Разбор задачи «Alphabet Contest»

Предположим, что стоя в очередной позиции i данной строки, мы знаем, что все предыдущие буквы могли произнести x детей, также знаем для всех этих детей последнюю произнесенную букву. Нужно понять, кто мог произнести букву на позиции i.

Если все дети уже произнесли её или произнесли одну из следующих в алфавите букв, то есть их последний символ совпадает с рассматриваемой буквой или стоит в алфавите позже, то никто из этих детей произнести эту букву не мог. Значит, нужно добавить ребёнка, для которого эта буква на текущем этапе будет первой и единственной произнессённой, к числу ошибок для данной конфигурации прибавить номер этой буквы в алфавите (в нумерации с нуля) и перейти к следующей букве в строке.

Если же существуют дети, которые до этой буквы ещё не дошли, то есть их последняя буква стоит в алфавите раньше, то из них выберем того, у кого последняя произнесенная буква имеет наибольший номер среди последних букв таких детей. Прибавим к текущему числу ошибок разность номеров рассматриваемой буквы и последней буквы выбранного ребенка. Скажем теперь, что текущая буква является последней буквой для этого ребенка, и перейдем к следующей букве в строке.

Начав в позиции 0 данной строки и обработав все буквы в строке описанным образом, сравним полученное число ошибок с данным: если полученное число больше, то данные некоректны («`Impossible`»), иначе получим ответ на задачу — минимальное число детей, участвовавших в олимпиаде.

Разбор задачи «Need More T-shirts!»

В задаче требовалось для каждого i решить, либо a_i — это число предметов типа i, либо a_i — это процентная составляющая i-го типа среди всех по числу предметов.

Разделим предметы на два множества: проценты и количества.

Пусть сумма a_i из первого множества равна p. И пусть всего предметов n штук. Тогда $n = n \cdot \frac{p}{100} + v$, где v — число предметов из второго множества, $v = \sum a_i - p$.

$$100n = n \cdot p + 100v$$

$$n = \frac{100v}{100 - p}$$

Проверим, что p процентов можно получить. Надо разбить на два множества, первое суммой p, второе суммой $\sum a_i - p$, это задача о рюкзаке. Но также надо не забыть об ограничении, что если a_i — это процент, то $\frac{a_i \cdot p}{100}$ должно быть целым числом. Поэтому в задачу о рюкзаке нужно включить только те типы, для которых $\frac{a_i \cdot p}{100}$ — целое.

В итоге решение:

1. Переберем p от 0 до 100;
2. Выделим только те a_i, для которых $\frac{a_i \cdot p}{100}$ — целое;
3. Решим задачу о рюкзаке для этих a_i;
4. Если мы смогли набрать p, то $n = \frac{100 \cdot (\sum a_i - p)}{100 - p}$ добавим в ответ.

Разбор задачи «The Final Countdown»

Обозначим как $f(x)$ число включенных сегментов в числе x. Посмотрим, на величину $f(x-1) - f(x)$. Если x не делится на 10, то все цифры, кроме последней, в числах x и $x+1$ одинаковые и разница $f(x) - f(x-1)$ равна разнице числа включенных сегментов в последних цифрах чисел x и $x-1$. Таким образом, если мы выпишем разности чисел, данных во вводе, в них должен повторяться паттерн $4, -3, 0, 1, -1, -1, 3, -4, 1$. Обнаружив этот паттерн, и удалив все

соответствующие числа, мы оставим только числа, которые делятся на 10. Применим для них те же рассуждения. В конце, когда останется одно число, задача сведется к следующей: дано число k, нужно найти число x таких, что $f(x) = k$, а так же найти m таких чисел x.

Чтобы найти число таких x, применим метод динамического программирования. Пусть $d[k]$ — ответ для числа k. Переберем последнюю цифру числа c, тогда если ее убрать, число сегментов в оставшихся цифрах будет равно $k - f(c)$. Для этого числа ответ посчитан в $d[k - f(c)]$. Сложив эти числа, получим ответ для k. Чтобы найти какие-то m подходящих чисел, достаточно запустить перебор с возвратом.

Отдельно следует разобрать случай, когда длина числа числа на дисплее меняется. Это происходит, когда число x равно $100\ldots00$, а $x - 1$ равно $99\ldots99$, этот случай легко обнаружить, потому что в этом случае $f(x - 1) - f(x)$ будет равно -2. Кроме этого, есть несколько тонкостей, которые следует аккуратно обработать в решении.

