Всероссийская олимпиада школьников по информатике 2019–2020

Региональный этап

Разбор задач

Разбор задач подготовили Андрей Станкевич, Николай Будин, Дмитрий Гнатюк, Даниил Орешников, Рамазан Рахматуллин, Иван Сафонов.

Условия задач, тесты и решения подготовили Михаил Анопренко, Николай Будин, Дмитрий Гнатюк, Александра Дроздова, Арсений Кириллов, Даниил Орешников, Рамазан Рахматуллин, Иван Сафонов, Дмитрий Саютин, Андрей Станкевич.

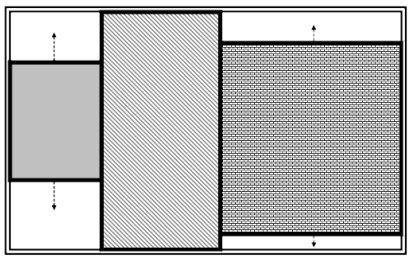
Ценные замечания по результатам тестирования задач сделали Никита Голиков, Михаил Первеев и Виктор Шарепо.

Задача 1. Транспортировка артефактов

Авторы задачи: Даниил Орешников, Андрей Станкевич

Заметим, что прямоугольники, с точностью до поворота плоскости на 90°, поворота каждого прямоугольника на 90° и их перестановки, могут быть расположены относительно друг друга двумя способами:

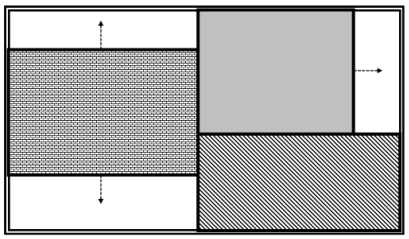
- правая сторона первого касается левой стороны второго, а правая сторона второго — левой стороны третьего



Как видно на картинке, все три прямоугольника расположены в ряд. Стрелками показано, что на самом деле прямоугольники можно двигать друг относительно друга в пределах максимума из их высот.
В этом случае прощадь баржи равна

$$(a_1 + a_2 + a_3) \cdot \max(b_1, b_2, b_3)$$

- слева находится первый прямоугольник, справа — блок из второго и третьего, при чем нижняя сторона второго касается верхней стороны третьего



Если прямоугольники расположены не в ряд, мы приходим к этой конфигурации. Не обязательно все три должны иметь общую точку, но если два касаются горизонтальными сторонами, и два — вертикальными, их можно сдвинуть, не увеличив площадь баржи.
В этом случае прощадь баржи равна

$$(a_1 + \max(a_2, a_3)) \cdot \max(b_1, b_2 + b_3)$$

Осталось только перебрать все возможные повороты прямоугольников и их перестановки. Суммарно надо перебрать $8 \cdot 6 \cdot 2 = 96$ вариантов. Это можно сделать с помощью нескольких циклов и/или ручного перебора вариантов.

Отметим, что благодаря потестовой оценке, если участник забывает рассмотреть некоторые случаи, он лишается только части баллов.

Задача 2. Скоростной транспорт

Автор задачи: Иван Сафонов

Для решения первой подзадачи можно перебрать все варианты новых границ отрезка и посчитать количество подходящих вариантов. Получится решение за $O(n^4)$.

```
long long tot = r1 - l1 + r2 - l2;
long long ans = 0;
for (long long ln1 = l1; ln1 <= r1; ln1++) {
    for (long long rn1 = ln1; rn1 <= r1; rn1++) {
        for (long long ln2 = rn1 + 1; ln2 <= l2; ln2++) {
            for (long long rn2 = r2; rn2 <= ln2 + tot; rn2++) {
                if (rn1 - ln1 + rn2 - ln2 == tot) {
                    ans++;
                }
            }
        }
    }
}
```

Заметим, что количество подходящих значений $rn2$ не более одного, вычислим его по формуле и проверим, что оно подходит. Получится решение за $O(n^3)$. Оно подходит также для второй подзадачи.

```
long long tot = r1 - l1 + r2 - l2;
long long ans = 0;
for (long long ln1 = l1; ln1 <= r1; ln1++) {
    for (long long rn1 = ln1; rn1 <= r1; rn1++) {
        for (long long ln2 = rn1 + 1; ln2 <= l2; ln2++) {
            long long rn2 = tot - (rn1 - ln1) + ln2;
            if (rn2 >= r2) {
                ans++;
            }
        }
    }
}
```

