Задача A. Новая игра

Автор задачи: Даниил Орешников, разработчик: Егор Юлин

Будем за $|a|$ обозначать длину числа a (не модуль), а за a' и b' обозначим $a + c$ и $b + c$, соответственно. Рассмотрим сначала случаи, когда ответом будет -1:

- Если $a > b$, то ответ всегда -1: это достаточно очевидно.
- Если $|a| = |b|$ и $b > a$, то есть варианты $|a'| = |b'|$ и $|a'| + 1 = |b'|$, рассмотрим их ниже.
- Если после прибавления c длины равны, то поскольку $a' < b'$, то и одно не входит в другое как подстрока.
- Если же длины становятся разными, то $|a'| + 1 = |b'|$, но тогда одно может входить в другое как подстрока, только если $b' = 10a' + 1$ или если $b' = d \cdot 10^{|a'|} + a'$. В обоих случаях величина $b' - a' = b - a$ оказывается заметно большей ($> 9a$), чем возможно для двух чисел a и b одинаковой длины.

Рассмотрим случай когда $|a| + 1 = |b|$. Есть несколько вариантов. Если a уже входит как подстрока в b, тогда $c = 0$ подходит. Иначе:

- Пусть мы хотим найти такое c, что $|a + c| = |b + c|$. Заметим, что такого подходящего c быть не может, так как такой случай был разобран выше и для него не существует ответа.
- Тогда должно выполняться $|a'| + 1 = |b'|$. Рассмотрим случай когда у нас a' — суффикс b'. Но тогда ответа не существует, так как при $b' - a' = b - a = d \cdot 10^{|a'|}$ было бы верно, что a изначально являлся суффиксом b.
- Соответственно, a' должен быть префиксом b'. Пусть $\overline{b} = \lfloor \frac{b}{10} \rfloor$, то есть этот самый префикс b. Покажем, что если $\overline{b} > a$, то ответ существует. Найдем его конструктивно: возьмем в качестве c число $\overline{b} - a$ и добавим его к a и b. Так как у нас $\overline{b}$ растет с таким действием на $\approx \frac{c}{10}$, а a растет на c, разница $\overline{b'} - a'$ будет уменьшаться, пока не станет нулем.

Случай, когда $|a| + 1 < |b|$, решается аналогично случаю, когда $\overline{b} > a$, но теперь в качестве $\overline{b}$ берем префикс минимальной длины, для которого $a < \overline{b}$. Поскольку разница между $\overline{b'}$ и a' каждый раз уменьшается примерно в 10 раз, достаточно будет $\mathcal{O}(\log_{10} a)$ шагов.

Задача B. Торговцы

Автор задачи и разработчик: Егор Юлин

Для решения данной задачи будем использовать дерево отрезком с отложенными операциями. Тогда первый и третий запрос — это стандартные запросы в дерево отрезков на изменение одного элемента и вычисление суммы на отрезке. Покажем, как можно выразить второй запрос.

Рассмотрим число x, которое меньше 2^k, и запишем инвертированный x как $\overline{x}$. Тогда $x \,\&\, \overline{x} = 0$. А тогда $x \oplus \overline{x} = x + \overline{x} = 2^k - 1$. Отсюда можно получить, что $\overline{x} = 2^k - 1 - x$. Такое изменение несложно поддерживается на отрезке, если его разложить в

1. умножение на отрезке на -1 (храним отложенный флаг, а сумму просто умножаем на -1);
2. прибавление $2^k - 1$ на отрезке (стандартное действие, также храним отложенную операцию, а сумму пересчитываем, прибавляя эту величину, умноженную на длину отрезка).

Задача C. Битвы с боссами

Автор задачи и разработчик: Егор Юлин

Обозначим за C_j множество боссов, которые могут быть на позиции j, чтобы они удовлетворяли всем условиям сражения Хорнет. Тогда если в конце есть хотя бы одно пустое множество C_j, то ответ «`No`».

Покажем, как мы будем пересчитывать C_j. Изначально скажем, что во всех позициях могут быть все боссы. Так же будем хранить два дополнительных множества W и L — боссы, которых Хорнет с текущими характеристиками может победить, и которым проиграет, соотвественно. При выполнении запросов множество W никогда не будет уменьшаться, при этом объединение W и L дает множество всех боссов.

Тогда запрос второго типа обрабатывается как $C_j \leftarrow C_j \cap W$, если Хорнет выиграла, и $C_j \leftarrow C_j \cap L$, если Хорнет проиграла.

