Всероссийская олимпиада школьников по информатике 2021–2022

Региональный этап

Разбор задач

Условия задач, тесты, решения и разбор задач подготовили Денис Акилов, Николай Будин, Савелий Григорьев, Михаил Первеев, Андрей Станкевич, Григорий Хлытин.

Ценные замечания по результатам тестирования задач сделали Никита Голиков, Мария Жогова, Андрей Левков, Михаил Первеев.

Задача 1. Чемпионат по устному счету

Автор задачи: Михаил Первеев

Подзадача 1

Заметим, что если команд второго типа нет, то достаточно поддерживать сумму чисел на доске и при изменении числа вычитать из суммы старое значение и прибавлять новое. Впрочем, ограничения первой подзадачи позволяют после при аккуратной реализации каждого изменения в цикле обновлять сумму массива.

Подзадача 2

Рассмотрим циклический сдвиг массива на 1. Для этого просто выполним присваивание $a[i] = a[i-1]$ для всех i в убывающем порядке, а $a[1]$ присвоим старое значение $a[n]$ (не забудем его сохранить). При ограничениях второй подзадачи это можно сделать за $O(n)$.

Подзадача 3

В этой подазадаче уже требуется выполнять циклический сдвиг для произвольного k. К счастью в условии приведен вид массива после циклического сдвига, осталось его аккуратно реализовать. Снова возможна реализация за $O(n)$.

Подзадача 4

Для решения задачи на полный балл необходимо научиться выполнять команды этого типа быстрее. Будем поддерживать переменную s — на сколько вправо сдвинут исходный массив на данный момент времени. Тогда в случае, если текущая команда второго типа, нужно увеличить число s на k. Осталось понять, как обрабатывать команды первого типа.

Заметим, что если некоторый элемент находился изначально на позиции i, то теперь он находится на позиции $(i+s) \bmod n$, если нумеровать позиции с нуля. Таким образом, если необходимо изменить элемент с номером p на x, изменим элемент массива с номером $(p-s) \pmod n$ на x. Вычислить новую сумму массива легко — нужно из старой суммы вычесть старое значение элемента, а затем прибавить новое значение.

Получили решение за $O(n+q)$.

Задача 2. Прыгающий робот

Автор задачи: Елена Андреева

Подзадача 1

Для решения первой подзадачи можно воспользоваться полным перебором. Заметим, что значения $a > \max\{d_i\}$ нет смысла перебирать.

Зафиксируем первую платформу и значение a. Проверим, подходит ли данное значение.

Асимптотика решения $O(n^2 \max\{d_i\})$.

Подзадача 2

Избавимся от перебора значений a. Пусть мы зафиксировали начальную платформу. Найдем минимальное значение a, которое подходит. Инициализируем a как 0. Пробежимся по всем платформам, если текущей ловкости не хватает, чтобы перепрыгнуть на следующую, мы знаем, на какую величину ее надо увеличить. При этом увеличение ловкости не влияет на возможность совершать прыжки между рассмотренными ранее платформам. Получаем решение за $O(n^2)$.

Подзадача 3

Здесь нам задана начальная платформа, поэтому можно не реализовывать ее перебор. Применим решение предыдущей подзадачи и найдем подходящее начальное значение ловкости.

Подзадача 4

Рассмотрим еще раз внимательно процесс поиска начальной ловкости после фиксирования стартовой платформы i:

```
int a = 0;
for (int k = 0; k < n; k++) {
    if (d[(i + k) % n] > a) {
        a = d[(i + k) % n];
    }
    a++;
}
a -= n;
```

Изменим немного этот фрагмент, избавимся от операции `a++`, включив накопленное значение в сравнение:

```
int a = 0;
for (int k = 0; k < n; k++) {
    if (d[(i + k) % n] > a + k) {
        a = d[(i + k) % n] - k;
    }
}
```

Перенося `+ k` на другую сторону неравенства, получаем окончательно код

```
int a = 0;
for (int k = 0; k < n; k++) {
    if (d[(i + k) % n] - k > a) {
        a = d[(i + k) % n] - k;
    }
}
```

в котором легко узнать поиск максимума в массиве $d[(i+k)\%n] - k$.

Заменим массив d на $d'[k] = d[k] - k$ и удвоим его: $d'[k+n] = d'[k] + n$. Тогда, чтобы учесть сдвиг индекса на i, надо взять максимум значений этого массива на полуинтервале $[i, i+n)$ и прибавить к нему значение i.

Таким образом мы свели задачу к задаче поиска максимума на отрезке. Эта задача широко известна и в этой подзадаче можно применить любую из известных структур данных, например дерево отрезков или разреженную таблицу.