Для дальнейшей оптимизации зафиксируем $rn1$ и $ln2$. Заметим, что число подходящих пар левых границ $ln1$ и правых границ $rn2$ выражается простой формулой. Получим решение за $O(n^2)$, которое укладывается в ограничения по времени также для третьей подзадачи.

```
long long tot = r1 - l1 + r2 - l2;
long long ans = 0;
for (long long rn1 = l1; rn1 <= r1; rn1++) {
    for (long long ln2 = rn1 + 1; ln2 <= l2; ln2++) {
        ans += max(0LL, rn1 - max(l1, rn1 + l2 + l1 - r1 - ln2) + 1);
    }
}
```

Следующий уровень оптимизации, который позволяет решить все подзадачи и набрать 100 баллов: избавиться от внутреннего цикла по $ln2$. Заметим, что число, которое прибавляется к ans внутри цикла кусочно-линейно зависит от $ln2$. Суммируя арифметические прогрессии, получаем значение, которое необходимо прибавить к ответу для фиксированного $rn1$.

```
long long tot = r1 - l1 + r2 - l2;
long long ans = 0;
```

```
    for (long long rn1 = l1; rn1 <= r1; rn1++) {
        ans += (rn1 - l1 + 1) * (l2 + 1 -
                max(max(rn1 + 1, l2 + l1 - r1), rn1 + l2 - r1));
        long long f = max(rn1 + 1, l2 + l1 - r1);
        long long t = min(rn1 + l2 - r1, l2 + 1);
        ans += (-l2 - l1 + r1 + 1) * (t - f);
        ans += (f + (t - 1)) * (t - f) / 2;
    }
```

Полученное решение работает за $O(n)$ и проходит все подзадачи.

Заметим, что можно сделать последний шаг и получить решение за $O(1)$, избавившись также от последнего оставшегося цикла. В результате для получения ответа нужно будет вычислить сумму нескольких значений, каждое из которых имеет порядок куба от входных данных. К счастью, для получения полного балла по задаче этого не требовалось.

Задача 3. Продукты в экспедиции

Автор задачи: Дмитрий Саютин

В первой подзадаче у есть только один тип продуктов. Нужно просто проверить, можно ли съесть этот тип продукта. За время t_1 все участники экспедиции смогут съесть $c \cdot t_1$ порций, поэтому если $k_1 \leqslant c \cdot t_1$, то ответ 1, иначе ответ 0. Время работы этого решения $O(1)$.

Как для множества типов продуктов проверить, можно ли его съесть полностью? Пусть множество состоит из m типов продуктов, таких что время, за которое они портятся это $t_1, t_2, \ldots, t_m$, количество порций продуктов $k_1, k_2, \ldots, k_m$, соответственно. Отсортируем их по возрастанию времени, то есть пусть $t_1 \leqslant t_2 \leqslant \ldots \leqslant t_m$. Заметим, что для всех $1 \leqslant i \leqslant m$ должно быть выполнено, что $k_1 + k_2 + \ldots + k_i \leqslant c \cdot t_i$. Это так, потому что время, за которое мы сможем есть первые i типов продуктов это t_i, при этом за это время мы должны будем съесть как минимум $k_1 + k_2 + \ldots + k_i$ порций. Докажем, что это также является достаточным условием того, что мы сможем все съесть. Заметим, что $k_1 \leqslant c \cdot t_1$, поэтому за первые t_1 дней мы сможем съесть все порции продукта первого типа. Затем, так как $k_1 + k_2 \leqslant c \cdot t_2$, мы за первые t_2 дней сможем съесть все порции первого и второго типа, так как на продукты первого типа мы потратили ровно k_1 возможностей съесть одну порцию. И так будет для любого i: поскольку $k_1 + k_2 + \ldots + k_i \leqslant c \cdot t_i$, мы сможем съесть за t_i дней все продукты первых i типов, потому что мы съели все продукты первых $i - 1$ типов, потратив на это ровно $k_1 + k_2 + \ldots + k_{i-1}$ возможностей съесть оджну порцию.

