

Задача А. Снеговика

Подзадача 1

При $n \leq 3$ есть всего несколько случаев, какие пары получившихся снеговиков в оригинале были одним снеговиком. Разобрать все такие случаи не составляет труда. В частности, при $n \leq 2$ ответом всегда будет сумма всех высот полученных снеговиков.

Подзадача 2

В этой и следующих подзадачах нужно уметь проверять, могла ли высота исходных снеговиков принимать некоторое фиксированное значение H . Для этого можно идти по порядку слева-направо и если высота очередного снеговика не равна H , то он должен быть объединён со следующим снеговиком. При этом, если сумма их высот отличается от H , то такой вариант не подходит. В данной подзадаче возможных значений H всего 3, ибо очевидно, что $H \leq 3$. Сложность решения будет $O(n)$.

Подзадача 3

По аналогии с предыдущей задачей нужно найти максимальную из подходящих высот. Легко видеть, что достаточно проверить значения $H = h_i$ (если хотя бы один снеговик не развалился) и значения $H = h_i + h_{i+1}$ (если снеговик развалился на два соседних). Сложность решения $O(n^2)$.

Подзадача 4

Развивая идею предыдущей подзадачи, можно заметить, что достаточно проверить значения $H = h_1$ и $H = h_1 + h_2$, поскольку у нас есть всего два варианта, как мог образоваться первый из увиденных снеговиков. Сложность решения $O(n)$.

Задача В. Достойные последовательности

Подзадача 1

При $Y = 0$ мы получаем геометрическую прогрессию. Тогда возможные значения для a_2 — числа, которые можно получить из N делением на X один или более раз. Количество таких чисел $C(N, X)$ легко найти последовательным делением N на X (пока делится). При этом значение a_1 может быть любым, то есть ответ нужно домножить на N . Также нужно отдельно рассмотреть случаи $X = 0$ и $X = 1$ (это просто).

Подзадача 2

Здесь исходная последовательность разбивается на две независимые геометрические прогрессии. Тогда по формуле включений и исключений ответ на задачу — количество подходящих прогрессий, умноженное на 2 (поскольку знаменатели обеих прогрессий равны Y), за вычетом случаев, когда число N находится в обеих прогрессиях. По аналогии с первой подзадачей количество нужных прогрессий считается через величину $C(N, Y)$ и ответ на задачу равен $N \cdot C(N, Y) \cdot 2 - C(N, Y)^2$.

Подзадача 3

Ограничения в этой подзадаче позволяют перебрать все допустимые значения a_1 и a_2 . При этом надо сгенерировать все элементы последовательности, пока два последних не будут превышать N (в силу неубывания элементов последовательности дальше генерировать новые значения будет бесполезно). При $X, Y \geq 1$ мы получим, что значения элементов растут экспоненциально (не медленнее, чем числа Фибоначчи), поэтому потребуется $O(\ln(N))$ элементов максимум. Итоговая сложность будет $O(N^2 \cdot \ln(N))$ на каждый тест.

Подзадача 4

При увеличении N придется перейти к другому подходу. По аналогии с предыдущей подзадачей заметим, что в последовательности может быть не более $O(\ln(N))$ элементов, не превосходящих N . Тогда можно перебрать позицию, на которой будет находиться элемент со значением N . Для каждой позиции можно посчитать, с какими коэффициентами в неё войдут значения a_1 и a_2 , просто промоделировав поведение последовательности. В частности, для позиции 1 коэффициенты будут равны $(1, 0)$, для позиции 2 — $(0, 1)$, для позиции 3 — (Y, X) , для позиции 4 — $(X \cdot Y, X^2 + Y)$ и так далее. Итого, для каждой возможной позиции i мы получим целочисленные коэффициенты p_i и q_i . Далее нам потребуется решить уравнение:

$$a_1 \cdot p_i + a_2 \cdot q_i = N$$

Это стандартное диофантово уравнение и нам нужно найти количество решений, в которых $0 \leq a_1, a_2 < N$. В этой подзадаче можно перебрать один из возможных коэффициентов, например, a_1 , после чего найти значение a_2 из получившегося линейного уравнения и проверить, что оно удовлетворяет неравенству. Итоговая сложность $O(N \cdot \ln(N))$ на каждый тест.

