

Минимумы на дугах

Автор задачи и разработчик: Александр Бабин

Подзадача 1.

В этой подзадаче перестановка восстанавливается следующим образом:

$$p = [1, f(1, 2), 2, f(2, 3), 3, \dots, n-1, f(n-1, n), n]$$

Подзадача 2.

В этой подзадаче можно было заранее сделать всевозможные запросы $f(i, j)$, а затем восстановить перестановку за время $O(n^2)$. Сделать это можно следующим образом:

- Пусть $g(x)$ — это количество таких y ($1 \leq y \leq n$), что $f(x, y) = 1$.
- Ясно, что любой элемент $x \neq 1$ находится на расстоянии $\frac{n+1}{2} - g(x)$ от единицы.

Поставим единицу на позицию $\frac{n+1}{2}$ в перестановке и произвольным образом расставим элементы, которые должны быть на расстоянии ровно 1 от нее. Так можно сделать, ввиду того, что функция f не изменяется от применения к перестановке операции циклического сдвига и разворота.

Для каждого элемента x на расстоянии $\leq \frac{n-3}{2}$ от единицы можно однозначно восстановить, в какой половине перестановки он находится, для этого достаточно проверить равенство $f(p_{\frac{n-1}{2}}, x) = 1$. В случае если это равенство выполняется, элемент лежит в правой половине, а иначе — в левой.

Расположение же элементов p_1 и p_n можно перебрать и проверить наивно корректность всех запросов. В некоторых случаях однозначно восстановить их взаимное расположение невозможно.

Подзадача 3.

Эта подзадача гарантировала, что решения лучше квадратичных смогут набрать хоть какие-то баллы.

Решение на 70+ баллов. (Вероятностное, сложное, но доказуемое)

Решение можно разбить на три фазы:

Фаза 1. Разбиение элементов на две половины относительно единицы.

Будем поддерживать 5 множеств элементов L — множество элементов, которые находятся левее единицы, R — множество элементов, которые находятся правее единицы, $L' \subset L$, $R' \subset R$ и C — множество элементов на расстоянии $\frac{n-1}{2}$ от единицы. Множества L' и R' будут содержать какое-то подмножество самых близких к единице элементов из множеств L и R .

Выберем случайный элемент x , для которого мы еще не смогли определить, с какой стороны он находится:

- Если $L = R = \emptyset$, то сформируем множество $R = R' = \{y \mid f(x, y) = 1\}$. В ином случае гарантируется, что $L' \neq \emptyset$ и $R' \neq \emptyset$ (см. следующий пункт) и тогда можно сформировать пару множеств $L(x) = \{y \in L' \mid f(x, y) = 1\}$ и $R(x) = \{y \in R' \mid f(x, y) = 1\}$.
 - Если $L(x) = R(x) = \emptyset$, то элемент x находится на расстоянии $\frac{n-1}{2}$ от единицы и он является *угловым*, поэтому он добавляется в множество C и следующий пункт алгоритма не производится.
 - Если $R(x) \neq \emptyset$, то $L(x) = \emptyset$. Значит, можно сформировать $R' \leftarrow R(x)$ и отнести x в левую половину.
 - Если $L(x) \neq \emptyset$, то $R(x) = \emptyset$ и надо присвоить $L' \leftarrow L(x)$.
 - Необязательно одновременно формировать множества $L(x)$ и $R(x)$, сначала можно проверить на пустоту одно, и только при необходимости сформировать второе множество.
- Положим, что мы определили, что x лежит в левой половине (т.е. в предыдущем пункте мы изменяли R'), тогда возьмем произвольный случайный элемент $y \in R'$ после обновления. И сформируем множество $L(y) = \{z \notin L \cup R \mid f(z, y) = 1\} \cup \{x\} \cup L$, присвоив $L \leftarrow L(y)$. Если $L' = \emptyset$, то присвоим $L' \leftarrow L$.

- Аналогично поступим, если x лежит в правой половине.