Во второй подзадаче переберем все 2^n подмножеств типов продуктов. Для каждого из них, за время $O(n \log n)$ или $O(n)$ (если предварительно отсортировать все t_i) мы сможем определить, можно ли съесть все порции выбранного множества типов продуктов. Среди всех таких подмножеств найдем подмножество максимального размера, это будет ответом. Время работы этого решения $O(2^n n)$.

Для того, чтобы решить третью подзадачу, отсортируем все t_i по возрастанию. Мы должны выбрать некоторое подмножество типов продуктов, которое удовлетворяет полученному нами критерию. Обозначим за $T = t_n$ — максимальное значение t_i. По ограничениям этой подзадачи $T \leqslant 2000$. Посчитаем $dp[i][s]$ — максимальное количество типов продуктов, которое можно выбрать, так чтобы его можно было съесть, сумма значений k_j по всем продуктам из множества равна s и продукт i это максимальный номер продукта, взятый в множество. Эти значения мы будем считать для $1 \leqslant i \leqslant n$ и $1 \leqslant s \leqslant T$. Тогда $dp[i][s] = \max\limits_{1 \leqslant j < i} dp[j][s - k_i] + 1$, если $s \leqslant t_i$ и $dp[i][s] = 0$, иначе. Это позволяет нам вычислить все значения за время $O(n^2 T)$, что недостаточно чтобы решить задачу. Для того, чтобы ускорить вычисление, обозначим $pref[i][s] \max\limits_{1 \leqslant j \leqslant i} dp[j][s]$. Тогда $dp[i][s] = pref[i-1][s - k_i] + 1$ и $pref[i][s] = \max(pref[i-1][s], dp[i][s])$. Вычислим значения за время $O(nT)$ и ответом будет являться максимум $dp[i][s]$ по всем $1 \leqslant i \leqslant n, 1 \leqslant s \leqslant T$.

Пропустим четвертую подзадачу и разберем сначала пятую. В этой подзадаче все значения t_i совпадают. Обозначим их за t. Тогда в этом случае критерием того, что мы можем съесть все продукты, количества которых $k_1, k_2, \ldots, k_m$ является одно неравенство, а именно $k_1 + k_2 + \ldots + k_m \leqslant c \cdot t$.

То есть мы хотим взять как можно больше типов продуктов, таких, что сумма их количеств не превышает $c \cdot t$. Для этого можно воспользоваться жадным алгоритмом: отсортируем все значения k_i и будем набирать их от меньшего к большему, пока сумма не превысит $c \cdot t$. Асимптотика решения этой подзадачи $O(n \log n)$.

Далее приведем концепцию полного решения. Отсортируем все типы продуктов по убыванию времени, за которое они портятся, то есть $t_1 \geqslant t_2 \geqslant \ldots \geqslant t_n$. Будем перебирать i от 1 до n и смотреть, какие продукты надо есть в обратном порядке времени. Будем хранить множество типов продуктов и для каждого типа сколько еще порций этого типа осталось. Перебирая i добавим тип продукта i и то, что осталось k_i порций этого типа. Теперь до следующего добавления типа продукта мы можем съесть $c \cdot (t_i - t_{i-1})$ порций (положим $t_0 = 0$). Заметим, что теперь выгодно начать есть те типы продуктов, которых осталось меньше всего, потому что разные типы продуктов к этому моменту равны между собой и невыгодно есть тип, которого больше. Поэтому будем есть продукты в порядке возрастания оставшихся порций этого типа, пока количество порций, которые мы можем сейчас съесть не будет равно 0 или пока все типы продуктов не кончатся. Если мы доедаем какой-то тип продуктов, то добавим его индекс в ответ. Таким образом мы получим множество типов продуктов максимального размера, которое можно съесть.

Для решения четвертой подгруппы можно наивно реализовать это решение — будем хранить массив типов продуктов и количество оставшихся порций. При выборе типа продукта, который надо съесть, линейным поиском найдем минимальное количество и будем есть порцию этого типа. Тогда реализация этого решения будет работать за время $O(n^2)$.