Подзадача 5

Это «учебная» подзадача, которая проверяет, что решение участников умеет генерировать массив по формуле для $f = 2$. Поскольку перебирать начальную платформу не надо, решение за линейное время с поиском максимума в массиве работает.

Поздадача 6

В этой подзадаче размер массива 10^7, поэтому разреженные таблицы не помещаются в память, а дерево отрезков сверху не проходит по времени. Можно либо использовать очень оптимизированное дерево отрезков в реализации снизу, дерево Фенвика, либо воспользоваться структурой данных для «максимума в окне».

Есть несколько разных реализаций этой структуры данных, одна из наиболее простых следующая. Нам нужно найти максимумы на полуинтервалах $[0, n)$, $[1, n + 1)$, $\ldots$, $[n; 2n)$. Посчитаем суффиксные максимумы на полуинтервалах $[0, n)$, $[1, n)$, $[2, n)$, и так далее, а также префиксные максимумы на полуинтервалах $[n, n + 1)$, $[n, n + 2)$, и так далее. Заметим, что максимум на нужном нам полуинтервале это максимум из двух предподсчитанных значений.

Задача 3. Треугольная головоломка

Автор задачи: Игорь Мамай

Посмотрим сначала, как можно использовать дополнительные ограничения в первых тестах.

Тест 3

Из любой четверки треугольников можно собрать треугольник.

Тесты 4 и 5

В этих тестах треугольник можно собрать только из четырех треугольников, одинаковых с точностью до поворота и сдвига.

Тесты 6–12

Дополнительные ограничения, наложенные на входные данные, позволяют уменьшить количество вариантов, которые нужно перебрать в решении по сравнению с полным.

Так, в тестах 6 и 7, а также 10, 11 и 12 треугольники не надо вращать.

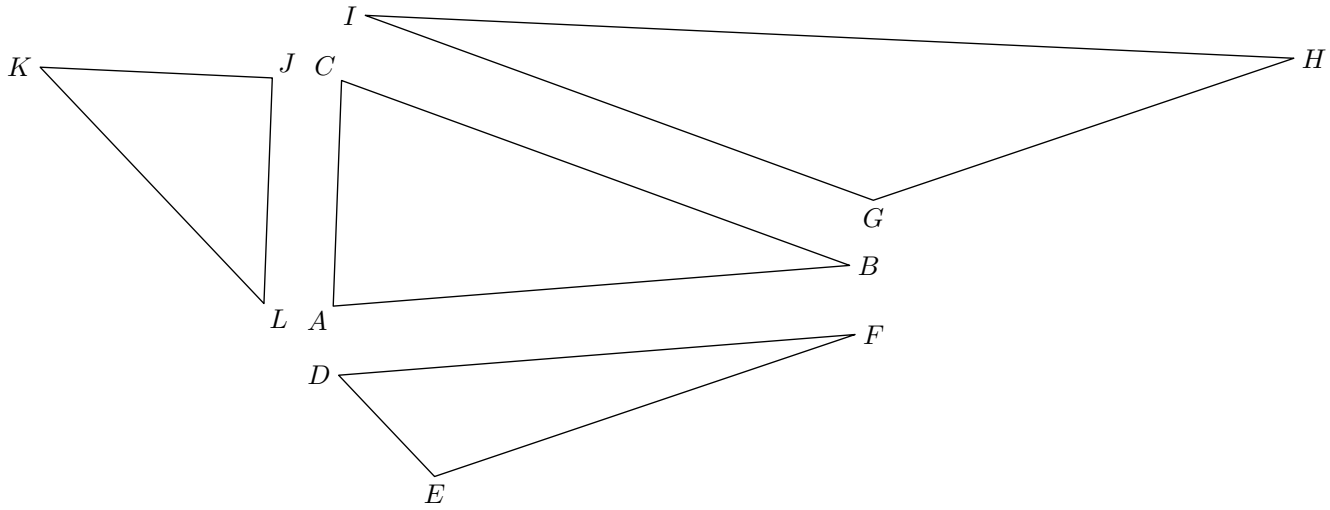
В тестах 8 и 9 треугольники может потребоваться повернуть, но легче проверить, что из четырех треугольников можно собрать один.

Полное решение

Переберем треугольник, который будет располагаться в центре, пусть это $\triangle ABC$. Переберем три треугольника, которые будут располагаться в углах и касаться сторон AB, BC и CA. А также, для каждого из трех треугольников, которые будут располагаться в углах, переберем циклический сдвиг вершин. Пусть $\triangle DEF$ касается стороны AB стороной DF, $\triangle GHI$ касается стороны BC стороной GI, и $\triangle JKL$ касается стороны CA стороной JL. Тогда эти четыре треугольника собираются в один большой, если:

- $|AB| = |DF|$
- $|BC| = |GI|$
- $|CA| = |JL|$
- $\angle BAC + \angle FDE + \angle JLK = 180^\circ$

- $\angle CBA + \angle IGH + \angle DFE = 180°$
- $\angle ACB + \angle LJK + \angle GIH = 180°$



Проверить, что стороны равны по длине можно абсолютно точно, вычислив квадрат длины стороны по формуле Пифагора.