Таким образом если повторять два пункта выше, то значения $2, \dots, n$ в конце концов разобьются на 3 группы L, R, C — левые значения, правые значения и угловые. Заметим, что в ходе выполнения алгоритма множества $L(x), R(x), L(y), R(y)$ при условии своей непустоты совпадали с $\{z \mid f(x, z) = 1\}$ или $\{z \mid f(y, z) = 1\}$ (доказательство в качестве упражнения). Более того, можно показать, что при каждой операции величина $n - |L(x)| - |R(x)|$ уменьшалась в среднем в два раза, а также примерно в два раза уменьшалась размер L' или R' . То есть первая фаза требует $O(n)$ запросов. Разумеется, можно произвести более аккуратный анализ и строго вычислить математическое ожидание количества запросов.

Обозначим $G(x) = \{y \mid f(x, y) = 1\}$. Так как мы уже определили для некоторых элементов $G(x)$, а также для всех элементов определили с какой стороны от единицы они располагаются, то для таких элементов мы уже однозначно умеем восстанавливать их позицию. Более того, различные значения вычисленных множеств $G(x)$ для $x \in L$ позволяют разбить все элементы из y на группы подряд идущих на цикле значений. Действительно, если $G(x_1) \subset G(x_2)$, $x_1, x_2 \in L$, то все элементы из $G(x_1)$ находятся к единице ближе, чем элементы из $G(x_2) \setminus G(x_1)$.

Фаза 2. Уточнение позиций элементов из L и R .

Разобьем все элементы из R , на блоки $L \setminus G(x_1), G(x_1) \setminus G(x_2), G(x_2) \setminus G(x_3), \dots, G(x_{k-1}) \setminus G(x_k)$, где $x_i \in L$ и $G(x_k) \subset G(x_{k-1}) \subset \dots \subset G(x_1) \subset L$, пусть это блоки B_1, \dots, B_k . Для каждого блока B_i можно определить множество позиций, которые занимают значения из B_i и между разными блоками эти множества позиций не пересекаются. Более того, ясно, что если в каком-то блоке содержится ровно одно значение $x \in B_i$, позиция которого неизвестна, то ее можно сразу восстановить.

Таким образом, чем больше значений $G(x)$ ($x \in L$) мы знаем, тем точнее можно расставить элементы из правой половины. Значения $x \in L$ точно также можно разбить по блокам, руководствуясь вычисленными множествами $G(y)$ ($y \in R$).

Если мы выберем случайный $x \in L$, для которого еще неизвестна его позиция, то необязательно проверять все $y \in R$, чтобы сформировать $G(x)$. Во-первых, есть значения $x' \in L$, для которых мы знаем $G(x')$ и которые гарантированно находятся или левее, или правее x (исходя из деления по блокам левой половины), то из этого мы можем заключить $G(x) \subset G(x')$ или $G(x') \subset G(x)$ в зависимости от взаимного расположения. Во вторых, зная значения $G(y)$ ($y \in R$) также можно сузить отрезок значений, для которых неизвестно, должны ли они лежать в $G(x)$ ($x \in G(y) \Leftrightarrow y \in G(x)$ + что-то известно про взаимное расположение блоков и уже поставленных на место элементов).

После этих отсечений остается только какой-то отрезок блоков из правой половины B_l, \dots, B_r , для элементов которых неизвестно лежат они в $G(x)$ или нет. Сначала будут идти блоки, которые не лежат в $G(x)$, потом, блок, который частично лежит в $G(x)$, а затем блоки, которые полностью содержатся в $G(x)$. Поэтому можно воспользоваться бинарным поиском и за $\log_2(r - l)$ определить пару блоков, которые возможно, частично лежат в $G(x)$. А затем уже за сумму размеров этих блоков определяем, какой из блоков лежит частично и делим его на пару блоков.