Для полного решения задачи будем хранить типы продуктов в структуре данных s типа, например, `std::set`, сортируя их по количеству оставшихся порций этого типа. При добавлении нового типа, просто положим его в s, если мы хотим съесть $c \cdot (t_i - t_{i-1})$ порций, будем вынимать из s тип продукта с минимальным количеством оставшихся порций и есть. Если количество оставшихся порций будет равно 0, то не будем возвращать этот тип в s, иначе вернем с новым количеством оставшихся порций. Суммарное время работы такого решения будет равно $O(n \log n)$, потому что каждый тип продуктов будет удален из s не более одного раза и мы n раз будем добавлять тип в s и n раз обновлять количество оставшихся порций.

Задача 4. Доставка почты

Авторы задачи: Михаил Ютман, Николай Будин

Немного переформулируем задачу. Дано дерево, подвешенное за вершину номер 1. У каждой вершины можно выбрать произвольный порядок детей, после чего, выполняется обход дерева из вершины 1, и все вершины выписываются в порядке первого посещения. Требуется посчитать количество способов выбрать порядок детей у всех вершин, чтобы вершина a_i была выписана раньше вершины b_i для всех i.

Пусть вершина a должна быть выписана раньше, чем b. Если a является предком b, это произойдет в любом случае. Аналогично, если b является предком a, это не произойдет никогда.

Иначе, рассмотрим c — наименьшего общего предка вершин a и b — наиболее глубокую вершину, являющуюся одновременно предком и a, и b. Обозначим за a' ребёнка вершины c, который является предком вершины a, и за b' — ребёнка c, являющегося предком b. Несложно заметить, что a' и b' существуют, единственны и не равны. Тогда a будет выписано до b, если из вершины c переход в a' будет выполнен раньше, чем в b'. Поэтому, требуется посчитать количество способов упорядочить детей вершин, чтобы не нарушалось ни одно из таких условий. Так как условия между вершинами независимы, можно посчитать количество способов упорядочить детей отдельно для каждой вершины, и затем перемножить результаты. Для того, чтобы решить задачу для отдельной вершины, нужно посчитать количество топологических сортировок в ориентированном графе, вершинами которого являются дети данной вершины, а рёбрами — наложенные ограничения.

Для нахождения наименьшего общего предка двух вершин, можно использовать либо наивный алгоритм, работающий за линейное время, что соответствует подгруппам 3–5. Либо любой оптимальный алгоритм, например, отвечающий на один запрос за время $O(\log n)$, что соответствует подгруппам 6–7.

Для нахождения количества топологических сортировок графа, содержащего не более D вершин, можно либо разобрать вручную все случаи для $D = 3$, либо перебрать все перестановки вершин для $D = 7$, либо использовать метод динамического программирования по подмножествам для $D = 12$.

Задача 5. Разность квадратов

Автор задачи: Даниил Орешников

Во-первых заметим, что числа 1, 2 и 4 невозможно представить в виде разности двух квадратов.

Рассмотрим решения для первых двух подзадач: его время работы $O(n)$.

Переберем все возможные числа x из интервала $(1, 2^{10})$. Зная уменьшаемое и разность, найдем вычитаемое. Если число является квадратом, то ответ найден. Если ни одно число из вычитаемых не является квадратом, то ответа нет.

Для подзадачи 3 одно из решений такое: заметим, что $x^2 - y^2 = (x - y) \cdot (x + y)$. Переберём делители n, для каждого делителя $p \leqslant \sqrt{n}$ проверим, есть ли решение в целых числах для системы уравнений

$$\begin{cases} x - y = p \\ x + y = n/p \end{cases}$$

Если такое решение есть, мы нашли ответ.

Интересно, что это решение легко модифицируется до $O(1)$. Если x и y одинаковой четности, то их разность квадратов делится на 4, а иначе не делится на 2. То есть ответ для n, которые делятся на 2, но не делятся на 4 - «`No`».

Если n нечетное, то давайте попробуем решить такую систему уравнений:

$$\begin{cases} x - y = 1 \\ x + y = n \end{cases} \quad \begin{cases} 2 \cdot x = n + 1 \\ x + y = n \end{cases} \quad \begin{cases} x = \frac{n+1}{2} \\ x + y = n \end{cases} \quad \begin{cases} x = \frac{n+1}{2} \\ y = \frac{n-1}{2} \end{cases}$$

То есть ответ для нечётного n всегда есть, если $n \geqslant 3$.