Для обработки запроса первого типа необходимо сделать небольшой предпосчет. Заведем многомерный массив $\mathsf{a}_{i,j}$, в котором i — номер характеристики, j — значение i-й характеристики. В данном массиве будем хранить индексы боссов p, у которых $b_{p,i} = j$. Тогда видно, что j должно быть не больше 500, так как у всех боссов характеристики ограничены 500. Значит при помощи такого предпосчета мы умеем узнавать всех боссов, у которых i-я характеристика не меньше, чем у Хорнет, в любой момент времени.

Чтобы понимать, что Хорнет может победить босса, будем для каждого босса хранить число характеристик, которые больше, чем у Хорнет. При увеличении характеристик Хорнет достаточно просмотреть, у каких боссов соответствующие характеристики были больше, а стали меньше, чем у нее, и обновить для этих боссов эту величину.

Таким образом, для решения задачи достаточно аккуратно поддерживать несколько вспомогательных величин и аккуратно пересекать множества возможностей.

Задача D. Скелеты, кости, кладбище, черепа

Автор задачи и разработчик: Павел Скобелин

Давайте поймем, что происходит при разрезании графа одной вертикальной чертой $x = C$. Он разбивается на два графа: все вершины с $x_i < C$, и все вершины с $x_i > C$. Тогда количество компонент связности в получившемся графе будет равно сумме количества компонент связности в левом и правом графе.

Предподсчитаем количество компонент связности в графе с вершинами левее C для каждого C. Для этого отсортируем вершины по x-координате и будем добавлять их в граф по очереди, поддерживая количество компонент связности с помощью системы непересекающихся множеств. Таким образом, для каждого C мы получим, сколько компонент связности в графе, все вершины которого левее C.

Сделаем аналогичный проход справа налево и получим ответ для правой части. Теперь, для ответа на запрос нужно воспользоваться бинарным поиском, чтобы найти запомненные для данного C значения, и сложить количество компонент связности в левой и правой части графа. Если изначально сделать сжатие координат, то можно обойтись и без бинпоиска.

Таким образом, сортировка работает за $\mathcal{O}(n \log n)$, предподсчет за $\mathcal{O}(n \cdot \alpha(n))$, и ответ на запрос за $\mathcal{O}(\log n)$.

Задача E. Путешествие по дереву

Автор задачи: Ильдар Гайнуллин, разработчик: Егор Юлин

Приведем конструктивное решение: построим новый граф, в котором есть все ребра (a_i, b_i), и найдем в этом графе остовное дерево. Утверждается, что размер этого остовного дерева и будет ответом.

Действительно, давайте докажем этот факт:

1. Если какие-то ребра в новом графе образуют цикл, то какие-то два соответствующих пути в исходном графе обязательно имели пересечение. Действительно, нельзя пройти по дереву по вершинам $v_1 \to v_2 \to \ldots \to v_k \to v_1$, не посетив никакое ребро дважды (так как в дереве не может быть циклов). Соответственно, ответ не может быть больше, чем величина остовного дерева в новом графе.

2. При этом существует граф, на котором такая оценка достигается: буквально возьмем старый граф равным найденному остовному дереву в новом графе, и в нем выбранное множество путей не пересекается, так как каждый путь состоит из одного уникального ребра.

Поэтому задача сводится к поиску размера остовного дерева графа, в котором (a_i, b_i) — ребра. А это можно сделать, например, с помощью обхода в глубину или в ширину.

Задача F. Убежища

Автор задачи: Даниил Орешников, разработчик: Павел Скобелин

Воспользуемся бинарным поиском по ответу D — максимальному расстоянию до ближайшего кокона. Для фиксированного D проверим, хватает ли имеющихся r коконов плюс не более $k - r$ новых, чтобы покрыть всех жителей.

Проверять будем жадно. Отсортируем $x_1 \leqslant \ldots \leqslant x_n$ и $a_1 \leqslant \ldots \leqslant a_r$. Каждый фиксированный кокон покрывает отрезок $[a_j - D, a_j + D]$. Будем перебирать x от меньших к большим и поддерживать указатель на имеющийся кокон. Если его не хватает (то есть ближайший из имеющихся коконов находится на расстоянии $> D$), то поставим новый — выгоднее всего нам поставить его в точку $a_i + D$, так как все жители левее уже покрыты, и значит имеет смысл покрыть как можно больше тех, что справа.

В конце посчитаем, сколько коконов нам пришлось добавить — если не более $k - r$, то такое значение D достижимо (причём, расстановку новых коконов мы получили), иначе нет. То есть, в зависимости от этого будем двигать или левый указатель, или правый.

