Dynamic Connectivity Problem: Online

pakhomovee

Compiled: 25 июля 2023 г.

1 Euler Tour Tree

1.1 Структура

Мы хотим научиться решать следующую задачу за $O(n \log n)$:

Задача 1.

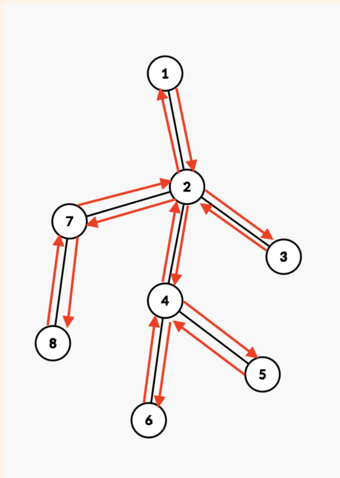
Дан лес на n вершинах. Нужно научиться обрабатывать запросы:

- Добавить ребро (u, v), u и v находятся в разных компонентах связности
- Удалить ребро (u, v)
- Сказать, лежат ли u и v в одной компоненте связности

Proposition 1.1

Рассмотрим любой эйлеров обход дерева, выпишем рёбра в порядке перехода по ним.
Сохраним ребра в декартовом дереве. Для каждой вершины сохраним $out(v)$ — указатель на ребро, которое выходит из неё, $in(v)$ — указатель на ребро, которое входит в неё.

- Проверка на связность: пусть $r(e)$ — корень ДД, в котором лежит ребро e. Проверим, что $r(in(u)) == r(in(v))$.
- Добавление ребра: введём функцию $make\_root(v)$, которая сделает вершину v корнем её дерева. Для этого циклически сдвинем ДД, чтобы $in(v)$ было первым ребром в массиве. Теперь выполним $make\_root(u), make\_root(v)$, после чего добавим в конец ДД вершины u ребро (u, v), ДД вершины v, ребро (v, u).
- Удаление ребра: для каждого ребра сохраним обратное ему ребро $back(e)$. Сделаем ребро e первым в ДД. Найдем индекс ребра $back(e)$ в ДД (для этого нужно для каждой вершине в ДД хранить её предка). Отрежем всё левее $back(e)$, удалим рёбра e и $back(e)$. Структура перестроилась корректно.



1.2 Другие задачи

Задача 2.

Дан лес на n вершинах. Нужно научиться обрабатывать запросы:

- Добавить ребро (u, v), u и v находятся в разных компонентах связности
- Удалить ребро (u, v)
- Найти XOR чисел на ребрах на пути (u, v)

Для этого найдем префиксный XOR до $in(u)$ и до $in(v)$. Тогда ответ равен XOR этих чисел.

Задача 3.

Дано дерево на n вершинах, корень постоянный. Нужно научиться обрабатывать запросы:

- Подвесить вершину u к вершине v, вершина u до этого не встречалась
- Удалить ребро (u, v), вершина v является листом
- Найти $lca(u, v)$

Если мы подвесили дерево, то у каждого ребра есть направление — «вниз» или «вверх». На ребрах «вниз» поставим число 1, а на ребрах вверх число -1. Достаточно просто вырезать отрезок от $in(u)$ до $in(v)$ и на нём найти вершину ДД с наименьшей префиксной суммой.

2 Dynamic Connectivity Problem

2.1 Постановка задачи

Задача 4.

Нужно научиться обрабатывать запросы:

- Добавить ребро (u, v) в граф
- Удалить ребро (u, v) из графа
- Проверить лежат ли u и v в одной компоненте связности

При этом граф в каждый момент может быть произвольным.

2.2 Алгоритм

Proposition 2.1

Введём веса для каждого ребра (граф невзвешенный, это веса, которые мы будем использовать в ходе алгоритма). Пусть l_e — вес ребра e, $0 \leq l \leq \lceil \log_2 n \rceil$.

