

## Разбор задачи «Перфокарты»

Автор: Дмитрий Саютин  
Разработчик: Рамазан Рахматуллин

Первая подзадача предназначена для переборного решения, пробуящего все возможные порядки выкладывания карт.

Посмотрим на самую верхнюю перфокарту в ответе. Все присутствующие в ней символы должны совпадать с соответствующими символами строки  $s$ . Теперь рассмотрим вторую сверху перфокарту. Все позиции в ней, за исключением тех, которые присутствовали в первой перфокарте, должны также совпадать с символами строки  $s$ . Данную идею можно обобщить: будем брать любую перфокарту, которую в данный момент можно положить под уже выбранные, и помечать непомеченные позиции этой перфокарты в строке  $s$ . После этого действия некоторые перфокарты можно будет положить под уже выбранные, причем если перфокарту можно положить, то это свойство останется.

Из этой идеи вырисовывается решение. Для каждой перфокарты храним число непомеченных позиций, которые не совпадают со строкой  $s$ . Также храним любым образом множество перфокарт, у которых это число равно нулю. Повторяем следующую процедуру:

- достаём любую перфокарту из множества, если оно непусто
- перебираем ее непомеченные позиции (они по предположению все совпадают с соответствующими символами строки)
- каждую непомеченную позицию помечаем
- уменьшаем число на единицу у всех перфокарт, у которых на этой позиции символ не совпадает с правильным
- если у какой-то перфокарты число обратилось в 0, добавляем её в множество

Если в множестве не осталось перфокарт, при этом не все из них положены, то ответ — -1. Также нужно не забыть про ситуацию, когда все карты выложить получилось, но при этом некоторые позиции остались непомеченными; это означает, что какая-то позиция не содержит нужной буквы ни в одной перфокарте, поэтому ответ также -1.

Осталось доказать, что если наш алгоритм не нашёл ответ, то его не существует. Предположим, что наш алгоритм зашёл в тупик, выложив перфокарты  $p_1, \dots, p_k$ , при этом существовал какой-то иной порядок  $q_1, \dots, q_n$ , реализующий заданную строку. Но тогда можно рассмотреть первую перфокарту  $q_j$  во втором порядке, которая не встречается среди чисел  $p_i$  — легко понять, что последовательность  $p_1, \dots, p_k, q_j$  является корректной, т.к.  $q_j$  можно было положить даже после некоторого подмножества предшествовавших перфокарт.

Вторая подзадача допускает неэффективные реализации идеи выше (например, вместо поддержания множества «нулевых» перфокарт, пробегаться по всему списку оставшихся перфокарт и искать подходящую).

Эффективная версия решения работает за время  $O(n + m + \sum_{i=1}^n k_i)$  и проходит на полный балл.

## Разбор задачи «Родные просторы»

Автор и разработчик: Николай Будин

Дана строка  $s$ . За ход можно взять два соседних символа  $s[i]$  и  $s[i + 1]$ , если  $A[s[i]][s[i + 1]]$ , и удалить из строки  $s[i + 1]$ . Нужно получить лексикографически минимальную возможную строку.

Рассмотрим граф, вершины которого соответствуют символам исходной строки. Когда берем пару соседних символов и удаляем правый, проведем в этом графе ориентированное ребро из исходной позиции левого символа в исходную позицию правого. Заметим, что получился подвешенный лес. Корнями деревьев являются те и только те символы, которые остались в итоговой строке. Вершины одного дерева образуют отрезок подряд идущих символов, корень является самым левым из них.

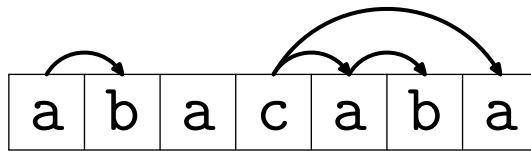


Рис. 1: Граф для первого теста.

