Разбор задачи «Достойный финал»

Для решения данной задачи воспользуемся методом динамического программирования. Состоянием будет характеризоваться позицией клетки, из которой начали путь, числом поворотов и направлением в котором идем. Будем хранить максимальное число шашек, которое перепрыгнули на таком пути.

Так как поворотов не более два, то если мы запретим поворачивать дважды в одной клетке, любой наш путь с не более, чем двумя поворотами не будет иметь самопересечений. Заметим, что интересных нам клеток немного, так как нас интересуют только клетки соседние с клетками с черными шашками или с белой шашкой, поэтому можно сжать координаты и тогда состояний динамики будет $O(n)$.

Также заметим, что значение динамики в клетке, зависит только от значения в соседней в заданном направлении клетки, и от значения в соседних в направлениях перпендикулярным нашему с числом поворотов на единицу меньше. То есть всего $O(1)$ переходов.

Ответом будет максимум посчитанных значений. Итоговая сложность $O(n)$ на подсчет состояний и $O(n \log n)$ на сжатие координат.

Разбор задачи «Карточная игра»

Будем пользоваться только операцией $Move$. Будем по одной откладывать карты сверху колоды и смотреть, что отвечает жюри. Как только в отложенной колоде будет $\frac{k}{2}$ карт, лежащих рубашкой вниз, мы победили (а такой момент точно наступит, так как в колоде $\geqslant k$ карт).

Также можно было заметить, что на самом деле это задача-шутка, которая может решаться и без использования ответов жюри. Сделаем следующие операции: верхние k карт из колоды перевернем и отложим во вторую колоду. Тогда, если среди этих карт было x карт, лежащих рубашкой вниз, теперь их стало $k-x$. В изначальной же колоде также осталось $k-x$ карт, лежащих рубашкой вниз, то есть их количество совпадает. Однако, в этом решении требовалось отдельно рассмотреть случаи $k = 0$ и $k = n$, чтобы после перекладывания колоды остались непустыми.

Разбор задачи «Просчет событий»

Для начала заметим, что если действие b_j можно выполнить, то $b_j = (a_{i_{0,0}} \wedge a_{i_{0,1}} \wedge \ldots \wedge a_{i_{0,k_0}}) \vee (a_{i_{1,0}} \wedge a_{i_{1,1}} \wedge \ldots \wedge a_{i_{1,k_1}}) \vee \ldots \vee (a_{i_{x-1,0}} \wedge a_{i_{x-1,1}} \wedge \ldots \wedge a_{i_{x-1,k_{x-1}}})$.

Для каждого b_j будем решать задачу по битам. Рассмотрим i-й бит b_j, который равен 1. Рассмотрим одно из слагаемых $a_{i_{x,0}} \wedge \ldots \wedge a_{i_{x,k_x}}$, в котором i-й бит тоже равен 1. Разумно это слагаемое сделать как можно меньшим числом, чтобы при последующих операциях $\vee$ добавлялось как можно меньше единиц. Обозначим минимальное значение этого слагаемого за c_x. Это значение равно $\wedge(bit(a_t, i) = 1)$, то есть $\wedge$-у всех чисел a_t, у которых i-й бит равен 1.

После предподсчета s_x для всех битов x, посчитать ответ для каждого b_j довольно просто: если $\vee$ всех s_x, таких, что $bit(b_j, x) = 1$, равен b_j, ответ «`YES`», иначе — «`NO`». Доказательство этого факта остается в качестве упражнения. Также нужно не забыть случай $b_j = 0$ — в этом случае ответ равен «`YES`», если $a_0 \wedge a_1 \wedge \ldots \wedge a_{n-1} = 0$.

Разбор задачи «Помогите спасти Землю!»