Чтобы проверить, что три угла в сумме равны 180°, нужно вычислить углы. Например, по теореме косинусов, воспользовавшись затем функцией acos. И проверить, что сумма углов достаточно близка к 180° — отличается не более чем на ε. При этом, необходимо выбрать достаточно маленький ε. А именно, если углы вычисляются в радианах, можно показать, что достаточно $\varepsilon \approx 10^{-12}$ при ограничениях на координаты точек до 10^6.

Таким образом, решение перебирает $n^4 \cdot 3^3$ вариантов.

Задача 4. Массивы-палиндромы

Автор задачи: Григорий Хлытин

Здесь и далее будут использоваться следующие обозначения:

$A[l, \ldots, r]$ — массив, полученный из элементов $A[l], A[l+1], \ldots, A[r]$ массива A. Будем называть массив $A[l, \ldots, r]$ *подмассивом* массива A. Индексация всех массивов с нуля, границы отрезков включительно.

Подзадачи 1 и 3

Для решения первой и третьей подзадачи можно воспользоваться следующим алгоритмом.

Будем перебирать позицию pos_1 в массиве A, а затем будем рассматривать массивы $A' = A[pos_1, \ldots, \min(n-1, pos_1 + m - 1)]$ и $B' = B[0, \ldots, \min(n - 1 - pos_1, m - 1)]$. Так как полученные массивы A' и B' имеют одинаковую длину $k = \min(n - pos_1, m)$, то Кай может поэлементно их просуммировать и получить массив C длины k.

Аналогично будем перебирать позицию pos_2 в массиве B, а затем будем рассматривать массивы $B' = B[pos_2, \ldots, \min(m-1, pos_2+n-1)]$ и $A' = A[0, \ldots, \min(m-1-pos_2, n-1)]$. Так как полученные массивы B' и A' имеют одинаковую длину $k = \min(m - pos_2, n)$, то Кай может поэлементно их просуммировать и получить массив C длины k.

Заметим, что теперь ответ на задачу — длина максимального палиндрома в массиве C.

Так как у каждого палиндрома есть его «центр» в массиве C, переберем (отдельно для четных и нечетных длин палиндромов) всевозможные центры x ($0 \leqslant x < k$), где k — длина массива C.

Для каждого такого центра x нам нужно найти наибольшую длину палиндрома len с центром в x. Для этого будем идти $i \geqslant 0$ в разные стороны от него и проверять элементы массива на равенство: $C[x - i] = C[x + i]$ для нечетного случая, либо $C[x - i] = C[x + i + 1]$ для четного случая. Если это равенство будет верным для всех $i \leqslant q$, то искомая длина палиндрома равна $len = 2 \cdot q + 1$ в нечетном случае и $len = 2 \cdot q + 2$ в четном случае. Таким образом, мы сможем найти длину максимального

палиндрома в массиве C со сложностью $O(k^2)$, так как сделаем $O(k)$ операций на перебор всех центров, а потом для каждого центра $O(k)$ операций для поиска наибольшей длины палиндрома с центром в нем.

Таким образом, асимптотика этого решения будет $O((n+m)\cdot k^2) = O((n+m)\cdot \min(n,m)^2)$. Этого хватит для решения первой подзадачи, но не хватит для решения третьей подзадачи.

Подзадачи 2 и 3

Заметим, что мы умеем искать наибольшую длину палиндрома len с фиксированным центром x за $O(\log(k))$.

Давайте посчитаем полиномиальные хеши $h[i] = C[0]\cdot P^{i-1} + C[1]\cdot P^{i-2} + \ldots + C[i-1]\cdot P^0$ для всех подмассивов $C[0,\ldots,i-1]$ массива C, $h[0] = 0$. Аналогично посчитаем полиномиальные хеши $h_{rev}[i]$ для всех подмассивов $C_{rev}[0,\ldots,i-1]$ массива C_{rev}, $h_{rev}[0] = 0$, где C_{rev} — массив, полученный переворачиванием массива C. Для фиксированного массива C сложность такого предпосчета $O(k)$.

Тогда для любого подмассива $C[l,\ldots,r]$ за $O(1)$ знаем прямой хеш $hash(l,r) = h[r+1] - h[l]\cdot P^{r-l+1}$ и обратный хеш $hash_{rev}(l,r) = h_{rev}[k-l] - h[k-r-1]\cdot P^{r-l+1}$.