Задача состоит из двух частей — в первой нужно разобраться с тем, какие строки вообще возможно получить такой процедурой, а во второй — научиться искать минимальный лексикографически ответ.

## Часть 1

Рассмотрим символ  $c$ , который останется в итоговой строке. Будем говорить, что отрезок символов, удаленных непосредственно после  $c$ , был удален с использованием  $c$ . Например, на картинке отрезок « $caba$ » был удален с использованием « $c$ ». Заметим, что если отрезок  $s[l+1 \dots r]$  можно удалить с использованием  $s[l]$ , то также с помощью  $s[l]$  можно удалить и  $s[l+1 \dots r-1]$ . Поэтому, для каждой позиции  $s[l]$  нужно найти максимальную длину отрезка  $s[l+1 \dots r]$ , который можно удалить с использованием  $s[l]$ .

### Способ 1.1

Можно воспользоваться методом динамического программирования.  $dp[c][l][r]$  — можно ли удалить отрезок  $s[l \dots r]$  с использованием символа  $c$ , написанного ровно перед ним. Если  $A[c][s[l]] = 0$ , то это в любом случае невозможно. Иначе, переберем  $m$  ( $l \leq m < r$ ). Если  $dp[c][l][m]$  и  $dp[c][m+1][r]$ , то  $dp[c][l][r] = true$ . Второй вариант — если  $dp[s[l]][l+1][r]$ , то  $dp[c][l][r] = true$ . Время работы такого алгоритма  $O(n^3 \cdot k)$ .

### Способ 1.2

Можно дополнительно ускорить предыдущий алгоритм.  $dp'[c][l]$  — максимальное  $r$ , для которого  $dp[c][l][r] = true$ . Пересчет происходит аналогично. Для фиксированных  $c$  и  $l$ , инициализируем  $dp'[c][l] = dp'[s[l]][l+1]$ , и перебираем  $r$  в порядке возрастания. Если  $r \leq dp'[c][l]$  и  $A[c][s[r]]$ , то  $dp'[c][l] = \max(dp'[c][l], dp'[c][r])$ . Получился алгоритм за  $O(n^2 \cdot k)$ .

### Способ 2

Это решение будет отличаться от предыдущих. Заведем стек  $st$  и пройдемся по строке  $s$  справа налево. Когда рассматриваем  $s[i]$ , пока стек не пуст и  $A[s[i]][s[st.top()]]$ , удалим верхний элемент из стека. Оказывается, что после этого максимальный отрезок, который можно удалить, используя  $s[i]$ , заканчивается в  $s[st.top() - 1]$  (либо в  $n$ , если стек пуст). После этого, добавим в стек  $i$ . Этот алгоритм работает за линейное время.

Докажем этот алгоритм. Пусть для какого-то  $i$  длина отрезка была найдена неправильно. Рассмотрим самый правый из таких  $i$ . Мы решили, что отрезок заканчивается в  $y$  ( $y = st.top() - 1$ ) и  $A[s[i]][s[y+1]] = 0$ , а на самом деле отрезок должен заканчиваться в  $r$  ( $y < r$ ). Рассмотрим дерево способа, в котором мы удалили отрезок до  $r$ . В этом способе  $s[y+1]$  не может быть непосредственным ребенком  $s[i]$ . Будем подниматься из  $s[y+1]$  вверх по дереву, пока не дойдем до ребенка  $s[i]$ , пусть это будет  $s[x]$ . По предположению, для  $s[x]$  мы нашли отрезок корректно. А значит, он должен заканчиваться не раньше  $s[y+1]$ , а значит  $y+1$  уже был удален из стека, когда мы рассматривали  $s[x]$ , значит его не могло быть в стеке и когда мы рассматривали  $s[i]$ . Противоречие.

