Студент: Милосердова Любовь Михайловна
Группа: 371 (SE)
Дата: 15 мая 2022 г.

Динамическая связность

Задача 1. *Задача динамического декрементального SSSP (single source shortest paths) на неориентированном невзвешенном графе. На вход дан граф $G = (V, E)$ и его фиксированная вершина-источник s. Нужно поддерживать запросы вида: дана вершина v, каково расстояние $d(s, v)$? Так как алгоритм декрементальный, рёбра только удаляется.*

Для начала мы препроцессим граф следующим образом: посчитаем BFS-дерево с корнем в s(просто запустим BFS из вершины s, и выпишем получившееся дерево). Каждой вершине v получившегося дерева присвоим уровень $l(v)$, значение которого есть расстояние от вершины s ($d(s, v)$). Очевидно, что $l(s) = 0$. С этим деревом будем работать как со структурой данных.

Также BFS посчитает для каждой вершины посчитает нам три множества её соседей N_1, N_2, N_3. Пусть $l(v) = i$, тогда $N_1(v)$ — соседи v, имеющие уровень $i - 1$; N_2 — соседи v с уровнем i; N_3 — соседи v с уровнем $i + 1$.

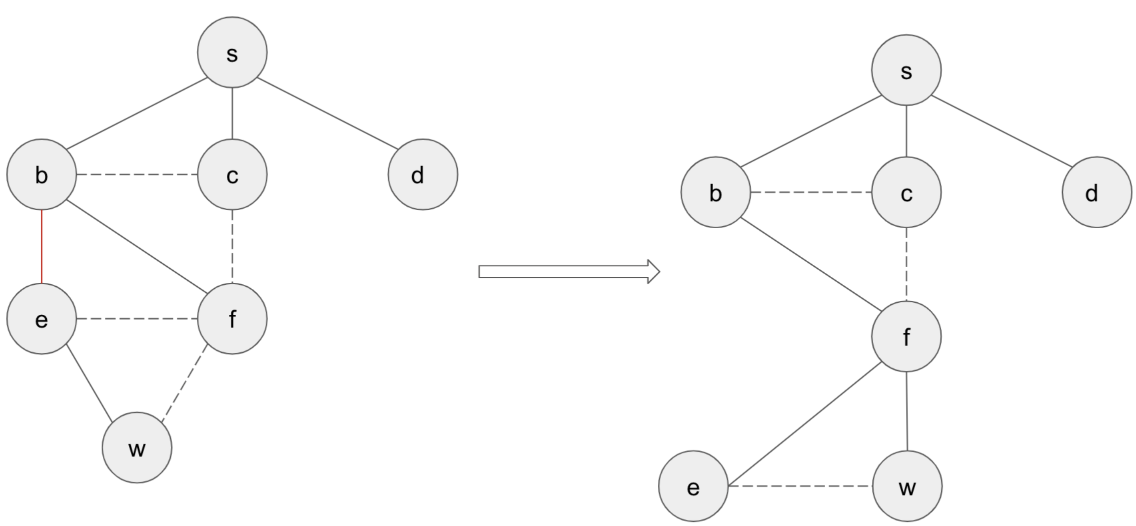
Наш алгоритм должен поддерживать удаления с помощью обновлений множеств N_1, N_2, N_3 для некоторых вершин и изменений их уровня в BFS-дереве. На запрос у нас уходит константное время — достаточно спросить у вершины её уровень.

Доказательство. Рассмотрим, что происходит, если мы удаляем из графа ребро (u, v).

Если $l(u) = l(v)$ (вершины на одном уровне), то удаление данного ребра не меняет расстояния от s, значит нужно просто удалить v из $N_2(u)$ и u из $N_2(v)$.

Пусть $l(v) = i$ и $l(u) = i-1$ (другой случай работает симметрично, так как граф неориентированный). Нам нужно удалить u из $N_1(v)$ и v из $N_3(u)$. Если во множестве $N_1(v)$ остались вершины, то расстояния не изменились, так как мы можем "привязать"вершину v к любой вершине из множества $N_1(v)$ в нашем дереве. Если же $N_1(v)$ стало пустым, то v должно “провалиться” вниз на новый уровень. ~~И более того, если v провалилась, то все вершины w, имеющие v в своем множестве $N_1(w)$ должны провалиться, и так далее~~! И более того, если v провалилась, то все вершины w, для которых $N_1(w) = \{v\}$ должны провалиться, и так далее!

Контр–пример к зачернутому утверждению:



Псевдокод процедуры $fall(v)$, которая для вершины v, такой, что $N_1(v) = \varnothing$, “роняет” v на правильный уровень BFS-дерева, корректно обновляет уровни соседей v и “роняет” те вершины, чей уровень изменился при падении v.

Algorithm 1 Обновление множеств N_1, N_2, N_3, пересчет уровней в BFS-дереве

```
procedure FALL(vertex)
    if N_1(vertex) ≠ ∅ then
        return
    q ← []
    q.add(vertex)
    while q ≠ ∅ do
        v ← q.get_first()
        level[v] = level[v] + 1
        for w : w ∈ N_3(v) do
            N_1(w).remove(v)
            if N_1(w) = ∅ then
                q.add(w)
            N_2(w).add(v)
        for w : w ∈ N_2(v) do
            N_2(w).remove(v)
            N_3(w).add(v)
        N_1(v) ← N_2(v)
        N_2(v) ← N_3(v)
        N_3(v) ← ∅
        if N_1(v) = ∅ then
            q.add(v)
```

Процедура получилась не рекурсивной, так как больше напоминает bfs, чем dfs, поскольку необходимо хранить кратчайшие пути, а для этого необходимо поддерживать корректность расстояний от верхнего уровня к нижнему.

Пояснение к алгоритму: будем повышать уровень вершины v, после чего необходимо правильно пересчитать множества. "Дети"вершины v становятся с ней на одном уровне, а для вершин, с которыми v была на одном уровне, v теперь становится ребенком. Если для какой-то из вершин не осталось предков, значит ее тоже необходимо обработать в очереди.

Для оценки сложности алгоритма будем считать $level(u, v)$ равным $max(level(u), level(v))$. При обработке вершины v мы рассматриваем все ребра, инциндентные v, и перекладываем их из одного множества в другое так, что один из концов ребра увеличивает свой уровень. Так как $1 \leq level(u) \leq n$ и $level(u)$ в процессе алгоритма может только увеличиваться, то $level(u, v)$ увеличится максимум $O(n)$ раз. Итого, так как всего у нас m ребер, суммарная сложность алгоритма на все апдейты — $O(mn)$.

При хранении BFS-дерева лишь с d уровнями, т.е. структура будет поддерживать только расстояния до вершин v, такие, что $d(s, v) \leq d$, уровень ребра повысится лишь $O(d)$ раз, а затем ребро будет удалено. Получаем, что суммарная сложность алгоритма на все апдейты — $O(md)$. $\square$