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.

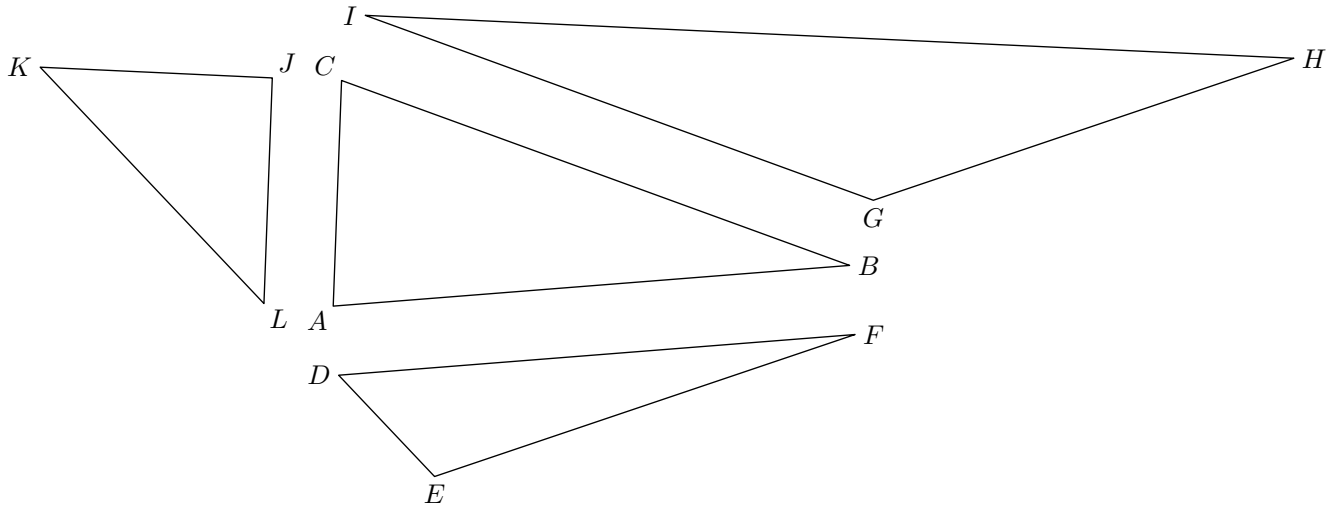
В тестах 8 и 9 треугольники может потребоваться повернуть, но легче проверить, что из четырех треугольников можно собрать один.

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.