Задача A. Обезвреживание бомбы

Автор задачи и разработчик: Даниил Орешников

Для решения этой задачи достаточно было заметить, что если есть ровно x кнопок, на которых написано число i, и y кнопок, на которых написано число $k - i$, то чтобы бомба не взорвалась, на эти $x + y$ кнопок придется потратить хотя бы $\min(x, y)$ секунд. Действительно:

- если сделать меньше $\min(x, y)$ нажатий, по принципу Дирихле останется хотя бы по одной кнопке в каждой группе, и будут две кнопки с суммой чисел k
- а ровно столько нажатий хватает, потому что можно просто нажать все кнопки из группы меньшего размера, и для оставшихся не будет пары, дающей с ними в сумме k

Очевидно также, что кнопки, на которых написаны числа $\geqslant k$, можно игнорировать, потому что они никак не влияют на ответ. Таким образом, давайте посчитаем количество кнопок, на которых написано i, для каждого i от 1 до $k - 1$, а затем сложим описанные выше минимумы. Отдельно следует обратить внимание на то, что для кнопок, на которых написано ровно $\frac{k}{2}$ (при четных k), надо сделать исключение — ровно одна кнопка с таким числом может остаться активной, и бомба не взорвется.

Первые две группы тестов можно было пройти, используя массив для подсчета количества кнопок (`cnt[i]` — количество кнопок с числом i). Третью группу тестов так пройти не получится, потому что завести массив длиной 10^9 не представляется возможным. Однако можно было использовать отношение (`map`, `dict`) вместо массива, а при подсчете ответа проходить только по тем числам, которые написаны хотя бы на одной кнопке. В таком случае факт того, что a_i могут достигать 10^9, не мешает нам посчитать ответ.

При этом первую группу тестов можно было пройти, написав полный перебор всех подмножеств кнопок, которые можно нажать, и выбрав оптимальный вариант, при котором бомба не взрывается. Подробности такого решения мы приводить здесь не будем.

Задача B. Поврежденный пароль

Автор задачи и разработчик: Николай Будин

Рассмотрим строку s', которая получается из строки s заменой символа на позиции x на c. Строка s' должна являться подпоследовательностью строки t.

Проверить, что строка a является подпоследовательностью строки b можно с помощью жадного алгоритма. Будем идти по строке a и поддерживать указатель p на символ строки b. Пусть мы стоим на i-м символе строки a. Пока $b[p] \neq a[i]$, увеличиваем p. Если вышли за границу b, то a не является подпоследовательностью b. Если нашли $b[p] = a[i]$, то увеличиваем p еще на один и переходим к $i + 1$. Алгоритм работает за $O(|a| + |b|)$.

Отсюда получается решение за $O(|s| \cdot \Sigma \cdot |t|)$, где Σ — размер алфавита. Переберем x и c, проверим что s' является подпоследовательностью t.

Заметим, что можно не перебирать c. Если мы зафиксируем x, тогда сначала выполним жадный алгоритм для префикса строки s до позиции x не включительно. Затем, нам нужно было бы идти по символам t, пока не найдется символ равный c. Очевидно, что нам выгодно выбрать в качестве c первый символ, который не будет равен $s[x]$. После чего, продолжить жадный алгоритм для оставшегося суффикса s начиная с позиции $x + 1$. Таким образом, получится решение за $O(|s| \cdot |t|)$.

Для решения на полный балл, снова переберем позицию x. Для префикса строки s до позиции x можно выполнить жадный алгоритм. Для суффикса строки s можно выполнить аналогичный жадный алгоритм, но идти по обеим строкам с конца. Тогда в строке t останется отрезок позиций, на котором нужно выбрать символ c (отличный от $s[x]$). Можно, например, либо взять символ на левой границе отрезка, если он отличается от $s[x]$, либо взять ближайший справа отличающийся от него. Для каждого символа найти ближайший справа символ, отличающийся от него, можно с помощью динамического программирования.

Наконец, заметим, что не нужно для каждого префикса (и суффикса) заново с начала выполнять жадный алгоритм. Можно за для всех префиксов найти эти значения один раз запустив жадный алгоритм для полной строки s.

