Задача A. Формула 2

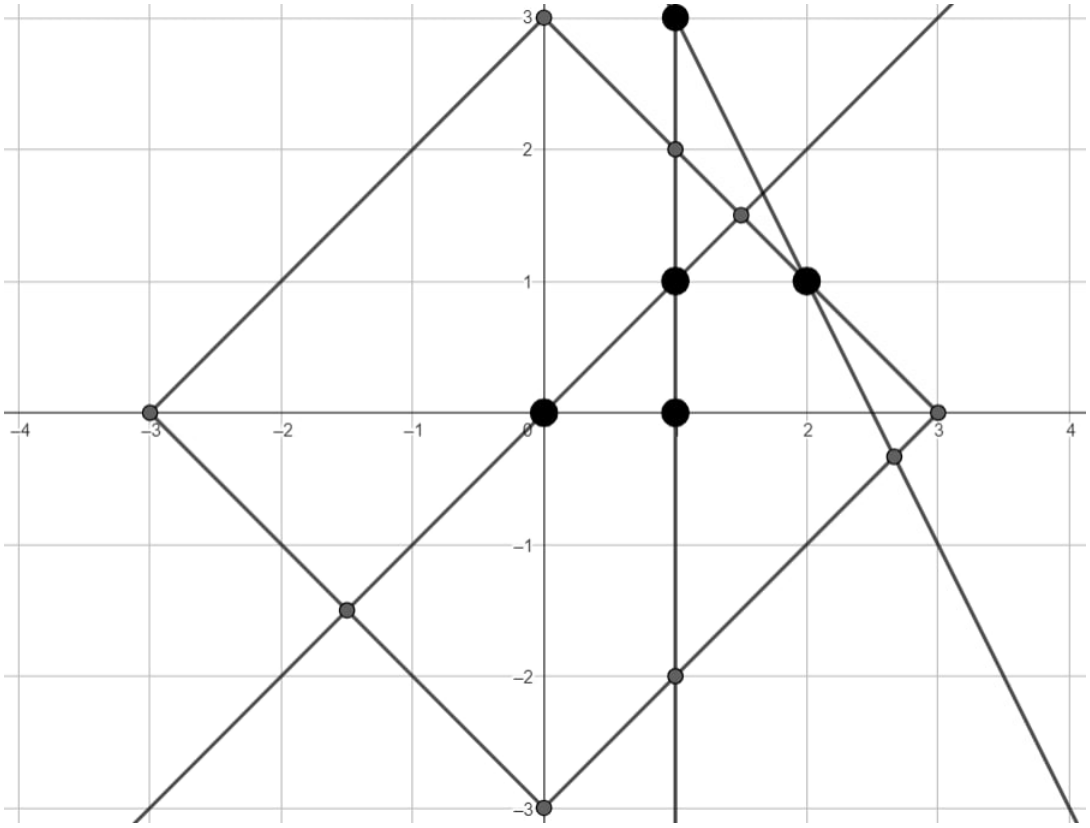
Заметим, что при каждом повороте налево второй гонщик проезжает на 2 больше, а при повороте направо — первый на 2 больше. Поэтому, достаточно сравнить количество поворотов налево и направо: если налево больше, то побеждает первый, если направо больше — второй, если поровну — ничья.

Задача B. Две дороги 2

В условии говорится о *Манхэттанской метрике* — расстояние от центра координат до станции (x, y) равно $|x| + |y|$. Вообще, множество точек, удалённых от центра координат на фиксированное расстояние d, является повёрнутым на 45 градусов квадратом (или же ромбом) с диагоналями равными $2d$. Таким образом, нам нужно найти минимальное такое d, что все точки пересечения прямых находятся внутри такого квадрата (ромба). В этой задаче может быть удобно заранее повернуть систему координат на 45 градусов, но это вопрос предпочтения.

Решение 1:

Будем делать бинарный поиск по ответу. Пусть мы проверяем текущую стоимость m, тогда нужно проверить, есть ли пересечение прямых вне предела ромба $[(m, 0), (0, m), (-m, 0), (0, -m)]$.



Во-первых, каждая прямая должна пересекать этот ромб: если прямая не проходит через ромб, то на ней точно есть точка пересечения, которая не лежит внутри ромба. Это можно каждый раз проверять внутри бинарного поиска, или же заранее подобрать для этого левую границу. Правую границу нужно выбрать достаточно большой: если две прямые задаются парами точек внутри квадрата стороны X, то координаты точки пересечения могут быть порядка X^3. Поэтому в качестве правой границы нужно выбрать число, не меньшее $8 \cdot 10^9$.

