Динамическая связность

Мирошников Владислав
371 группа

2. Придумайте, как усовершенствовать алгоритм, чтобы научиться поддерживать декрементально (только удаления рёбер) минимальный остовный лес во взвешенном неориентированном графе также за $O(log^2 n)$ амортизированно.

Решение:

Чтобы алгоритм работал корректно, добавляем третий инвариант (*):

Если ребро е является ребром максимального веса среди рёбер некоторого цикла C, то у е самый низкий уровень среди всех рёбер C.

В случае удаления ребра, принадлежащего остовному дереву, необходимо искать замену данному ребру, чтобы сохранить остовное дерево. В противном случае, если ребра-замены не нашлось, то остовное дерево распадется на два дерева. Чтобы поддерживать декрементальную динамическую связность неориентированного графа, нужно перебирать уровни с i до $logn$. Внесем изменения в данный алгоритм и будем перебирать уровни в обратном порядке с $logn$ до i. Также введем для каждого уровня порядок перебора ребер. Будем искать подходящее ребро с наименьшим весом. Подходящее ребро-замена должно лежать концами в обоих деревьях, образовавшихся после удаления ребра, то есть оно должно как бы "склеить"и оставить остовное дерево целостным. Если рассматриваемое ребро не подходит, то ему присваивается уровень $i - 1$.

Теперь нужно показать, что данный алгоритм получения ребра-замены позволит поддерживать минимальный остовный лес при удалениях ребер.

Введем утверждение.

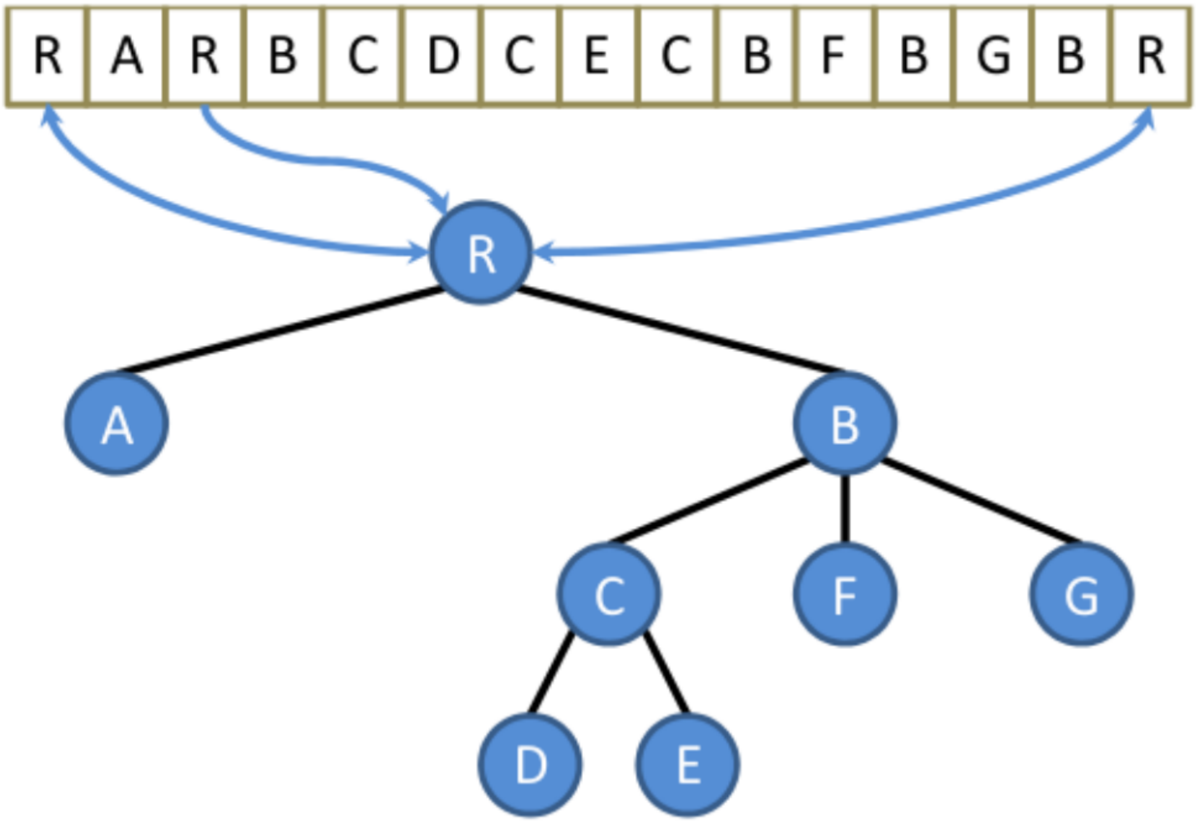
Утверждение (модифицированное под задачу из статьи): Пусть выполняется (*) и F - минимальное остовное дерево. Тогда для любого ребра е из дерева F, ребро с наименьшим весом имеет наибольший уровень среди всех вариантов на ребро-замену.

