Разбор задачи «Сбор сторонников»

Заметим, что если один человек приходит каждые a дней, а другой каждые b дней, то вместе они ходят каждые НОК(a, b) дней, где НОК — наименьшее общее кратное. Обобщая это на случай нескольких человек, получаем, что все вместе n людей ходят каждые НОК$(d_1, d_2, \ldots, d_{n-1}, d_n)$. Значит ответ — это просто прибавить к этому числу s, и вывести остаток по модулю 7 получившегося числа.

Разбор задачи «Ньют в пещере»

Для решения задачи переберем ширину w чемодана и найдем по фиксированной ширине максимальную длину.

Будем делать это с помощью структуры данных очередь с максимумом, поддерживающей максимум среди всех элементов, находящихся в данный момент в ней. Заведем две таких очереди: up и $down$, при этом в up в любой момент времени будут хранится $w + 1$ (почему не w будет объяснено позже) последовательных a_i, в $down$ — b_i. Эти очереди будут отвечать за отрезок ширины пещеры, покрытый нашим чемоданом.

Пусть сейчас чемодан занимает столбцы с i-го по $i + w$, при сдвиге чемодана вправо, следует перестать учитывать столбец i и начать учитывать $i + w + 1$-й. Это легко поддерживать в очереди. Кроме того, в любой позиции, чемодан не может быть длиннее, чем $n - max(up) - max(down)$, где $max(up), max(down)$ - максимумы в очередях. Иначе чемодан будет пересекать стену пещеры в каком-то из столбцов.

Тогда max - максимальная сумма $max(up) + max(down)$ на всех отрезках длины $w + 1$, и максимальной площадью чемодана для фиксированного w будет $n - max$.

В очереди следует каждый раз хранить сведения о $w + 1$ подряд идущих столбцах, так как при сдвиге чемодана вправо и вниз или вверх, мы **сначала** двигаем вниз или вверх и важно, чтобы после такого сдвига чемодан не выходил за пределы стен пещеры. То есть важно, чтобы чемодан не пересекал пещеру как на отрезке $[i, ..., i + w - 1]$, так и на отрезке $[i + 1, ..., i + w]$.

Разбор задачи «Добрых снов»

Посчитаем количество нюхлеров каждого типа в массив cnt. Тогда для типа i ответом будет строчка подряд идущих точек-нюхлеров длиной $cnt[i]$. Расположим строчки так, чтобы если одна заканчивается в координате j по Ox, то следующая начинается в координате $j+1$, при этом каждая из них лежит по Oy в координате i, соответствующей номеру типа.

Разбор задачи «Нюхли в министерстве»

Заметим, что ни один нюхль не пойдет по лестнице вниз: действительно, в этом случае нюхль заметит, что его этаж пропустили, и всё забудет.

Второе наблюдение. Не может быть такого, что нюхль вышел из лифта, а лифт поехал дальше вверх. В таком случае было нюхлю было бы выгоднее проехать на лифте еще хотя бы один этаж.

Пусть $maxf$ — максимальный этаж, на который могут поднять лифт нюхли. Тогда все, кому нужно попасть на этаж не выше $maxf$, попадут на него с помощью лифта, а все остальные выйдут на этаже $maxf$ и дальше пойдут пешком. Таким образом ответ — $\sum_{i=1}^{n} \max(0, f_i - maxf)$.

Как вычислить $maxf$? Можно использовать один из следующих подходов:

- Будем ехать по этажам вверх и поддерживать текущее множество нюхлей в лифте. Пусть мы находимся на этаже F. Все нюхли, которым нужно на этаж F, выходят из лифта. Мы сможем поехать на следующий этаж только если после этого в лифте есть нюхль высотой хотя бы $h + F + 1$, иначе $maxf = F$. Если лифт остался пустой, то каждый нюхль смог доехать до своего этажа.

  Этажей слишком много, чтобы проходить по всем этажам по очереди, такое решение не уложится в отведенное программе время. Поэтому будет посещать только «интересные» этажи — это этажи, на которые нужно попасть нюхлям, максимальные этажи, на которые нюхли могут нас довести, а также первый этаж.

- На какой этаж нас сможет доставить нюхль i? Во-первых, на первом этаже мы точно сможем оказаться. Во-вторых, он не поедет выше этажа f_i. В-третьих, он не сможет нажать на кнопки выше этажа $g_i - H + 1$. Таким образом, нюхль i сможет доехать не выше этажа $maxf_i = \max(1, \min(f_i, g_1 - H + 1))$. А значит $maxf = \max\limits_{i=1\ldots n} maxf_i = \max\limits_{i=1\ldots n} \max(1, \min(f_i, g_1 - H + 1))$.

Разбор задачи «Фоторобот Грин-де-Вальда»

В задаче требовалось реализовать то, что сказано в условии. Проверку того, что пиксель (i, j) можно закрасить, удовлетворив условию $c_{i,j}$, можно было двумя способами:

- разобрать все возможные случаи руками;
- заметить, что $c_{i,j}$ — битовая маска возможных цветов, где единицы стоят только в битах, соответствующих разрешенным цветам. Тогда пусть $m_{i,j} = 1$, если $p_{i,j} = R$, $m_{i,j} = 2$, если $p_{i,j} = G$ и $m_{i,j} = 4$, если $p_{i,j} = B$. Тогда достаточно проверить, что $m_{i,j} \& c_{i,j} = m_{i,j}$, где $\&$ означает битовое «и».