## Часть 2

Теперь осталось разбить строку  $s$  на отрезки  $a_1, a_2, \dots, a_k$ , такие что весь отрезок  $a_i$ , кроме первого символа, может быть удален с использованием  $a_i[1]$ . А строка  $a_1[1] + a_2[1] + \dots + a_k[1]$  — лексико-

графически минимальная возможная, где  $+$  — операция конкатенации строк.

### Способ 1

Вспользуемся методом динамического программирования.  $ans[i]$  — ответ для строки  $s[i \dots n]$ . Тогда пересчет  $ans[i] = \min(ans[i], s[i] + ans[j])$ , если  $s[i + 1 \dots j - 1]$  можно удалить с использованием  $s[i]$ . Такой алгоритм работает за  $O(n^3)$ .

### Способ 2

Вместо того, чтобы явно хранить в  $ans[i]$  строки, сохраним  $next[i]$  — индекс второго символа  $ans[i]$ . Тогда  $ans[i] = s[i] + s[next[i]] + s[next[next[i]]] + \dots$ . Ссылки  $next[i]$  образуют подвешенное дерево. А каждое  $ans[i]$  — путь из вершины  $i$  до корня. В этом дереве можно вычислить двоичные подъемы, и для каждого двоичного подъема хеш. Чтобы сравнить лексикографически две строки, нужно найти их максимальный совпадающий префикс, и сравнить следующий символ. Найти максимальный совпадающий префикс двух путей в дереве можно за  $O(\log(n))$  с помощью двоичных подъемов и хешей.

#### Способ 2.1

Если применить такое сравнение строк к первому способу, получится алгоритм за  $O(n^2 \log(n))$ .

#### Способ 2.2

Построим дерево отрезков на  $ans[i]$ , которые мы сравниваем с помощью двоичных подъемов и хешей. Тогда  $ans[i] = s[i] + \min_{j=i+1}^{dp[s[i]][i+1]} ans[j]$ , это один запрос к дереву отрезков. Получается  $O(\log^2(n))$  на один запрос. И алгоритм работает за  $O(n \log^2(n))$ .

#### Способ 2.3

Объединим алгоритмы первой и второй части. В тот момент, когда мы рассматриваем  $s[i]$  и выкидываем из стека индекс  $st.top()$ , сделаем  $ans[i] = \min(ans[i], s[i] + ans[st.top()][1 \dots])$ . Такой алгоритм сделаем  $O(n)$  сравнений и будет работать за  $O(n \log(n))$ .

### Заключение

В зависимости от способа, выбранного в первой и второй части, решение может получить разные баллы. Полное решение работает за  $O(n \log(n))$ .

## Разбор задачи «Игра с тайным смыслом»

Автор: Михаил Дворкин  
Разработчики: Павел Кунявский, Михаил Дворкин

В отличие от большинства задач олимпиады, в этой задаче нет четкого разделения, какое количество баллов можно получить за различные идеи решения. Кроме того, скорее всего будет много решений, сильно отличающихся от авторских, да и авторские решения достаточно сильно могут зависеть от деталей реализации и подбора использующихся в них констант. Из-за необычного формата, задача также отличалась сложностью отладки решения, поэтому важно было не бросаться сразу писать сложное решение, а постепенно усложнять решение, экспериментируя с различными идеями и оптимизациями. Жюри надеется, что этот эксперимент понравился участникам.

Решение, которое передаёт один бит за игру, уже набирает 10 баллов. Для этого рассмотрим ход (1, 2) или любой другой ход, который сразу же может сделать программа и в случае, когда решение ходит первым, и когда бот делает первый ход, а программа — второй. Тогда если нужно передать

бит 1, то программа делает этот ход, а следующим ходом сделает проигрышный ход  $(1, 1)$ . Для передачи бита 0 программа сразу же делает проигрышный ход  $(1, 1)$ .