Также есть один случай, который нужно учесть отдельно: существуют последовательности с $X, Y > 0$, в которых число N встречается 2 раза. Например, при $N = 6, X = 1, Y = 2, a_1 = 3, a_2 = 0$, мы получим последовательность $3, 0, 6, 6, 18 \dots$. Общий вид таких последовательностей предлагаем найти самостоятельно. **Примечание:** для нахождения этого случая очень помогает стресс-тестирование с решением из предыдущей подзадачи.

Подзадача 5

Подход аналогичен предыдущей подзадаче, но нужно научиться эффективно находить количество решений диофантова уравнения. Здесь также достаточно использовать стандартный подход при решении таких уравнений. Сначала нужно найти одно из решений (A_1, A_2) (если они существуют) при помощи расширенного алгоритма Евклида за $O(\ln(N))$. Далее решения будут идти с фиксированным шагом и иметь вид $(A_1 + k \cdot q_i, A_2 - k \cdot p_i)$ (при взаимно-простых p_i и q_i) для любого целого k . Тогда границы значений k , которые подходят для приведения коэффициентов в диапазон $[0, N)$ можно найти за $O(1)$. Общая сложность решения будет $O(\ln(N)^2)$ на каждый тест.

Задача С. Соты

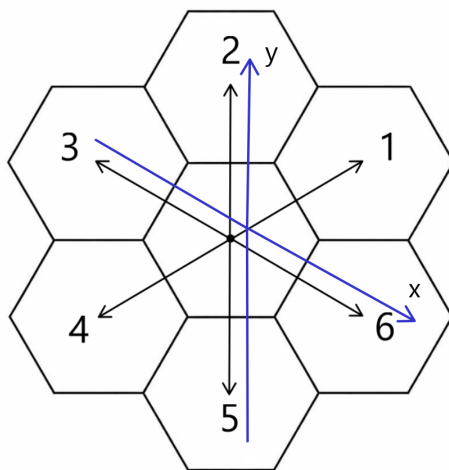
Подгруппа 1

Для решения первой подгруппы достаточно вернуться тем же путём, заменяя все направления в строке s на противоположные.

Решение за $O(n)$.

Подгруппа 2

Поскольку $n \leq 5$ и $m \leq 5$, то можно перебрать все возможные маршруты от соты B до соты A . Чтобы было удобнее ориентироваться на шестиугольной сетке, можно ввести оси, например, следующим образом:



Тогда перемещение по направлению 1 будет соответствовать сдвигу $(+1, +1)$, по направлению 2 : $(0, +1)$, по направлению 3 : $(-1, 0)$, по направлению 4 : $(-1, -1)$, по направлению 5 : $(0, -1)$, а по направлению 6 : $(+1, 0)$.

Решение за $O(6^n)$.

Подгруппа 3

В третьей подгруппе строка s задаёт кратчайший путь до соты B . Если n строго меньше m , то ответа не существует. Иначе можно доказать, что существует путь длины ровно n . Если n строго больше m , найдем соту B и соту, предшествующую ей в пути. Найдётся непосещённая сота, через которую можно пройти, увеличив путь на 1. Выполним «обход» через неё. Далее повторим всё сначала, выберем соту B и новую соту достроенного пути, предшествующую ей. Точно также соединим

их, снова увеличив путь на 1. Таким образом, спустя нужное количество повторений получим путь нужной длины.

