Олимпиада школьников по программированию «ТехноКубок» 2021

Финальный тур (заключительный этап Олимпиады)

Разбор задач

A. Побег

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

Каждая сломанная стена увеличивает число связных областей плоскости максимум на 1. Поскольку в конце каждая клетка должна быть в той же области, что и внешнее пространство, в конце должна быть только одна связная область, а значит, ответ хотя бы nm (потому что вначале областей было $nm + 1$).

Добиться цели, удалив ровно nm стен, можно, например, сломав верхнюю стену в каждой клетке.

B. Восстановление модуля

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

Давайте отдельно разберем случай $c == 0$. Для этого проверим, что для любого i такого, что $1 \le i < n$ верно, что $arr[i] == arr[i + 1]$. Если это так, то модуль может быть бесконечно большим.

Если это не так и есть пара $arr[i] == arr[i + 1]$, значит с одной стороны, $(x + c)\%mod = x$, а значит $c = 0$, но этот случай мы уже отсекли. Значит ответ -1, ведь решения нет.

Теперь посмотрим на то, что $(x + c)\%mod$ может изменить x либо на $x + c$, либо на $x - (mod - c)$.

Посмотрим на все положительные дельты. Они должны быть одинаковыми. Аналогично и с отрицательными.

Если нет отрицательно или положительной дельты - модуль может быть сколь угодно большим. В противном случае модуль равен сумму модулей положительной и отрицательной дельт. Осталось только проверить, что он подходит.

C. Базовая дипломатия

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Сначала для каждого дня произвольно выберем друга из списка. Заметим, что при таком выборе максимум один друг будет играть больше $\left\lceil \frac{m}{2} \right\rceil$ раз. Назовем его f. Теперь мы хотим немного подправить расписание так, что f играет ровно $\left\lceil \frac{m}{2} \right\rceil$ раз. Это будет гарантировать корректность нашего ответа. Для этого просто пройдемся по всем дням и, если в некоторый день назначен f и может играть кто-то другой, назначим на этот день кого угодно, кроме f. Такие замены будем производить пока f играет больше $\left\lceil \frac{m}{2} \right\rceil$ раз. Существует только один случай, когда добиться этого невозможно: если в больше чем $\left\lceil \frac{m}{2} \right\rceil$ дней f является единственным другом, который может играть.

D. Плейлист

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Будем называть пару песен *плохой*, если НОД их жанров равен 1. Для двух данных песен это легко проверить за $O(\log C)$, используя алгоритм Евклида.

Так как удалений будет не более n, то мы можем просо симулировать процесс, и это уложится в ограничение по времени, если мы сможем быстро находить песню, которая будет удалена следующей. Будем поддерживать упорядоченное множество песен в плейлисте, а также другое упорядоченное множество плохих пар соседних песен в плейлисте. Легко понять, что после того, как мы удалили песню из плейлиста, в этих множествах нужно сделать лишь константное число изменений. Пусть мы удалили песню b, и в этот момент предыдущая песня в плейлисте была a, а следующая песня — c. Индексы a и c легко найти, используя первое из поддерживаемых множеств. Тогда нам нужно:

- удалить b из множества, в котором хранится плейлист;
- удалить (a, b) из множества плохих пар;
- удалить (b, c) из множества плохих пар, если это плохая пара;
- добавить (a, c) в множество плохих пар, если это плохая пара.

Затем можно перейти к следующей песне, которую нужно удалить. Это легко сделать, используя множество плохих пар.
Если мы будем поддерживать данные множества в достаточно быстрой структуре данных (например, сбалансированном дереве поиска, стандартного `set` в языке C++ достаточно), то мы получим достаточно быстрое время работы ($O(n(\log n + \log C))$).

Также эту задачу можно решить с помощью двусвязных списков или СНМ, тогда асимптотика будет $O(n \log C)$.

E. Панорама города

ограничение по времени на тест: 2.5 секунд
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Будем решать задачу методом динамического программирования. Наивный алгоритм за $O(n^2)$ выглядит следующим образом: обозначим через dp_i максимальную красоту, которую можно получить на подотрезке от 1 до i. Проверим все возможные варианты для последнего отрезка: $dp_i = \max\limits_{j \le i}(dp_{j-1} + b_{j..i})$. Осталось ускорить это решение.

Предположим, в данный момент мы считаем dp_i. Заметим, что если j — позиция ближайшего меньшего числа слева от i (то есть $h_j < h_i$, и для всех $j < j' < i$ выполнено $h_{j'} > h_i$), и если оно попадёт на одно фото со зданием i, то в таком случае наибольшее значение, которое может принять dp_i — это dp_j, поскольку красота последнего фото будет определяться зданием, которое находится не правее j.

Осталось лишь проверить все позиции от $j + 1$ до i в качестве левого здания на последнем фото. Но мы знаем, что все здания между j и i выше h_i, поэтому красота последнего фото в таком случае будет определяться зданием i, так что ответ в таком случае будет равен $dp_{k-1} + b_i$, где k — выбранная позиция. Чтобы найти это значение, нам нужна структура, поддерживающая максимум на отрезке и обновляющая значение в точке, например, дерево отрезков. Итоговое решение имеет сложность $O(n \log n)$.

