Всероссийская олимпиада школьников по информатике 2019–2020

Региональный этап

Разбор задач

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

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

Ценные замечания по результатам тестирования задач сделали Никита Голиков, Александра Дроздова, Илья Збань, Павел Кунявский, Михаил Первеев и Виктор Шарепо.

Задача 1. 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 2. 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 3. 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 $\sum_c ((n - d + 1) - D(d, c))$. 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

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

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 4. 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.

Наконец, 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.

Поскольку 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.

Задача 5. Максимальное произведение

Автор задачи: Даниил Орешников

В первой подзадаче можно просто перебрать все возможные точки разреза, для каждой найти сумму элементов, перемножить и выбрать оптимальное значение. При этом сумму можно каждый раз вычислять заново, получив время работы $O(n^2)$. Благодаря тому, что сумма элементов всего массива не превышает 10^9, все действия можно безопасно выполнить в 64-битном типе данных.

Для решения четвертой подзадачи можно заметить, что вместо суммирования элементов заново, каждый раз при перемещении границы вправо на один элемент можно пересчитывать сумму за $O(1)$. Следовательно решение работает за $O(n)$.

Для решения остальных подзадач метод просуммировать элементы и перемножить не подходит — произведение получается слишком большое и не влезает в 64-битный тип данных. Заметим, что решения на языке Python или в случае доступности 128-битного типа данных, в принципе, лишены этого недостатка. Но есть решение, которое не выполняет операций умножения.

Решим сначала подзадачи, где все значения равны. Пусть массив поделен на части размера k и l. Заметим, что произведение равно $(k \cdot a) \cdot (l \cdot a)$, где a — значение элементов массива, а $k+l = n$. Так как $(ka)(la) = (kl)a^2 = k(n-k)a^2$, достаточно максимизировать величину $k(n-k)$. Эта величина максимальна, когда k примерно равно $n/2$, а именно равно $n/2$ при чётном n и $(n \pm 1)/2$ при нечётном n. Таким образом получаем решение за $O(1)$: вывести $\lfloor n/2 \rfloor$. Это решение подходит для второй и пятой подзадачи, также можно не находить математически максимум $k(n-k)$, а перебрать значения k. Во второй подзадаче для вычисления $k(n-k)$ при этом достаточно 32-битного типа данных.

Наконец, для решения на полный балл заметим, что идея, что значение $k(n-k)$ максимально при k в районе $n/2$ обобщается следующим образом. Пусть s — сумма всех элементов массива, а t_k — сумма первых k элементов. Тогда произведение $t_k(s-t_k)$ максимально, когда t_k максимально близко к $s/2$. Можно перебирать k и вычислять t_k за $O(n)$ в первых трёх подзадачах, либо вычислять t_k за $O(1)$ с помощью префиксных сумм, или пересчитывать за $O(1)$ при увеличении k на 1, получая полное решение.

Задача 6. Планировка участка

Автор задачи: Андрей Станкевич

Заметим, что если $a > n$, то $ab - cd > ab - ad = a(b-d) \geqslant a > n$ и никакие значения b, c и d не подходят. Следовательно $a \leqslant n$. Аналогично доказывается, что $b \leqslant n$.

Переберём a и b от 2 до n, а также c от 1 до $a-1$ и d от 1 до $b-1$. Проверив для каждого варианта, что $ab - cd = n$, получим решение первой подзадачи за $O(n^4)$, которое несложно модифицируется, чтобы убеждаться, что $a \neq x$ и $b \neq x$, тем самым решая подзадачу 2.

Пусть мы зафиксировали a, b и c. Тогда заметим, что d определяется единственным образом из уравненя $ab - cd = n$: если $ab - n$ кратно c, то $d = (ab-n)/c$, иначе никакие значения d не подходят. Следует также не забыть проверить, что $1 \leqslant d < b$. Получаем решение за $O(n^3)$, оно подходит для подзадачи 3, а добавляя проверки $a \neq x$ и $b \neq x$, также и для подзадачи 4.

Для решения подзадачи 5 можно применить забавный трюк. Давайте подсчитаем предыдущим алгоритмом за $O(n^3)$ решения для всех возможных значений n. Решение за $O(n^4)$ работает очень долго для n порядка 3000, но у нас почти все время тура. Все полученные значения поместим в константный массив. Такой метод известен как *предподсчёт*. К сожалению, сделать предподсчёт для шестой подзадачи не получается, это и слишком долго, и занимает слишком много места, такой большой исходный файл не удастся отправить на проверку.

