Задача A. Фотографія

Автор: Андрій Холод
Роздробник: Павло Цікалишин
Розбір підготовано: Костянтин Денисов

Фотографія має розмір $n \times n$. Рамка товщини t додає смугу шириною t з кожного боку, тому зовнішній розмір усієї конструкції дорівнює $(n + 2t) \times (n + 2t)$. Площа рамки дорівнює різниці площ зовнішнього квадрата та фотографії.

Площа фотографії: $S_{\text{in}} = n^2$.
Площа зовнішнього квадрата: $S_{\text{out}} = (n + 2t)^2$.
Отже, площа рамки: $S = S_{\text{out}} - S_{\text{in}} = (n + 2t)^2 - n^2$.

Спростимо вираз: $(n + 2t)^2 - n^2 = n^2 + 4nt + 4t^2 - n^2 = 4t(n + t)$.

Тому відповідь: $S = 4t(n + t)$.

Складність. Час: $O(1)$, пам'ять: $O(1)$.

Зауваження. За $n \leqslant 10^9$ значення площ може бути порядку 10^{18}, тому слід використовувати 64-бітний цілий тип (наприклад, `long long`).

Задача B. Пікнік

Автор: Андрій Холод
Роздробник: Павло Цікалишин
Розбір підготовано: Костянтин Денисов

Потрібно розбити масив $a_1, \ldots, a_n$ на $m = k + 1$ **послідовних** частин так, щоб сума в кожній частині була однакова, і використати всі шматки. Нехай $S = \sum_{i=1}^{n} a_i$. Тоді кожна частина має вагу $T = S/m$, отже обов'язково $m \mid S$.

Через те, що $n \leqslant 5000$, можна зробити дуже просту перевірку: перебрати всі можливі m від найбільшого до 1 і перевірити, чи можна нарізати на m частин сумою T жадібно зліва направо. Оскільки ми знаємо наперед суму в кожній частині, то далі перший відрізок визначається однозначно, за ним другий і так далі.

Перевірка для фіксованого m. Обчислюємо $T = S/m$ і йдемо по масиву, накопичуючи cur та підраховуючи кількість друзів cnt:

- $cur := cur + a_i$.
- якщо $cur = T$, то "закриваємо"частину: $cnt := cnt + 1$, $cur := 0$;
- якщо $cur > T$, то таке m неможливе (бо всі $a_i \geqslant 1$ і ми вже перебрали суму).

В кінці потрібно, щоб $cur = 0$ і $cnt = m$.

Складність. Перебираємо m не більше ніж n разів, кожна перевірка за $O(n)$, отже загалом $O(n^2)$. За $n \leqslant 5000$ це проходить. Пам'ять $O(n)$.

Задача C. Гра

Автор: Костянтин Денисов
Роздробник: Павло Цікалишин
Розбір підготовано: Костянтин Денисов

Очевидно, що на відрізку $[l, r]$ треба ставити якнайбільші числа, тобто $a_i = m$ для всіх $l \leqslant i \leqslant r$. Тоді цей відрізок має найбільший потенціал, і далі наша задача — зробити так, щоб жоден інший підвідрізок не міг з ним зрівнятися за сумою, та ще й серед усіх таких масивів максимізувати суму невід'ємних елементів.

Окремий випадок $l = r$. Тут максимальна сума має бути рівно m і досягатися **лише** на позиції l. Тому поза l не можна ставити m, інакше одиночний підвідрізок з цієї позиції теж матиме суму m. Отже поза l найкраще ставити $m - 1$. Але треба ще зіпсувати всі підвідрізки, які включають l і ще щось: для цього достатньо розставити $-m$ на кожній другій позиції, починаючи від сусідів $l - 1$ та $l + 1$. Тоді будь-яке розширення відрізка за межі $[l, l]$ неминуче підхопить хоча б один $-m$, і сума стане строго меншою за m.

Випадок $l < r$.

Поза відрізком $[l, r]$ оптимальна конфігурація має максимально багато m, але такі m доводиться ставити через один, чергуючи з $-m$; інакше з'являється занадто великий підвідрізок поза $[l, r]$ або підвідрізок, що "приклеюється"до $[l, r]$.

Тому для **префікса** довжини $s = l - 1$ оптимально ставити (зліва направо):

- якщо s **парна**, то $m, -m, m, -m, \ldots, m, -m, m - 1, -m$; доводиться ставити одну $m - 1$, щоб сума елементів була від'ємна, бо інакше буде декілька елементів з однаковою максимальною сумою.
- якщо s **непарна**, то $m, -m, m, -m, \ldots, m, -m, -m$.

Для **суфікса** довжини $t = n - r$ робимо аналогічну (дзеркальну) побудову за парністю t.