Если n делится на 4, то решим такую систему:

$$\begin{cases} x - y = 2 \\ x + y = \frac{n}{2} \end{cases} \quad \begin{cases} 2 \cdot x = \frac{n}{2} + 2 \\ x + y = \frac{n}{2} \end{cases} \quad \begin{cases} x = \frac{n+4}{4} \\ x + y = n \end{cases} \quad \begin{cases} x = \frac{n+4}{4} \\ y = \frac{n-4}{4} \end{cases}$$

То есть ответ всегда существует, если n делится на 4 и больше 4.

Заметим, что решение с перебором делителей автоматически найдёт указанные решения, если n не даёт остаток 2 при делении на 4. Таким образом для его доработки достаточно рассмотреть указанный случай.

Наконец, не забудем, что 0 можно получить например из $x = 1, y = 1$

Задача 6. Превышение скорости

Автор задачи: Андрей Станкевич

Предположим, максимальное превышению скорости автомобиля на дороге не превышает d. Это значит, что на i-м участке автомобиль ехал со скоростью не выше $v_i + d$ и проехал его за время не меньше $\frac{l_i}{v_i+d}$. Общее время, за которое автомобиль проедет дорогу не меньше

$$\sum_i \frac{l_i}{v_i + d}.$$

Таким образом, превышение не больше чем на d возможно, если

$$s + \sum_i \frac{l_i}{v_i + d} \geqslant t.$$

Решение за $O(nmq)$: для каждой верхней границы отрезков штрафов проверим, возможно ли, чтобы автомобиль превышал не более чем на d. Заметим, что если для некоторого i невозможно,

чтобы превышение было не больше a_{i-1}, то гарантировано можно назначить автомобилю штраф f_i. Из всех таких значений надо выбрать максимальное возможное.

Заметим, что проверку можно делать с использованием вещественной арифметики, не опасаясь проблем с точностью, благодаря ограничению в условии, что небольшое изменение времени въезда или выезда во входных данных не может изменить штраф.

Это решение проходит все подзадачи, кроме подзадачи 7.

Чтобы решить подзадачу 7, заметим, что свойство «можно проехать с превышением не больше d» является монотонным: если можно проехать с превышением не больше d, то можно проехать с превышением не больше d' для всех $d' \geqslant d$. Поэтому для определения максимального возможного штрафа можно применить двоичный поиск. Полученное решение работает за $O(nq \log m)$.

Отметим также частичные решения, решающие некоторые подзадачи.

Для решения подзадач с $n = 1$ можно воспользоваться тем, что в этом случае мы можем легко вычислить превышение: $d = \max(0, (t - s)/l_1)$.

В подзадачах с $m = 1$ требуется лишь проверить, есть ли превышение. Для этого можно вычислить, за какое время может проехать дорогу автомобиль, соблюдающий ограничения скорости.

В подзадаче 5 можно применить линейный поиск вместо двоичного, перебрав значения d превышения от 0 до 10.

Задача 7. Борьба с рутиной

Автор задачи: Сергей Шедов

Для решения первых двух подзадач можно непосредственно реализовать определение из условия. Переберём d, для заданного d переберём все отрезки длины d. Для каждого отрезка переберём все числа на нём и посчитаем число различных.

В подзадаче 1 для этого можно использовать массив элементов типа `bool`, отмечая встретившиеся значения, в подзадаче 2 встретившиеся значения можно сложить, например, в `std::set`, либо в массив, отсортировать и найти число различных значений. Наконец, отметим, что поскольку $n \leqslant 50$, можно обойтись даже без сортировки, для каждого элемента проверяя, не встречался ли он раньше.

Решение с `std::set` при аккуратное реализации может также пройти подзадачу 3, альтернатива — отсортировать значения на отрезке с помощью `std::sort` и посчитать количество различных значений.

Улучшим это решение.

Зафиксируем длину отрезка d и посмотрим для каждого числа c в массиве то, в скольких отрезках оно не встречается. Посмотрим на два соседних вхождения числа, если расстояние между ними $x \geqslant d$, то существует ровно $d-x+1$ отрезок, лежащий между ними. Просуммировав эту величину по всем парам соседних вхождений числа, а также между началом массива и первым числом и между последним числом и концом массива, получим количество отрезков, которые не включают в себя ни одно вхождение данного числа.

