A. Чемпионат мира

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Изначально нужно понять следующий факт. Так как количество команд в каждом раунде было чётным, то n обязательно является степенью двойки.

Будем решать задачу в 0-индексации команд, то есть сразу уменьшим a и b на единицу.

Будем узнавать номер матча, в котором играют команды с изначальным номерами a и b в каждом раунде. Команда a будет играть в матче номер $a / 2$, а команда b в матче номер $b / 2$.

Если $a / 2 = b / 2$, то эти команды будут играть в одном матче, и нужно вывести номер текущего раунда в качестве ответа. Если при этом количество оставшихся команд равно двум, то это будет финальный матч.

Если же номера матчей не совпадают, то нужно перейти к следующему раунду. При этом номер команды a станет равным $a / 2$, а номер команды b станет равным $b / 2$. Количество команд которые перейдут в следующий раунд уменьшится вдвое, то есть нужно $n = n / 2$. Этот процесс всегда конечен, так как рано или поздно команд останется всего 2 и в этом раунде состоится всего один матч, который будет финалом турнира.

B. Лабораторная работа

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Так как нужно, чтобы среднее значение измерений Ани было равно среднему значению измерений Кирилла, то и сумма всех измерений Ани должна быть равна сумме всех измерений Кирилла.

Пусть минимальное число в измерениях Кирилла равно min, а максимальное max. Тогда, если $(max - min)$ меньше либо равно единице, то Аня не сможет записать ни одного измерения, которого не будет у Кирилла, поэтому все её измерения совпадут с его измерениями.

Остаётся случай когда $(max - min) = 2$. По условию, каждое измерений Ани должно быть не менее min и не более max. Тогда переберём сколько из её измерений будет равно min от 0 до n. Тогда количество измерений, равных $(min + 1)$ и maxопределяется однозначно.

Пусть sum — это необходимая сумма всех измерений, которая равна сумме всех измерений Кирилла, а $cntMin$ — это текущее количество минимумов, которые Аня может записать в качестве результатов измерений. Тогда Ане нужно набрать за оставшиеся измерения сумму $leftSum = sum - cntMin \cdot min$.

Минимальная сумма, которую может набрать Аня за оставшиеся измерения $minSum = (n - cntMin) \cdot (min + 1)$, а максимальная — $maxSum = (n - cntMin) \cdot max$. Тогда, если $leftSum < minSum$ или $leftSum > maxSum$, то нельзя взять $cntMin$ минимальных значений и получить нужную сумму.

В противном случае, Аня должна записать измерения, равные $(min + 1)$ в количестве $(leftSum - minSum)$ штук, а все оставшиеся измерения будут равны значению max. После этого нужно обновить ответ количеством совпадающих значений min, $(min + 1)$ и max в измерениях Ани и в измерениях Кирилла.

После обновления ответа переходим в переборе к следующему значению $cntMin$.

C. Необычная яблоня

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Рассмотрим яблоки, которые могут попасть в корневую вершину дерева в момент времени t. Так как задано корневое дерево, то это те яблоки, которые находились изначально на расстоянии t от корня.

Поймем следующий факт. Предположим, что яблоки аннигилируют только тогда, когда попадают в корень. Это действительно так, ведь количество яблок, оставшихся в корне дерева в момент времени t зависит только от чётности количества яблок, попавших в него. Поэтому если два яблока одновременно оказались в какой-то вершине, отличной от корня, то можно считать, что они аннигилируют в корневой вершине позднее, ведь это не изменит чётности количества яблок, которые попадают в корень дерева в каждый из моментов времени.

Поэтому посчитаем для каждого момента времени t величину cnt_t — сколько яблок окажется в корне в момент времени t. Это можно сделать с помощью обычного обхода в ширину. Также можно использовать обход в глубину, так как заданный граф ориентированный и обладает таким свойством, что из каждой вершины выходит ровно одно ребро (за исключением корня).

Таким образом, ответ на задачу это сумма всех $cnt_t \bmod 2$ ($a \bmod b$ означает взятие числа a по модулю b) для всех t от 0 до d, где d — максимальное расстояние от какой-то из вершин дерева до корня.