Можно модифицировать это решение, научив его выигрывать партию. Рассмотрим два хода  $(1, 2)$  и  $(2, 1)$ . Заметим, что решение всегда может сделать эти два хода в начале. Тогда мы можем передавать один бит при помощи порядка этих ходов. При этом такое решение ещё и будет всегда выигрывать игру, поэтому оно набирает 20 баллов. Интересной особенностью такого решения является то, что оно может игнорировать ходы бота и вообще не хранить игровое поле, но развивать дальше эту идею вряд ли получится.

Начнем с обсуждения, как выигрывать игры. Игра «Щелк» известна специалистам по теории игр одним интересным свойством — в этой игре всегда выигрывает первый игрок, но уже для небольших полей не известна выигрышная стратегия. Посмотрим на ход  $(n, n)$ , закрашивающий правый верхний угол. Этот ход либо выигрышный, либо у второго игрока есть ответный выигрышный ход. Но в этом случае первый игрок может вместо хода  $(n, n)$  сделать этот ход и выиграть. Но это доказательство не позволяет найти выигрышную стратегию.

В случае квадратного поля (как в нашей задаче) есть симметричная выигрышная стратегия. Первый игрок может сделать ход  $(2, 2)$ , оставив от игрового поля только первый столбец и первую строку. После этого он будет симметрично повторять ходы второго игрока. В этой задаче решение может играть как первым, так и вторым, но так как бот делает случайные маленькие ходы, то бот не сможет сделать ход  $(2, 2)$  в начале игры, поэтому решение может воспользоваться симметричной стратегией и даже если ходит вторым. При этом решение не обязано сразу же пользоваться симметричной стратегией, поскольку противник не делает большие ходы. Можно в начале игры передавать информацию, игнорируя ходы противника, а в конце игры переключится на симметричную стратегию, чтобы выиграть партию. Пока оставшиеся два ряда длинные, можно передавать по одному биту за ход выбором стороны и несколько бит, выбирая длину хода. Это позволяет передать несколько десятков бит за игру и получить 15–20 баллов и ещё 10 баллов за выигрыш всех игр.

Другой идеей является кодирование бит координатами хода. Например, можно каждым ходом кодировать 10 бит — 5 бит номером строки и 5 бит номером столбца. У этого подхода есть два недостатка. Во-первых, игра будет заканчиваться достаточно быстро, так как часто придётся делать ходы, закрашивающие много клеток. Во-вторых, если нужный ход сделать невозможно, нужно как-то указать, что этот ход не передаёт информацию на самом деле. Избавиться от этих недостатков можно, если передавать информацию только одной координатой хода, выбирая вторую координату так, чтобы закрашивать как можно меньше клеток. Это даёт возможность сделать больше ходов, передающих информацию, а если ход сделать нельзя, то решение делает ход  $(2, 2)$  и переключается в режим симметричной стратегии для выигрыша игры. Таким способом можно получить 40–60 баллов в зависимости от деталей реализации.

Проблема описанных подходов в том, что в одном из них съедается много клеток, а в другом — слишком мало вариантов хода. Нужен промежуточный вариант! Заметим, что необязательно передавать информацию именно координатами. Можно перенумеровать все возможные ходы по какому-то правилу и таким образом передавать информацию. Выберем константу  $k$  и рассмотрим все ходы, которые закрашивают не более  $k$  клеток. Пусть количество таких возможных ходов оказалось равно  $C$ . Тогда если оба запуска перенумеруют все эти ходы одинаковым способом, то номером выбранного хода можно передать примерно  $\log_2 C$  информации. В таком решении необходимо подобрать константу  $k$ . Если её выбрать небольшой, то будет передаваться мало информации за ход, а при большом  $k$  ходы быстро закончатся. Значение  $k$  можно подобрать экспериментально, запустив решение несколько раз с разными значениями  $k$  или отправив его в тестирующую систему. Например, в экспериментах жюри оптимальным был выбор  $k \approx 10$ , но это значение может быть разным в зависимости от деталей реализации. Подобные решения позволяют передавать около 700 бит за игру и получить 80–90 баллов, если добавить в это решение переход к выигрышной стратегии не слишком поздно. Одной из особенностей такого решения является то, что количество передаваемых бит за ход может быть различным в зависимости от значения  $C$ , которое зависит от игровой ситуации.