Теперь найдем ответ при помощи бинарного поиска по длине len палиндрома с центом в x (отдельно для четных и нечетных длин палиндромов). Будем проверять на равенство прямой и обратный хеш: $hash(x-i,x) = hash_{rev}(x,x+i)$ для нечетного случая, либо $hash(x-i,x) = hash_{rev}(x+1,x+i+1)$ для четного случая, а затем сдвигать границы бинарного поиска.

Такое решение будет работать со сложностью $O((n+m)\cdot k\cdot \log(k))$, что равно $O((n+m)\cdot \min(n,m)\cdot \log(\min(n,m)))$.

Заметим, что найти самый длинный палиндром в массиве C можно еще быстрее, например при помощи алгоритма Манакера, который работает со сложностью $O(k)$.

Итоговая асимптотика решения в этом случае будет $O((n+m)\cdot \min(n,m))$.

Для решения второй подзадачи заметим, что нам всего лишь нужно найти длину максимального палиндрома в массиве A, так как известно, что все элементы массива B одинаковые.

В массиве A найдем длину максимального палиндрома нечетной длины t_1, а также длину максимального палиндрома четной длины t_2 при помощи хешей или алгоритма Манакера, как было описано выше.

Обозначим длину массива B как m. Если m меньше чем t_1, то $t_1 = m$ или $t_1 = m-1$ чтобы длина t_1 оставалась нечетной. Аналогично если m меньше чем t_2, то $t_2 = m$ или $t_2 = m-1$ чтобы длина t_2 оставалась четной.

Ответом на задачу будет максимум из длин t_1 и t_2.

Подзадача 4

Предпосчитаем прямые и обратные полиномиальные хеши для каждого из массивов A и B, как это было расписано выше.

Воспользуемся бинарным поиском по ответу ans (отдельно для четных и нечетных длин палиндромов). Внутри бинарного поиска:

Пусть длина палиндрома $ans = 2\cdot q + 2$ или $ans = 2\cdot q + 1$ в четном и нечетном случае соответственно. Пусть константа $f = 1$ для четного случая и $f = 0$ для нечетного случая.

- Инициализируем множество $set = [\ldots]$, в котором будем хранить целые числа.
- Перебираем все центры x массива A. Для каждого такого центра x добавляем (`add`) в `std::set` S разность хешей подмассивов $hash(A[x-q,\ldots,x]) - hash_{rev}(A[x+f,\ldots,x+q+f])$.
- Перебираем все центры y массива B. Для каждого такого центра y проверяем (`contains`), содержит ли S разность хешей подмассивов $hash_{rev}(B[y+f,\ldots,y+q+f]) - hash(B[y-q,\ldots,y])$. Если содержит — значит можно сделать палиндром длины ans, иначе — нельзя.
- Сдвигаем границы бинарного поиска.

Чтобы избежать коллизий, необходимо делать вычисления сразу по нескольким простым модулям, например $M_1 = 10^9 + 7$ и $M_2 = 10^9 + 9$, и как результат хранить пару из двух целых чисел $\{u, v\}$ — результат вычисления по первому и второму модулю соответственно. Пусть для примера $hash_1 = \{u_1, v_1\}$ и $hash_2 = \{u_2, v_2\}$, тогда $hash1 + hash2 = \{(u_1 + u_2) \mod M_1, (v_1 + v_2) \mod M_2\}$ и аналогично для других операций.

Решение работает за $O((n+m) \cdot \log(\min(n, m)) \cdot W)$, где $O(W)$ — сложность операции добавления (`insert`) и проверки принадлежности (`count`) для `std::set`.

Если использовать `«std::unordered_set»` в C++, то $O(W)$ будет в среднем $O(1)$.

Задача 5. Новый год в детском саду

Автор задачи: Денис Акилов

Для начала давайте переформулируем условие задачи в более простой форме: для данных n, a и b нужно посчитать количество пар целых чисел (x, y) таких, что $0 \leqslant x \leqslant a$, $0 \leqslant y \leqslant b$, $x + y > 0$ и $(x + y)$ делится на n. Стоит обратить внимание, что в некоторых подзадачах (4, 5, 7, 8) ответ может быть очень большим, и чтобы не было переполнения в таких языках, как C++, нужно использовать целочисленный тип данных, который умеет работать с числами порядка 10^{18} (в C++ это, например, `long long int`).