Также можно пройтись по зданиям слева направо, поддерживая стек возрастающих зданий, чтобы всегда знать, где находится ближайшее менее высокое здание, и заодно поддерживать оптимальные значения dp, соответствующие каждому зданию из стека, что даёт линейное по времени решение, но это было не обязательно.

F. Полезные рёбра

ограничение по времени на тест: 5 секунд
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Посчитаем все попарные расстояния между вершинами алгоритмом Флойда за $O(n^3)$. Рассмотрим все тройки с фиксированной первой вершиной, назовём её v. Найдем все полезные рёбра только для этих троек. Ребро (a, b, w) полезное, если существует тройка (v, u_i, l_i), такая что $dist(v, a) + w + dist(b, u_i) \le l_i \Leftrightarrow -l_i + dist(u_i, b) \le -w - dist(v, a)$. Заметим что справа стоит величина, зависящая только от фиксированной вершины v, и ребра. Поэтому достаточно минимизировать левую часть по всем возможным тройкам. Это можно сделать алгоритмом Дейкстры за $O(n^2)$, если изначально проинициализировать расстояние до u_i значением $-l_i$. После этого можно для каждого ребра проверить является ли оно полезным хоть для какой-то тройки. Итоговая асимптотика $O(n^3)$.

G. Ва-банк

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Разделим наше решение на две части. Для начала найдем верхнюю границу этого самого M. Для этого будет делать запросы по степеням двойки: $1, 2, 4, \ldots$. Тогда нам всегда будет хватать денег на очередной запрос. Когда-то мы проиграем и получится, что ответ лежит на отрезке $[2^k, 2^{k+1}]$.

Это займет не более 47 запросов.

Теперь можно было бы сделать например такой бинпоиск: один раз взять левую границу, далее каждый раз брать левую границу и середину. Тогда если мы победили - все хорошо. Если проиграли - потеряли $L + \frac{R-L}{2}$. Заметим, что L мы уже взяли, а вот сумма $\frac{R-L}{2}$ будет не больше L, так как $R - L = L$, а длина каждый раз уменьшается вдвое.

Однако это решение требует еще $2\log(10^{14})$ операций, что много.

Давайте заметим, что если левая часть не больше половины, то этой дополнительной L хватит, чтобы покрыть поражения. Теперь заметим, что мы никак не используем то, что после победы мы получаем M. Давайте это исправим.

Попробуем построить оптимальное решение. Пусть текущий отрезок равен (l, r). Текущий баланс не меньше $l \cdot y + (r - l)$. Тогда, если $y == 0$ то мы должны сделать запрос в l, а если $y > 1$ - можно сделать любой запрос на отрезке.

Заметим, что после неудачного запроса наш баланс не меньше $l \cdot (y - 1) + (r - l)$ (для новых l и r), а после удачного пока предположим, что наш баланс не меньше $l \cdot (y + 1) + (r - l)$ (для новых l и r). Это не всегда так, обсудим это позже. Заметим, что при разумных действиях часть $(r - l)$ сохраняется, поэтому не будем ее рассматривать. Таким образом баланс зависит только от y.

Пусть $dp[x][y]$ такое максимальное число d, что за x запросов можно найти ответ на отрезке $(l, l + d)$ имея исходный баланс $y \cdot l + d$. В таком случае $dp[x][y] = dp[x - 1][y - 1] + dp[x - 1][y + 1]$, эти величины легко вычислить, а в решении — следовать соответствующему разбиению.

Можно показать, что $dp[k][0] = C_k^{k//2}$. Тогда можно заметить, что $k \le 49$.

Таким образом получается 97 запросов.

Однако стоит заметить, что если мы перешли от (l, r, y) в $(m, r, y + 1)$, то на самом деле у нас не y левых границ, так как баланс был $y \cdot l$ и стал $y \cdot l + m$, а нужно $(y + 1) \cdot m$.

Можно показать, что нам хватит 3 доп запросов в исходной l.

H. Экзамен

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Зафиксируем конкретную строку и найдём все рёбра выходящие из этой строки. Для каждой позиции i нашей строки найдём значение $left_i$ — позиция, в которой начинается наидлиннейшая подстрока, равная одной из строк нашего словаря, и заканчивающаяся в позиции i. Легко понять, что рёбра могут идти только в такие строки, и никакие другие. Осталось понять какие из этих строк лишние.

Строка является лишней, когда у неё есть вхождение в нашу строку, в котором она накрывается вхождением какой-либо более длинной строки. Пусть $ind_i = min\{left_j, j > i\}$. Тогда лишними будут все строки из нашего словаря, которые являются суффиксами подстроки $[ind_i, i]$, для какого-либо i. Их можно находить с помощью структуры Ахо-Корасик. Так как суффиксные ссылки в Ахо-Корасик образуют дерево, то достаточно найти вершины соответствующие подстрокам $[ind_i, i]$, и покрасить путь по суффиксным ссылкам до корня дерева. После этого можно просто проверить покрашены ли вершины, соответствующие подстроками $[left_i, i]$. Чтобы не красить путь наивным образом, можно воспользоваться структурой Дерева Отрезков или std::set. Итоговая асимптотика $O(n \cdot logn)$.

Видео-разбор задач заключительного этапа доступен по ссылке:
https://youtu.be/Bfhmjg2YNeE?t=2521