Наконец, решение за $O(n^2 \log n)$, которое решает задачу полностью.

Вместо c будем перебирать $a - c$. Заметим, что $ab - cd = ab - bc + bc - cd = (a-c)b + c(b-d)$. Поскольку $c > 0$ и $b > d$, второе слагаемое положительно, и следовательно $(a-c)b < n$. Таким

образом, достаточно перебирать только $O(n/b)$ значений. Значит для каждого a будет перебираться $n+n/2+n/3+\ldots+n/n = n(1+1/2+1/3+\ldots+1/n)$ пар значений. Сумма $1+1/2+1/3+\ldots+1/n$ хорошо известна — это частичная сумма гармонического ряда, она равна $O(\log n)$, следовательно суммарно для фиксированного a перебирается $O(n \log n)$ значений, а общее время работы алгоритма равно $O(n^2 \log n)$. Финальное замечание, что так как a и b перебираются явно, нет проблемы убедиться, что $a \neq x$ и $b \neq x$.

Задача 7. Банкомат

Автор задачи: Даниил Орешников

Рассмотрим сначала решение первых четырех подзадач, в которых $q \leqslant 5$. В этом случае каждый запрос можно обрабатывать независимо от других, поэтому проигнорируем факт, что у нас несколько значений b_i и будем решать задачу для одного значения b.

В первой подзадаче $b \leqslant 500$, $n \leqslant 500$, поэтому можно применить динамическое программирование. Обозначим как $dp[i]$ количество купюр, которое будет выдано, если запросить сумму i. Тогда $dp[0] = 0$, и если $m(i)$ — максимальный номинал купюры, которая не превосходит i, то $dp[i] = dp[i - m(i)] + 1$. Ответ — максимальное значение среди $dp[1] \ldots dp[b]$. Решение работает за за $O(nb)$.

Чтобы распространить описанное решение на третью подзадачу, избавимся от множителя n в времени работы. Для этого заметим, что $m(i)$ только возрастает при возрастании i, поэтому в процессе вычисления $dp[i]$ можно поддерживать текущее значение $m(i) = a_j$ и при переходе к следующему значению i проверять, нельзя ли перейти к следующему номиналу a_{j+1}. Время работы будет $O(b+n)$ и третья задача подрешена.

Во второй подзадаче номиналы купюр являются последовательными степенями двойки. Заметим, чтобы выдать сумму c банкомат выдаст купюры с номиналами, соответствующими позициям единиц в двоичной записи числа c. Значит задача свелась к поиску числа, не превосходящего b, содержащего максимальное число единиц в двоичной записи. Пусть двоичная запись числа b содержит k разрядов. Тогда число единиц в ответе не превышает k, и число $2^{k-1} - 1$ заведомо меньше b и содержит ровно $k - 1$ двоичную единицу. Значит ответ — либо b, либо $2^{k-1} - 1$, получившееся решение работает за $O(\log b)$.

Решение четвертой подзадачи уже приближает нас к полному решению: ограничения на n, a_i и b_i в ней максимальны и только $q \leqslant 5$. Научимся решать задачу для одного запроса b за $O(n)$. Правильное решение основывается на жадном алгоритме.

Сделаем инверсию задачи. Пускай mn_x — минимальное число c такое, что банкомат выдаст x купюр для суммы c.

Будем по-прежнему обозначать как $m(v)$ максимальный номинал купюры, не превышающий v. Докажем, что $mn_x = mn_{x-1} + m(mn_x)$.

С одной стороны $mn_{x-1} \leqslant mn_x - m(mn_x)$, поскольку из набора купюр, дающего сумму mn_x, всегда можно убрать самую большую купюру (равную $m(mn_x)$), и получим корректный набор из $x-1$ купюр, дающих в сумме $mn_x - m(mn_x)$. С другой стороны, предположим, что мы зафиксировали x, и $mn_{x-1} < mn_x - m(mn_x)$. Можно заметить, что в этом случае величина $mn_{x-1} + m(mn_x)$ меньше, чем mn_x, и банкомат вернет при запросе такой величины ровно x купюр — одну купюру $m(mn_x)$ (она максимальная, не превышающая этой суммы), и еще $x - 1$ купюр, на которые раскладывается сумма mn_{x-1}. Противоречие, исходное утверждение доказано.

