt \leq n$.

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 $l, l + 2, l + 4, \dots$).

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 $dp[i][l] = true, dp[i][l + 2] = true, \dots$).

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 Е. Три запити

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

Будемо розв'язувати нашу задачу в офлайн. Тобто зчитуємо 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 : $0 \leq a_i < n + q$.

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 $last \geq l$, то враховуємо його у підрахунку для поточного запиту.

Оскільки особливих значень не більше ніж $C \cdot 2$, 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 $O(n \cdot \sqrt{q})$.