Подгруппа 4

Для начала нужно научиться строить кратчайший маршрут из одной соты в другую. Раз мы возвращаемся в соту A с координатами $(0, 0)$, то длина кратчайшего пути (обозначим переменной d) в, введённой во 2-й подгруппе, системе координат — это $\max(|x|, |y|)$ при x и y одного знака или $|x| + |y|$, если x и y разных знаков.

Если значение d больше, чем значение n , то Вася не успеет вернуться в соту B и нужно вывести -1 . Иначе можно построить ответ длины ровно n . Существует несколько способов его построения. Например, построим предварительно кратчайший путь из соты B в соту A , а затем, подобно решению в 3 подгруппе будем расширять путь на 1 дополнительный шаг за каждую итерацию.

Подгруппа 5

Можно выполнить поиск в ширину из клетки B . Переход разрешён в любую соседнюю клетку, если она ещё не посещалась и из неё можно добраться до A не более чем за оставшееся число шагов. Так как глубина поиска не превышает 100, такой алгоритм укладывается в ограничения.

Подгруппа 6

Снова обратимся к решению 3-й подгруппы. Поскольку теперь длина пути из A в B короче, чем обратного из B в A , то ответ всегда существует и достаточно увеличить длину исходного пути на нужную разность, а затем пройти по новому пути длины n в обратном направлении. Идея, описанная в 3 подгруппе, здесь не всегда применима, так как из клетки B и предшествующей ей теперь может не существовать «обходного» маршрута (если, например, все соседние клетки уже заняты). Чтобы исправить это, будем расширять путь не из соты B , а из самой верхней, а затем самой левой соты на пути. Тогда можно будет сделать достаточное количество шагов в направлении влево-вверх.

Задача D. Сортировочная машина

Подзадача 1

В этой подзадаче можно рассмотреть каждый случай отдельно. При $n = 1$ ответ 1. При $n = 2$, если массив отсортирован, то ответ 1, иначе ответ 2.

При $n = 3$: если массив отсортирован, то ответ 1; если максимальный элемент стоит на первой позиции или минимальный элемент стоит на последней позиции, то ответ 3; иначе ответ 2.

Подзадача 2

Создаем копию массива, сортируем, пусть новый массив будет b .

Перебор k от 1 до n . Для каждого k последовательно сортируем подотрезки длины k , и объединяем их в массив c . Если $c = b$, то данный k подходит.

Решение $O(n^2 \cdot \log(n))$

Подзадача 3

Если все элементы равны, то ответ 1.

Иначе, последняя единица и первая двойка должны лежать в одном подотрезке, иначе при сортировке в сортировочной машине двойка окажется впереди единицы и итоговый массив не будет отсортирован.

Подзадача 4

Давайте переберем все блоки длины k от 1 до n , тогда при фиксированном k будет $\frac{n}{k}$ блоков, суммарно $n \log n$. У каждого блока есть своя правая и левая граница, тогда чтобы блок подходил, все элементы внутри должны быть не меньше, чем все элементы из предыдущих блоков и не более, чем все элементы из следующих блоков. Сделаем копию исходного массива и отсортируем его. Теперь, зная границы блока, можно узнать, в каком диапазоне находятся числа в нем. Теперь для каждого блока нужно проверить, что все элементы больше, чем элемент на правой границе блока, впервые встречаются правее, и наоборот, что все элементы меньше, чем элемент на левой границе, в последний раз встречаются левее. Для этого можно перебрать каждое число (их не более 1000 по ограничениям подзадачи). Если все блоки для определенного k подходят, то наименьший такой k и будет ответом.

Полное решение

Используем подход из предыдущей подгруппы, только для определения пригодности блока посчитаем префикс-максимумы и суффикс-минимумы, после этого проверка пригодности блока будет происходить за $O(1)$. Проверяем, что максимум на префиксе не больше, чем элемент на левой границе блока, и минимум на суффиксе не меньше, чем элемент на правой границе блока.