Теперь нужно на каждом шаге бинарного поиска проверить, есть ли точка пересечения вне ромба. Если каждая прямая пересекает этот ромб, то из него будет «торчать» $2 \cdot N$ лучей. Заметим, что два луча могут пересечься только тогда, когда они выходят из одной стороны либо из смежных сторон ромба. Давайте отсортируем лучи, исходящие из каждой стороны, по координатам точки, из которой они исходят. Луч, исходящий из вершины, можно отнести к любой стороне. Тогда если два луча на одной стороне пересекаются, то обязательно пересекаются два соседних луча. Иначе, если два луча из двух разных сторон пересекаются, то пересекаются два ближайших к общей вершине

луча. В любом случае, если мы запишем лучи в порядке обхода, то достаточно будет проверить на пересечение только пары соседних.

Таким образом, для каждой стороны нужно отсортировать лучи по их началу, что делается за $O(N \cdot log(N))$, затем за линию проверить все пары соседей на пересечение. Общая сложность будет равна $O(N \cdot log(N) \cdot log(\frac{MAXX^3}{EPS}))$, где EPS — требуемая точность.

Решение 2:

Будем использовать идеи из решения 1, но рассмотрим ответ поподробнее. Если постепенно увеличивать размер ромба, то в какой-то момент последняя точка пересечения попадёт внутрь него — эта точка и будет ответом. Тогда эта точка обязательно была точкой пересечения двух соседних лучей в порядке сортировки. Но после того как все точки пересечения попали в ромб, исходящие из него лучи больше не пересекаются, значит они будут отсортированы не только по стартовой точке, но и по углу.

Порядок по стартовым точкам зависит от размера ромба, а вот порядок по углу мы знаем заранее. Таким образом, достаточно отсортировать все прямые по углу и проверить пересечения только пар соседних. Это даст нам решение за $O(N \cdot log(N))$.

Задача C. Принцип Дирихле 2

По формуле дискриминанта, корень существует тогда и только тогда, когда $B^2 - 4AC \geqslant 0$. Это значит, если среди чисел есть хотя бы одно положительное и хотя бы одно отрицательное, можно взять их в качестве A и C. Тогда $4AC$ будет отрицательным, и уравнение будет иметь хотя бы один корень.

Остался случай, когда все числа неотрицательные или неположительные. Возьмём наименьшее по модулю число на место коэффициента C. Из оставшихся возьмём наименьшее по модулю ненулевое число на место коэффициента A. Из оставшихся после этого возьмём наибольшее по модулю на место коэффициента B. Наконец, проверим, выполняется ли $B^2 \geqslant 4AC$.

Задача D. Прямоугольники 3

Рассмотрим пару смежных сторон исходного листа-прямоугольника и выпишем два набора отрезков, на которые закрашенные строки и столбцы разбивают эти стороны. Заметим, что любой белой прямоугольной области в качестве длины и высоты соответствует пара отрезков, взятых с каждой из этих двух исходных сторон. Тем самым, число различных прямоугольников — это произведение количеств различных длин в этих наборах отрезков. Чтобы эффективно найти количество различных чисел в каждом наборе, можно воспользоваться сортировкой или положить все его элементы в set.

Задача E. Шифровка 5

Нам понадобится интересный факт: два соседних простых числа из отрезка $[0; 1048575]$ различаются не более, чем на 114. Это означает, что для любого числа из этого отрезка его разница с соседним простым не больше 113, либо оно само простое, тогда ответ 0. Этот факт можно либо проверить с помощью программы, либо просто догадаться, что эта константа небольшая.

Это означает, что для числа a_i можно просто проверить все значения x от 0 до 127, так как результаты побитового исключающего «ИЛИ» образуют последовательный отрезок чисел длины 128. По выше замеченному факту, среди них точно найдётся простое число.

Для того, чтобы ускорить программу, можно заранее посчитать все простые из отрезка $[0; 1048575]$ с помощью решета Эратосфена.

Задача F. Руны в поле 2

Для удобства переформулируем задачу: нужно посчитать, какое наименьшее количество букв может остаться в строке. Легко заметить, что при замене любой подстроки задача решается независимо для оставшихся подстрок. Значит, можно решить задачу с помощью одномерного динамического программирования. Для каждой позиции будем считать ответ — какое наименьшее число рун останется на этом префиксе. Будем обновлять значение в префиксе, перебирая суффикс. Далее нам понадобится несколько ключевых утверждений:

Лемма №1: если мы можем забрать подстроку с помощью общего префикса и суффикса, которые пересекаются, то мы могли бы забрать её с помощью общего префикса и суффикса без пересечений.

