Разбор задачи «Проверка»

Для решения сделаем предобработку последовательности: найдем для позиции i значение $next_i$ — следующий символ в a_n после позиции i, который равен a_i. Такую предобработку можно сделать за $O(n)$:

1. Будем идти по последовательности справа налево и поддерживать массив $last_c$ — последний встреченный символ c. После индекса i потребуется обновление $last_{a_i} = i$.

2. Тогда $next_i = last_{a_i}$.

После этого сделаем в нашей последовательности m указателей p_i, i-й из которых будет указывать на первое вхождение буквы b_i. Для инициализации достаточно воспользоваться $last_{b_i}$. Но такие p_i не будут задавать искомую подпоследовательность, пока не выполнено $p_i < p_{i+1}$.

Чтобы p_i монотонно возрастали, будем пользоваться следующим жадным алгоритмом *починки*:

1. p_1 оставим неподвижным.

2. Пока $p_2 \leqslant p_1$, двигаем p_2 вправо до следующего подходящего символа: $p_2 = next_{p_2}$.

3. Переходим к p_3 и так далее.

Тогда p_m является искомой правой границей для левой границы 1. А что происходит при сдвиге границы? На самом деле, достаточно сделать $p_1 = next_{p_1}$ и заново запустить алгоритм починки, это будет гарантировать корректность следующей границы. В самом деле, новая левая граница будет больше предыдущей (и потому подойдет нам), а алгоритм починки поправит сколько-то первых значений p_i, чтобы условие упорядоченности сохранилось.

Осталось понять, почему этот алгоритм работает быстро. Для этого нужно заметить, что все символы b_i различны, поэтому множества индексов p_i на протяжении всего алгоритма не пересекаются. А значит, сумма размеров таких множеств равна размеру объединения, который не превосходит n. С учетом того, что инициализация потребовала m указателей, итоговая асимптотика равна $O(n+m)$.

Разбор задачи «Ремонт дороги»

В рамках разбора для удобства будем считать, что дорога — бесконечный массив с такой же нумерацией, как и у участков дороги. В каждой ячейке массива записан уровень воды на соответствующем участке дороги.

Посмотрим, как изменяется этот массив. Если ливень прошёл над ячейкой с 0, то 0 заменяется на 1. Теперь разберём случай, когда ливень идёт над ячейкой под номером i с значением 1. Рассмотрим отрезок массива, на котором во всех ячейках 1, содержащий ячейку i. Из всех таких отрезков рассмотрим отрезок $[l, r]$ наибольшей длины. В частности, это означает, что в ячейках $l-1$ и $r+1$ воды нет. Пусть участок i является k-м слева на этом отрезке. С помощью рекурсивной симуляции описанных в условии задачи процессов можно видеть, как меняются уровни воды. А именно: все ячейки с $l-1$ по $r+1$ после ливня будут 1, кроме ровно одной под номером $(r-k+1)$, в ней будет 0. Таким образом, если ливень прошёлся над ячейкой с 1, то весь отрезок из 1 расширится влево и вправо на 1 ячейку, а в точке, симметричной месту ливня, будет 0. Остаётся эффективно поддерживать отрезки из единиц с возможностью их менять. В наивных реализациях можно было сохранять массив всех участков дорог с их уровнями воды (или `unordered_map`) или сохранять только номера участков дорог, где есть вода. Это позволяло набрать частичные баллы, всё ещё применяя идею о том, как меняются уровни воды после ливня. Чтобы набрать полный балл, было достаточно поддерживать `set` из пар, в котором хранятся границы отрезков. Тогда если ливень идёт над ячейкой с 0, то в сет просто добавляется отрезок с этой ячейкой. Если ливень идёт над ячейкой с 1, то отрезок пересчитывается, как описано выше. При этом нужно "склеивать"образовавшиеся соседствующие отрезки, потому что мы поддерживаем отрезки из 1 наибольшей длины. Теперь чтобы ответить на запрос значения в ячейке, мы просто проверяем, существует ли отрезок в нашем сете, содержащий позицию запроса, это можно сделать бинпоиском.

Разбор задачи «Новые кампусы!»