D. Игра со строкой

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Переберем, какая буква c_1 будет первой в строке t. Переберем, про какую по номеру d вторую букву спросит Вася. Теперь, если пара букв (c_1, c_2) встречается единственный раз на расстоянии d, то, если во второй раз откроется буква c_2, то Вася сможет однозначно определить сдвиг.

Рассмотрим все буквы на расстоянии d от всех букв c_1, посчитаем для каждого символа c_2 количество таких букв. Это можно сделать за $O(cnt(c_1))$, где $cnt(c_1)$ — количество букв c_1 в исходной строке. Теперь, если мы выберем такое d при открытии буквы c_1, которое максимизирует число p уникальных пар (c_1, c_2) на расстоянии d, то это и будет оптимальным выбором d, а условная вероятность победы при первой букве c_1 равна $p / cnt(c_1)$.

Осталось лишь просуммировать условные вероятности для различных букв c_1. Вероятность выпадения буквы c_1 равна $cnt(c_1) / n$, поэтому итоговая вероятность равна $\sum \frac{p}{cnt(c_1)} \cdot \frac{cnt(c_1)}{n} = \frac{\sum p}{n}$.

E. Федя не врёт

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Первым делом мы хотели бы для каждой из точек от 1 до m узнать, скольким отрезкам она принадлежит. Это можно делать, например, методом сканирующей прямой за $O(m + n)$, либо деревом отрезков с прибавлением на отрезке за $O(n \cdot log(m))$.

Далее, как легко заметить, набор точек является плохим $\iff cnt(x_i)$ до некоторой позиции не убывают, а затем не возрастают, в противном случае существует тройка индексов $i < j < k$, для которой верно $cnt(i) > cnt(j), cnt(j) < cnt(k)$, наличие которой показывает, что существуют два непересекающихся отрезка(первый — обладающий самым левым из правых концов среди всех, содержащих i и второй — обладающий самым правым из левых концов среди всех отрезков, содержащих k), из чего следует, что не существует точки, принадлежащей всем отрезкам.

Осталось найди самую длинную такую последовательность x_i. Чтобы её найти, будем перебирать серединный элемент – тот, после которого cnt начинает убывать. Для него нам хотелось бы узнать длину наибольшей неубывающей подпоследовательности, заканчивающейся в нём, а также длину набольшей невозрастающей подпоследовательности, начинающейся с него. Ответ на оба эти вопроса можно получить за $O(1)$, если предпосчитать для каждого элемента массива $cnt[i]$ с помощью классического динамического программирования длину наибольшей неубывающей подпоследовательности исходной последовательности, оканчивающейся в нем, и аналогично для развернутой. Асимптотика подсчёта этой динамики: $O(m \cdot log(m))$

Итоговая сложность решения $O(m \cdot log(m))$ или $O((n+m) \cdot log(m))$ для решения с деревом отрезков.