У будь-якому допустимому масиві поза $[l, r]$ не можна ставити багато великих додатних підряд: кожен додатний m змушений мати поруч достатню "компенсацію"$-m$, інакше з'являється підвідрізок з занадто великою сумою, який конкурує з $[l, r]$. Отже m можна ставити не частіше, ніж через один, тобто шаблон $m, -m, m, -m, \ldots$ дає максимальну кількість m. У парному випадку рівно один раз доводиться замінити m на $m - 1$, щоб уникнути рівності максимумів.

Складність. Побудова займає $O(n)$ часу і $O(1)$ додаткової пам'яті (окрім виводу).

Задача D. Василько і комп'ютер

Автор: Костянтин Денисов
Роздробник: Павло Цікалишин
Розбір підготовано: Костянтин Денисов

Можна розв'язувати жадібно: робити перший підмасив якомога довшим, потім другий якомога довшим, і т.д.

Чому так можна: припустимо, існує оптимальне розбиття, де перший підмасив закінчується раніше, ніж це можливо. Нехай другий підмасив починається з елемента x. Якщо додавання x в кінець першого підмасиву **не ламає** можливість відсортувати його стеком, то ми можемо просто "перекласти"x з початку другого підмасиву в кінець першого. Другий підмасив при цьому стає коротшим, але він все одно відсортовуваний (бо ми лише забрали з нього перший елемент). Отже завжди існує оптимальне розбиття, в якому перший підмасив максимально довгий. Те саме міркування працює для кожного наступного підмасиву.

Тому достатньо йти зліва направо і додавати елементи в поточний підмасив, доки це можливо; коли стає неможливо — робимо розріз і починаємо новий.

Ми будуємо один підмасив зліва направо і симулюємо, як би працювало сортування стеком. Коли ми кладемо елементи в стек, іноді ми **змушені** викидати деякі зі стеку у вихідну послідовність: якщо прийшов новий елемент x, який більший за верхівку стека, то цю верхівку вже не можна тримати далі, бо інакше x опиниться над меншим елементом і ми ніколи не зможемо отримати відсортований вихід.

Тому правило таке: поки $top < x$, ми викидаємо top зі стека. Серед усіх викинутих значень запам'ятовуємо найбільше m. Коли прийдемо до випадку $top > x$, то в цьому випадку вже не можна класти верхівку стеку до масиву, бо тоді новий елемент x точно не буде стояти на своїй позиції.

m — це фактично найбільше число, яке ми вже **викинули** зі стека у цьому підмасиві. Тобто у фінальній відсортованій послідовності воно вже стоїть десь зліва, і ми вже не можемо поставити перед ним щось менше.

Якщо в якийсь момент приходить елемент x та виконується $x < m$, то все: поточний підмасив вже неможливо відсортувати стеком. Чому: число m уже "пішло"у вихід і мусить стояти **до** x у фінальній послідовності, але $x < m$, отже порядок буде неправильний і це вже не виправити ніякими майбутніми діями зі стеком.

У такому випадку ми **зобов'язані** розрізати масив перед x і почати новий підмасив: очищаємо стек, скидаємо m і збільшуємо лічильник частин.

В інших випадках ($top >= m$) нескладно збагнути, що можна просто перекинути весь стек у кінцевий масив, щоб він був відсортованим.

Складність. Кожен елемент один раз заходить у стек і не більше одного разу виходить зі стека, тому час $O(n)$, пам'ять $O(n)$.

Задача E. Подарунки Васильку

Автор: Костянтин Денисов
Роздробник: Павло Цікалишин
Розбір підготовано: Костянтин Денисов

1) Жадібний алгоритм в лоб (для одного запиту). Нехай робот запускається на підмасиві $a[l..r]$, пам'ять розміру m. Підтримуємо мультимножину S (вміст пам'яті) і покажчик i, який спочатку дорівнює l.

Поки будуємо b:

- поки $|S| < m$ і $i \leqslant r$, додаємо a_i в S і $i := i + 1$;
- інакше беремо мінімальний елемент з S, записуємо в b, і видаляємо його з S.

На кожній позиції b ми беремо найменше число, яке взагалі доступне зараз (тобто вже завантажене в пам'ять). Якщо на деякому кроці не взяти мінімум, то перша позиція, де ми відхилились, стане більшою — лексикографічно гірше.

2) Яка відповідь на k-ту позицію. Позначимо $len = r-l+1$ і $p = l+k+m-2$ (це $l+(k+m-1)-1$).

- Якщо $p > r$, тобто $k+m-1 > len$, то робот встиг завантажити весь підмасив, і відповідь — просто k-та порядкова статистика на $[l,r]$: $b_k = k$-те найменше на $a[l..r]$.
- Якщо $p \leqslant r$, тоді відповідь така: $b_k = \min\Big(a_p,\ k\text{-те найменше на } a[l..p]\Big)$.

Доведення формули індукцією по k (випадок $p \leqslant r$). Позначимо $T(k) = l+k+m-2$ (тобто $T(k) = p$) і $X(k)$ — k-те найменше на $a[l..T(k)]$. Твердження: $b_k = \min(a_{T(k)}, X(k))$.