В первой подзадаче достаточно перебрать каждую пару вершин и посчитать за $O(n)$ расстояние между ними в каждом из деревьев. Асимптотика $O(n^3)$.

Во второй подзадаче можно зафиксировать одну из вершин, затем посчитать расстояние от нее до всех остальных в обоих деревьях, и потом перебрать вторую вершину. Асимптотика $O(n^2)$.

В третьей подзадаче достаточно рассмотреть два случая: если расстояние в первом дереве равно 1, то в пару точно входит вершина, которая соединена со всеми в первом дереве, иначе расстояние в первом дереве равно 2 — в этом случае нужно обновить ответ диаметром второго дерева.

Далее рассмотрим полное решение задачи. Для этого будем использовать идею, похожую на центроидную декомпозицию.

Будем итеративно строить центроидную декомпозицию первого дерева: на первой найдем центроид и удалим его, на второй – найдем центроиды во всех компонентах и удалим их, ...

Теперь научимся на каждой итерации обновлять ответ. Для каждой вершины v из первого дерева найдем расстояние $d_1[v]$ до центроида ее компоненты. Теперь у каждой вершины есть два параметра $comp[v]$ — в какой компоненте она находится и $d_1[v]$ — расстояние до соотвествующего центроида. Теперь нам нужно найти во втором дереве пару вершин (u, v), таких что $comp[u] = comp[v]$ и $d_1[u] + d_1[v] + d_2(u, v)$ максимально, где $d_2(u, v)$ — расстояние между u и v во втором дереве.

Эту задачу можно решить за $O(n \cdot \log(n))$ используя технику переливаний.

Опишем решение за $O(n)$. Для каждого цвета отдельно можно построить неявное (виртуальное) дерево, один раз обойдя все дерево целиком. Затем для каждого цвета можно решить задачу за линейное время относительно количества вершин этого цвета, используя простую динамику на построенном виртуальном дереве.

Итак, итоговая асимптотика составляет $O(n \cdot \log(n)^2)$ или $O(n \cdot \log(n))$. Для полного решения задачи необходимо использовать неасимптотические оптимизаций, такие как перенумерация вершины в порядке обхода Heavy-Light Decomposition (hld-reorder) и так далее.

Во многих решениях требуется отвечать на запрос LCA пары вершин для большого количества пар вершин оффлайн. На практике в этом случае быстро работает алгоритм Тарьяна.

Разбор задачи «Древо жизни»

Посчитаем вклад каждой вершины v в итоговую сумму. Для этого посчитаем количество расстановок событий, в которых существует путь до v, проходящий только по радостным событиям. Пусть всего k из n радостных событий и количество вершин на пути от корня до v равно h. Если $h > k$, то подходящих расстановок событий нет. Иначе всего $k \cdot (k-1) \cdot \ldots \cdot (k-h+1) \cdot (n-h)!$ подходящих расстановок, так как нужно выбрать, какие из событий будут стоять на пути от корня до v, а затем выбрать любое расположение остальных событий. Каждое радостное событие будет стоять в вершине v в одинаковом количестве расстановок. Поэтому, чтобы получить искомый вклад вершины v в сумму, нужно умножить найденное количество расстановок на среднее значение величин a_i по всем радостным событиям. Предподсчитаем все значения $k!$ и $\frac{1}{k!}$ для k до n, тогда для каждой вершины v можно посчитать значение формулы за $O(1)$. Итоговая асимптотика $O(n)$.

Разбор задачи «Битовый хаос»

В первой подгруппе можно было после каждого изменения пробегаться по всему отрезку и насчитывать *and*, *or* или *xor*.

Во второй подгруппе можно было сделать дерево отрезков, внутри которого поддерживать *xor* в поддереве. Ключевая идея: если у вас есть элементы, *xor* которых равен a, и вы хотите для каждого элемента сделать *xor* с числом x, то в случае четного числа элементов *xor* не изменится, иначе станет равен $a \oplus x$.

В третьей подгруппе можно для каждой операции сделать свое дерево отрезков.