Осталось определить, когда нужно останавливать бинарный поиск. Можно было решить задачу, используя тип `double` с достаточной точностью, но при внимательном рассмотрении становится понятно, что ответ всегда будет полуцелый (то есть, целое число, или целое число, поделенное на 2). Таким образом, если домножить все координаты на 2, можно было обойтись целочисленным бинарным поиском.

Тогда решение работает за $\mathcal{O}(n + r)$ на одну проверку для бинарного поиска, и один запрос работает за $\mathcal{O}(\log C \cdot (n + r) + r \log r)$. Суммарное время работы решения — $\mathcal{O}(n \log n + (nq + R \log R) \log C)$, что легко укладывается в лимиты при заданных ограничениях.

Задача G. Песнь Нити

Автор задачи и разработчик: Павел Скобелин

Зафиксируем насекомое, которое будет Певчим, то есть переберем его из n вариантов.

Поймем, как звук будет доходить до точки $(0, 0)$. Очевидно, первый клич, дошедший до $(0, 0)$, должен быть кличем от Певчего насекомого (кратчайшее расстояние между двумя точками — по прямой между ними). Так как по условию он должен быть единственным, то несложно понять, что для выполнения такого на отрезке от Певчей до Слушателя не должно находиться насекомых Эхо (если такие есть, то выкинем их из рассмотрения).

Как следующие кличи будут доходить до Слушателя? Несложно понять, что они будут иметь траекторию: Певчая $\to$ Эхо $\to$ Слушатель. Тогда для каждого насекомого посчитаем сумму расстояний от него до Слушателя и Певчей, и отсортируем по этому параметру. Далее сгруппируем кличи, которые приходят в один момент, и получим количества кличей, которые приходят в разные моменты (также нужно быть аккуратным с точностью нецелых чисел).

Далее мы получили порции кличей, которые приходят к нам в какие-то моменты — пусть это $x_1, x_2, \ldots x_m$. Теперь, из них нам нужно выбрать такие $s - 1$ насекомых, чтобы размеры их групп были $c_2, c_3, \ldots c_k$. Это можно сделать с помощью динамического программирования: $\mathtt{dp}_{i,j}$ — количество способов выбрать первые j кличей из первых i моментов времени. Пересчет будет выглядеть так:

$$\mathtt{dp}_{i,j} = \mathtt{dp}_{i-1,j} + \mathtt{dp}_{i-1,j-1} \cdot C_{x_i}^{c_j}.$$

Так как каждый раз мы можем или не брать текущий x_i в рассмотрение, или выбрать из него c_i насекомых, не учитывая порядок, с помощью числа сочетаний. Ответ ДП будет лежать в $\mathtt{dp}_{n,k}$.

Таким образом, мы можем посчитать ДП за $\mathcal{O}(nk)$. С учетом перебора Певчего насекомого, решение будет работать за $\mathcal{O}(n^2 k)$.

Задача H. Головоломка отрезков

Разработчик: Даниил Орешников

Переформулируем задачу. Есть n блоков, у каждого есть выступ b_i над основанием a_i. Поскольку блоки можно поворачивать, а надо разместить их как можно более плотно, очевидно, что имеет смысл рассматривать минимальную величину, на которую каждый блок будет увеличивать длину выступа. Таким образом, если в ответе будут k блоков, то:

- если $k = 1$, то единственный блок вносит вклад в ответ b_i;
- иначе, центральные $k - 2$ из них будут вносить вклад a_i, а два крайних — $\min(b_i + c_i, a_i - c_i)$ (это b_i плюс минимальное расстояние от выступа до края основания).

Собственно, обозначим за d_i величину $\min(b_i + c_i, a_i - c_i)$, а также отдельно рассмотрим возможность ответа с $k = 1$.

Теперь проверим наличие ответа с $k > 1$. Для этого отсортируем все блоки по возрастанию a_i, и жадно выберем t первых их них так, что $\sum\limits_{i=1}^{t} a_i \leqslant W$. Ключевое утверждение следующее:

1. Во-первых, ответ теперь гарантированно равен либо t (уже нашли подходящий), либо $t + 1$, либо $t + 2$. Для $k = t + 3$ уже $t + 1$ центральный блок будут занимать больше места, чем доступная ширина W (следует из того, как мы выбрали t).
2. Во-вторых, для фиксированного k минимальные $k - 2$ по a_i блоков гарантированно будут входить в ответ. Действительно, у нас есть $k - 2$ центральных блоков, которые дают вклад a_i в ширину, и единственная причина не выбрать $k - 2$ минимальных по a_i в качестве центральных — это если какой-то из них более выгодно располагается с краю (но тоже входит в ответ).