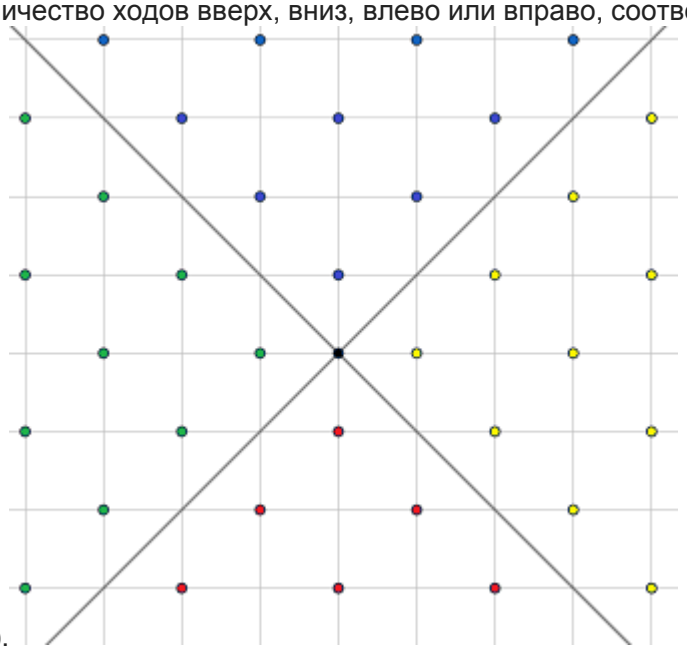
F. Игра с фишками

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Заметим, что если черная и белая фишка находятся в начале игры в точках $\{x, y\}$ с одинаковой четностью $x + y$, то перед ходом белых черная фишка не будет на манхэтеннском расстоянии 1 от белой и не может ее заблокировать. То есть черная фишка с нечетным $x + y$ может хоть как-то повлиять на игру, только если белая фишка имеет четное $x + y$, и наоборот, черная фишка с четным $x + y$ влияет на игру, только если белая фишка имеет нечетное $x + y$. Значит, можно независимо решить задачу для двух наборов черных фишек с одинаковой четностью и сложить ответы. Заметим, что решение для одной четности сводится к решению для другой четности, например, сдвигом всех фишек на 1 вверх. Поэтому дальше будем считать, что для всех черных фишек $x + y$ нечетно. Тогда нас интересуют только положения белой фишки с четным $x + y$.

Посмотрим на картинку. Если черная фишка находится в черной точке, то она может помешать белой фишке бесконечно двигаться вверх, вниз, влево, вправо, если белая фишка находится в красных, синих, желтых и зеленых точках, соответственно (то есть какие бы ходы

не сделали бы белые, количество ходов вверх, вниз, влево или вправо, соответственно, не может быть бесконечным).

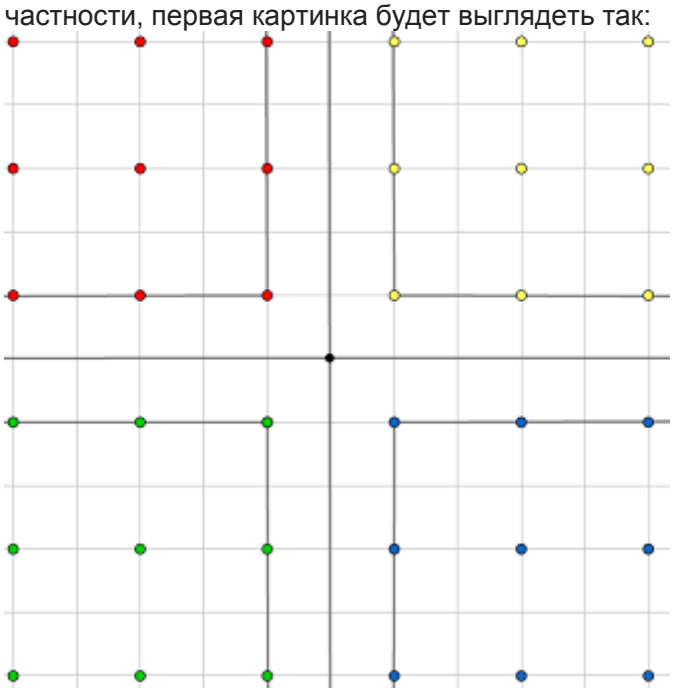


Заметим, что одна черная фишка может остановить белую фишку при движении в максимум одном направлении. Если найдутся четыре черных фишки таких, что каждая может остановить белую фишку в одном из направлений, то черные могут выиграть, используя только эти четыре фишки, и произвольно передвигая остальные. Если найдется направление, при движении в котором, белую фишку никто не может остановить, то белые выиграли.

Следовательно, каждая черная фишка порождает четыре угла разных типов. Если точка $\{x + y\}$ находится на пересечении четырех углов разных типов и $x + y$ четно, то такую точку нужно посчитать в ответе.

Заменим каждую точку $\{x, y\}$ на точку $\{x + y, x - y\}$. Тогда четыре типа углов сохранятся, а четными должны быть обе координаты белой фишки, а не только сумма координат. В

частности, первая картинка будет выглядеть так:



Оставим только точки с четными координатами и поделим каждую координату на два. По-прежнему каждая черная фишка порождает четыре угла, белая фишка должна находится на пересечение четырех углов разных типов, но больше нет требований четности чего-либо.

