Разбор задачи «Сложная задача»

Решение первой и второй подзадачи

Эти подзадачи являются упрощением задачи о наибольшей общей убывающей подпоследовательности и решаются стандартными алгоритмами за $O(n^2m)$ и за $O(nm)$ соответственно.

Решение третьей позадачи

Заметим, что любая неубывающая последовательность из нулей и единиц имеет следующий вид: сначала сколько-то нулей (возможно, ни одного), а потом сколько-то единиц (возможно, ни одной). Пусть a состоит из a_0 нулей и a_1 единиц, а b — из b_0 нулей и b_1 единиц. Тогда очевидно, что ответ равен $min(a_0, b_0)+min(a_1, b_1)$. Решение работает за один линейный проход по последовательностям, асимптотика времени работы составляет $O(n + m)$

Полное решение

Воспользуемся наблюдением из прошлой подзадачи. Переберем количество нулей в ответе, обозначим его за k и найдем максимальное число единиц, которое может следовать за таким количеством нулей в какой-то общей подпоследовательности a и b. Заметим, что максимальное число единиц, которое может следовать за k нулями в какой-то подпоследовательности, для каждой из последовательностей равняется числу единиц после позиции, на которой в ней встречается k-й ноль. Для того, чтобы быстро определять, сколько единиц идет после k-го нуля, запомним для каждой из последовательностей позиции всех нулей. Также, для всех суффиксов каждой из последовательностей посчитаем, сколько единиц содержит этот суффикс. Весь предподсчет может быть выполнен за линейное время. Теперь, когда мы умеем быстро определять для каждого k максимальное количество единиц, которое может следовать за k нулями в общей подпоследовательности, переберем количество нулей, и для каждого из них посчитаем максимальный ответ. Из всех получившихся ответов возьмем максимальный, он и будет являться ответом на задачу. Асимптотика времени работы этого решения составляет $O(n + m)$.

Разбор задачи «IPvX»

Для удобства каждый адрес будем кодировать числом от 0 до $2^{64} - 1$ (то есть беззнаковым 64-битным числом) следующим образом: $encode(a_1.a_2.a_3.\ldots.a_x) = a_1 \cdot 256^{x-1} + a_2 \cdot 256^{x-2} + \ldots + a_{x-1} \cdot 256 + a_x$.

Теперь посмотрим на отрезок $encode(a)..encode(b)$. Ответ существует, если на этом отрезке есть два подряд свободных адреса. Чтобы понять это, найдем все адреса на этом отрезке, отсортируем их и проверим за $O(n)$ этот факт. И если ответ существует, то он равен $encode(b)-encode(a)-between-1$, так как все остальные адреса на отрезке надо заполнить. Здесь за $between$ обозначено количество занятых адресов, лежащих между a и b.

Разбор задачи «Наскальная живопись»

Удвоим строку, тогда палиндром, являющийся ответом, будет подстрокой удвоенной строки. Найдем для каждой позиции длину максимального палиндрома с центром в этой позиции. Уменьшим длину палиндрома на минимальное четное число так, чтобы она стала не больше n, так как ответ не может превышать n. Заметим, что можно выбрать такой циклический сдвиг, чтобы палиндром такой длины с центром в этой позиции являлся подстрокой этого циклического сдвига. Обновим этим значением ответ.

Если для каждого центра увеличивать длину палиндрома на 1, пока он остается палиндромом, получится решение за $O(n^2)$, которое проходит первые три группы тестов.

Если посчитать полиномиальные хеши для прямой и развернутой строки, и для каждого центра искать длину палиндрома двоичным поиском, сверяя хеши прямого и развернутого палиндрома, получится решение за $O(n \cdot \log n)$. Из-за большого количества операций взятия по модулю это решение работает довольно медленно, поэтому оно могло не проходить последнюю группу тестов. Если использовать полиномиальный хеш по модулю 2^{64}, то решение должно получить вердикт `wrong answer` из-за тестов со строкой Туэ-Морса.

Если для нахождения максимальных длин палиндромов для всех позиций воспользоваться алгоритмом Манакера, асимптотика времени работы программы составит $O(n)$. Такое решение набирает 100 баллов.