В четвертой и пятой подгруппах можно было сделать дерево отрезков для каждого бита (заметим, что по каждому биту задачу можно решать независимо). Например, если мы делаем *and* с числом x, то нам нужно будет во всех битах со значением 0 присвоить 0 на заданном отрезке для этого бита.

Дальше обсудим полное решение. Сделаем дерево отрезков, внутри которого для поддерева будем хранить:

- AND — побитовое И в поддереве
- OR — побитовое ИЛИ в поддереве
- XOR — побитовый XOR в поддереве
- ALL_ONE — некоторое число, в i-м бите которого находится единица только в том случае, если во всех числах в поддереве i-й бит равен единице.
- ALL_ZERO — некоторое число, в i-м бите которого находится единица только в том случае, если во всех числах в поддереве i-й бит равен нулю.

Всё это мы умеем пересчитывать через левое и правое поддерево за $O(1)$. Нам осталось научиться справляться с операциями обновления и проталкиванием вниз. Для этого дополнительно будем поддерживать:

- psh_POS — некоторое число, в i-м бите которого находится единица только в том случае, если мы хотим изменить значение этого бита для всех элементов поддерева на заданное.
- psh_VAL — некоторое число, в i-м бите которого храним заданное значение из предыдущей переменной (и 0, если этот бит не меняем).
- psh_XOR — некоторое число, в i-м бите которого храним единицу только в том случае, если нужно будет сделать xor в этом бите для всех элементов поддерева (для удобства реализации в битах, в которых мы знаем значение, будем хранить 0, то есть сделаем так, чтобы числа psh_POS и psh_XOR не пересекались по битам).

Дальше давайте поймем, что делает каждая из операций:

- $AND\ X$ — в битах числа X равных нолю, мы хотим присвоить 0 на отрезке.
- $OR\ X$ — в битах числа X равных единице, мы хотим присвоить 1 на отрезке.
- $XOR\ X$ — просто делаем XOR на отрезке.

Заметим, что теперь мы решаем задачу не по битам отдельно, а для всех одновременно. Итоговое решение получается за $O(n \log n)$, но с большой константой из-за большого пересчета, который опущен в разборе (его можно будет посмотреть в авторских решениях). Тем не менее это решение быстрее чем решение для каждого бита отдельно и заходит на полный балл.

Разбор задачи «Финальная Битва»

Покажем, как выделить небольшое число клеток, таких что ответ лежит среди них. Предположим, что мы знаем, что в строке и столбце клетки (x, y) суммарно deg клеток, не считая клетки (x, y). Обозначим за r_x сумму в строке x, а за c_y сумму в стобце y. Тогда эта клетка прибавит к сумме в матрице $w_{xy} = r_x + c_y - (3 + deg)a_{xy}$.

Рассмотрим порядок, в котором клетки отсортированы по убыванию w_{ij}, будем жадно идти в такой порядке и брать очередную клетку в множество интересных, если в ее столбце и строке выбрано меньше k клеток и суммарно выбрано не больше $k - 1 + \lfloor \frac{(k-1)\cdot 2k}{deg+1} \rfloor$ клеток. Докажем, что можно заменить рассмотренную клетку на одну из множества интересных и не ухудшить ответ. Случай, когда клетка (x, y) входит в множество интересных очевиден.

Назовем множество клеток ответа, кроме рассмотренной, дополнением. Степенью строки (столбца) назовем количество клеток дополнения в этой строке (этом столбце). Назовем степенью клетки, не принадлежащей дополнению, сумму степени ее строки и степени ее столбца.

Рассмотрим два случая:

1. В множестве интересных клеток меньше, чем $k+\lfloor\frac{(k-1)\cdot 2k}{deg+1}\rfloor$ клетка. В частности из этого следует, в множестве интересных клеток в строке x или столбце y выбрано хотя бы k клеток (иначе мы могли бы добавить клетку (x, y) в множество интересных, а мы этого не сделали). Рассмотрим первую в порядке сортировки клетку такую, что после ее добавления в строки x или столбце y выбрано k клеток. Пусть, не умаляя общности, эта клетка находится в строке x. Рассмотрим интересную клетку из этого строки, в столбце которой нет клеток дополнения, тогда ее степень $=$ (степень строки x) $\leqslant$ (степень строки x + степень столбца y) $= deg$. Заметим, что эта клетка в порядке сортировки шла не позже клетки, до добавляения которой в строке x и столбце y было выбрано $< k$ клеток, а значит она шла раньше клетки (x, y), так как (x, y) мы в множество интересных добавить не смогли. Пусть ее кооридианты (x, t), тогда ее вес $w_{xt} >= w_{xy}$. А ее вклад в ответ $w = w_{xt} + a_{xt} \cdot$ (степень строки $x - deg$) $>= w_{xt} + 0 >= w_{xy}$. Значит можно заменить клетку (x, y) на клетку (x, t).