Улучшить это решение можно следующим образом. Заметим, что не все выбранные ходы одинаково ценны: большие ходы (закрашивающие много клеток) быстрее приводят к окончанию игры,

поэтому делать их невыгодно. Давайте передавать короткими ходами мало бит информации, а большими ходами — много бит информации, то есть “невыгодные” ходы будут полезны хотя бы тем, что они передают больше бит информации. Рассмотрим все рассматриваемые решением потенциальные ходы на данном шаге, пусть опять-таки их количество равно  $C$ . Ранее мы предлагали каждому из таких ходов сопоставить двоичную строку длины  $\approx \log_2 C$ . Теперь же сопоставим разным ходам разные двоичные строки: ходам, закрашивающим мало клеток будут сопоставлены короткие строки, а ходам, закрашивающим много клеток — длинные строки. Предположим, что решение в среднем передаёт  $s$  бит, закрашивая одну клетку. Константа  $s$  — эвристическая и в решениях жюри использовалось значение  $s \approx 1,1$ . Нам нужно для каждой из возможных  $C$  клеток выбрать двоичное кодовое слово  $w_i$  (передаваемая этим ходом информация), так чтобы максимизировать среднее значение величины  $\text{len}(w_i) - e_i \times C$ , где  $e_i$  — количество клеток, закрашенных этим ходом.

Чтобы сопоставить возможным ходам такие строки, можно воспользоваться эвристическим жадным решением, похожим на алгоритм построения кода Хаффмана (оптимального префиксного кода) или динамическим программированием. Например, в алгоритме Хаффмана строится представление множества строк в виде дерева, где у каждой вершины, не являющейся листом, — два потомка, а рёбрам соответствуют передаваемые биты 0 или 1. В листьях записаны ходы (выбранные клетки), а передаваемое кодовое слово  $w_i$  есть последовательность нулей и единиц на пути из корня дерева. Для построения дерева выбираются две вершины, для которых количество закрашиваемых клеток максимально, и они объединяются вместе. Это продолжается, пока не останется одна вершина — корень дерева.

Конкретные значения константы  $s$ , а также значения  $k$  (отсечения, насколько большие ходы допускаются), нужно подобрать экспериментально.

## Разбор задачи «Доклад инвесторам»

Автор: Ильдар Гайнуллин

Разработчик: Егор Чунаев

Назовем дерево с числами на листьях *деревом поиска*, если возможно упорядочить детей у каждой нелистовой вершины, чтобы значения на листьях возрастали в порядке обхода.

Нам дано дерево из  $n$  вершин, где у каждой вершины не более чем  $k$  сыновей; требуется для каждого листа выбрать одно значение из множества возможных значений для данного листа, чтобы получилось дерево поиска. Обозначим за  $M$  суммарное количество возможных значений в листьях.

В первой подзадаче дерево достаточно маленькое, и максимальное количество сыновей не превосходит двух. Поэтому в ней работает полный перебор — мы можем перебрать все способы расставить все значения в листья и перебрать порядок детей в каждой из вершин. У нас есть  $2^n$  вариантов упорядочить детей и примерно  $2^{\frac{M}{2}}$  вариантов зафиксировать числа в листьях в худшей конструкции ( $\frac{n}{2}$  листьев, в каждом два варианта), получаем  $O(2^{n+\frac{M}{2}})$ .

Во второй подзадаче  $n$  и  $M$  все ещё достаточно маленькие, но максимальное количество детей уже не ограничено. Заметим, что если числа в листьях зафиксированы, то порядок детей в каждой из вершин восстанавливается однозначно. Каждому поддереву можно сопоставить два числа — минимальное и максимальное значение в этом поддереве. Тогда в каждой вершине надо упорядочить детей по минимальному значению в поддереве и проверить что отрезки значений поддеревьев не пересекаются.

