Технокубок 2020 - Финал(Разбор)

A. Контест для роботов (500 баллов)

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт

Баллы для задач, которые решат/не решат оба участника, ни на что не влияют.

Пусть есть x задач, которые решит первый участник, но не второй. Аналогично, пусть есть y задач, которые решит первый участник, но не второй. Если $x = 0$, то второй участник всегда наберёт не меньше баллов, чем первый, значит, ответа не существует.

Иначе выгоднее всего назначить задачам первого участника p баллов, а задачам второго — один балл. Тогда должно выполняться $xp > y$, или $p > \frac{y}{x}$, поэтому ответ на задачу равен $p = \left[\frac{y+1}{x}\right]$. В заданных ограничениях также можно было перебирать p по возрастанию и проверять, какие из них подходят.

B. Планирование путешествия (1000 баллов)

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест:256 мегабайт

Перепишем равенство из условия в следующем виде: $c_{i+1} - b_{C_{i+1}} = c_i - b_{C_i}$. Это означает, что для всех городов в нашем пути значение $i - bi$ будет одинаковым; более того, все города с одинаковым значением могут быть посещены в возрастающем порядке.
Поэтому можно сгруппировать все города по $i - bi$, посчитать сумму красот в каждой группе и найти максимум. В заданных ограничениях размеры групп для каждого значения $i - bi$ можно поддерживать в одном массиве (будьте аккуратны с отрицательными значениями $i - bi$!)

C. Удаление соседних (1250 баллов)

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт

Оптимальный ответ может быть получен следующим алгоритмом: выберем максимально возможную (по алфавиту) букву, которую мы можем удалить, и удалим ее. Мы можем делать это наивно и тогда асимптотика решения будет равна $O(n^2)$.
Почему это всегда правильно? Допустим, мы удаляем максимальную букву i, которая может быть использована, чтобы удалить какую-то другую букву j (очевидно, что в этом случае $s_i + 1 = s_j$).
Если между s_i и s_j нет никаких других букв, то s_i не является максимальной буквой, таким образом мы получаем противоречие с нашим алгоритмом.
Теперь предположим, что мы можем удалить все буквы между s_i и s_j. Тогда по нашему алгоритму мы сначала выберем s_j и только затем s_i.

Рассмотрим последний случай — между s_i и s_j находится хотя бы одна буква s_k. Так как мы не можем удалить s_k, то остается только два случая: $s_k > s_j + 1$ или $s_k + 1 < s_i$. Тогда мы вообще не можем использовать s_i, чтобы удалить s_j.

D. Система навигации (1750 баллов)

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт

Обозначим за d_v длину кратчайшего пути из v в t. Пусть на маршруте мы совершаем переход из вершины v в вершину u. Тогда:

- маршрут обязан перестроиться, если $d_u > d_v - 1$;
- маршрут мог перестроиться, если существует вершина $w \neq u$, такая что $d_w = d_v - 1$ (в случае, если навигатор проложил маршрут через w).

Найдем величины d_v обходом в ширину из t по обратным ребрам. Теперь мы можем легко определить для каждой вершины, нужно ли ее прибавлять к минимальному количеству (вершин, где маршрут обязан перестроиться) и максимальному количеству (вершин, где маршрут мог перестроиться).

E. World of Darkraft: Battle for Azathoth (2250 баллов)

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт

Пусть S_a — множество монстров, которых можно победить оружием с атакой a. Тогда выгода, которую мы получим, если воспользуемся оружием i с атакой a и броней j с защитой b равна $-ca_i - cb_j +$ (сумма z_k всех монстров k в множестве S_a с $y_k < b$).

Будем рассматривать варианты оружия по возрастанию их атаки и обновлять S_a. Для каждого варианта брони будем поддерживать величину $D_j = -cb_j +$ (суммарное количество монет монстров с атакой меньше b_j). Если варианты брони упорядочены по возрастанию защиты, то добавление нового монстра в S_a прибавляет какое-то число к суффиксу значений D_j. Таким образом, можно поддерживать значения D_j в дереве отрезков.

В итоге решение выглядит так:

- Построим дерево отрезков на значениях $-cb_j$, где варианты брони упорядочены по возрастанию b_j.
- Рассматриваем варианты оружия и монстров по возрастанию a_i / x_k. Для каждого нового оружия i добавим всех новых неучтенных монстров с $x_k < a_i$ в множество (двигая указатель в отсортированном массиве).
- Добавление каждого монстра — прибавление z_k к суффиксу D_i.
- После этого для текущего оружия i попробуем улучшить ответ значением $-ca_i + \max D_j$.