Задача Е. Красивая строка

Подзадача 1

В подзадаче могут быть только два подряд одинаковых элемента. Тогда, если мы их встречаем, то лучше удалить символ с наименьшим значением; так мы минимизируем итоговую стоимость удалений.

Подзадача 2

Если у нас есть k подряд одинаковых элементов, то нужно оставить элемент с максимальной стоимостью удаления; таким образом минимизируется итоговая стоимость удалений.

Подзадача 3

Здесь после каждого запроса просто пересчитываем ответ для новой получившейся строки.

Подзадача 4

Когда стоимость удалений одинаковая, то важно знать, какая будет конечная длина строки. Тогда ответом будет $a_i \cdot (n - \text{len}(s))$, где n — начальная длина строки, а $\text{len}(s)$ — конечная длина строки после удалений.

Давайте изначально строку разделим на блоки и сохраним их границы.

В каждом запросе при изменении элемента s_i нужно смотреть, в каком блоке этот элемент и на какой элемент меняется. Тогда может быть несколько случаев: меняется элемент внутри блока (не на границе), тогда изменения элемента образует три новых блока (а старый не забываем удалить). Если меняется элемент на границе блока, то текущий блок уменьшается на 1 элемент; также нужно еще смотреть на соседний блок, если новый элемент совпадает с элементами соседнего блока, то соседний блок расширяется, иначе просто образуется новый блок из одного элемента.

Блоки можно хранить в структуре данных `set` в виде левой и правой границы; тогда поиск блока по позиции элемента можно будет осуществить за $O(\log(cnt))$, где cnt — это текущее количество блоков.

Полное решение

Заметим, что если у нас есть блок из одинаковых элементов, то в нем лучше удалить все элементы, кроме элемента с максимальным a_i . Воспользуемся методом из предыдущей подгруппы для поддержания блоков, а также реализуем, например, Дерево Отрезков для поддержания максимума на отрезке. Таким образом мы будем знать, какой a_i в каждом блоке будет оставаться и сможем пересчитывать итоговую стоимость удалений после каждого изменения в блоках.

Задача F. Четвёрки

Подзадача 1

В этой подзадаче достаточно каждый раз заново находить подходящую четвёрку вершин и добавлять ребро, поэтому можно тривиально просимулировать процесс. Поскольку всего может произойти не более $O(n^2)$ операций (так как в графе всего $O(n^2)$ рёбер), а каждый поиск работает за $O(n^4)$, то такое решение работает за $O(n^6)$ и проходит только тесты из первой группы.

Подзадача 2

Для решения второй подгруппы отдельно найдём все четвёрки вершин, подходящие изначально, после чего для каждого нового ребра перебирать пару других вершин и добавлять в очередь следующее добавляемое ребро. Таким образом, время работы составит $O(n^2)$ на ребро — всего $O(n^4)$.

Подзадача 3

Для полного решения найдём все треугольники в графе и будем перебирать четвёртую вершину среди тех, которые соединены с двумя из трёх вершин треугольника. Найти такие вершины для очередного треугольника можно за $O(n/64)$, если поддерживать матрицу смежности с помощью `'std::bitset'`. Для каждой такой найденной вершины добавим в граф ребро между ней и третьей вершиной треугольника, а в очередь добавим все треугольники, содержащие данное ребро. Таким образом, в очереди окажется всего не более $O(n^3)$ треугольников, каждый из которых обработан за $O(n/64)$. Общее время работы — $O(n^4/64)$.

Задача G. Правильное питание

Будем считать, что одна единица продукта состоит из 10^9 *наноединиц*. Таким образом, каждая единица i -го продукта состоит из p_i наноединиц белков, f_i наноединиц жиров и c_i наноединиц углеводов. Тогда, если получена одна единица рациона, содержащая ровно P наноединиц белков и ровно F наноединиц жиров, то она содержит $10^9 - P - F = C$ наноединиц углеводов. Получается, что соотношение белков, жиров и углеводов в таком рационе совпадает с требуемым. Значит, достаточно получить единицу рациона с нужным количеством белков и жиров, не учитывая углеводы. Этим фактом мы будем пользоваться при решении задачи.