Разбор задачи «Магический замок»

Построим двойственный граф, где вершины — треугольники, на которые разбит данный n-угольник, а ребро между двумя вершинами проведено в случае, если у соответствующих треугольников есть общее ребро. Несложно заметить, что получившийся граф является деревом.

Также несложно заметить, что удаление одной магической связи соответствует удалению одного ребра в дереве, а также удаляет из исходной триангуляции два треугольника. Поэтому новая формулировка задачи звучит так: удалить из построенного дерева минимальное количество ребер так, чтобы у каждой вершины было удалено хотя бы одно ребро. Это стандартная задача, которая решается динамическим программированием на дереве: $dp[v][flag]$ — минимальное количество ребер, которое надо удалить в поддереве вершины v, чтобы у каждой вершины было удалено хотя бы одно ребро, где $flag$ соответствует одному из двух значений — было ли уже удалено у вершины v ребро или нет. Пересчет этой динамики довольно простой и остается в качестве упражнения.

Разбор задачи «Каждой твари — по паре»

Заметим, что нас не интересуют конкретные положения тварей на плоскости, а лишь знак их координаты. Пусть есть X_+ тварей-мальчиков с положительной x-координатой и X_- тварей-мальчиков с отрицательной x-координатой. Аналогично, пусть есть Y_+ тварей-девочек с положительной y-координатой и Y_- тварей-девочек с отрицательной y-координатой. Посчитаем эти значения в начале решения.

Переберем, сколько будет пар, в которых обе твари будут иметь положительную координату. Пусть это значение будет равняться P_{++}. Зная это значение, мы можем определить количество пар, в которых мальчик будет иметь положительную координату, а девочка — отрицательную. Несложно понять, что это количество $P_{+-} = X_+ - P_{++}$ (так как $P_{++} + P_{+-} = X_+$). Аналогично можно определить количество пар, в которых мальчик будет иметь отрицательную координату, а девочка — положительную $P_{-+} = Y_+ - P_{++}$ и количество пар, в которых обе твари будут иметь отрицательную координату $P_{--} = X_- - P_{-+}$.

Зная эти значения, мы можем вычислить количество разбиений на пары при таких значениях. Легко понять, что достаточно для каждой твари определить знак координаты твари, с которой она будет в паре. Таким образом, нам нужно из X_+ тварей с положительной x-координатой выбрать какие-то P_{++}, в паре с которыми будет тварь с положительной y-координатой. Чтобы сделать это есть $C_{X_+}^{P_{++}}$ вариантов, где C_n^k — количество сочетаний из n по k. Напомним, что число сочетаний может быть вычислено по формуле $C_n^k = \frac{n!}{k!(n-k)!}$. Аналогично, нам нужно из X_- тварей с отрицательной x-координатой нужно выбрать какие-то P_{-+}, в паре с которыми будет тварь с положительной y-координатой. Для этого есть $C_{X_-}^{P_{-+}}$ вариантов. Аналогично, есть $C_{Y_+}^{P_{++}}$ и $C_{Y_-}^{P_{+-}}$ вариантов для

тварей-девочек. Таким образом, чтобы получить ответ, переберем значение P_{++}, и если все значения P_{+-}, P_{-+}, P_{--} получились неотрицательными, прибавим к ответу значение $C_{X_+}^{P_{++}} \cdot C_{X_-}^{P_{-+}} \cdot C_{Y_+}^{P_{++}} \cdot C_{Y_-}^{P_{+-}}$.

Чтобы быстро вычислять значения C_n^k, посчитаем в начале программы значения $n!$ и обратные к ним значения по нужному модулю, это позволит вычислять C_n^k за время $\mathcal{O}(1)$. Для того, чтобы искать обратное по модулю число, можно использовать алгоритм быстрого возведения в степень, работающий за время $\mathcal{O}(\log M)$, где M — модуль, по которому производятся вычисления. Таким образом, асимптотика времени работы программы составит $\mathcal{O}(n \log M)$.

Разбор задачи «Нюхли»

Заметим, что описанная в условии конструкция представляет из себя граф, а именно — дерево. А две искомые вершины это его листы.

С помощью поиска в глубину, найдем для каждой вершины наименее глубокий лист среди ее потомков. Для это корнем дерева нужно взять не лист (число соседей больше одного).

Рассмотрим пути между листами. Утверждение: такой путь единственен (дерево) и сначала идет вверх до какого-то предка (наименьшего общего предка двух листов), а потом идет вниз, до второго листа.

Таким образом можно найти для каждой вершины два наименее удаленных друг от друга листа, путь между которыми «перегибается» в ней. Для этого посмотрим уже посчитанные наименее глубокие листы по каждому ребенку и если детей хотя бы два, то ответ это сумма расстояний до найденных листов.

Ответом же на задачу будет минимум по всем вершинам из найденных наименьших расстояний. Итоговая сложность $\mathrm{O}(n)$.

Замечание: возможна сложность $\mathrm{O}(n \log n)$, если для упрощения поиска двух минимумов из детей использовать быструю сортировку.