Заметим, что состояние однозначно задает множество супергероев, находящихся на Земле, и планета, на которой находится корабль. Получаем $2^n \cdot 2$ различных состояний. Переходы из состояния — это все возможные подмножества множества супергероев, находящихся на той планете, на которой стоит корабль. Выбранное подмножество полетит на корабле на другую планету. Суммарно получается $3^n \cdot 2$ перехода. Далее, в получившемся графе нужно найти любой путь от состояния, задающего стартовую конфигурацию, в состояние, задающее требуемую конфигурацию. Например, с помощью алгоритмов bfs или dfs. Так как максимальное возможное количество вершин в графе не превышает 100 000, любой найденный простой путь по длине не будет превышать это ограничение.

Разбор задачи «Поиск корабля»

В условии задачи описан неориентированный невзвешенный граф. Обозначим через d_i длину кратчайшего пути от вершины s до вершины i. Значения d_i можно найти, запустив в графе поиск в ширину.

Обозначим через T_i множество всех вершин, лежащих на хотя бы одном кратчайшем пути из вершины s в вершину i. Найдем множества T_i для всех вершин графа. Для этого упорядочим вершины по возрастанию значения d_i, и в этом порядке будем строить искомые множества для вершин. Заметим, что если какой-то кратчайший путь из вершины s заканчивается в вершине v, то предпоследняя вершина этого пути может быть расположена в вершине u тогда и только тогда, когда $d_v = d_u + 1$, а также вершины u и v соединены ребром. Таким образом, множество T_v — это объединение множеств T_u для всех подходящих вершин u. Также в множество T_v нужно не забыть добавить саму вершину v.

Для того, чтобы быстро построить множества T_i, воспользуемся битовым сжатием: его можно реализовать самостоятельно, либо с помощью структуры данных bitset. Эта структура данных позволяет хранить множества чисел от 1 до n и производить операцию объединения за время $O(\frac{n}{w})$, где w — машинное слово, равное 32 или 64, в зависимости от параметров системы.

Для того, чтобы ответить на запрос, необходимо выяснить, какие вершины лежат на кратчайшем пути от s до v, причем расстояние до самих этих вершин от s равно k. Для того, чтобы получить множество всех таких вершин, возьмем множество вершин, которые лежат на кратчайших путях из s в v, и удалим из него все вершины, кроме тех, для которых $d_i = k$. Заметим, что так как мы упорядочили вершины в порядке возрастания величины d_i, искомые вершины будут образовывать непрерывный подотрезок, а значит, необходимые операции удаления с использованием битового сжатия можно сделать за время $O(\frac{n}{w})$, используя операции битового сдвига. После того, как в множестве остались лишь подходящие вершины, необходимо вывести ответ: если множество пусто, то ответ 0, если в нем более одного элемента, то ответ -1, а в противном случае ответ — это вершина, которая содержится в найденном нами множестве. Для проверки всех этих условий за время $O(\frac{n}{w})$ также можно использовать битовое сжатие.

Асимптотика времени работы программы, а также памяти, используемой программой, составляет $O(\frac{n(m+q)}{w})$.

Разбор задачи «Возвращение к домашней работе»

Будем хранить строку в персистентном декартовом дереве, где одна вершина соответствует одному символу, в вершине хранится флаг, означающий, развернуто ли ее поддерево. Также, будем хранить в каждой вершине матрицу d размера 4×4, где d_{ij} — длина самой большой монотонной последовательности, начинающейся с числа i и заканчивающаяся числом j (i может быть больше j).

Чтобы найти длину последовательности в поддереве, если строка не развернута, нужно взять $\max\limits_{i \leqslant j} d_{ij}$, а если развернута, $\max\limits_{i \geqslant j} d_{ij}$.

Чтобы дублировать строку, воспользуемся тем, что декартово дерево персистентно. Для начала, научимся удваивать строку. Пусть мы хотим удвоить строку, которой соответствует поддерево вершины v. Тогда, для этого нужно сделать операцию merge вершины v самой с собой. Персистентное дерево позволяет это сделать, потому что оно никогда не меняет старые вершины, а всегда создает их копии. Для того, чтобы продублировать строку s k раз, нужно, как в бинарном возведении в степень, получить строку s, продублированную 2^0 раз, 2^1 раз, 2^2 раз, 2^3 раз, и так далее. А потом сделать операцию merge логарифма нужных из данных строк.