Обозначим эту величину за $D(d, c)$. Тогда ответ для заданной длины отрезка — это $\sum\limits_c ((n - L + 1) - D(d, c))$. Если посчитать эту сумму наивно, получим решение за $O(n^2)$, которое проходит подзадачи 3–5. Заметим, что для решения подзадачи 4 этим методом не требуется перебор значений, которые встречаются в массиве, можно перебирать значения от 1 до 5000.

Для получения полного решения этим методом нужно избавиться от суммирования по c для каждого d. Для этого запишем все значения расстояний между соседними вхождениями c в один массив a для всех c. Тогда

$$\sum_c ((n - L + 1) - D(d, c)) = \sum_i \max(a[i] - d + 1, 0).$$

Чтобы посчитать эту сумму можно найти префиксные суммы массива a_i и двоичным поиском находить позицию, где $\max(a[i]-d+1, 0)$ начинает равняться 0 для данного d. Альтернативно можно заметить, что эта позиция только растёт при росте d.

Задача 8. Олимпиада для роботов

Авторы задачи: Александра Дроздова, Рамазан Рахматуллин

Отметим три важных свойства операций `and` и `or`: сохранение 0, сохранение 1 и монотонность:

- 0 `and` $0 = 0$ и 0 `or` $0 = 0$, поэтому если все входные переменные равны 0, то значение любой функции, которую можно задать БМЛП, равно 0.
- 1 `and` $1 = 1$ и 1 `or` $1 = 1$, поэтому если все входные переменные равны 1, то значение любой функции, которую можно задать БМЛП, равно 1.
- Если увеличить значение аргумента, то значение `and` и `or` не уменьшается. Значит если входное значение изменяется с 0 на 1, то значение функции, которую вычисляет БМЛП либо не изменяется, либо изменяется с 0 на 1.

Изначально положим $z_i = 0$. Поскольку $x < 0$ ложно для всех неотрицательных x, все начальные значения val будут 0, а значит в силу сохранения 0 все значения функций равны 0.

Изменим одно из значений $z_i < m$ на $z_i + 1$. Заметим, что в этом столбце ровно одно значение $x_{j,i}$ равно z_i. Только у j-й булевой функции изменятся входные переменные, причем 0 заменится на 1, а значит в силу монотонности количество программ, возвращающих единицу, либо не изменится, либо увеличится на один.

Наконецц, если все $z_i = m$, то все входные переменные равны 1 и значение всех функций равно 1.

Сделав эти наблюдения, получаем первое решение задачи. Будем последовательно выбирать любое значение $z_i < m$ и увеличивать его на один. После этого будем пересчитывать количество программ, возвращающих единицу, то в какой-то момент мы обязательно получим s программ. Такая наивная реализация работает за $O((nm)^2)$ и решает подзадачи 1 и 3.

Заметим, что поскольку входные данные изменяются только для одной функции, то можно пересчитывать значение только для неё. Время работы усовершенствованного решения равно $O(mn^2) = O((nm)n)$, поэтому оно решает подзадачи с небольшим n: 1, 2, 3 и 6.

Посольку порядок изменений z_i не важен, можно, например, сначала увеличивать z_1, пока оно не станет равно m, потом z_2 и так далее. Теперь для зафиксированного i можно сделать двоичный поиск по количеству изменений. Полученное решение работает за $O(nm \log nm)$ и при достаточно оптимальной реализации может пройти тесты для всех подзадач.

Рассмотрим теперь решение без двоичного поиска. Для каждой инструкции нас интересует первый момент, когда она станет равна единице. Заметим, что результат каждой инструкция, кроме последней в каждой программе, является входом ровно одной другой инструкции. Поэтому, когда мы увеличиваем z_i, мы изменяем ровно одно выражение ($x_{j,i} < z_i$) с нуля на единицу, которое в свою очередь может изменить выход той инструкцию, в которой это выражение было входом, и так далее. Следуя этой логике, будем рекурсивно подниматься по дереву разбора программы и изменять выходы инструкций с нуля на единицу, пока не встретим единицу или не прекратим изменение. Так как каждая инструкция будет изменена не более одного раза, итоговое время работы составит $O(nm)$. В этом решении используется бесповторность программы: факт, что каждая ячейка массива val используется как вход ровно один раз.