Подзадача 1

В первой подзадаче достаточно перебрать все пары (x, y) и для каждой отдельно проверить — подходит она или нет.

```
ans = 0
for x in range(a + 1):
    for y in range(b + 1):
        ans += (x + y > 0 and (x + y) % n == 0)
print(ans)
```

Подзадача 2

В этой подзадаче нужно найти количество целых чисел $0 < y \leqslant b$, которые делятся на n. Нам подходят значения $n, 2n, 3n, \ldots, \left\lfloor \frac{b}{n} \right\rfloor n$, то есть всего вариантов $\left\lfloor \frac{b}{n} \right\rfloor$.

Подзадача 3

Можно заметить, что сумма $0 < (x + y) < 2n$ и делится на n, значит, она равна n. Это значит, что для каждого конкретного значения x мы можем точно сказать, что $y = n - x$. Тогда можно перебрать все x от 0 до a и проверить, что пара $(x, n - x)$ является корректной.

Подзадача 4

На самом деле мы можем перебирать только x и в этой подзадаче. Только теперь мы не знаем, чему точно равно значение y. Но мы можем сказать, какой оно имеет вид: остаток от деления y на n должен быть сравним с $-x$ (чтобы сумма делилась на n). Теперь мы свели исходную задачу к следующей: для данных n, r и b найти количество целых чисел $0 \leqslant y \leqslant b$, которые дают остаток r при делении на n. Эту задачу несложно свести к случаю $r = 0$: можно заметить, что $y \geqslant r$, тогда можно вычесть из y и из b значение r. Другими словами, нужно найти количество $0 \leqslant y \leqslant b - r$, делящиеся на n. Мы уже научились это делать во второй подзадаче, за исключением некоторых деталей:

- нужно отдельно обрабатывать случай $b - r < 0$;
- $y = 0$ теперь подходит, то есть теперь формула имеет вид $\left\lfloor \frac{b-r}{n} \right\rfloor + 1$;
- нужно из ответа в конце вычесть 1, так как мы посчитали пару $(0, 0)$, которая нам не подходит.

Подзадача 5

Вернемся к решению первой подзадачи. Заметим, что мы можем перебирать не сами значения x и y, а их остатки по модулю n. Тогда для каждой пары остатков можно проверить, подходит она или нет, после чего найти количество вариантов для x и количество вариантов для y.

Подзадача 6

А теперь давайте вспомним про третью подзадачу. У нас для каждого остатка x подходит только один остаток y.

```
    def num_of_r(x, r, m):
        return 0 if x < r else (x - r) // m + 1


    t = int(input())
    for tn in range(t):
        n, a, b = map(int, input().split())
        ans = 0
        for ra in range(n):
            rb = (n - ra) % n
            ans += num_of_r(a, ra, n) * num_of_r(b, rb, n)
        print(ans - 1)
```

Подзадача 7

Рассмотрим 4 группы пар (x, y):

1. $k_x n \leqslant x < (k_x + 1)n < a$; $k_y n \leqslant y < (k_y + 1)n < b$
2. $\left\lfloor \frac{a}{n} \right\rfloor n \leqslant x \leqslant a$; $k_y n \leqslant y < (k_y + 1)n < b$
3. $k_x n \leqslant x < (k_x + 1)n < a$; $\left\lfloor \frac{b}{n} \right\rfloor n \leqslant y \leqslant b$
4. $\left\lfloor \frac{a}{n} \right\rfloor n \leqslant x \leqslant a$; $\left\lfloor \frac{b}{n} \right\rfloor n \leqslant y \leqslant b$

По сути, мы разбили возможные x и y на блоки размера n (последний блок может быть меньшего размера, если число не делится нацело на n). Нам это поможет, так как нам на самом деле неважно, какой именно из полных блоков мы рассматриваем.
В первом случае у нас для каждого x ровно один подходящий вариант ($x = k_x n + r$, тогда $y = k_y n + n - r$), всего пар полных блоков $\left\lfloor \frac{a}{n} \right\rfloor \cdot \left\lfloor \frac{b}{n} \right\rfloor$, в каждом блоке n подходящих x.
Во втором случае у нас также для каждого x будет ровно один подходящий y (так как второй блок полный), но в этом случае у нас всего $a\%n + 1$ возможных x.
Третий случай аналогичен второму, только x и y поменялись местами.
В четвертом случае нам подходят только те x, для которых $x + b\%n \geqslant n$, то есть $n - b\%n \leqslant x \leqslant a\%n$, для них также будет всего один подходящий y.
Суммируя все 4 случая, получаем формулу:

```
t = int(input())
for tn in range(t):
    n, a, b = map(int, input().split())
    print((a // n) * (b // n) * n + (a % n + 1) * (b // n) +
          (b % n + 1) * (a // n) + max(0, a % n + b % n - n + 1))
```

Задача 6. Сортировка дробей

Автор задачи: Федор Царев

Для начала отметим, что поскольку значения в массивах не превышают 10^6, две неравные дроби различаются хотя бы на 10^{-12}. Это дает нам возможность при сравнении двух дробей использовать вещественные числа, хотя в целом возможно и решение этой задачи в целых числах.

