

# Динамическая связность

Кузнецов Илья Александрович

371 группа

25.05.2022

**1(a).** Придумайте рекурсивную процедуру  $fall(v)$ , которая для вершины  $v$ , такой, что  $N_1(v) = \emptyset$ , "роняет"  $v$  на правильный уровень BFS-дерева, корректно обновляет уровни соседей  $v$  и "роняет" те вершины, чей уровень изменился при падении  $v$ .

*Решение.* Положим, что перед вызовом рекурсивной процедуры  $fall(v)$ :  $N_2(v) \neq \emptyset$ . В противном случае будет образована новая компонента связности.

---

**Algorithm 1**

---

```
1: function FALL( $v$ )
2:    $l(v) \leftarrow l(v) + 1$ 
3:   for  $u \in N_2(v)$  do
4:      $N_2(u) \leftarrow N_2(u) \setminus \{v\}$ 
5:      $N_3(u) \leftarrow N_3(u) \cup \{v\}$ 
6:   for  $u \in N_3(v)$  do
7:      $N_1(u) \leftarrow N_1(u) \setminus \{v\}$ 
8:      $N_2(u) \leftarrow N_2(u) \cup \{v\}$ 
9:    $N_1(v) \leftarrow N_2(v)$ 
10:   $N_2(v) \leftarrow N_3(v)$ 
11:   $N_3(v) \leftarrow \emptyset$  ▷ Заполняется при вызове  $fall$  от вершин-детей
12:  for  $u \in \{w \mid w \in N_2(v) \text{ and } N_1(w) = \emptyset\}$  do
13:     $fall(u)$ 
```

---

**1(b).** Докажите, что если в графе  $n$  вершин и  $m$  рёбер изначально, на все обновления суммарно при удалении  $m$  рёбер уйдёт время  $O(mn)$ .

*Доказательство.* Пусть  $deg(v)$  — степень вершины  $v$ . Обработка одной вершины внутри описанной рекурсивной процедуры при этом будет занимать  $O(deg(v))$  времени.

В случае, когда BFS-дерево вырождается в список, его высоту можно сравнить с  $n$ . Тогда наибольшее число вызовов рекурсивной процедуры  $fall$ , начатых из вершины  $v$ , тоже можно сравнить с  $n$ . Отсюда, работа процедуры  $fall$  для вершины  $v$  занимает  $O(n \deg(v))$  времени. Удаление ребра занимает  $O(1)$  времени.

Таким образом, имеем:  $O(m + \sum_{v \in V} n \deg(v)) = O(m + n \sum_{v \in V} \deg(v)) = O(m + nm) = O(mn)$ .  $\square$

**1(c).** Пусть вместо всего BFS-дерева нам разрешено хранить только BFS-дерево с  $d$  уровнями, т.е. структура будет поддерживать только расстояния до вершин  $v$ , такие, что  $d(s, v) \leq d$ . Докажите, что суммарное время на все обновления в этом случае равно  $O(md)$ .

*Доказательство.* Пусть  $deg(v)$  — степень вершины  $v$ . Обработка одной вершины внутри описанной рекурсивной процедуры при этом будет занимать  $O(deg(v))$  времени.

В случае, когда BFS-дерево вырождается в список, его высоту можно сравнить с  $d$ . Тогда наибольшее число вызовов рекурсивной процедуры  $fall$ , начатых из вершины  $v$ , тоже можно сравнить

с  $d$ . Отсюда, работа процедуры *fall* для вершины  $v$  занимает  $O(d \deg(v))$  времени. Удаление ребра занимает  $O(1)$  времени.