2. В множестве интересных клеток $k + \lfloor\frac{(k-1)\cdot 2k}{deg+1}\rfloor$ клетка. Назовем клетку из множества интересных непригодной, если она совпадает с одной из клеток дополнения или ее степень хотя бы $deg + 1$. Заметим, что непригодных клеток не больше $k - 1 + \lfloor\frac{(k-1)\cdot 2k}{deg+1}\rfloor$. Действительно, сумма степеней l непригодных клеток не из дополнения $\geqslant l \cdot (deg + 1)$, но каждая клетка дополнения прибавляет к сумме степеней $\leqslant 2k$. Рассмотрим клетку из множества интересных, не являющуюся непригодной, обозначим ее координаты за (s, t), ее степень за $d \leqslant deg$.

   Рассмотрим два случая:

   (a) В строке/столбце (x, y) в множестве интересных есть k клеток — доказательство совпадает с доказательством случая 1.

   (b) В строке/столбце (x, y) в множестве интересных меньше k клеток, заметим тогда, что вес клетки $w_{st} \geqslant w_{xy}$, вклад клетки (s, t) в ответ $w = w_{st} - a_{st} * (deg - d) \geqslant w_{st} - 0 \geqslant w_{xy}$. Тогда можно заменить клетку (x, y) на клетку (s, t).

Таким образом, можно перебрать deg от 0 до $k-1$ и выделить не более $k+\lfloor\frac{(k-1)\cdot 2k}{deg+1}\rfloor$ кандидатов для каждого deg. Для $k = 5$ это не более 116 кандидатов, аккуратный рекурсивный перебор дает решение за $\binom{116}{5}$.

Для получения полного решения переберем все возможные расстановки 6 клеток с точностью до перестановки строк и столбцов, для каждой посчитаем степени каждой из клеток ответа, а также количество кандидатов каждой степени, которого хватит, чтобы найти такую конфигурацию. Из таких оставим только невложенные наборы количеств кандидатов. Будем отдельно перебирать такой набор, а затем перебирать ответ только из клеток, которые попали в список необходимых для какой-либо степени. Оптимизацая, которая дает существенное ускорение — не перебирать клетки с $deg = 0$, а определять их оптимальную расстановку после перебора клеток с $deg > 1$ с помощью алгоритма `min-cost max-flow`. Авторское решение запускает поиск потока размера $\leqslant 6$ на графе из 61 вершины в худшем случае $\sim 10^6$ раз, что на практике работает достаточно быстро.

Разбор задачи «Гармоничные шарфы»

Поймем, когда строку можно разделить на подстроки, содержащие две различные буквы. Для этого сожмем идущие подряд блоки равных символов и будем рассматривать строку как упорядоченный набор пар (c, cnt_c) — символ и количество его вхождений. Строку можно разделить если выполняется хотя бы одно из трех условий:

- В сжатом виде строка содержит четное число блоков.
- Найдется блок на четной позиции с $cnt_c > 1$.
- Надутся три последовательных блока вида aba с центральным блоком на четной позиции.

В каждом из случаев легко предъявить подходящее разбиение, необходимость выполнения одного из условий можно показать по индукции.

Можно выполнить наивную проверку критерия для всех подстрок начинающихся в фиксированной позиции l за время $O(n)$ и получить решение за $O(n^2)$.