Подзадача 1

В первой подзадаче во всех продуктах количество углеводов равно 0. Тогда тройка (p_i, f_i, c_i) имеет вид $(p_i, 10^9 - p_i, 0)$. При этом, поскольку $C = 0$, то и целевое соотношение имеет вид $(P, 10^9 - P, 0)$. Проведя рассуждение, аналогичное тому, что описано выше, получаем, что в этом случае достаточно получить единицу рациона, содержащую ровно P белков.

Пусть имеется всего 2 продукта: $(p_1, 10^9 - p_1, 0)$ и $(p_2, 10^9 - p_2, 0)$. Тогда, если $p_1 \leq P \leq p_2$, то можно использовать $\frac{p_2 - P}{p_2 - p_1}$ единиц первого продукта и $\frac{P - p_1}{p_2 - p_1}$ единиц второго продукта и получить единицу смеси с целевым соотношением, поскольку

$$\frac{p_2 - P}{p_2 - p_1} + \frac{P - p_1}{p_2 - p_1} = 1$$
$$\frac{p_2 - P}{p_2 - p_1} \cdot p_1 + \frac{P - p_1}{p_2 - p_1} \cdot p_2 = \frac{p_2 p_1 - P p_1 + P p_2 - p_1 p_2}{p_2 - p_1} = \frac{P p_2 - P p_1}{p_2 - p_1} = P$$

При этом, несложно понять, что любая единица смеси, полученная из продуктов с содержанием белков p_1, p_2, \dots, p_n содержит не менее $\min(p_1, p_2, \dots, p_n)$ и не более $\max(p_1, p_2, \dots, p_n)$ белков. Значит, достаточно проверить, что

$$\min(p_1, p_2, \dots, p_n) \leq P \leq \max(p_1, p_2, \dots, p_n)$$

Это удобно проверить, поддерживая мультимножество p_1, p_2, \dots, p_n в контейнере (например, используя `std::multiset` или сохраняя пары из p_i и количества продуктов с данным p_i в `std::set<std::pair<int, int>`).

Таким образом, операции вставки удаления, поиска максимума и минимума работают за $O(\log(n + q))$, поэтому временная сложность такого решения — $O(q \cdot \log(n + q))$.

Подзадача 2

В этой подзадаче нет запросов на изменение множества продуктов, поэтому достаточно лишь проверить, можно ли из имеющегося набора продуктов составить требуемый рацион, используя α_1 единиц первого продукта, α_2 единиц второго продукта, ... α_n единиц n -го продукта. Представим каждый продукт как вектор $\vec{v}_i = (p_i, f_i)$. Тогда требуется найти такие коэффициенты $\alpha_1, \alpha_2, \dots, \alpha_n$, что выполнены 2 условия:

$$\sum_{i=1}^n \alpha_i = 1$$

(так как нужно получить единицу смеси)

$$\sum_{i=1}^n \alpha_i \vec{v}_i = (P, F)$$

(так как смесь должна содержать ровно P белков и F жиров)

Несложно заметить, что уравнения выше описывают центр масс системы материальных точек $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ с весами $\alpha_1, \alpha_2, \dots, \alpha_n$. Также несложно проверить, что центр масс такой системы всегда лежит внутри (или на границе) выпуклой оболочки множества точек $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$, и для любой точки внутри (или на границе) выпуклой оболочки можно подобрать веса соответствующим образом.

Тогда во второй подзадаче достаточно построить выпуклую оболочку всех n точек и проверить, лежит ли в ней точка (P, F) .

Построение выпуклой оболочки занимает $O(n \log n)$ времени, а проверка принадлежности точки выпуклому многоугольнику — $O(\log n)$. Таким образом, решение работает за $O(n \log n)$.