Используя это наблюдение, можно посчитать все значения mn_x за время $O(n)$. Заметим, что для двух последовательных чисел a_i, a_{i+1} для всех значений x таких, что $m(mn_x) = a_i$ (иными словами, самая большая купюра, если банкомат выдаёт x купюр, равна a_i), значения mn_x образуют арифметическую прогрессию с разностью a_i и значениями в промежутке $[a_i; a_{i+1})$. Посчитав такую арифметическую прогрессию для промежутка $[a_i; a_{i+1})$, можно перейти к промежутку для $[a_{i+1}; a_{i+2})$: если последнее значение в прогрессии равно y ($y < a_{i+1}$), то следующая прогрессия начнется с y, и будет состоять из чисел $y + k \cdot a_{i+1}$, меньших a_{i+2}. Чтобы найти все эти числа и их количество можно просто поделить длины соответствующих отрезков нацело на a_{i+1}. Таким образом, последнее число mn_x, не превышающее b, и соответствующее значение x являются ответом на задачу.

Для решения последней группы можно заметить, что не обязательно решать независимо задачу для разных b_j, а достаточно найти двоичным поиском последнее $a_i < b_j$, и посмотреть на соответствующую арифметическую прогрессию из значений mn_x.

Задача 8. Плакаты

Автор задачи: Алексей Толстиков

В первой подзадаче можно просто перебрать, кто из друзей поднимет плакаты, для каждого варианта проверить, что нет трёх поднятых подряд плакатов, и посчитать суммарную красочность поднятых плакатов. Время работы решения $O(2^n \cdot n)$.

Во второй подзадаче можно реализовать тот же алгоритм, но повторить его q раз, после каждого изменения. Время работы $O(2^n \cdot nq)$.

Третья и четвертая подзадача решаются с помощью динамического программирования. Заметим, что друзья 1, 2, 3 и 4 не могут одновременно поднять плакат. Переберём, кто из них не поднял плакат, и переставим его и друзей, которые идут до него, в конец нумерации. Теперь стоящий последним друг не поднял плакат, поэтому из задачи на окружности мы получили задачу на прямой. Используем для её решения динамическое программирование: обозначим как $dp[i][j]$ максимальную суммарную красочность плакатов, поднятых первыми i друзьями, причём последние j из них подняли плакат ($0 \leqslant j \leqslant 3$). Тогда $dp[i][0] = \max(dp[i-1][0], dp[i-1][1], dp[i-1][2], dp[i-1][3])$, а для $j > 0$ выполнено $dp[i][j] = dp[i-1][j-1] + a_i$.

В решении третьей подзадачи запустим этот алгоритм $q+1$ раз, для начальной позиции и после каждого изменения, получив решение за $O(nq)$. В четвертой подзадаче один запуск этого алгоритма работает за $O(n)$.

Наконец, для полного решения задачи необходимо научиться изменять величину красочность плакатов и пересчитывать значения $dp[i][j]$. Потребуется чуть более сложная конструкция: построим дерево отрезков на индексах от 1 до n. В узле, соответствующем отрезку от l до r будем хранить матрицу 4×4: $best[a][b]$ равно максимальной суммарной красочности плакатов, поднятых друзьями, с номерами от l до r, в начале a стоящих подряд друзей подняли плакат, а в конце — b подряд стоящих друзей. Тогда объединение двух соседних отрезков происходит по алгоритму, аналогичному пересчёту значений динамического программирования в одной из предыдущих задач. Изменяя значение красочности i-го плаката, нам необходимо пересчитать $O(\log n)$ значений в узлах на пути к листу, соответствующему отрезку от i до i.

Время работы получившегося решения есть $O(n + q \log n)$, напоследок отметим, что, несмотря на небольшие по меркам получившейся асимптотики ограничения, пересчёт значения в узле дерева отрезков получился довольно трудоёмким, поэтому константа, спрятанная в O довольно велика, и для того, чтобы уложиться в ограничение по времени, необходима достаточно аккуратная реализация.