Задача A. Оптимизация Матрицы

Автор задачи и разработчик: Константин Бац

Формализуем условие задачи. Нам дан некоторый массив $w_1, w_2, \ldots, w_n$. Мы должны выбрать один раз некоторую позицию j, и после этого значение w_i для всех i между $\max(j-k, 1)$ и $\min(j+k, n)$ заменятся на значение w_j. Нам требуется найти такое j, после которого сумма w_i по всему массиву будет максимальной, и вывести эту сумму.

Для решения первой и, возможно, второй подзадачи можно было разобрать случаи для всех возможных n и k, прописав каждый случай в коде явно. Ниже приведен код для $n \leqslant 3$. В случае $k = 0$ замены вообще не производятся, а при $k \geqslant 2$ выбранное значение покрывает весь массив, так что достаточно выбрать максимальное.

```
if k == 0:
    print(sum(w))
elif k == 1:
    if n <= 2:
        print(max(w) * n)
    elif n == 3:
        print(max(w[0] * 2 + w[2], w[1] * 3, w[0] + w[2] * 2))
else:
    print(max(w) * n)
```

В третьей подзадаче $k = 0$, а это значит, что выбор j не меняет значение никаких w_i. Следовательно, вне зависимости от j ответ на задачу будет равен сумме w_i. На самом деле, как нетрудно убедиться, приведенный выше код также решает и эту подзадачу.

В четвертой подзадаче не было дополнительных ограничений на k, однако $n \leqslant 1000$ позволяло просто перебрать все j, для каждого посчитать сумму w_i в k-окрестности вокруг него и выбрать максимальный из них. Ответ на задачу можно записать в виде формулы

$$\max_{j=1}^{n} \left\{ \sum_{i=1}^{n} \begin{cases} w_j, & \text{если } j-k \leqslant i \leqslant j+k \\ w_i, & \text{иначе} \end{cases} \right\}.$$

В коде это представляется в виде двух вложенных циклов: первый цикл по j, второй по i. Время работы такого решения равно $\mathcal{O}(n^2)$.

Для прохождения последней подзадачи давайте немного оптимизируем нашу формулу. Записанную выше сумму можно разбить на случаи, когда $i < j-k$, $i > j+k$ и i находится на расстоянии не больше k от j. Запишем эту сумму следующим образом:

$$\left(\sum_{i<j-k} w_i\right) + \left(\sum_{j-k\leqslant i\leqslant j+k} w_j\right) + \left(\sum_{i>j+k} w_i\right) = \left(\sum_{i<j-k} w_i\right) + \left(\sum_{i>j+k} w_i\right) + (d \cdot w_j),$$

где за d обозначено количество i между $j-k$ и $j+k$, то есть $\max(j-k, 1) - \min(j+k, n) + 1$.

Для того, чтобы для выбранного j находить такую сумму за $\mathcal{O}(1)$, достаточно посчитать массив префиксных сумм $\texttt{pref}[t] = \sum_{i=1}^{t} w_i$. Это можно сделать за линейное время, использовав тот факт, что $\texttt{pref}[0] = 0$, а для $t > 0$ верно $\texttt{pref}[t] = \texttt{pref}[t-1] + w_t$. Тогда для подсчета ответа для фиксированного j можно пользоваться следующими равенствами:

$$\sum_{i<j-k} w_i = \texttt{pref}[\max(1, j-k) - 1]$$

$$\sum_{i>j+k} w_i = \texttt{pref}[n] - \texttt{pref}[\min(j+k, n)]$$

Учитывая, что массив pref предподсчитывается один раз за линейное время, осталось перебрать все возможные j, для каждого найти результат за $\mathcal{O}(1)$ описанным выше способом, и выбрать максимальный. Таким образом, общее время работы программы — $\mathcal{O}(n)$.

Задача B. Нужно больше энергии

Авторы задачи и разработчики: Дмитрий Грунтов и Александр Яцук

Базовая идея решения — использовать динамическое программирование. Первая подзадача при этом решается аккуратным перебором (например, рекурсивным перебором) всех вариантов расположения криокамер. Тогда как во второй подзадаче можно использовать динамику наподобие $\texttt{dp}[i][j][lst][up]$, где i — количество уже расставленных криокамер, j — количество энергетически выгодных криокамер среди первых i, lst — расстояние последней криокамеры от стены, и up — флаг (либо 0, либо 1), означающий, имело ли место возрастание расстояний от стены между последними двумя камерами (то есть, может ли последняя криокамера стать энергетически выгодно расположенной, если следующую расположить ближе).