Подзадача 3

В этой подзадаче для ответа на каждый запрос можно строить выпуклую оболочку заново и проверять, лежит ли точка (P, F) внутри. Каждая итерация работает за $O((n+q) \log(n+q))$. Таким образом, временная сложность решения — $O(q(n+q) \cdot \log(n+q))$.

Подзадача 4

В этой подзадаче достаточно заметить, что ответ всегда изменяется монотонно — либо ответ всегда "No,,", либо существует такой номер запроса $0 \leq j \leq q$, что ответ на все запросы до j -го — "No,,", а после j -го — "Yes,,". Поэтому можно найти j бинарным поиском. На каждой итерации бинарного поиска строим выпуклую оболочку и проверяем, лежит ли точка (P, F) внутри (или на границе). Поскольку все точки, которые могут быть в выпуклой оболочке, известны заранее, можно один раз упорядочить их по одной из координат (или по полярному углу, в зависимости от выбранного алгоритма построения выпуклой оболочки) — тогда построение выпуклой оболочки на каждой итерации будет линейным по времени.

Таким образом, время работы составит $O((n+q) \log(n+q))$ на сортировку и $O(n+q)$ на итерацию (всего $O(\log q)$ итераций). Значит, время работы алгоритма составит $O((n+q) \log(n+q))$.

Подзадача 5

Формализуем условие задачи. S — множество точек на плоскости, t — точка на плоскости. Поступают запросы на добавление и удаление точки из множества S . Требуется после каждого запроса проверить, лежит ли точка t в выпуклой оболочке S . Ниже приведены два способа решить эту задачу за $O((n+q) \log(n+q))$, где $n = |S|$.

Решение 1

Сдвинем начало координат в t . Тогда теперь требуется проверить, что начало координат лежит внутри выпуклой оболочки S . Заметим, что если это не так, то, поскольку выпуклая оболочка S является выпуклым множеством, то можно провести прямую через начало координат таким образом, что S лежит строго по одну сторону от этой прямой. Упорядочим все точки из S по полярному углу относительно начала координат. Тогда среди направленных углов между соседними точками найдётся угол больше 180 градусов.

Значит, достаточно проверять, что угол между любыми двумя соседними точками меньше 180 градусов. Для этого можно, например, хранить множество точек S в 'std::multiset' упорядоченным по полярному углу. Тогда после каждой операции вставки или удаления изменяется $O(1)$ пар соседних углов. Для каждой такой пары проверка занимает $O(1)$ времени (достаточно найти псевдоскалярное произведение соответствующих векторов). Сравнение двух векторов по полярному углу также занимает $O(1)$ времени, а вставка или удаление из 'std::multiset' — $O(\log(n+q))$. Таким образом, $n+q$ операций с множеством S и последующие проверки занимают $O((n+q) \log(n+q))$ времени.

Решение 2

Воспользуемся тем, что все запросы заранее известны. Для каждого продукта рассмотрим время, когда он был доступен — набор непересекающихся подотрезков отрезка $[0, q]$. Концами этих отрезков могут быть $0, q$ и номера запросов, когда этот продукт был добавлен или удалён. Несложно понять, что в совокупности все такие наборы содержат $O(n)$ отрезков.

Построим дерево отрезков на отрезке $[0, q]$, в каждой вершине которого будем хранить список продуктов. Тогда каждый отрезок времени жизни некоторого продукта можно разбить на $O(\log q)$ отрезков, соответствующих вершинам дерева. Добавим этот продукт в списки соответствующих вершин.