Разбор задачи «Painting»

В задаче ответ может быть одного из трёх типов:

- Не существует ни одного решения
- Существует ровно одно решение
- Существует хотя бы два решения

Для начала, научимся отличать случай, в котором решения не существует, от случаев, в которых решение существует. Для каждого цвета рассмотрим множество клеток, покрашенных в этот цвет. По условию, это множество не пусто. Рассмотрим bounding box этого множества. Bounding box — это наименьший по размеру прямоугольник, содержащий все клетки множества. Несложно заметить, что если существует какое-то решение, существует решение, в котором в i-й цвет был покрашен прямоугольник, совпадающий с bounding box-м множества i-го цвета. Действительно, если прямоугольник, который был покрашен в i-й цвет, не покрывает bounding box i-го цвета, то какая-то клетка i-го цвета точно не будет покрашена в i-й цвет. С другой стороны, все клетки, вне bounding box-а i-го цвета, в итоге не должны быть покрашены в i-й цвет. Значит, от того, что мы их не покрасим, решение не перестанет быть верным. Поэтому, можно прямоугольник, который мы покрасим в i-й цвет, обрезать до bounding box-а.

Рассмотрим клетки, покрытые bounding box-м i-го цвета. Если среди них есть клетки цвета 0, то решения не существует. Иначе, построим граф, в котором вершины будут соответствовать цветам. Проведем ориентированные ребра из вершины, соответствующей цвету i, в вершины, соответствующие цветам, попавшим в bounding box i-го цвета. Ребро из вершины u в вершину v означает, что прямоугольник цвета u должен быть покрашен раньше прямоугольника цвета v. Поэтому, если граф построен, то несложно найти ответ. Если в графе есть цикл, то решения нет. Иначе, построим топологическую сортировку графа и в таком порядке покрасим прямоугольники.

Для того, чтобы найти все цвета, встречающиеся в прямоугольнике, можно воспользоваться битовым сжатием и методом сканирующей прямой. Запустим вертикальную сканирующую прямую слева направо. В каждой строке будем поддерживать множество открытых прямоугольников, покрывающих эту строку, с помощью bitset-а. Для каждой клетки обновляем множество прямоугольников, содержащих цвет этой клетки. Получается время работы $\frac{n \cdot m \cdot k}{64}$.

Осталось проверить, единственный ли ответ. Если ответ не единственный, то существует второе решение, в котором все прямоугольники равны bounding box-ам, а один из прямоугольников равен bounding box-у, объединенному со столбцом клеток, касающихся его с одной из сторон. Переберем прямоугольник и сторону. Нужно проверить, продолжит ли существовать ответ, если прямоугольник расширить на 1 в эту сторону. Если среди добавляемых клеток есть клетка цвета 0, то ответа существовать не будет. Иначе, в граф нужно добавить ребра из вершины, соответствующей цвету изменяемого прямоугольника, в вершины, соответствующие цветам присоединенных клеток. И если в графе не появился цикл, то ответ существует.

Заметим, что все ребра, которые добавляются в граф, выходят из одной вершины. Поэтому, если в графе появится цикл, появится также и цикл, проходящий ровно по одному из добавленных ребер. С помощью битового сжатия, сделаем транзитивное замыкание исходного графа. Теперь нужно только проверять при добавлении ребра из u в v, что в графе не было пути из v в u.

Транзитивное замыкание графа можно построить за время $\frac{k^3}{64}$. А перебор прямоугольника и проверка, можно ли его расширить в одну из сторон, за $k \cdot (n+m)$.

Разбор задачи «Chicken Farm»

Решение будет работать за время $O(nm \cdot \log(m))$.

Зафиксируем позицию, которую должны покрывать кормушки, и оставим только подходящие под это условие. Данная часть работает за $O(n \cdot m)$. Теперь, идя слева направо, будем хранить кормушки, покрывающие текущую позицию.

Заметим, что текущего цыпленка выгодно кормить из той кормушки, которая заканчивается раньше остальных. Допустим обратное, обозначим описанную раннее кормушку за x, а ту, из которой мы покормим цыпленка за y. По предположению, цыпленка покормили из y и не кормили хотя бы одним зерном из x. Пусть зерном из y покормили текущего цыпленка a, а зерном из x покормили цыпленка b, который правее a(либо никого не кормили, тогда b это фиктивный цыпленок). Заметим, что можно вместо этого покормить a зерном из x, а b зерном из y, так как x заканчивается раньше y. Поэтому существует оптимальный ответ, в котором цыпленка a кормят сначала зернами из кормушки x.

Из этого следует, что выгодно кормить цыпленка из кормушки, которая кончается раньше всех до тех пор, пока цыпленок не съест свой максимум, либо пока не кончится кормушка. Во втором случае нужно выбрать следующую кормушку и так далее. Таким образом, необходимо поддерживать множество кормушек, уметь добавлять в него кормушку, удалять из нее кормушку, а также брать кормушку, которая заканчивается раньше всех, и изменять/удалять ее. Данные операции поддерживает структура данных set. Данная часть работает за $O((n+m) \cdot \log(m))$ на один проход.