Как же находить число точек на пересечении четырех углов разных типов эффективно? Давайте для каждого типа углов и для каждой x-координаты найдем полуинтервал y-координат, что точка, лежащая в данном полуинтервале, покрыта углами текущего типа. Если мы сможем это найти, то нужно просуммировать для каждой x-координаты длину пересечения соответствующих полуинтервалов каждого типа, и это будет ответом.

Рассмотрим углы только одного типа, так как для других типов все делается симметрично. Пусть это углы, стороны которых направлены вверх и вправо. Тогда заметим, что для каждой x-координаты полуинтервал будет вида $[L_x, \infty)$, где L_x - это минимальная y-координата у вершин углов, находящихся не правее x. Следовательно, можно отсортировать все вершины углов по x-координате и написать довольно простой сканлайн.

G. Выставка монет

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Будем обозначать монету, лежащую вверх аверсом, за 0, а монету, лежащую вверх реверсом, за 1. Тогда нам требуется посчитать количество бинарных строк длины k таких, что на n заданных отрезках был бы хотя бы один 0, а на m заданных отрезков — хотя бы одна 1.

Воспользуемся методом динамического программирования. Пусть, для начала, $dp[i][l_0][l_1]$ — количество бинарных строк длины i таких, что последний ноль стоит на позиции l_0, последняя единица — на позиции l_1, и все ограничения на наличие нулей и единиц на заданных отрезках выполнены для всех отрезков, правая граница которых не превосходит i. Переходы построить достаточно просто: переберем, какая цифра будет стоять на позиции $i+1$, обновим l_0 и l_1 в соответствии с этой информацией, пусть новые значения будут l_0' и l_1'. Теперь рассмотрим все данные нам отрезки, заканчивающиеся в позиции $i+1$. Если среди них есть отрезки $[l, r]$, требующие наличия нуля, и при этом $l > l_0'$, или требующие наличия единицы, и при этом $l > l_1'$, то такое продолжение строки нам не подходит. Иначе добавим к ответу $dp[i+1][l_0'][l_1']$ значение $dp[i][l_0][l_1]$. Такое решение работает за $O(k^3 + n + m)$, если заранее для всех величин r сохранить отрезки, заканчивающиеся в позиции r. Такое решение слишком медленное и требует большой объем памяти.

Первое, что можно улучшить в этом решении — заметить, что либо $l_0 = i$, либо $l_1 = i$. Действительно, на позиции i должен стоять либо 0, либо 1. Поэтому вместо предыдущих подзадач достаточно рассмотреть подзадачи $dp_0[i][l_1]$ (при этом подразумевается $l_0 = i$) и $dp_1[i][l_0]$ (здесь $l_1 = i$). Переходы осуществляются аналогично предыдущему решению, асимптотика $O(k^2 + n + m)$, что тоже слишком медленно.

Далее заметим, что все позиции, в которых не заканчивается ни один интересный нам отрезок, можно рассматривать аналогично, так как переходы осуществляются одинаково. В то же время, не имеет значения, находится ли последний 0 в позиции l_0 или в позиции $l_0 + 1$, если в позиции $l_0 + 1$ не начинается ни один отрезок. Аналогично для l_1. Поэтому осуществим операцию, известную как сжатие координат. А именно, найдем все точки x_i такие, что позиции x_i и $x_i + 1$ покрываются различным подмножеством данных нам отрезков. Теперь воспользуемся методом динамического программирования, пусть $dp_0[i][l_1]$ — количество бинарных строк длины x_i таких, что последний символ — 0, последняя единица находится в диапазоне позиций от $x_{l_1 - 1} + 1$ до x_{l_1}, и все ограничения на присутствие единиц и нулей выполнены для всех данных нам отрезков, правая граница которых не превосходит x_i. Аналогично определим $dp_1[i][l_0]$. Рассмотрим возможные переходы в таком методе динамического программирования, без ограничения общности будем считать, что переход осуществляется из состояния $dp_0[i][l_1]$ в некоторое другое состояние $dp_?[i+1][?]$. Переходы из $dp_1[i][l_0]$ рассматриваются аналогично. В этом случае мы должны дописать бинарную строку длины $(x_{i+1} - x_i)$ к уже существующей. Возможны три случая:

