Алгоритмы на графах во внешней памяти Напомним, что мы хотим делать алгоритмы с менее, чем линейным числом обращения к памяти. Например, сложность $O(\frac{n \cdot Poly(logn)}{B})$ нас устраивает, а сложность $O(n)$ нас категорически не устраивает. Всякие $O(\frac{Poly(n)}{\sqrt{B}})$ или $O(\frac{Poly(n)}{\log B})$ нам не нравятся, но иногда мы не можем лучше.

Dfs. Если мы смогли сохранить ребра так, что для каждой вершины ребра лежат в памяти последовательно, то мы можем доставать соседей v блоками. Но, к сожалению, это не делает сложность алгоритма хорошей, потому что у нас есть массив $used$, к которому нам придется обращаться. Обращения будут произвольными. Тогда, если $n < M$, то алгоритм имеет смысл, потому что мы можем сохранить массив в оперативной памяти, а иначе этот алгоритм работает за $O(n + \frac{m}{B})$.

Работа со списками. Имеем какой-то односвязный список. Он реализован как массив и указатели. То есть, в каждой ячейке массива есть указатель на следующую ячейку массива. Научимся с ним работать.

Групповые операции. Чтобы считать несколько значений из списка, мы можем либо спросить про каждый элемент запроса, либо про все элементы списка — $O(q)$ или $O(\frac{n}{B})$. Чтобы сделать групповые изменения, можно отсортировать все запросы заранее, после чего обрабатывать запросы группами за линейное время блоками. Сложность будет равняться $O(sort(q, M, B) + \frac{n+q}{B})$.

Задача list ranking. Мы хотим для каждого элемента узнать его порядковый номер в соответствующем списке. Это тривиально делается в оперативной памяти, потому что мы можем просто пройти по списку. Но для внешней памяти алгоритм не подойдет — прыжки по памяти произвольны.

Во-первых, мы хотим сделать список двусвязным. Это то же самое, что и набор запросов «Присвой следующему элементу мой индекс в поле предка». Групповой запрос на изменение мы уже научились обрабатывать.

Теперь сделаем что-то типа двоичных подъемов. На итерации k считаем, что мы правильно посчитали ранги для всех вершин, у которых ранг не более 2^k. Кроме этого, будем поддерживать для всех вершин указатель на вершину на расстоянии 2^k от нее (и вперед, и назад). Как задать новые ранги? Для тех вершин, у которых уже известны ранги, есть суперссылка в вершину без ранга. Но тогда ее ранг — это ранг старой вершины, увеличенный на 2^k. Осталось пересчитать суперссылки. Их можно пересчитывать двоичными подъемами — брать суперссылку от нашей суперссылки. Но у нас доступа в произвольную точку памяти. Поэтому мы сделаем так: пусть у нас есть вершина v, ее суперпредок u, и ее суперсын w. Тогда новый суперсын от u это w, а суперпредок от w это u. Это тоже операция группового присваивания.

Пусть у нас был оракул, который умел говорить, четный или нечетный ранг для элемента. Пусть мы выкинули из списка все нечетные элементы, а указатели на следующий элемент списка теперь указывают на элемент «через один» от нас. Например, список $1 \to 3 \to 4 \to 2$ перейдет в $1 \to 2$. Мы уменьшили длину списка в два раза. Пусть мы посчитали ранги в четном списке. Тогда мы на самом деле знаем ранги и для нечетного списка тоже — это просто ранг предка, увеличенный на единицу. Таким образом, мы получим сложность $t(n) = t(\frac{n}{2}) + sort(n, M, B)$.

Как создать такого оракула? Мы сделаем его недетерменированным. А именно, для каждого элемента случайно решим, выкинем мы его или нет. Но нам нужно, чтобы для элемента остался в новом списке либо он, либо его предок. Тогда мы выкидываем элемент, только если мы случайно решили, что хотим выкинуть его, и случайно решили, что мы не хотим выкинуть его предка. Так мы отбросим четверть элементов. Получаем сложность $t(n) = t(\frac{3n}{4}) + sort(n, M, B)$. Так мы можем проталкивать величину «чему равен номер моего списка».

Но поскольку у нас нумерация сместилась произвольным образом, ранги пока что пересчитываются не так просто. Раньше мы просто умножали ранг на 2 и прибавляли 1, но сейчас мы не выкидывали некоторые элементы.

Пусть рекурсия вернула наш новый список с рангами. Теперь мы для каждого нерассмотренного элемента знаем, чему должен быть равен ранг относительно предка. Но ранги предков сейчас поедут. Предварительно скажем, что ранг новых вершин — это ранг предка, увеличенный на 1. После чего мы получили два массива рангов — для старых вершин и для новых. Теперь нам надо сделать линейный проход, и при встрече равных рангов сначала брать старую вершину, а потом увеличивать все ранги на суффиксе на 1 (каким-нибудь счетчиком). По сути, мы просто избавились от равных чисел за сложность сортировки. Таким образом, мы решили list ranking за сложность $O(sort(n, M, B))$.

Поиск минимального остова во внешней памяти с помощью алгоритма Борувка. Отсортируем ребра по первой вершине. Теперь за $O(\frac{n}{B})$ мы находим минимальное ребро из каждой вершины. Теперь нам надо сделать самый сложный шаг — найти компоненты.

У нас есть какое-то множество ребер, которое задает лес. Мы хотим для каждой вершины узнать номер ее компоненты. Тогда мы потом сожмем компоненты, уменьшим размер в два раза, и тд (см. алгоритм Борувки). Разобьем каждое ребро на два ориентироанных, и для каждой вершины зададим каждому ребру следующее, как в эйлеровом обходе. Теперь мы получаем списки ребер, и нам надо для каждого ребра просто понять номер его списка. Видим, что это и есть задача list ranking. Таким образом, мы сможем найти остов за $O(sort(n, M, B) \log n)$.