Разбор задачи «Прибытие Таноса»

Заметим, что перестановок из 8 цифр всего 8!, то есть 40320. Значит можно просто перебрать все перестановки из 8 цифр, и проверить, что первые 4 образуют корректный год, следующие две цифры образуют корректный месяц, а последние две цифры образуют корректный день в этом месяце. Что бы перебрать все перестановки, достаточно написать рекурсивный перебор или воспользоваться стандартный функцией *next_permutatuoion* в c++, или аналогичными в других языках. Что бы вывести только уникальные даты, достаточно их положить в стандартную структуру данных множество, например *set* в c++.

Разбор задачи «Зал брони»

Пусть мы выбрали точку в которой будет располагаться люк. Тогда если увеличить эту координату на единицу, сумма расстояний до нее от всех костюмов увеличится на число костюмов, которые находятся в меньших координатах и уменьшится на количество костюмов, которые до увеличения находились в больших координатах.

Отсюда получим, что если можно поставить люк между какими-то двумя отсеками, можно поставить в каком-то из них, и ответ не ухудшится.

Тогда переберем все точки, в которых находится хотя бы один отсек, поддерживая сколько костюмов находятся до и после текущей координаты. И таким образом найдем оптимальную точку.

`Замечание:` Из выше описанных рассуждений можно заметить, что оптимальный ответ всегда будет достигаться в первой координате, такой что сумма костюмов в ней и предыдущих точках больше либо равна половине всех костюмов.

Разбор задачи «Ресторан»

Сперва для каждого ресторана нужно выписать тройку чисел – какое место этот ресторан занимает в списке каждого из мстителей. После этого сравнивать пару ресторанов становится очень легко – тот у которого хотя бы два числа из тройки меньше, тот и лучше. А значит можно поступить так: сперва одним проходом найти потенциально-лучший ресторан (каждый раз сравнивать текущий лучший с новым и обновлять, если потребуется), а затем еще одним проходом проверить является ли этот ресторан действительно лучшим. Сложность этого решения всего $O(N)$.

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 $a_0 \wedge a_1 \wedge \ldots \wedge a_{n-1} = 0$.