Пересчитывать такую динамику можно по следующим формулам:

$$\texttt{dp}[i][j][lst][0] = \texttt{dp}[i-1][j][lst][*] + \sum_{prv=lst+1}^{x} (\texttt{dp}[i-1][j][prv][0] + \texttt{dp}[i-1][j-1][prv][1])$$

$$\texttt{dp}[i][j][lst][1] = \sum_{prv=1}^{lst-1} \texttt{dp}[i-1][j][prv][*]$$

Здесь под $[up = *]$ имеется в виду сумма значений динамики в состояниях с $[up = 0]$ и $[up = 1]$. Идея следующая — если последняя криокамера не дальше предыдущей, тогда предыдущая либо на том же расстоянии и не расположена энергетически выгодно, либо на большем расстоянии и расположена энергетически выгодно (j без последней меньше на 1) тогда и только тогда, когда камера перед ней была ближе к стене ($[up = 1]$). Во втором же случае достаточно просто просуммировать состояния, в которых предыдущая камера расположена строго ближе, и тогда она в любом случае не расположена энергетически выгодно, поэтому j не уменьшается. Такое решение за $\mathcal{O}(nx^2k)$ решало вторую подзадачу, только если не пытаться завести статический массив размера $500 \times 500 \times 500$.

Для решения третьей подзадачи можно заметить, что используемые в формулах пересчета динамики суммы можно поддерживать и считать за $\mathcal{O}(1)$, используя префиксные суммы $\texttt{ps}[i][j][lst][up] = \sum_{t=1}^{lst} \texttt{dp}[i][j][t][up]$. При пересчете динамики внутри одного слоя (i, j) по возрастанию lst можно было считать такие префиксные суммы параллельно с основной динамикой, что позволяет получить решение за $\mathcal{O}(nxk)$.

Чтобы не получить вердикт `ML` в четвертой подзадаче, достаточно избавиться от первого измерения массива `dp`. Действительно, формула пересчета для $\texttt{dp}[i]$ всегда обращается к $\texttt{dp}[i-1]$, поэтому достаточно хранить только два последних слоя, тем самым сокращая используемую память с $\mathcal{O}(nxk)$ до $\texttt{O}(xk)$. Время решения при этом не меняется.

Достаточно аккуратное решение четвертой подзадачи проходило и пятую, но если нет, то можно было добавить различные оптимизации, например, считать реже выполнять операцию взятия по модулю $10^9 + 7$, либо хранить только один слой, и пересчитывать `dp` только через `ps`, таким образом следуя логике «(посчитать `dp` слоя i) $\to$ (посчитать `ps` слоя i) $\to$ (посчитать `dp` слоя $i + 1$) $\to \ldots$». Асимптотика времени решения все та же, $\mathcal{O}(nxk)$.

Задача C. Финальное противостояние

Авторы задачи: Владимир Рябчун и Даниил Орешников, разработчик: Даниил Орешников

Для решения первой подзадачи достаточно было перебрать группы подряд идущих волн в любом порядке, для каждой группы найти пересечение всех отрезков и проверить, что длина этого пересечения находится в интервале $[m_1, m_2]$. Для пересечения нескольких отрезков достаточно рассмотреть максимальную из их левых границ $l_{\max}$ и минимальную из их правых границ $r_{\min}$, тогда длина их пересечения будет вычисляться как $\max(0, r_{\min} - l_{\max})$. Такое решение работает за $\mathcal{O}(n^3)$.

В третьей подзадаче работало то же самое решение, только перебор групп следовало осуществлять по возрастанию левой границы, а при фиксированной левой границе — по возрастанию правой. В таком решении пересечение всех отрезков волн можно было пересчитывать за $\mathcal{O}(1)$, что дает решение за $\mathcal{O}(n^2)$.

Решения остальных подзадач основаны на методе двух указателей. Для каждой левой границы группы найдем минимальную и максимальную подходящие правые границы, после чего добавим их разность в ответ. Оба указателя двигаются слева направо, и остается только пересчитывать за небольшое время пересечение всех отрезков в группе при движении указателей.