Доказательство. Пусть ребра e_1 и e_2 — кандидаты на ребро-замену ребра e. Эти ребра при добавлении в остовное дерево порождают циклы. Обозначим эти циклы как C_i для каждого ребра e_i, где i = 1, 2. Пусть e_1 легче, чем e_2. Покажем, что тогда $level(e_1) >= level(e_2)$. Теперь рассмотрим цикл $C = (C1 \cup C2) \setminus (C1 \cap C2)$. F — минимальное остовное дерево, а значит e_i будет ребром с самым большим весом в C_i. Получается, что e_2 — ребро с самым большим весом в цикле C. А значит, по инварианту (*), у него же будет и самый низкий уровень в C. А значит, $level(e_1) >= level(e_2)$. □

Таким образом, благодаря данному факту получится обеспечивать сохранение минимальности остовного леса при поиске ребер-замен.

Какими операциями необходимо дополнить структуру Эйлеров обход + BST для работы с весами ребер?

1. В BST каждая вершина остовного дерева может встретиться более одного раза. Происходит это из-за Эйлерового обхода. Например, как на фото ниже вершина R повторяется 3 раза.



Чтобы не хранить во всех вершинах дерева списки, среди повторяющихся вершин будем выбирать по одной называемой *репрезентативной* вершине. Таким образом, это позволит уменьшить общие расходы по памяти.

2. Минимальные остовные деревья мы храним в виде ET_i (те же структуры Эйлеров обход + BST), но с модификацией. Для таких ET_i в репрезентативных вершинах мы храним ребра, инцидентные с данной вершиной в оригинальном остовном дереве (то дерево, которое изначально поддерживаем на графе), но не принадлижащее этому дереву, так как среди этих ребер мы и будем искать ребро-замену.

3. Также для каждой вершины структуры ET_i мы храним количество инцидентных (вне дерева) с поддеревом ребер и количество репрезентативных вершин, а также указатель на соответствующую ей репрезентативную вершину.

4. Добавляем операцию $GetNonTreeEdgesSorted(v)$, которая возвращает список ребер, отсортированных по возрастанию весов. При этом данные ребра инцидентны с поддеревом с корнем v и при этом не принадлежат данному поддереву. Таким образом, получится перебирать репрезентативные вершины, добавлять их в итоговый список и выполнять слияние за $O(nlogn)$ (существует алгоритм за $O(nlogn + mlogm)$ в общем случае, где n - длина первого массива, а m - второго, в нашем случае n = m). А итоговая сложность амортизированно составит $O(logn)$ (так как делим на n для амортизированности).

Итого поиск смежных ребер остается аналогичным с алгоритмом поддержки декрементальной динамической связности неориентированного графа, где амортизированная сложность $O(log^2 n)$ на одну операцию удаления, а меняется только алгоритм выбора ребра-замены. Таким образом, амортизированная сложность остается $O(log^2n)$ на одну операцию удаления ребра.