Пусть F_i — максимальный остовный лес на рёбрах веса $\leq i$. Мы будем поддерживать следующий инвариант:

1. $F_{\lceil \log_2 n \rceil} \subseteq F_{\lceil \log_2 n \rceil - 1} \subseteq \ldots \subseteq F_1 \subseteq F_0$.
2. В F_i все компоненты связности имеют размер $\leq \frac{n}{2^i}$.

Каждое остовное дерево будем хранить в ETT. Все ребра веса i, которые не лежат в F_i сохраним в $g[i]$ Тогда проверить связанные ли u и v можно за одну проверку их связности в F_0.

Для того, чтобы добавить ребро (u, v) сделаем следующее:

- Если u и v не связаны в F_0, то добавим ребро (u, v) в F_0, назначив ему вес 0
- Иначе добавим ребро в $g[0][v]$ и $g[0][u]$.

Инвариант не испортился.

Для того, чтобы удалить ребро (u, v):

- Если $(u, v) \notin F_0$, то просто удалим (u, v) из списка ребер.
- Иначе пусть $level[(u, v)] = L$. Тогда $(u, v) \in F_0, F_1, \ldots, F_L$. Заметим, что, так как $F_i \subseteq F_{i+1}$, компоненты T_u и T_v, образовавшиеся при удалении в $F_0, \ldots, F_L$ ребра (u, v) не соединены ни в одном F_i. Отсюда же следуем, что ребра замены нет в E_k для $k > L$, где E_k — множество ребер веса k. Для определенности скажем $|T_u| \leq |T_v|$, $|T_u| + |T_v| \leq \frac{n}{2^L}$ по инварианту, так как до удаления они были одной компонентой. Значит, $T_u \leq \frac{n}{2^{L+1}}$. Значит, можно добавить все рёбра веса L из T_u в F_{L+1}, увеличить их вес на 1. Инвариант после такого изменения сохраняется. После этого рассмотрим все рёбра веса L, инцидентные T_u, не лежащие в T_u, которые мы еще не смотрели (то есть их вес равен текущему L). Есть 2 случая:
  1. Второй конец ребра лежит в T_v. В этом случае добавим это ребро в $F_L, F_{L-1}, \ldots, F_0$. На этом закончим выполнение *del_edge*.
  2. Иначе оба конца ребра лежат в T_u (так как иначе оно лежало бы в T_u, ведь до удаления (u, v) T_u и T_v составляли единую полную (максимальную по включению) компоненту связности). Увеличим l_e на 1 (в F_{L+1} оно соединяет вершины из одной компоненты, так как мы перенесли T_u в F_{L+1}). Если на уровне L мы не нашли замену, то перейдем на уровень $L - 1$ и поищем её там. Если замены нет и в F_0, то замены просто нет.

2.3 Реализация

Чтобы не умереть во время реализации алгоритма:

1. Не забываем $g[level][vertex]$ для рёбер не из F_{level}, смежных с $vertex$
2. Для каждой вершины остовного леса храним флаг: есть ли ребро из неё, не лежащее в лесу. Для этого

фиксируем ребро-представитель из ETT для каждой вершины. Модно, например, сделать петлю.

2.4 Асимптотика

По алгоритму видно, что вес ребра никогда не превосходит $\lceil \log_2 n \rceil$, а каждое обращение к ребру меняет его вес на 1, поэтому всего будет $O(m \log_2 n)$ изменений, каждое из которых обращается к ETT, откуда итоговая асимптотика получается равной $O(m \log_2^2 n)$.

3 Decremental Minimum Spanning Forests

Мы хотим научиться решать следующую задачу:

Задача 5.

Дан взвешенный граф на n вершинах. Нужно научиться обрабатывать следующие запросы:

- Удалить ребро (u, v)
- Найти вес минимального остовного дерева компоненты связности, в которой лежит вершина u

Proposition 3.1

Обозначим за $w(e)$ вес ребра в графе, за $l(e)$ уровень ребра из предыдущего пункта.