Для сокращения дроби перед выводом необходимо воспользоваться алгоритмом Евклида, во всех решениях необходимо это сделать непосредственно перед выводом ответа.

Подзадачи 1 и 2

В подзадачах 1 и 2 можно сложить все дроби в массив, отсортировать по возрастанию и выводит ответы на запросы. В подзадаче 1 можно воспользоваться любой сортировкой, в том числе квадратичной. В подзадаче 2 нужно воспользоваться быстрой сортировкой, например, встроенной в язык программирования.

Начиная с подзадачи 3 значение n может достигать 10^5 и сгенерировать все дроби, сложив их в массив или просто рассмотрев каждую по разу, не хватает времени и памяти.

Разобьем дроби на классы по значению знаменателя b_i. Отсортируем массивы a и b по возрастанию.

Подзадача 3

В этой задаче все запросы не превышают 100, то есть надо найти первые 100 дробей. Научимся генерировать дроби в порядке возрастания.

Заметим, что при фиксированном знаменателе минимальное значение — значение с минимальным числителем. При этом некоторые из этих значений могут быть уже использованы при генерации.

Получаем следующий алгоритм: для каждого знаменателя b_i храним минимальный не использованный числитель $a_{pos[i]}$. Чтобы получить значение следующей по величине дроби, находим минимальное значение $a_{pos[i]}/b_i$ за $O(n)$, отправляем эту дробь в вывод, а значение $pos[i]$ увеличивается на 1.

Следует особо обратить внимание, что в процессе генерации дробей в порядке возрастания дроби с каким-либо знаменателем могут «закончиться» и следует аккуратно рассмотреть этот случай.

Решение работает за $O(n \max\{c_j\})$.

Подзадача 4

В этой подзадаче можно также сгенерировать необходимое количество дробей в порядке возрастания, так как $c_j \leqslant 10^5$. Однако генерация одной дроби за $O(n)$ уже не подходит, необходимо получать минимальную неиспользованную дробь быстрее.

Сложим все дроби $a_{pos[i]}/b_i$ в приоритетную очередь, дерево отрезков или `std::set`. Теперь мы можем находить минимальную неиспользованную дробь за $O(\log n)$, получаем решение за $O(n + \max\{c_j\} \log n)$.

Подзадача 5

Рассмотрим теперь два решения полной версии задачи.

Решение 1. Рассмотрим один запрос c_k. Найдем вещественное значение ответа f с использованием двоичного поиска. Чтобы проверить, верно ли, что $f \geqslant m$ для некоторого значения m, найдем d — количество дробей, меньших f. Для этого используем метод двух указателей: заметим, что при фиксированном знаменателе b_j подходят дроби с числителем, $a_i < b_j \cdot m$. При переходе к $b'_j > b_j$ старые значения a_i также подходят и, возможно, появляются новые. Получаем следующий код:

```
        i = 0;
        for (int j = 0; j < n; j++) {
            while (i < n && a[i] < b[j] * m) {
                i++;
            }
            d += i;
        }
```

Если $d < c_k$, то $f > m$, иначе $f \leqslant m$.

Завершив двоичный поиск, мы получаем вещественное число f, осталось найти, какой дробью это значение реализуется. Тут следует отметить, что f найдено не точно, но достаточно точно, если отрезок двоичного поиска сужен менее чем до 10^{-12}.

Чтобы найти подходящую дробь, возьмем минимальное значение $a_i/b_j \geqslant f$ и максимальное значение $a_i/b_j < f$. Проверим их оба и выберем то, которое подходит.

Время ответа на один запрос $O(\log(1/\varepsilon) \cdot n)$, на все запросы $O(\log(1/\varepsilon) \cdot n \cdot q)$, тут играет роль ограничение $n \cdot q \leqslant 10^5$.

Решение 2. Недостаток предыдущего решения — использование вещественной арифметики. Рассмотрим полностью целочисленное решение на основе классического алгоритма Хоара поиска k-й порядковой статистики в массиве.

Напомним основную идею алгоритма: как в алгоритме быстрой сортировки, выбираем разделяющий элемент x и делим массив на две части: меньшие или равные x и большие x. После этого алгоритм быстрой сортировки запускается рекурсивно от обеих половин массива, а алгоритм поиска k-й порядковой статистики — только от той части, где лежит ответ.

В нашей задаче мы не можем в явном виде хранить массив. Но снова посмотрим на дроби с фиксированным знаменателем. Обратим внимание, что подходящие числители еще оставшихся в рассмотрении дробей с этим знаменателем образует отрезок отсортированного массива a. Будем хранить границы этого отрезка для каждого знаменателя и уточнять его границы с помощью двоичного поиска. В качестве x можно взять случайным образом выбранную дробь из остающихся в рассмотрении.