Таким образом мы придумали достаточно экономный по запросам способ делить блоки на еще меньшие блоки. Предлагаемое решение выбирает случайно элемент x для которого неизвестно $G(x)$ (выбираются равновероятно и элементы из L и элементы из R), а затем выполняется описанная процедура разделения блоков. В конце концов все блоки делятся на блоки размера 1 и происходит победа. Разумеется, множества $G(x)$ не поддерживаются напрямую, вычисляется только разбиение на блоки.

Эту часть можно реализовать так, чтобы она работала за $O(n \log n)$ по мат ожиданию времени работы. Для этого достаточно работать за время $O(|B_x| + |B_a|)$ — где B_x — размер блока, где находится x , а $|B_a|$ — суммарный размер блоков из другой половины, которые оказались интересными для рассмотрения.

Фаза 3.

После третьей фазы восстановились все значения p_2, \dots, p_{n-1} ($p_{\frac{n+1}{2}} = 1$), остается только расположить элементы из C (угловые элементы) на позициях p_1 и p_n . Это не всегда можно сделать однозначно, но утверждается, что достаточно рассмотреть значения $f(c, p_2)$ и $f(c, p_{n-1})$ ($c \in C$) и по ним восстановить любое подходящее расположение угловых элементов.

Не легко, но плюс-минус интуитивно, можно показать, что математическое ожидание количества запросов у такого решения не превышает $O(n \log n)$.

Решение на 85+ баллов. (Теория информации)

Пусть мы находимся на второй фазе алгоритма. По сути, уже определены C — угловые элементы (на расстоянии $\frac{n-1}{2}$ от единицы), определены позиции некоторых элементов, а остальные элементы разбиты на блоки B_1, \dots, B_k , где каждый блок состоит из каких-то значений, для которых известно, что они находятся на каком-то множестве подряд идущих значений, на которых еще не стоят восстановленные элементы перестановки (какой-то найденный элемент при этом может стоять между какими-то двумя возможными позициями для одного блока), при этом эти множества позиций не пересекаются.

Тогда, имея эту информацию возможно $2^J = |B_1|! \cdot |B_2|! \cdot \dots \cdot |B_k|!$ возможных перестановок, количество информации J вычисляется, соответственно, как $J = \sum_{i=1}^k \log_2 |B_i|! = \sum_{i=1}^k \sum_{j=1}^{|B_i|} \log_2 j$. Во многих задачах применима эвристика на основе теории информации — достаточно выбирать жадную стратегию, которая на каждом шаге максимизирует отношение уменьшения количества информации к количеству потраченных запросов, т.е. максимизируется математическое ожидание величины $-\frac{\Delta J}{\Delta q}$ по всем допустимым шагам, где ΔJ — это количество информации после очередного шага, а Δq — потраченное на этот шаг количество запросов.

В нашем случае, шаг заключается в том, что мы выбираем какой-то элемент x с неизвестной позицией, а затем находит для него $G(x)$ ($G(x) = \{y \mid f(x, y) = 1\}$), после чего узнаем позицию элемента x , а также разрезаем какой-то блок на две части.

Рассчитаем количество запросов и уменьшение количества информации, которое произойдет, если мы определим позицию элемента с номером i :

- $-\Delta J = \log_2 |B(i)| - \log_2(X!) - \log_2(Y!) + \log_2((X+Y)!)$, где $|B(i)|$ — размер блока, в котором содержится элемент i , а X и Y — это размеры блоков, на которые разрезается блок, после нахождения блока i .
- $\Delta q \approx X + Y + \log_2 M(i)$, где X и Y — те же величины, что и в прошлом пункте, а $M(i)$ — количество блоков, которые может разрезать элемент x' , находящийся в том же блоке, что и x .

Соответственно, математическое ожидание отношения полученной информации к затраченным действиям $E[-\frac{\Delta J}{\Delta q}]$ для конкретного блока B_j равно среднему арифметическому $-\frac{\Delta J}{\Delta q}$ для всех возможных индексов $i \in B_j$, где могут лежать элементы из B_j . И жадная стратегия будет заключаться в том, чтобы взять случайное значение из того блока, в котором это математическое ожидание как можно больше.