Предыдущий алгоритм несложно поменять для решения этой задачи:

1. Изначально F_0 — минимальный остовный лес графа
2. При просмотре рёбер, которые инцидентны T_u, но не лежат в T_u, будем идти в порядке возрастания их весов ($w(e)$) (мы всё еще выбираем их по одному и завершаемся, если нашли замену).

Theorem 3.2

Этих изменений достаточно для решения DMSF.

Доказательство. К нашему инварианту из предыдущего пункта добавим также утверждение: любой цикл C графа имеет ребро не из F_i, что $w(e) = \max_{f \in C} w(f)$ и $l(e) = \min_{f \in C} l(f)$
Докажем, что тогда мы будем находить ребро-замену минимального возможного веса.

Lemma 3.3 Пусть F — минимальный остовный лес и выполняется наш инвариант. Тогда для любого ребра e из леса самое легкое его ребро-замена лежит на максимальном уровне.

Доказательство. Пусть e_1 и e_2 — ребра замены для e. Обозначим за C_i цикл, который возникнет, если добавить в дерево ребро e_i. $e_i \in C_i$. Пусть $w(e_1) < w(e_2)$. Достаточно показать $l(e_1) \geq l(e_2)$.
Рассмотрим цикл $C = (C_1 \cup C_2) \backslash (C_1 \cap C_2)$. По нашему инварианту e_i — наиболее тяжелое ребро не из F на C_i. Значит, e_2 — уникальное самое тяжелое ребро на C не из F. То есть e_2 имеет наименьшее $l(e)$ на C. В частности, $l(e_1) \geq l(e_2)$.
😎

Очевидно, что наш алгоритм сохраняет инварианты (i) и (ii) из DCP.
Докажем, что он также сохраняет и новый инвариант (iii).

Lemma 3.4 Алгоритм сохраняет: любой цикл C графа имеет ребро не из F_i, что $w(e) = \max_{f \in C} w(f)$ и $l(e) = \min_{f \in C} l(f)$

Доказательство. Изначально (iii) выполнено, так как все ребра на уровне 0. При простом удалении ребра всё сохраняется: новое множество циклов будет подмножеством старого множества.
Нужно доказать, что во время поиска замены удаленному ребру мы ничего не портим, когда увеличиваем уровень просмотренного ребра / добавляем его в остовный лес. Заметим, что можно смотреть только на ребра, являющие самыми низкими по уровню и в то же время самыми тяжелыми по весу в своём цикле, так как иначе инвариант нарушиться не может. Докажем, что:

1. все рёбра из C смежные с T_u имеют уровень $> l(e)$
2. используя (1) покажем, что C не сможет покинуть T_v

Из (2) станет ясно, что e не будет ребром-заменой. Из (1) и (2) мы поймем, что e имеет уровень строго меньший уровней всех ребер в C, поэтому e не сломает ничего, если повысит свой уровень.
Итак, докажем (1): Пусть $i = l(e)$. Пусть мы сейчас будем смотреть на e в поиске замены. Тогда все рёбра из T_u имеют уровень $> i$ (мы их до этого повысили). Также любое ребро инцидентное T_u, которое легче e имеет уровень $> i$, так как мы посмотрели его раньше. Более того, так как e — единственное самое «низкое» ребро на C ребро наибольшего веса не из леса, любое ребро такого же веса будет иметь строго больший уровень. ч.т.д;
Докажем (2): пусть C покинет T_v. Тогда есть хотя бы 2 ребра в C, которые покидают T_v, одно из которых не e. Назовем это ребро f. Если $l(f) \geq i$, то f является кандидатом на замену, но по (1) нет ребра-замены на уровне $> i$, то есть $l(f) \leq i$, значит $l(f) = i$, что противоречит (1).
Значит, (iii) не нарушается. 😎

Инварианты (i), (ii) и (iii) сохраняются, поэтому, если начать с минимального остовного леса F, по 3.3 при удалении ребра, оно должно быть заменено самым лёгким вариантом замены в установленном порядке перебора, поэтому остовный лес останется корректным. 😎

Итоговая реализация алгоритма работает за $O(m \log_2^2 n)$