Пусть существует решение частного случая задачи, когда можно удалить только последний добавленный продукт. Тогда обойдём дерево отрезков. При входе в очередную вершину выполним запросы на добавление всех продуктов из её списка. Затем, если данная вершина — лист, запишем результат последнего запроса в ответ на исходный запрос, соответствующий этому листу. При выходе удалим столько же продуктов, сколько добавили при входе. Несложно понять, что в таком случае

при записи ответа на исходный запрос в листе u во множество продуктов будут входить те, и только те, которые указаны в списках предков u — а это, в свою очередь, продукты, которые доступны в момент времени, соответствующий листу u . Таким образом, мы получим корректное состояние множества и корректный ответ на запрос в листе u . Осталось решить частный случай, описанный в начале.

Для решения частного случая будем поддерживать в стеке все состояния. Сдвинем начало координат в t . Каждое состояние — либо метка о том, что начало координат принадлежит выпуклой оболочке текущего множества точек, либо пара (v_1, v_2) векторов-касательных к этой выпуклой оболочке. Тогда, если требуется добавить продукт, то либо начало координат уже принадлежит выпуклой оболочке (тогда просто добавим в стек текущее состояние), либо возникают три случая:

Первый вариант — точка, соответствующая продукту, лежит между векторами v_1 и v_2 . В таком случае состояние не изменяется, поскольку добавление новой точки не повлияет на касательные. Тогда добавим в стек состояние, равное текущему.

Второй вариант — новая точка w лежит внутри угла, вертикального с углом, образованным v_1 и v_2 . В этом случае начало координат лежит в треугольнике v_1v_2w , поэтому добавим в стек состояние с меткой об этом.

Третий вариант — w лежит вне угла, но и вне вертикального к нему. В этом случае добавление w во множество приводит к тому, что одна из касательных (ближайшая по углу) переходит в w . Тогда добавим в стек обновлённое состояние.

В случае, если требуется удалить последний добавленный продукт, достаточно удалить из стека последнее состояние. Текущим мы здесь считаем состояние, которое хранится в вершине стека.

Таким образом, мы получили решение частного случая, которое обрабатывает каждый запрос за $O(1)$. Поскольку запросов к этому решению будет сделано $O((n + q) \log q)$, то и время работы такого решения — $O((n + q) \log q)$, и этого достаточно для полного решения задачи.

Задача Н. ИИ

Задачу можно решать двумя принципиально разными подходами

Подход 1

Можно придумать критерии, которые позволят отличить знаки в высоком качестве. В **подзадаче 1** достаточно посчитать количество компонент связности из белых пикселей. У цифры 6 есть дополнительная компонента по сравнению с цифрой 5. В **подзадаче 2** нужно выделить черные пиксели, которые относятся к стрелке, а не окружности (например, учитывать расстояние до центра картинки) и определить, в какую сторону идёт поворот — по часовой стрелке или против часовой стрелки. Это значение не зависит от поворота знака и его можно определить через векторное произведение. Для этого нужно черные пиксели стрелки разбить на 2 отрезка, например, запустив поиск в ширину от границы. В **подзадаче 3** нужно учесть и количество компонент связности (у чисел их больше из-за цифры 0), и поворот стрелки, и аналогично посчитать ориентацию цифры 6/9 относительно 0. В **подзадаче 4** имеет смысл предварительно устранить «шумы» (перекрашенные пиксели), например, при помощи медианного фильтра.

Подход 2

Поскольку в задаче всего 50 тестов и известна сумма баллов за каждую посылку, то можно применить подходы, которые как раз используются в некоторых алгоритмах машинного обучения. Возьмём какой-нибудь признак (значение конкретного пикселя, количество черных пикселей, размер картинки и т.п.) и «разобьём» тесты по этому признаку. Для этого будем отвечать, например, 'left', если этот признак принимает конкретное значение, и '?' в остальных случаях. Сделав несколько посылок с разными ответами, мы поймём, у каких знаков чаще встречается этот признак. Далее множество этих знаков разобьём по другому признаку и «уточним» ответы. Продолжая действовать таким образом, можно при помощи небольшого количества признаков и большого количества посылок получить правильные ответы на все тесты.