Разбор задачи «Энергия»

В задаче требовалось найти в дереве k непересекающихся путей так, чтобы сумма весов всех вершин на путях была максимальна.

Решение для маленьких k

В первых четырех подгруппах, при небольших значениях k, достаточно было перебрать путь в дереве, либо же два пути, и выбрать наилучший ответ. В подзадаче 4 требовалось соптимизировать этот перебор, запомнив для каждого поддерева максимальный вес пути в нем.

Решение для маленьких n

В подзадаче 5 достаточно было написать перебор.

Можно перебрать множество вершин, которые войдут в ответ, и проверить с помощью динамического программирования, что это множество вершин можно разбить на k путей.

Можно реализовать рекурсивный перебор, перебрав все возможные множества из k непересекающихся путей. Этот перебор необходимо реализовать аккуратно, чтобы не рассматривать по несколько раз одни и те же множества путей.

Полное решение

Для полного решения этой задачи воспользуемся динамическим программированием по поддеревьям.

Подвесим дерево за произвольную вершину, и посчитаем динамику. Состоянием динамики будет пара чисел (u, s) — вершина, поддерево которой мы рассматриваем, и количество путей, которые мы уже выбрали в этом поддереве. Для пересчета этой динамики в состояние также необходимо добавить булевый параметр $flag$ — верно ли, что вершина u является концом какого-то из взятых путей. Значением динамики будет искомый ответ. Таким образом, $dp[u][s][flag]$ — это максимальная сумма, которую мы можем получить, взяв из поддерева вершины u ровно s путей, при этом, если $flag = 1$, вершина u должна являться концом одного из путей.

Для пересчета этой динамики в каждой вершине будем считать вспомогательную динамику на префиксах ее детей. Пусть $dp'[u][p][i][flag]$ — это максимальная сумма, которую можно набрать, если рассматривать только поддеревья первых p сыновей вершины u. При этом параметр $flag$ может принимать одно из трех значений: $flag = 0$ означает, что вершина u не была задействована ни в одном из путей, $flag = 1$ означает, что вершина u задействована в одном из путей и является его концом, а $flag = 2$ означает, что вершина u задействована в одном из путей, но не является его концом.

Пересчитывать динамику dp' можно через значения dp для очередного сына каждой вершины. Достаточно лишь перебрать, сколько путей мы возьмем из поддерева сына, а также, что случится с путем, содержащим этого сына: он может либо продлиться наверх на один, либо склеиться с каким-то другим путем. Каждый из этих случаев дает нам переход в свое состояние динамики dp'.

Ответ на задачу будет равен $min(dp[root][k][0], dp[root][k][1])$, где $root$ — корень дерева.

Оценим время работы данного решения. Всего есть $O(nk)$ состояний динамики, всего переходов от сына к отцу будет сделано $O(n)$, так в дереве $n - 1$ ребро. При каждом из переходов нам нужно перебрать, сколько путей мы берем из поддерева сына, а также сколько путей уже было взято. Итоговая оценка времени работы составляет $O(nk^2)$, чего достаточно для прохождения шестой подгруппы.

Для прохождения седьмой подгруппы применим несложную оптимизацию: не будем переходить из тех состояний динамики, которые невозможны. Пусть мы считаем значение динамики для вершины u и сейчас обновляем их для ее сына v. Тогда, перебирая, сколько путей мы возьмем из сына, будем перебирать это значение до величины $min(k, size_v)$, где $size_v$ — размер поддерева вершины v. Также, когда мы перебираем, сколько путей мы берем из нашей текущей динамики dp', будем перебирать это значение до величины $min(k, size'_u)$, где $size'_u$ — суммарный размер всех обработанных детей u. Заметим, что ограничив перебор количества путей этими значениями, мы все еще учитываем все возможные переходы: действительно, нас не интересуют значения, большие k, так как всего мы хотим взять k путей, а также, нас не интересуют значения, большие размера соответствующих поддеревьев, так как из поддерева невозможно выбрать больше путей, чем в нем есть вершин.

Можно доказать, что благодаря этим оптимизациям асимптотика времени работы программы уменьшится до $O(nk)$, этого достаточно для получения 100 баллов.