Для решения последующих подзадач нам надо воспользоваться методом динамического программирования.

Обозначим за  $dp(v, x)$  минимально возможное максимальное значение в поддереве, если мы можем выбирать в этом поддереве только числа больше или равные  $x$ , либо  $+\infty$ , если это поддерево невозможно упорядочить в таких условиях.

Переменная  $x$  может принимать значения от 0 до максимального значения элемента по всем листьям  $M$ .

Обсудим, как инициализировать значение динамического программирования в листе. Если лист содержит числа  $a_1 < a_2 < \dots < a_r$ , то

$$\begin{aligned} dp(v, 0) &= dp(v, 1) = \dots = dp(v, a_1) = a_1 \\ dp(v, a_1 + 1) &= dp(v, a_1 + 2) = \dots = dp(v, a_2) = a_2 \\ &\dots \\ dp(v, a_{r-1} + 1) &= dp(v, a_{r-1} + 2) = \dots = dp(v, a_r) = a_r \\ dp(v, a_r + 1) &= dp(v, a_r + 2) = \dots = dp(v, M) = +\infty \end{aligned}$$

Обозначим детей вершины за  $u_1, u_2, \dots, u_m$ . Тогда верно следующее:

$$dp(v, x) = \min_{\text{перестановка } p} dp(u_{p_1}, dp(u_{p_2}, dp(\dots dp(u_{p_m}, x) \dots)))$$

Если сжать значения в листах, то эта динамика имеет  $O(nM)$  состояний, поэтому это решение работает за  $O(nM \cdot k!)$  и проходит все подгруппы с  $k \leq 5$  и  $n, M \leq 2000$ , а также какие-то из групп с  $k = 8$  и меньшими  $n$ .

Пересчет в вершине можно улучшить, введя вспомогательную динамику от подмножества уже взятых детей  $S = \{a_1, a_2, \dots, a_t\}$  и числа  $x$ , обозначающего текущее значение максимума. Положим:

$$f(S, x) = \min_{\text{перестановка } p} dp(a_{p_1}, dp(a_{p_2}, dp(\dots dp(a_{p_t}, x) \dots)))$$

Такую динамику в вершине можно посчитать за  $m2^m$ , где  $m$  — количество детей вершины.

Вершин со степенью  $m$  не более чем  $\frac{n}{m}$ , поэтому это решение будет работать за  $O(\sum_{m=1}^k M \cdot \frac{n}{m} m 2^m) = O(2^k nM)$

Такого метода пересчета уже достаточно, чтобы пройти все группы, где  $n, M \leq 2000$ .

Для решения трёх последних подзадач надо делать что-то аналогичное описанной динамике, но хранить значение минимума не по всем возможным значениям  $x$ , а использовать какой-то сжатый вид.

Для каждой вершины  $v$  будем хранить  $P_v$ : множество упорядоченных пар  $(a, b)$  таких, что возможно так выбрать значения в листьях в поддереве вершины  $v$ , чтобы получилось дерево поиска, и при этом минимальное и максимальное значение листьев равно  $a$  и  $b$  соответственно.

**Утверждение 1:** Если есть две пары  $(a, b) \in P_v$  и  $(c, d) \in P_v$ , что  $a \leq c \leq d \leq b$  и  $(a, b) \neq (c, d)$ , тогда пару  $(a, b)$  можно выбросить из  $P_v$ .

*Доказательство:* Решение, в котором поддерево соответствует паре  $(a, b)$ , можно поменять, заменив это поддерево на решение для пары  $(c, d)$ . Тогда легко видеть, что дерево останется деревом поиска.