F. Древляндия и вирусы (2750 баллов)

ограничение по времени на тест: 3 секунды
ограничение по памяти на тест: 512 мегабайт

Мы можем выкинуть все вершины, не лежащие ни на каком пути между v_i и u_i. Более того, мы можем сжать цепочки в новом поддереве, оставив только вершины степени >2. Найти множество вершин такого «сжатого» дерева можно так: упорядочим все вершины v_i и u_i по времени входа обхода в глубину, после чего добавим в множество LCA всех пар соседних вершин в таком порядке.

Восстановить рёбра сжатого дерева можно, рассматривая все вершины по возрастанию времени входа и поддерживая путь до текущей вершины в стеке. Для новой вершины нужно откинуть из стека все последние вершины, не являющиеся предками нашей, тогда вершина стека является нашим родителем.

На сжатом дереве решать задачу можно несколькими способами:

- Алгоритм Дейкстры. Расстоянием до вершины v будем считать пару (время заражения v, минимальный номер вируса, который доберется до v за такое время). Инициализируем парами $(0,i)$ стартовые вершины v_i. Для текущей вершины в Дейкстре попробуем заразить всех ее соседей ее вирусом.
- ДП по поддеревьям. Сперва для каждой вершины найдем вирус, который ее заразит, если рассматривать только ее поддерево: это либо вирус, который в ней начинает, либо вирус, заражающий ее ребенка. После этого пойдем в обратную сторону: сверху вниз попробуем заразить каждого ребенка каждой вершины v ее вирусом.

В любом случае, общая сложность составляет $O((k+m) \log n)$ на запрос.

G. Достижимые строки (3250 баллов)

ограничение по времени на тест: 3 секунды
ограничение по памяти на тест: 256 мегабайт

Когда одну строку можно превратить в другую? Очевидно, количество единиц в двух строках должно совпадать. Также заметим, что сохраняется следующий инвариант: если из строки удалить все пары соседних единиц, то набор позиций оставшихся единиц должен сохраниться.

Можно показать, что этих инвариантов достаточно: если сдвинуть все пары единиц вправо, строки совпадут, если совпадают инварианты (по факту, сдвиг всех пар единиц вправо равносилен их удалению).

Одно из наиболее простых в идейном плане решений — использовать дерево отрезков, и для каждой вершины хранить:

- количество удаленных пар соседних единиц;
- хэш множества позиций оставшихся единиц;
- символы на краях соответствующего отрезка (чтобы можно было удалять появляющиеся пары соседних единиц).

При слиянии двух вершин мы можем попытаться удалить пару из единиц, если она появляется на стыке, и пересчитать хэш для новой вершины.
Существует также множество других решений, в том числе детерминированное решение с помощью суффиксного массива.

H. Блоки и сенсоры (3250 баллов)

ограничение по времени на тест: 3 секунды
ограничение по памяти на тест:512 мегабайт

Изначально предположим, что пустых ячеек нет. Будем удалять кубики, которых не может быть ни в одном ответе, и поддерживать, какой кубик видит каждый сенсор в текущей конфигурации.
Мы обязаны удалить кубик, если:

- его видит сенсор, в позиции которого не должно быть ни одного кубика;
- его видят два сенсора, которые должны видеть разные цвета.

В любом из этих случаев мы удаляем кубик. После этого в каждой проекции для соответствующего сенсора, если нужно, сдвинем указатель на следующий кубик, который он видит. Если в результате этого находится сенсор, который должен видеть кубик, но не видит, то ответа нет.
Если все хорошо и кубики удалять больше не нужно, то мы нашли ответ. Действительно — сенсоры, которые не должны видеть кубики, действительно их не видят, и для каждого кубика сенсоры, которые его видят, согласны насчет его цвета: назначим этот цвет данному кубику (если кубик не видит ни один сенсор, его можно удалить или назначить произвольный цвет).
Поддерживать кубики, которые нужно удалить, можно в стеке или очереди; можно даже написать рекурсивную функцию, которая удаляет кубик, пересчитывает сенсоры, которые видели его, и удаляет другие кубики, для которых нарушается условие.