Задача C. Кибер-взлом

Автор задачи и разработчик: Николай Будин

Чтобы решить третью подзадачу, можно было заметить, что каждому из игроков выгодно, независимо от действий второго, ходить так, чтобы он смог сделать как можно больше ходов. Для каждой вершины можно вычислить $len[v]$ — длину самого длинного пути, начинающегося в вершине v. Для всех вершин, из которых достижим цикл, $len[v] = \infty$, после чего можно удалить все циклы, граф станет ациклическим, можно будет построить его топологическую сортировку и вычислить значения для оставшихся вершин с помощью метода динамического программирования. Пара вершин v, u входит в ответ, если $len[v] > len[u]$.

Для решения оставшихся подзадач, нужно было заметить, что в этой задаче был описан процесс, который является несимметричной комбинаторной игрой. Поймем, какие у этой игры состояния. Перед тем, как ход сделает первый игрок, нам важна только информация о позициях токенов первого и второго игроков. Перед ходом второго игрока, нам помимо позиций токенов также нужно знать символ, по которому перешел первый игрок на последнем ходу.

Можно построить граф, в котором вершины соответствуют описанным состояниям, а ребра соответствуют ходам игроков и всегда ведут из состояния первого типа в состояние второго и наоборот. Состояние, из которого нет ни одного перехода, является проигрышным. Если из состояния есть переход в проигрышное, то оно выигрышное. Если из состояния все переходы ведут в выигрышные, то оно проигрышное. Ответом является количество выигрышных состояний первого типа.

Таким образом, если бы граф состояний был ациклическим, можно было бы сделать его топологическую сортировку и для каждого состояния вычислить выигрышность/проигрышность с помощью метода динамического программирования. Можно доказать, что если исходный граф является ациклическим, то и граф состояний является ациклическим. Поэтому, это позволяет решить четвертую подзадачу. Для ускорения, можно не строить граф явно и воспользоваться ленивым динамическим программированием.

Если же граф состояний содержит циклы, то некоторые состояния являются ничейными, это означает, что игра из этого состояния никогда не завершится. Для того, чтобы определить тип каждого из состояний в циклической игре, можно воспользоваться стандартным методом — ретроспективным анализом. Его суть заключается в том, что он старается определить типы вершин начиная с тех, из которых нет переходов (напомним, что эти вершины являются проигрышными). Если в какой-то момент из вершины, тип которой мы еще не определили, появляется ребро в проигрышную, то эту вершину мы помечаем выигрышной. Если в какой-то момент из вершины, тип которой мы еще не определили, все ребра начинают вести в выигрышные, вершина становится проигрышной. Если таких вершин не осталось, процесс прерывается. Все вершины, тип которых мы не определили, помечаются ничейными.

Задача D. Побег из здания

Автор задачи: Даниил Орешников, разработчик: Арсений Кириллов

Общая идея

Заметим, что если k раз использовать чип на этаже x, то в следующий раз нужно будет использовать чип на этаже $x + 2 \cdot k$. Посчитаем методом динамического программирования величину dp_i — минимальное количество энергии, которое нужно потратить, чтобы оказаться вместе с полицией на этаже i.

Решение за $O(n^2)$

Заметим, что если этажи i и j находятся на расстояние, кратном $2 \cdot t_i$, то можно попасть с этажа i на этаж j, потратив $q_i \cdot \frac{j-i}{2 \cdot t_i}$ энергии. Обновим значение dp_j величиной $dp_i + q_i \cdot \frac{j-i}{2 \cdot t_i}$. Мы научились считать эту динамику за $O(n^2)$.

Полное решение

Заметим, что $\frac{j-i}{2 \cdot t_i}$ это тоже самое, что $\lfloor \frac{j}{2 \cdot t_i} \rfloor - \lfloor \frac{i}{2 \cdot t_i} \rfloor$, ведь i и j имеют одинаковый остаток по модулю $2 \cdot t_i$. Тогда значение dp_j можно посчитать, подставив $x = \lfloor \frac{j}{2 \cdot t_i} \rfloor$ в уравнение $q_i \cdot x + dp_i - q_i \cdot \lfloor \frac{i}{2 \cdot t_i} \rfloor$. Заведём для каждого остатка по модулям 2, 4 и 6 своё дерево Ли Чао. Тогда значение dp_j можно получить как минимальное значение, полученное из этих деревьев, для каждого числа t_i с нужным остатком. Получилось решение за $O(n \log n)$.