Для ускорения также переберем левую границу в порядке уменьшения. Отдельно посчитаем подстроки, содержащие четное число блоков. Для подсчета полезных подстрок, состоящих из нечетного числа блоков, будем хранить для каждой четности ближайший справа блок, удовлетворяющий условию 2 или 3. Тогда все правые границы, содержащие его, нам также подойдут. Количество таких правых границ можно посчитать с помощью суффиксных сумм, итоговая асимптотика $O(n)$.

Разбор задачи «Заработок без вложений»

Разложим число n в систему счисления по основанию 4. Пусть соответствующие цифры равны $d_0, d_1, \ldots, d_l$, тогда $n = \sum_{i=0}^{l} d_i \cdot 4^i$. Научимся получать все числа вида $d_i \cdot 4^i$ в качестве сумм первых $k-1$ ячеек. Тогда мы сможем накапливать необходимую сумму в последней ячейке, прибавляя в нужные моменты времени к ее значению $d_i \cdot 4^i$. Начнем с прибавления 1 четыре раза к значению в первой ячейке, так мы получим значения $1, 2, 3, 4$. Дальше будем действовать рекурсивно, пусть сумма в первых s ячейках равна 4^s, три раза применим операцию прибавления префиксной суммы к ячейке с номером $s+1$, так мы получим суммы первых $k-1$ ячейки равные $2 \cdot 4^s, 3 \cdot 4^s, 4^{s+1}$ и сделаем шаг $s \to s+1$. Несложно убедиться, что это решение требует $\lceil \log_4(n) \rceil$ ячеек памяти и $4 \cdot \lceil \log_4(n) \rceil$ операций. Возможно написать перебор с отсечениями, который делает 36 операций в худшем случае для 10 ячеек и $n \leqslant 10^6$.

Разбор задачи «Прибытие»

Сделаем бинпоиск по ответу. Нужно научиться проверять, что ответ не меньше R. Давайте мысленно покрасим все точки на расстоянии не больше R от прямоугольника и точек, то есть сдвинем внутрь границы прямоугольника и закрасим окружности радиуса R с центрами в исходных точках. В такой формулировке нам нужно найти 2 точки, не лежащие строго внутри покрашенных областей, на расстоянии ровно $2 \cdot R$. Как научиться такое проверять? Можно доказать, что есть такая пара точек существует, существует и пара точек такая, что одна из них лежит на пересечении какой то пары окружностей (радиуса R построенных на исходных точках), или на пересечении сдвинутых на R границ, или на пересечении окружности и прямой. Заметим важный факт: всего точек пересечения $O(n^2)$, и можно показать, что незакрашенных точек пересечения $O(n)$. Осталось научиться проверять, что существует непокрашенная точка на расстоянии $2 \cdot R$ от зафиксированной точки. То есть нужно проверить, что на окружности радиуса $2 \cdot R$ есть точка , не покрытая набором из окружностей радиуса R и полуплоскостей. Если такая точка существует, понятно, что она может быть точкой пересечения нашей окружности и одного из объектов в наборе, соответственно можно перебрать, с каким объектом будет пересечение и проверить, что точка не покрыта исходными окружностями. Таким образом в зависимости от реализации получается полиномиальное решение за $O(n^4 \log(C))$ или $O(n^3 \log(C))$.

Далее можно различными способами улучшать это решение.

Часть с выделением кандидатов можно ускорить, заметив, что можно рассматривать только точки пересечения окружностей построенных на точках соединённых ребром в триангуляции Делоне, таким образом кандидатов на проверку становится $O(n)$. Проверку того, что точка не покрыта окружностями можно ускорить, заметив, что можно проверять только окружность, построенную на точке в ячейке диаграммы Вороного которой лежит точке, которую нужно проверить. Это можно сделать за $O(n \log(n))$ различными способами.

Часть с проверкой кандидатов можно сделать за $O(n^2 \log(n))$ сделав сканлайн на окружности или за $O(n^2)$ сделав тот же сканлайн, но сортировать окружности друг относительно друга заранее. Так же есть и другой вариант решения за $O(n^2)$: можно выделить непокрытые отрезки рёбер диаграммы Вороного и проверить, что существует пара точек на одном или двух отрезках на расстоянии $2 \cdot R$.

Итого различными комбинациями этих решений можно добиться авторской асимптотики: $O(n^2 \cdot log(C))$.