Разбор задачи «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 $x = 1, y = 1$

Разбор задачи «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.

Разбор задачи «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 $\sum_c ((n - L + 1) - D(d, c))$. 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.

Разбор задачи «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.