Доказательство: если это произошло, то забранная подстрока имеет вид `s+t+s+t+s`, где `s` — непустая подстрока, являющаяся общей частью префикса и суффикса, а `t` может быть пустой. Забрали всю это подстроку с помощью префикса и суффикса `s+t+s`. Но это значит, что всю эту подстроку можно было забрать с помощью `s`.

Назовём подстроку, которую можно использовать в качестве префикса и суффикса при заборе рун *интересной*. Заметим, что она непустая. Тогда каждую операцию можно представить в виде забора двух равных непересекающихся интересных подстрок и всего, что лежит между ними.

Лемма №2: для достижения оптимального ответа достаточно рассматривать интересные подстроки, в которых символ на последней позиции встречается ровно один раз во всей интересной подстроке.

Доказательство: от противного, пусть в оптимальном разбиении была унесена подстрока `c` с помощью интересной подстроки `s+'d'+t+'d'` (`s` и `t` могут быть пустыми), то есть подстрока `c` имела вид `s+'d'+t+'d'+...+s+'d'+t+'d'`. Есть два случая:

1. `s` — пустая, тогда всю подстроку можно унести с помощью интересной подстроки `'d'` (первый символ префикса и последний суффикса), что удовлетворяет условию леммы;

2. `s` — не пустая, тогда унесём в две операции: первая — с помощью интересной подстроки `s` унесём `s+'d'+t+'d'+...+s`, в итоге останется `'d'+t+'d'`, что можно унести с помощью интересной подстроки `'d'`. Эти две операции удовлетворяют условию леммы.

Значит можно рассматривать не все интересные подстроки, а только те, в которых последний символ встречается ровно один раз. Таких интересных подстрок не более $\sigma \cdot N$, где σ — размер алфавита. Это потому, что если зафиксировать этот последний символ, всего будет не больше N подстрок, заканчивающихся на такой символ и больше не содержащих его внутри.

Пусть `dp[i]` — оптимальный ответ для i-го префикса изначальной строки. Будем перебирать позиции слева направо. Когда мы стоим в i-й позиции, нужно перебрать все интересные подстроки, заканчивающиеся в этой позиции, и обновить ответ на префиксе. Здесь проблема в том, что текущая интересная строка могла встречаться много раз, и мы не можем перебрать все её вхождения ранее.

Решается это просто: нужно для каждой интересной строки завести словарь `mp[s]`, где хранится минимальное значение `dp` до её вхождения. Например, для строки `'dfabfhabdfab'` будет `mp['ab'] = min(dp[1], dp[5], dp[9])`, если работать в 0-индексации. Тогда можно посчитать `dp[i]` как минимум из значений `mp[`s_j`]` для каждой интересной строки s_j, заканчивающейся в позиции i. Когда же мы заходим при пересчёте динамики в подстроку s_j, мы обновляем `mp[`s_j`] = min(mp[`s_j`], dp[`k`])`, где k — номер позиции непосредственно перед вхождением s_j.

Если считать хэши от подстрок и использовать `hash_table` для `mp`, то общая сложность будет равна $O(\sigma \cdot N)$.

Задача G. Градостроение 2

Решим для каждой компоненты связности отдельно, затем ответы перемножим.

Во-первых, если в компоненте есть вершина степени 5 и больше, то ответ равен 0. Так как у каждой стойки есть 4 слота, это значит, что степень вершин должна быть 4 или меньше. Во-вторых, если в компоненте рёбер больше, чем вершин, то ответ равен 0. Это просто потому, что у каждой стойки одна лента.

Таким образом, в каждой компоненте количество рёбер меньше либо равно количеству вершин. Но рёбер у связного графа не может быть меньше, чем количество вершин минус один. Поэтому компоненты связности делятся на 2 вида:

1. Компонента — дерево. То есть, рёбер на одно меньше, чем вершин. В этом случае есть ровно одна стойка, из которой не выходит лента. И если мы зафиксируем эту стойку, все остальные ленты должны вести в её сторону, поэтому конфигурация определяется однозначно. Единственная загвоздка — у соответствующей вершины степень не может быть равна 4, потому

что в стойку может вести максимум 3 ленты. Поэтому ответ для такого случая — количество вершин в компоненте, степень которых не равна 4.

