Task4 2022

m.pelogeiko

May 2022

1 Условие задачи

По умолчанию число вершин во входном графе равно n, а число рёбер — m.Давайте научимся решать задачу динамического декрементального SSSP (single source shortest paths) на неориентированном невзвешенном графе, используя идею с уровнями немного в другом ключе.
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 (без вставок).
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 ($d(s, v)$) (см. картинку). 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 $N1$, $N2$, $N3$. Пусть $l(v) = i$, тогда $N1(v)$ — соседи v, имеющие уровень $i\ \ 1$; $N2$ — соседи v с уровнем i; $N3$ — соседи v с уровнем $i + 1$.
Наш алгоритм должен поддерживать удаления с помощью обновлений множеств $N1$, $N2$, $N3$ для некоторых вершин и изменений их уровня в BFS-дереве. На запрос у нас уходит константное время — достаточно спросить у вершины её уровень. 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 $N2(u)$ и u из $N2(v)$. Пусть $l(v) = i$ и $l(u) = i\ \ 1$ (другой случай работает симметрично). Нам нужно удалить u из $N1(v)$ и v из $N3(u)$. Если во множестве $N1(v)$ остались вершины, то расстояния не изменились (подумайте, почему). Если же $N1(v)$ стало пустым, то v должна"провалиться" вниз на новый уровень. И более того, если v провалилась, то все вершины w, для которых $N1(w) = v$ должны провалиться, и так далее!

(a) Придумайте рекурсивную процедуру f all(v), которая для вершины v, такой, что $N1(v) =$, "роняет" v на правильный уровень BFS-дерева, корректно обновляет уровни соседей v и "роняет" те вершины, чей уровень

изменился при падении v.
(b) Докажите, что если в графе n вершин и m рёбер изначально, на все обновления суммарно при удаления m рёбер уйдет время O(mn).
(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) d. Докажите, что суммарное время на все апдейты в этом случае равно O(md).

2 Решение A

```python
def fall(vert):
  level[vert] = level[vert] + 1

  for v in N2(vert):
    N2(v).remove(vert)
    N3(v).add(vert)

  temp_set = set()

  for v in N3(vert):
    N1(v).remove(vert)

    if N1(v) == set():
      N1(v).add(vert)
      temp_set.add(v)
      N1(vert).remove(v)
      fall(v)
      continue
    else:
      N2(v).add(vert)

  N1(vert) = N1(vert) + N2(vert)
  N2(vert) = N3(vert)
  N3(vert) = temp_set

  if N1(vert) == set():
    if N2(vert) == set() and N3(vert) == set():
      vertex_miss_callback(vert)
    else:
      fall(vert)
```

Пояснение к алгоритму: повысим уровень вершины vert, после чего необходимо правильно пересчитаем множества. Вершины N2(vert) т.е. находящиеся на том же уровне, становятся родителями, т.е. на уровень выше. Вершины из N3(vert) т.е. находящиеся уровнем ниже, становятся на одном уровне с vert. Заметим, что для вершин v из N3(vert) есть 2 случая:
1) N1(v) $\setminus\{$vert$\} = \emptyset$

В этом случае необходимо увеличить уровень вершины v, запустив функцию fall(v).
2) N1(v) $\setminus\{$vert$\} \neq \emptyset$

В данном случае уровень вершины в сохраняется за счет оставшихся элементов в множестве N1(v) и надо просто переместить vert из N1(v) в N2(v).

Далее если в конце алгоритма N1(vert) все еще пусто, надо еще раз совершить понижение

3 Решение B

У нас в графе m ребер и n вершин. При чем для каждой вершины
$0 <= l(vertex) < n$ и в процессе работы алгоритма для каждой вершины l(vertex) не уменьшается. Данный алгоритм может обработать одну вершину максимум 3 раза и в самом негативном случае надо обработать n-1 вершину т.е. O(3n-3) = O(n). Если нам надо удалить m ребер,
то мы имеем m * O(n) = O(mn), ч.т.д.

4 Решение C

Мы храним BFS-дерево лишь с d уровнями, т.е. структура будет поддерживать только расстояния до вершин v, такие, что d(s, v) <= d. В таком случае уровень максимальной глубины = d и мы каждую вершину будем обрабатывать за O(d) (если уровень повысится больше d, то мы остановим обработку и удалим ребро, т.к. мы не можем хранить больше, чем d уровней). А суммарная сложность удаления m ребер = O(md) ч.т.д.