Остается только научиться пересчитывать все описанные значения. Удивительно, но это все можно делать наивно, суммарное количество изменений ΔJ и Δq по всем индексам оказывается не очень большим.

Решение на 100+ баллов. (Дерево минимумов)

Предположим, что мы разделили все элементы на 3 группы L, R, C (читать первую фазу алгоритма на 70 баллов). Тогда можно потратить $n \log_2 n$ запросов на построение дерева минимумов по элементам L и по элементам R . Деревом минимумов называется дерево, которое строится для массива $[a_1, \dots, a_n]$ следующим образом:

- Выбирается минимальный по значению элемент a_i , он становится корнем дерева, а левым и правым сыновьями этой вершины становятся деревья минимумов, построенные для подотрезков $a[1 \dots, i-1]$ и $a[i+1, \dots, n]$.

Легко заметить, что если мы выбираем два значения $x, y \in L$, то $f(x, y)$ будет равняться LCA вершин x и y в дереве минимумов для подотрезка перестановки, где располагаются элементы из L .

Для построения дерева минимумов для множества L можно постепенно добавлять в него элементы в порядке возрастания. Пусть мы добавляем элемент x , тогда можно выделить путь из корня в уже построенном дереве, который всегда идет в большее поддерево. Если этот путь заканчивается в листе y , то после запроса $f(x, y)$ мы понимаем, LCA вершин x и y , то есть тот момент, в который

надо сойти с этого пути. Сошли с пути — зашли в поддереву в два раза меньше, можем повторить запрос, за $O(\log n)$ запросов смогли понять, куда подвесится вершина x . Единственным нюансом является тот факт, что в построенном дереве непонятно, какие сыновья левые, а какие правые, но даже без этой информации построенное дерево помогает ускорить вторую фазу алгоритма.

Заметим, что когда разрежется какой-то блок или из блока мы узнаем значение какого-то элемента и его понадобится вырезать из дерева минимумов, то мы все спокойно сможем пересчитать и не будет никак проблем. Достаточно разрезать исходное дерево минимумов на минимальное количество деревьев, а потом корни полученных деревьев соединить в одно дерево тем же алгоритмом.

Теперь рассмотрим то, почему дерево минимумов в принципе может помогать. Пусть у нас есть элемент x и набор блоков с построенными деревьями B_1, \dots, B_n , а в данный момент мы разрезаем элементом x какой-то блок B . Обозначим $a(v) = 1$ ($v \in B$), если $f(x, v) = 1$ и $a(v) = 0$ в ином случае. Есть следующие отсечения:

- Если v — корень, а l и r — какие-то вершины из его разных поддеревьев, то если $a(v) \neq a(l)$, то $a(v) = a(r)$.
- Если v — корень, а в одном из его поддеревьев есть y ($a(y) \neq a(v)$), то можно отделить от дерева v сына, в поддереве которого лежит вершина y , и однозначно восстановить какое дерево левее, а какое правее (то есть дополнительно разделить все на блоки).
- Вычислять значения $a(v) = f(x, v)$ выгодно в порядке убывания размеров поддеревьев v .

Эта пара отсечений работает хорошо, так как разрезает блок, не на один блок, а по крайней мере на $\Omega(\log(n))$ по матожиданию, если разрезать блоки в случайных местах. Более того, такой способ дает некоторые гарантии (сейчас будет рукомахание, но все же):

- Мы режем дерево в случайном месте, поэтому чаще всего $f(v, x) = 1$ и $f(v, x) \neq 1$ будет встречаться примерно с соизмеримой вероятностью. Поэтому будет очень часто работать отсечение по первому пункту и мы сделаем мало запросов на сбалансированном дереве.
- Если же мы режем дерево не в равной пропорции, то мы сделаем много запросов, но и разрежется много ребер, потому что мы достаточно глубоко спустимся.