Таким образом, имеем:  $O(m + \sum_{v \in V} d \deg(v)) = O(m + d \sum_{v \in V} \deg(v)) = O(m + dm) = O(md)$ .  $\square$

**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.* По условию, введем для алгоритма следующий инвариант: *если ребро  $e$  является ребром максимального веса среди рёбер некоторого цикла  $C$ , то  $u$  — самый низкий уровень среди всех рёбер  $C$  [\*].*

В случае удаления ребра, принадлежащего остовному дереву, необходимо искать замену этому ребру, чтобы сохранить остовное дерево. 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  $\log n$ . Внесем изменения в данный алгоритм и будем перебирать уровни в обратном порядке: с  $\log n$  до  $i$ .

В оригинальном алгоритме порядок перебора ребер на уровне не был определен, поэтому теперь будем перебирать ребра в порядке увеличения веса. Подходящее ребро-замена должно лежать концами в обоих деревьях, образовавшихся после удаления ребра, то есть оно должно "склеить" и оставить остовное дерево целостным. Если рассматриваемое ребро не подходит, то ему присваивается уровень  $i - 1$ .

Теперь покажем, что данный алгоритм получения ребра-замены позволит поддерживать минимальный остовный лес при удалениях ребер. Для этого введем утверждение.

*Утверждение (изменено под задачу, из статьи).* Пусть выполняется инвариант [\*] и  $F$  — минимальное остовное дерево. Тогда для любого ребра  $e$  из дерева  $F$ , ребро с наименьшим весом имеет наибольший уровень среди всех кандидатов 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  $\text{level}(e_1) \geq \text{level}(e_2)$ .

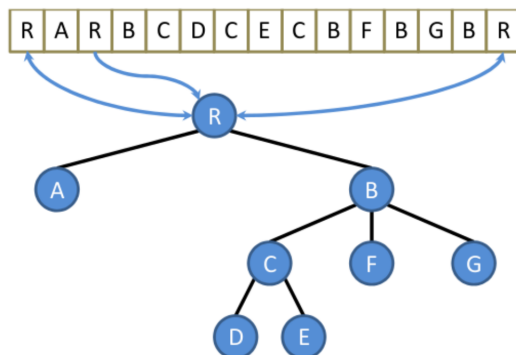
Рассмотрим цикл  $C = (C_1 \cup C_2) \setminus (C_1 \cap C_2)$ .  $F$  — минимальное остовное дерево, а значит  $e_i$  будет ребром с самым большим весом в  $C_i$ . Получается, что  $e_2$  — ребро с самым большим весом в цикле  $C$ . А значит, по инварианту [\*], у него будет и самый низкий уровень в  $C$ . А значит,  $\text{level}(e_1) \geq \text{level}(e_2)$ .  $\square$

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

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

1. В BST каждая вершина остовного дерева может встретиться более одного раза. Происходит это из-за выполнения Эйлера обхода.

Например, на рисунке ниже вершина  $R$  повторяется 3 раза.



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

2. Минимальные остовные деревья мы храним в виде  $ET_i$  (структуры Эйлеров обход + BST), но с модификацией: для  $ET_i$  в репрезентативных вершинах мы храним ребра, инцидентные с данной вершиной в оригинальном остовном дереве (то дерево, которое изначально поддерживаем на графе), но не принадлежащие 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 отсортированных по возрастанию весов ребер. При этом данные ребра инцидентны с поддеревом с корнем  $v$  и не принадлежат данному поддереву. Таким образом, получится перебирать репрезентативные вершины, добавлять их в итоговый список и выполнять слияние за  $O(n \log n)$  (существует алгоритм за  $O(n \log n + m \log m)$  в общем случае, где  $n$  — длина первого массива, а  $m$  — длина второго массива; в нашем случае  $n = m$ ). Тогда итоговая сложность амортизированно составит  $O(\log n)$  (так как делим на  $n$  для амортизации).

Таким образом, поиск смежных ребер остается таким же, как и в алгоритме поддержки декрементальной динамической связности неориентированного графа, где амортизированная сложность составляет  $O(\log^2 n)$  на одну операцию удаления. Меняется только алгоритм выбора ребра-замены, поэтому амортизированная сложность остается  $O(\log^2 n)$  на одну операцию удаления ребра.