Алгоритм работает за $O(n \log^2 n)$ (один log от двоичного поиска для разделения каждого массива, второй — от глубины рекурсии).

Мы снова получили оценку времени работы только для одного запроса, но умножая на число запросов и учитывая ограничение $n \cdot q \leqslant 10^5$, получаем приемлимое время работы.

Задача 7. Оптические каналы связи

Автор задачи: Андрей Станкевич

Задачу можно сформулировать так: требуется удалить часть ребер из дерева, чтобы степень каждой вершины была не больше k. При этом требуется оставить как можно больше ребер, а при равном количестве оставить ребра с максимальной суммой.

Большинство подзадач этой задачи требуют до той или иной степени применения технологии динамического программирования на дереве. Основная идея следующая: состояние — это вершина дерева, для поддерева которой решается задача, а также информация о решении в поддереве, которая необходима для использования решения для этого поддерева снаружи от него.

Подзадачи 1, 2 и 3

В этих подзадачах можно реализовать перебор всех ребер, которые мы оставляем за $2^n \cdot poly(n)$.

Подзадачи 1, 4 и 5

В этих подзадачах требуется найти паросочетание в дереве. Невзвешенная задача решается жадным алгоритмом, а для подзадачи 5 надо использовать динамическое программирование. Состояние

динамического программирования для поиска максимального взвешенного паросочетания: (u, b), где u — вершина дерева, а b — флаг, можно ли задействовать вершину u в паросочетании с её родителем.

Подзадачи 6 и 7

В этих подзадачах оставшиеся ребра образуют пути. Каждый путь в дереве состоит из двух частей: вверх и вниз (одна из них может отсутствовать). Используем динамическое программирование, состояние: (u, b), где u — вершина дерева, а b — флаг, является можно ли продлить из вершины u путь вверх.

Рассмотрим теперь решения для произвольного k.

Заметим общую тенденцию решения подзадач для $k = 1$ и $k = 2$: флаг b в состоянии динамического программирования показывает, были ли все k ребер для корня поддерева u уже сохранены в выборанном оптимальном решении для поддерева, либо можно выбрать ребро из u в родителя.

Обобщим это для полного решения: состояние — пара (u, b), где флаг b показывает, были ли все k ребер для корня поддерева u сохранены для решения в поддереве.

Как пересчитать значение динамического программирования. Рассмотрим вершину u. Чтобы посчитать $(u, 1)$ надо выбрать не более $k-1$, а чтобы посчитать $(u, 0)$ — не более k ребер в поддерево из корня. Для каждого выбранного ребра uv к значению прибавляется $(v, 0)$, а для невыбранного $(v, 1)$.

Для вычисления значений воспользуемся вспомогательным динамическим программированием, подобным задаче о рюкзаке. Посчитаем $opt[i][j]$ — рассмотрели первые i ребер и взяли из них j, какое оптимальное решение можно получить таким образом. Переход выполняется за $O(1)$, мы либо берем очередное ребро, либо нет.

Вычисление массива opt для вершины, у которой t детей, работает за $O(tk)$, суммируя по всем вершинам получаем время $O(nk)$.

Значение основного динамического программирования для состояния $(u, 0)$ равно максимуму по j от 0 до k, а для состояния $(u, 1)$ — от 0 до $k - 1$.

Наконец, сформулируем четко, что мы храним в качестве значения динамического программирования. Если задача невзвешенная (подзадачи 1, 2, 4, 6, 8, 10, 12, где $w_i = 0$), значение динамического программирования — это количество взятых ребер. Если же задача взвешенная, то в качестве значения будем хранить пару из количества взятых ребер и суммарного их веса. Количество ребер будет первым критерием оптимизации, а их суммарный вес — вторым, при равенстве количества ребер.

Менее эффективные решения, которые реализуют динамическое программирование за $O(n^2k)$ или $O(nk^2)$ могут проходить подзадачи 8 и/или 9, и 10 и/или 11, соответственно.

Задача 8. Подарки

Автор задачи: Савелий Григорьев

Подзадача 1

Самое наивное решение данной задачи — перебрать все возможные пары l и r, найти k максимальных чисел на отрезке $[l, r]$ массива (например, сортировкой), а затем вычислить сумму всех чисел без k максимумов. Такое решение работает за $O(n^3 \log n)$ и проходит первую группу тестов.