При движении правого указателя добавляется новый отрезок — достаточно просто пересечь текущее посчитанное пересечение с новым отрезком за $\mathcal{O}(1)$. При движении левого указателя надо удалить одну волну из группы — в таком случае пересчет пересечения не так тривиален, однако в четвертой подзадаче можно было просто поддерживать целиком все покрытие отрезками прямой и пересчитывать ответ за $\mathcal{O}(r_i - l_i) = \mathcal{O}(1)$. А в последней подзадаче можно было, например, хранить `multiset` (упорядоченное мультимножество) для всех задействованных левых границ и еще одно для всех правых границ, в таком случае удаление отрезка из рассмотрения занимает $\mathcal{O}(\log n)$ времени, а поиск минимума и максимума — $\mathcal{O}(1)$.

Альтернативно для поиска минимальной и максимальной из границ отрезков можно было построить разреженные таблицы. Авторское же решение основано на методе «разделяй и властвуй»: можно разбить все волны на две равные части, рекурсивно посчитать ответ в левой и правой частях, после чего найти количество опасных групп, содержащих две центральные волны, методом двух указателей, предподсчитав пересечения отрезков на суффиксах правой части и префиксах левой. Полное решение, таким образом, работает за $\mathcal{O}(n \log n)$.

Задача D. Распределенная Матрица

Автор задачи и разработчик: Владимир Рябчун

Для решения первой подзадачи достаточно аккуратно реализовать то, что написано в условии. Для этого достаточно хранить граф, для каждого узла хранить время его присоединения, и при каждом запросе заново делать `dfs`. Асимптотика решения будет $\mathcal{O}(nm)$.

Решение второй подзадачи отличается лишь тем, что для каждой вершины следует хранить флаг, находится ли она в рабочем состоянии, и время последнего восстановления. При обходе `dfs` достаточно считать сумму времен последнего восстановления и длину пути, чтобы после посчитать суммарную ненадежность узлов. Чуть более эффективное решение заключается в том, чтобы сразу пройти по всем запросам, целиком построить дерево, и хранить для каждого узла предка и его глубину в дереве. Тогда ответ на запрос может заключаться в постепенном подъеме из двух вершин до корня. Время работы решения все еще $\mathcal{O}(nm)$.

Для решения третьей подзадачи можно выписать в каждой вершине момент времени, когда она была добавлена. Обозначим за $\mathrm{lca}(x, y)$ узел, являющийся наименьшим общим предком узлов x и y, а за $\mathtt{s}[v]$ — сумму чисел в узлах на пути от v до корня. Будем считать, что в данный момент запрос номер t, а количество различных узлов, от которых зависят узлы x и y, равно p. Тогда ответ в случае, если обе вершины добавлены, равен $t \cdot p - (\mathtt{s}[x] + \mathtt{s}[y] - \mathtt{s}[\mathrm{lca}(x, y)])$. Поддерживать и считать суммы чисел на путях, наименьшего общего предка и количество узлов на пути можно при помощи любой удобной техники, которая отвечает за запрос за время порядка $\mathcal{O}(\log n)$, например с помощью двоичных подъемов. Итоговая асимптотика будет $\mathcal{O}(m \log n)$.

В четвертой подзадаче необходимо делать ровно то же самое, что и в третьей, но надо быстро понимать, получает ли узел энергию. Если узел v отказал, то он и все его потомки никогда больше не будут получать энергию. Можно проставить отметки об отсутствии энергии обычным обходом в глубину, который не будет заходить в уже помеченные вершины. Пометки с вершин не снимаются, и каждую мы пометим не более одного раза, поэтому суммарное время всех обходов в глубину будет $\mathcal{O}(n)$.

Для полного решения необходимо заметить прекрасное свойство обхода дерева. Будем обходить дерево в глубину и записывать вершину в список, когда заходим в нее. Тогда все вершины поддерева v образуют в этом массиве непрерывный отрезок (от вхождения v до вершины, которая была в его конце, когда `dfs` покидал v). Будем в каждом элементе массива обхода хранить, сколько узлов-предков v отказали. Отказ вершины v приводит к увеличению всех значений на отрезке ее поддерева на 1. И, наоборот, восстановление приводит к уменьшению значений на отрезке на 1. Для проверки того, получает ли вершина энергию, достаточно посмотреть на значение в ее ячейке массива обхода. Если там записано 0, то нет ни одной вершины выше нее, которая отказала.

Авторское решение для реализации этих операций использует дерево отрезков. Величину $\mathtt{s}[v]$ можно поддерживать так же: при восстановлении узла значения $\mathtt{s}[u]$ поменяются только у потомков v на одинаковую величину. В итоге решение будет отвечать на каждый запрос за $\mathcal{O}(\log n)$ времени, соответственно итоговая сложность будет равна $\mathcal{O}(m \log n)$.