- Все дописываемые символы равны 0, тогда переход осуществляется в состояние $dp_0[i+1][l_1]$ с коэффициентом 1, так как существует единственный способ дописать строку из нулей.
- Все дописываемые символы равны 1, тогда переход осуществляется в состояние $dp_1[i+1][i]$ с коэффициентом 1, так как существует единственный способ дописать строку из единиц, а последний ноль остается на месте x_i.
- Среди дописываемых символов есть как 0, так и 1. Тогда переход осуществляется в состояния $dp_0[i+1][i+1]$ и $dp_1[i+1][i+1]$ с коэффициентами $2^{x_{i+1} - x_i - 1} - 1$, так как

существует именно столько способов дописать строку, содержащую и нули и единицы при фиксированном последнем символе. Такой переход возможен, только если $x_{i+1} - x_i > 1$.

Кроме того, при выполнении переходов необходимо рассмотреть все отрезки, заканчивающиеся в x_{i+1} и не учитывать те переходы, которые не удовлетворяют заданным ограничениям. Такое решение работает за $O((n+m)^2 \cdot \log(k))$ и все еще работает слишком медленно. Здесь логарифм появился из-за того, что необходимо быстро вычислять значения 2^q для некоторых целых q, что можно сделать быстрым возведением в степень.

Остался последний шаг для получения полного решения. Заметим, что в динамическом программировании, описанном выше, можно разделить переходы, связанные с дописыванием цифр к строке, и учет ограничений, наложенных данными отрезками. Так, можно сначала выполнить все переходы из $dp_*[i][*]$ в $dp_*[i+1][*]$, не учитывая отрезки, заканчивающиеся в x_{i+1}, а затем учесть эти отрезки, занулив значения $dp_0[i+1][l_1]$, где $l_1 < l$, где $[l, x_i]$ — некоторый отрезок, ставящий ограничение на наличие единицы, и значения $dp_1[i+1][l_0]$, где $l_0 < l$, где $[l, x_i]$ — некоторый отрезок, ставящий ограничение на наличие нуля. Кроме того, заметим, что переходы, имеющие коэффициент, отличный от 1, затрагивают только значения $dp_*[i+1][i]$ и $dp_*[i+1][i+1]$, а значения $dp_*[i+1][j]$ при $j < i$ равны $dp_*[i][j]$ или 0. Поэтому можно не хранить ответы для подзадач с различным первым параметром динамического программирования отдельно, а хранить только два массива $dp_0[l_1]$ и $dp_1[l_0]$, подразумевая параметр i равный текущему значению. Теперь при переходе от i к $i+1$ требуется обновить значения $dp_*[i]$ и $dp_*[i+1]$, а также обнулить некоторый префикс dp_0 и некоторый префикс dp_1 в зависимости от левых концов отрезков, заканчивающихся в x_i. Обновленные значения $dp_*[i]$ и $dp_*[i+1]$ легко вычислить через сумму элементов в этих массивах и величину $x_{i+1} - x_i$. Теперь можно использовать одну из многочисленных структур данных для обнуления на префиксе и подсчета суммы и получить решение, имеющее временную асимптотику $O((n+m)\log(n+m)\log(k))$ и использующее $O(n+m)$ памяти. Такого решение уже достаточно для полного решения задачи.

Кроме того, для упрощения реализации и асимптотики можно заметить, что, так как обнуление всегда происходит на префиксе, который потом не будет обновляться никак иначе, кроме как обнуляться снова, то можно вместо структуры данных поддерживать указатель на первый еще не обнуленный элемент и увеличивать его по мере обнуления. Это также позволяет легко поддерживать текущую сумму всех элементов без использования структур данных. Такое решение имеет временную $O((n+m)\cdot(\log(n+m)+\log(k)))$ асимптотику, первый логарифм в асимптотике берется из сортировки, необходимой для сжатия координат. Именно такая идея и реализована в авторском решении.