База $k = 1$. До першого вибору робот може завантажити рівно m елементів, тобто $a[l..l+m-1]$. Тому b_1 — мінімум серед цих m елементів, а це рівно $\min(a_{l+m-1}, \text{1-ше найменше на } a[l..l+m-1])$, тобто формула виконується.

Перехід $k \to k+1$. Припустимо, що після k кроків робот видав $b_1, \ldots, b_k$ за формулою, і розглянемо крок $k+1$.

До моменту вибору $(k+1)$-го елемента робот може завантажити $m+k$ перших елементів підмасиву, тобто рівно $a[l..T(k+1)]$. На перших k кроках жадібний алгоритм завжди забирав мінімально можливе, отже він забрав рівно k найменших елементів серед $a[l..T(k+1)]$, окрім можливої заміни на $a_{T(j)}$ там, де це було менше — але це якраз і означає, що на кроці $k+1$ серед кандидатів лишається:

- або $a_{T(k+1)}$, якщо він виявився "дуже малим"і пробиває статистику,
- або $X(k+1)$ як найменший серед тих, що лишились після вилучення k найменших.

Отже жадібний вибір на кроці $k+1$ дає $b_{k+1} = \min(a_{T(k+1)}, X(k+1))$.

Таким чином твердження доведене для всіх k.

Випадок $p > r$. Тут $k+m-1 > r-l+1$, робот встигає завантажити весь $a[l..r]$, тому b є просто відсортованим $a[l..r]$, і b_k — k-те найменше на $[l,r]$.

Як відповісти на q запитів швидко.

Нам треба багато разів знаходити k-те найменше число на відрізку $[L,R]$. Для цього будуємо персистентне дерево відрізків по "значеннях де в **листках** зберігаємо, скільки разів кожне значення зустрілося на префіксі.

1) Щоб були листки $0..n$. Значення a_i великі, тому спочатку впорядковуємо всі елементи. Сортуємо пари (a_i, i) за a_i і замінюємо кожен a_i на номер його позиції у відсортованому списку (число від 0 до n). Масив d зберігає цей відсортований список: якщо листок має номер pos, то справжнє значення дорівнює $d[pos].first$.

2) Персистентні версії для префіксів. Будуємо корені $c[0], c[1], \ldots, c[n]$. $c[i]$ — це дерево для префікса $[1..i]$.

У листку з номером pos зберігаємо число cnt: скільки разів на префіксі $[1..i]$ зустрілося значення з номером pos.

У внутрішній вершині зберігаємо суму cnt по її дітях (тобто скільки елементів префікса попадає в її діапазон листків).

Перехід від $c[i-1]$ до $c[i]$: ми збільшуємо на 1 значення в листку, що відповідає a_i. Персистентність означає: копіюємо тільки вершини на шляху від кореня до цього листка, і в них оновлюємо суми; всі інші вершини спільні з попередньою версією. Це $O(\log n)$ нових вершин (функція `add`).

3) Як отримати відрізок $[L,R]$ з двох префіксів. Нехай ми хочемо працювати з підмасивом $[L,R]$. Для будь-якого листка (тобто конкретного значення) кількість його поява на $[L,R]$ дорівнює: $cnt_{[L,R]} = cnt_{[1,R]} - cnt_{[1,L-1]}$, тобто різниця того самого листка в деревах $c[R]$ і $c[L-1]$.

4) Пошук k-го найменшого. Маємо дві версії дерева: одна для префікса $[1..R]$, друга для $[1..L-1]$. Розглянемо будь-яку вершину дерева, яка відповідає деякому діапазону значень $[x, y]$. У цій вершині зберігається число елементів префікса, що попали в $[x, y]$. Тоді кількість елементів на відрізку $[L, R]$ у цьому діапазоні дорівнює різниці цих чисел у двох версіях.

Щоб знайти k-те найменше, спускаємося від кореня до листка. Нехай зараз ми в діапазоні $[x, y]$ і $mid = (x + y)/2$. Дивимось, скільки елементів з $[L, R]$ лежить у лівій половині $[x, mid]$: це різниця значень у **лівих дочірніх вершинах** двох версій. Позначимо це число як $left$.

- Якщо $left \geqslant k$, то шукане значення лежить у лівій половині, і ми переходимо в лівих дітей обох версій.
- Інакше воно лежить у правій половині, тоді переходимо в правих дітей, а k замінюємо на $k - left$.

Так ми за $O(\log n)$ кроків дійдемо до листка, який і відповідає шуканому числу (потім переводимо його назад у початкове значення через відсортований список).

Складність. Побудова: n додавань, кожне $O(\log n)$, разом $O(n \log n)$. Кожен запит: $O(\log n)$. Пам'ять: $O(n \log n)$ вершин.