2. Компонента — не дерево. То есть, рёбер столько же, сколько и вершин. В общем случае, все такие связные графы представляют из себя единственный цикл с подвешенными к нему деревьями. Заметим, что лента из стойки, входящей в цикл, может вести только в другую стойку из этого цикла. Если бы лента вела в одно из подвешенных к циклу деревьев, то мы могли бы удалить это дерево, и получили бы граф в котором есть цикл, но одна лента не используется. Это является противоречием, следовательно у такой компоненты ленты направлены вдоль цикла. Можно выбрать одну из двух ориентаций цикла, а оставшаяся конфигурация определяется однозначно: все другие ленты будут направлены в сторону цикла. Поэтому ответ для такого случая всегда равен 2.

Задача H. На планете Иворил... 2

Заведём счётчик для всех части речи — сколько окончаний не начальной формы могут быть у данного слова. Если после обработки всех 9 окончаний счётчик равен 0, то ответ «`NONE`», если он больше 1 — ответ «`INDEFINITE`». Иначе находим подходящее окончание, удаляем его и вставляем окончание соответствующей начальной формы.

Задача I. Эксперимент с соком 2

Заметим, что сливаться в один сосуд будут только одинаковые объёмы, поэтому формулу для температуры можно упростить:

$$t_f = \frac{V_1 \cdot t_1 + V_2 \cdot t_2}{V_1 + V_2} = \frac{t_1 + t_2}{2}.$$

Также заметим, что t_f всегда будет строго между t_1 и t_2, поэтому формулу для времени тоже можно упростить:

$$T_f = \frac{|t_f - t_1| \cdot V_1 + |t_f - t_2| \cdot V_2}{c_w} = V \cdot |t_1 - t_2|.$$

Так как считать температуру не требуется, можно вместо этого ввести величину Q, равную произведению температуры и объёма сока в сосуде, и вместо этого считать ответ через Q:

$$Q_f = Q_1 + Q_2,$$

$$T_f = |Q_1 - Q_2|.$$

Чтобы отвечать на запросы, воспользуемся структурой `sparse_table`. В ней для каждого подотрезка, длина которого равна степени двойки, будем хранить величину Q и ответ T. Проинициализируем их как $Q[i][0] = t_i, T[i][0] = 0$. Пересчёт отрезка с i-го по $(i + 2^j - 1)$-й сосуд ($i + 2^j \leqslant 2^N$) происходит по следующим формулам:

$$Q[i][j] = Q[i][j-1] + Q[i + 2^{j-1}][j-1],$$

$$T[i][j] = max(T[i][j-1], T[i + 2^{j-1}][j-1]) + \left|Q[i][j-1] - Q[i + 2^{j-1}][j-1]\right|.$$

Это работает, потому что алгоритм делает ровно эти операции. В начале программы сделаем предподсчёт всех значений T, тогда ответ для запроса (l, p) равен $T[l][p]$.

Задача J. Две прогрессии 2

Известный факт: пересечением двух возрастающих арифметических прогрессий (то есть, последовательность чисел, входящих в две прогрессии) является возрастающая арифметическая прогрессия, либо их пересечение пустое. Из этого следует, что пересечение любого количества возрастающих арифметических прогрессий есть возрастающая арифметическая прогрессия, либо пустое множество. Значит, для подсчёта можно воспользоваться формулой включения-исключения.

Для этого нужно научиться находить количество чисел в отрезке $[1; N]$, входящих в одну прогрессию, и находить пересечение арифметических прогрессий.

1. Пусть мы пытаемся найти количество чисел в отрезке $[1; N]$, входящих в прогрессию с началом b и шагом d. Если $b > N$, то таких чисел 0, иначе это количество находится по формуле $1 + (N - b) \operatorname{div} d$;

2. Пусть мы пересекаем прогрессии с началами b_1 и b_2 и шагами d_1 и d_2 соответственно. Их пересечением является множество чисел $k = b_1 + d_1 x = b_2 + d_2 y$, где x и y — целые неотрицательные числа. Таким образом, нам нужно решить диофантово уравнение с помощью расширенного алгоритма Евклида. Тогда, если решения существуют, началом полученной прогрессии будет минимальное число k, удовлетворяющее уравнению при неотрицательных значениях x и y, а шагом — НОК(d_1, d_2).