**Утверждение 2:** Для вершины  $v$  с детьми  $c_1, c_2, \dots, c_k$ , можно найти  $P_v$  за  $O((|P_{c_1}| + |P_{c_2}| + \dots + |P_{c_k}|) \cdot (2^k + \log n))$

*Доказательство:*

Нам нужно сделать следующее, выбрать пары  $(a_i, b_i) \in P_{c_i}$  так, чтобы эти пары не пересекались как отрезки, а затем добавить  $(\min a_i, \max b_j)$  в  $P_v$

Выпишем в один список все  $a_i$  и  $b_i$ , отсортируем и оставим среди них только различные; получим последовательность  $s_1, s_2, \dots, s_m$ . После этого рассмотрим следующую динамику:

Значение  $dp_{i,S}$ : пусть мы уже выбрали непересекающиеся отрезки для детей из множества  $S$ , все они имеют  $a_j \geq s_i$ , и минимальное возможное значение  $\max b_i$  среди них равно  $dp_{i,S}$ .

Затем,  $dp_{i,S} = \min$  среди

- $dp_{i+1,S}$
- $dp_{j,S \cup \{t\}}$ , если есть пара  $(s_i, s_j)$  в  $P_{c_t}$
- $j$ , если есть пара  $(s_i, s_j)$  в  $P_{c_t}$  и  $S = \{t\}$

Мы можем найти значение этой динамики в порядке убывания  $i$ .

Затем, достаточно поставить  $P_v = \{(s_i, s_{dp_{i,C}}) | 1 \leq i \leq m\}$ , где  $C$  это множество всех детей.

Сортировка и динамика вместе работают за  $O((|P_{c_1}| + |P_{c_2}| + \dots + |P_{c_k}|) \cdot (2^k + \log n))$

После этого, как это следует из утверждения 1, можно удалить бесполезные пары из  $P_v$ . Это делается простым линейным проходом.

Также это можно сделать менее оптимально, просто перебирая все возможные порядки детей, тогда эта часть будет работать за  $O((|P_{c_1}| + |P_{c_2}| + \dots + |P_{c_k}|) \cdot (k! + \log n))$

**Утверждение 3:**

Обозначим за  $z_v$  суммарный размер множеств всех листьев в поддереве вершины  $v$ .

Пусть  $v$  это вершина с хотя бы двумя детьми.

Рассмотрим последовательность  $a_1, a_2, \dots, a_k$ , где  $a_i = z_{c_i}$

Тогда  $|P_v| \leq 2 \cdot (\sum a_i - \max a_i)$

*Доказательство:*

Пусть  $j$  это сын с наибольшим значением  $a_j$

Заметим, что каждая пара  $(a, b) \in P_v$  имеет либо  $a \notin P_j$  либо  $b \notin P_j$ .

А также, после удаления бесполезных пар, для каждого  $x$  в множестве содержится не более одной пары  $(x, b)$  и не более одной пары  $(a, x)$ . Это значит, что каждый  $i \neq j$  поспособствует добавлению не более двух пар в  $P_v$ . А из замечания выше мы знаем, что это покрывает все пары в  $P_v$ .

Таким образом,  $|P_v| \leq 2 \cdot (\sum a_i - \max a_i)$

**Утверждение 4:**

$|P_1| + |P_2| + \dots + |P_n|$  это  $O(n \log n)$ , где  $n$  это сумма размеров всех списков листьев и количества вершин.

*Доказательство:*

Доказательство тривиально из стандартной техники «меньшее к большему» и утверждения 3.

В итоге можем найти  $P_v$  для всех вершин за  $O(n \log n \cdot (2^k + \log n))$ . После этого ответ «Yes» только если  $|P_{root}| \neq 0$ , а дальше требуется восстановить ответ, что является несложным техническим упражнением с помощью посчитанных динамик.

Менее оптимальные решения, использующие перебор все перестановок детей и работающие за  $O(n \log n \cdot (k! + \log n))$ , проходят все группы кроме последней.