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 «Волшебные замки»

Так как $n \cdot m \leqslant 160$, $min(n, m) \leqslant 12$. Задача решается методом динамического программирования по изломанному профилю. Нас интересуют профиль, сообщающий, где находятся концы простых путей, которые были постренены. Заметим, что все такие концы лежат на изломанном профиле, и их концы образуют правильную скобочную последовательность. При $n = 12$, количество различных состояний получается равным $15\,511$. При добавлении очередной клетки, нужно попробовать 4 варианта добавления новых ребер, соединяющих эту клетку с предыдущими, и сделать переходы по корректным.

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 Ох, то следующая начинается в координате $j+1$, при этом каждая из них лежит по Оу 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 — $\sum\limits_{i=1}^{n} \max(0, f_i - maxf)$.

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 выже 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 $maxf = \max\limits_{i=1...n} maxf_i = \max\limits_{i=1...n} \max(1, \min(f_i, g_1 - H + 1))$.

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 «Волшебная шахта»

Переберем столб, в котором в итоге будет минимальная глубина. Сделаем двоичный поиск по глубине, которая будет в итоге в этом столбе. Пусть мы выбрали столб x и глубину h. Нужно посчитать, какое минимальное количество минут потребуется, чтобы у столба x стала высота h. Чтобы x-ый столбец мог иметь высоту h, его соседи должны иметь высоту не более $h + 1$, их соседи должны иметь высоту не более $h + 2$ и так далее. Найдем ближайший слева столбец i, что $h_i \leqslant h + (x - i)$. Заметим, что для всех $j \leqslant i$ верно, что $h_j \leqslant h + (x - j)$, а для всех $i < j \leqslant x$ верно $h_j > h + (x - j)$. Значит, столбы левее j-го уменьшать не нужно, а правее — нужно. Аналогично справа от x. Значит, чтобы найти количество минут, которое потребуется, чтобы выкопать столб x до высоты h, можно сложить высоты на отрезке, и вычесть сумму двух арифметических прогрессий. Границы отрезка также можно найти двоичным поиском.

Разбор задачи «Упражнения в умножении»

Для того, чтобы избежать работы с длинной арифметикой целых чисел, которая не укладывается в ограничение по времени, применим логарифмирования. Посчитаем вместо искомой величины значение $\log \frac{a_1 a_2 \ldots a_n}{b_1 b_2 \ldots b_m}$, где логарифм можно брать по любому удобному основанию. После этого, чтобы получить ответ, нужно лишь возвести основание в степень получившегося логарифма.

Исходя из правил логарифмирования, можно получить, что $\log \frac{a_1 a_2 \ldots a_n}{b_1 b_2 \ldots b_m} = \log a_1 + \log a_2 + \ldots + \log a_n - \log b_1 - \log b_2 - \ldots - \log b_m$. Таким образом, необходимо было лишь прологарифмировать все числа a_i и b_j и посчитать их сумму с нужными знаками. Для того, чтобы избежать проблем с точностью вещественных чисел, можно было отсортировать их по возрастанию, и после этого прибавлять к ответу либо очередное число вида $\log a_i$, либо очередное число вида $\log b_j$, причем делать это в таком порядке, чтобы значение ответа всегда было не очень большим по модулю. Например, можно действовать так: если текущее значение ответа отрицательно, прибавим к нему очередное a_i, а если положительное, то вычтем из него очередное b_j. Так мы сможем получить искомое значение с достаточной точностью.

Асимптотика времени работы решения составляет $\mathcal{O}(n + m)$.

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 ктором производятся вычисления. Таким образом, асимптотика времени работы программы составит $\mathcal{O}(n \log M)$.

Разбор задачи «Сила волшебных заклинаний»

Рассмотрим мага на позиции i. Чтобы посчитать сколько единиц урона нанесет i-й маг, нужно знать какими отрезками заклинаний покрывается этот маг. Пусть номер минимального отрезка, который содержит в себе i равен x, тогда i-й маг нанесет $(x-1) \cdot s_i$ единиц урона.

Воспользуемся техникой динамического программирования. Пусть $dp[i, msk] = minDamage$, i — позиция мага, которого сейчас рассматриваем, msk — число в троичной системе счисления, характеризующее отрезки.

- Если i-я цифра в msk равна нулю, то будем считать, что подотрезок i-го отрезка еще не открывался.
- Если i-я цифра в msk равна единице, то подотрезок i-го отрезка сейчас открыт.
- Если i-я цифра в msk равна двум, то подотрезок i-го отрезка уже был закрыт.

Возможные переходы:

1. Открыть отрезок j: $dp[i, msk'] = dp[i, msk], l_j \leqslant i \leqslant r_j$.
2. Закрыть отрезок j: $dp[i, msk'] = dp[i, msk], l_j + 1 \leqslant i \leqslant r_j + 1$.
3. Перейти к следующему магу: $dp[i+1, msk] = dp[i, msk] + cost, cost = s_i \cdot$ индекс минимального открытого отрезка.

Ответ находится в $dp[n, msk]$, если msk не содержит открытых отрезков.
Время работы: $\mathcal{O}(n \cdot 3^m \cdot m)$.

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 $O(n)$.

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