Наконец, нужно применить формулу включения-исключения, так как мы умеем считать пересечение прогрессий, а от нас просят объединение. Сначала возьмём в качестве ответа сумму количеств входящих в прогрессии чисел от 1 до N. Так мы посчитаем по несколько раз все числа, входящие в две прогрессии. Затем возьмём пересечения всех пар прогрессий и вычтем соответствующие количества из ответа. Но так мы по несколько раз вычтем все числа, входящие в три прогрессии, поэтому надо добавить их обратно. И так далее.

Всё это можно сделать, например, перебором всех подмножеств прогрессий. Если в подмножество входит нечётное число прогрессий, то возьмём количество чисел из пересечения со знаком плюс; если чётное — со знаком минус. Таким образом мы посчитаем любое число, входящее в объединение, ровно один раз.

Задача K. Мета-условие 2

Пусть M — искомый ответ. Заметим, что ответ не меньше, чем пожелание каждого из спонсоров, так как каждому из них нужна уникальная задача. А также, ответ не может быть меньше $\left\lceil \frac{\sum s_i}{K} \right\rceil$, так как всего упоминаний $\sum s_i$ должно быть не больше, чем $M \cdot K$. Значит,

$$M = \max(\max_{i=1}^{N}(s_i), \left\lceil \frac{\sum s_i}{K} \right\rceil).$$

Пусть задачи нумеруются с нуля. Тогда, чтобы построить пример, нужно по очереди каждого из спонсоров записывать в задачи по порядку, то есть первого спонсора записать в задачи с номерами $0, 1, \ldots, (s_1 - 1) \mod M$, второго спонсора — в $s_1 \mod M, (s_1 + 1) \mod M, \ldots, (s_1 + s_2 - 1) \mod M$, третьего — в $(s_1 + s_2) \mod M, (s_1 + s_2 + 1) \mod M, \ldots, (s_1 + s_2 + s_3 - 1) \mod M$ и так далее. Таким образом, ни один из спонсоров не окажется в одинаковых задачах, а также в каждой из задач будет не больше K записанных спонсоров.

Задача L. Мета-условие 3

Решим эту задачу методом динамического программирования. Пусть $dp[i][j]$ — количество способов записать i первых спонсоров в комплект, причём ровно j задач — с одним спонсором. Тогда $dp[0][0] = 1$, $dp[0][i] = 0$.

Будем заполнять массив динамического программирования вперёд из $dp[i][j]$. Заметим, что количество задач без единого спонсора в условии f равно $M - (\sum_{q=1}^{i} s_q - j)/2 - j$. Если $\sum_{q=1}^{i} s_q - j$ не делится на 2 либо отрицательно, то этот вариант надо пропустить.

Для текущего спонсора $i + 1$ можно дописать его название как в задачи без спонсоров, так и в задачи с одним спонсором. Переберём, во сколько условий с одним спонсором запишется название этой компании. Назовём это количество h, тогда оно пробежит отрезок от 0 до $min(s_{i+1}, j)$. Тогда количество задач без спонсоров, куда запишется название этой компании, будет равно $e = s_{i+1} - h$. Если $e > f$, то этот вариант надо пропустить. Иначе увеличим значение $dp[i + 1][j - h + e]$ на $dp[i][j] \cdot C_j^h \cdot C_f^e$ — это соответствует тому, что мы добавили очередного спонсора, заняв e пустых задач и h задач, в которых уже есть один спонсор. Здесь C_n^k — это количество сочетаний из n по k, которое можно предподсчитать заранее с помощью треугольника Паскаля.

Ответом будет число $\sum_{i=0}^{M} dp[N][i]$.

С первого взгляда этот алгоритм работает за $O(N \cdot M^2)$, однако суммарно h переберётся $O(M \cdot \sum_{i=1}^{N} s_i) = O(M^2)$ раз, поэтому суммарная сложность (включая предподсчёт количества сочетаний C_n^k за $O(M^2)$) равна $O((N+M) \cdot M)$.

Задача M. Гусеницы 2

Решение делится на три случая:

1. Все три точки на одной прямой. Тогда ответ равен длине наибольшего из трёх отрезков, образованных точками, делённой пополам;

2. Три точки образуют тупоугольный треугольник. Тогда им выгодно собраться в центре самой длинной стороны. Ответ также равен длине наибольшего из трёх отрезков, делённой пополам;

3. В остальных случаях ответ равен радиусу описанной окружности этого треугольника. Его можно вывести по формуле $R = \frac{a \cdot b \cdot c}{4 \cdot S}$, где a, b, c — длины сторон треугольника, а S — его площадь.