Дальше можно решать следующим образом: переберем k от $t + 1$ до $t + 2$ (ответ для $k = t$ мы уже получили) и проверим, можно ли построить ответ с таким k. Для этого изначально расположим $k - 2$ минимальных по a_i в центре, а затем переберем варианты выбора крайних блоков:

- либо центральные без изменений, тогда крайние — это два минимальных по d_i из оставшихся;
- либо один из центральных надо переместить в край (тогда это минимальный по d_i из центральных, а на его место в центр надо поместить минимальный по a_i из оставшихся);
- либо надо сделать две замены аналогично предыдущему случаю.

Итого мы имеем $\mathcal{O}(1)$ вариантов, которые надо перебрать, и из них выбрать оптимальный. Здесь есть тонкости с разбором случаев, и если хотелось их избежать, можно было просто написать чуть более примитивный перебор: выбрать по два минимальных по d_i из первых $k-2$ по a_i и из оставшихся $n - k + 2$, и перебрать шесть вариантов их расположения с краю; для каждого выбрать оставшиеся $k - 2$ центральных как минимальные по a_i из оставшихся. Этого тоже было достаточно, чтобы уложиться в лимит по времени.

Задача I. Коллекция ягод

Автор задачи и разработчик: Егор Юлин

Во первых заметим, что суммарную красоту ягод на отрезке можно считать как $g(r) - g(l - 1)$, где $g(x) = \sum\limits_{j=0}^{x} f(j)$. Тогда научимся находить только $g(x)$, и этого будет достаточно, чтобы решить задачу.

Для этого будем идти итеративно. Найдем самую большую степень двойки, не превосходящую x (по сути старший бит числа): пусть она равна c, то есть $x = 2^c + d$, где $d < 2^c$. Тогда очевидно, что все числа от 0 до 2^c нам подходят, и посчитать сумму $\sum\limits_{j=0}^{2^c-1} f(j)$ несложно: это будут произведения всех подмножеств $\{a_i\}_{i=0}^{c-1}$, то есть

$$1 + a_0 + \ldots + a_{c-1} + a_0a_1 + \ldots + a_{c-2}a_{c-1} + \ldots + a_0a_1 \ldots a_{c-1}.$$

Такая сумма легко записывается в виде выражения $(1 + a_0) \cdot \ldots \cdot (1 + a_{c-1})$: действительно, из каждой скобки можно выбрать либо a_i, либо единицу, и получить слагаемые, соответствующие всем возможным подмножествам $\{a_i\}$.

После остается только посчитать $\sum_{j=2^c}^{x} f(j)$, но тут достаточно заметить, что каждое $f(j)$ будет делиться на a_c, а именно, будет равно $a_c \cdot f(j-2^c)$. Таким образом, эта сумма считается как $a_c \cdot \sum_{j=0}^{x-2^c} f(j)$, и эту сумму уже можно посчитать рекурсивно тем же способом. Повторим такие действия, пока не получим $x = 0$, и это займет не более $\log_2 x$ таких переходов.

Соответственно, на запрос можно ответить за $\mathcal{O}(\log r + \log l)$, то есть $\mathcal{O}(n)$.

Задача J. Шёлковая лестница

Автор задачи: Даниил Голов, разработчик: Павел Скобелин

Минимальное число поворотов равно $3n - 2$. Достаточно нарисовать картинку и попробовать нарисовать подходящий путь, чтобы убедиться в ответе, но распишем доказательство ниже.

Чтобы впервые попасть в (i, i) из $(0, 0)$ для любого $i > 0$, нужна смена направления $\Rightarrow$ как минимум один раз. При этом для каждого перехода $i \to j$ неизбежны три поворота:

1. в (i, i) полный разворот назад или же поворот, чтобы пойти в другую сторону;
2. в $(i, 0)$ или $(0, i)$: поворот на $90°$;
3. в $(j, 0)$ или $(0, j)$: аналогично.

Итого $\geqslant 1 + 3(n - 1) = 3n - 2$. Маршрут

$$(0,0) \to (1,0) \uparrow (1,1) \downarrow (1,0) \to (2,0) \uparrow (2,2) \downarrow \cdots \to (n,0) \uparrow (n,n)$$

дает ровно $3n - 2$ поворотов.