Подзадача 2

Для прохождения второй группы тестов заметим следующее: зафиксируем левую границу отрезка l и будем увеличивать правую границу r. В таком случае надо уметь поддерживать k максимумов (и их сумму) и добавлять по одному числу к отрезку. Такую задачу можно решить при помощи `std::set` с асимптотикой $O(n^2 \log n)$.

Подзадача 4

Теперь давайте научимся решать задачу для $k = 0$. Нам требуется найти подотрезок исходного массива с максимальной суммой. Посчитаем префиксные суммы $pref_i = a_1 + a_2 + \ldots + a_i$. Будем перебирать правую границу отрезка r, тогда нам нужно выбрать такое $l \leqslant r$, что $a_l + \ldots + a_r = pref_r - pref_{l-1}$ было максимальным. Для этого возьмём минимальное значение $pref_i$ среди всех $i < r$, обозначим его за m, тогда ответ для фиксированной границы r — это $pref_r - m$. Итоговая асимптотика — $O(n)$.

В дальнейшем будем считать, что $k \geqslant 1$, так как случай $k = 0$ разобран.

Заметим, что ответ всегда $\geqslant 0$, так как можно взять любой отрезок длины k и все подарки отдать Маше.

Подзадача 3

Для третьей подзадачи сделаем следующее: пусть все подарки исходно не помечены. Будем помечать их в порядке возрастания характеристики (если несколько подарков имеют одинаковую характеристику, то их помечаем в любом порядке). В получившемся массиве рассмотрим все отрезки, на которых есть хотя бы k непомеченных подарков, они являются кандидатами для ответа.

Действительно, для любого отрезка настанет момент, когда на нём будут не помечены только k максимумов. В то же время, если на отрезке не помечены более k максимумов, то его общее удовольствия будет только меньше (это неверно, если какой-то из непомеченных подарков имеет отрицательную характеристику, однако в таком случае все помеченные точно имеют отрицательное общее удовольствие, что меньше 0 и не влияет на ответ).

Пусть зафиксированы помеченные элементы. Решим задачу методом двух указателей: будем перебирать правую границу r, тогда l — максимальная левая граница, что на $[l, r]$ есть хотя бы k непомеченных элементов. При увеличении r граница l не может уменьшиться. Тогда в качестве ответа для фиксированного r надо взять такое $l_0 \leqslant l$, что $pref_r - pref_{l_0-1}$ — максимально (здесь $pref_i$ обозначает не сумму первых i чисел массива, а сумму помеченных чисел на префиксе i). Это делается аналогично подзадаче 4. Итоговая асимптотика $O(n^2)$ позволяет пройти тесты в третьей группе.

Подзадача 5

Разберём случай $k = 1$. Так же будем помечать элементы от меньшего к большему. Дополнительно поддерживаем `std::set` отрезков из подряд идущих помеченных элементов. Тогда при рассмотрении очередного элемента x мы должны выбрать суффикс с максимальной суммой на отрезке слева от x и префикс с максимальной суммой на отрезке справа от x, а после объединить эти два отрезка. При объединении максимальный префикс и суффикс пересчитываются с помощью вычисленных значений для предыдущих отрезков: обозначим левый отрезок за s, а правый — за t, тогда $maxpref_s = \max(maxpref_s, sum_s + x + maxpref_t)$, $maxsuf_t = \max(maxsuf_t, sum_t + x + maxsuf_s)$, где sum_s и sum_t обозначают сумму на отрезках s и t соответственно. Заметим, что это можно реализовать и без помощи сета, для этого в первом элементе отрезка будем хранить информацию о максимальном префиксе и о последнем элементе, а в последнем — информацию о максимальном суффиксе и первом элементе. Эта информация так же пересчитывается при объединении. Асимптотика $\varnothing(n \log n)$.

Подзадачи 6 и 7

Перейдём к полной задаче: как и ранее будем помечать подарки в порядке возрастания удовольствия. Кандидаты на ответ — отрезки с k непомеченными элементами. Также будем хранить отрезки подряд идущих помеченных элементов, как в подзадаче 5. Пусть мы зафиксировали, какие k непомеченных элемента лежат на искомом отрезке. Тогда левее самого левого надо взять максимальный суффикс среди помеченных, правее самого правого надо взять максимальный префикс среди помеченных, а между — сумму помеченных.

При рассмотрении очередного элемента x этот элемент превращается из непомеченного в помеченный, следовательно, появляются новые возможности выбрать k непомеченных элемента для отрезка. Таких возможностей не более $k + 1$ варианта. Переберём их все и обновим ответ.

Чтобы уметь для непомеченных элементов сдвигаться вперёд/назад на k шагов от заданного, будем поддерживать их в связном списке. Решения с `std::set` или другими более тяжеловесными структурами данных проходят только 6 подзадачу.

Итоговая асимптотика $O(nk + n \log n)$.