Алгоритм Левита

Симонов Александр

Матмех СПбГУ 18.Б13-мм

4 декабря 2019

План

Алгоритм Левита

- Находит во взвешенном графе (ребра могут быть отрицательными) без отрицательных циклов кратчайшие расстояния от заданной вершины до всех остальных

Алгоритм Левита

- Находит во взвешенном графе (ребра могут быть отрицательными) без отрицательных циклов кратчайшие расстояния от заданной вершины до всех остальных
- Обозначим:
  s - начальная вершина (от которой ищем расстояние)
  w(ij) - Вес ребра ij
  D[i] - текущая кратчайшая длина пути из s в i
  (изначально $D[i] = \infty$ при $i \neq s$, $D[s] = 0$)

Алгоритм Левита

- Находит во взвешенном графе (ребра могут быть отрицательными) без отрицательных циклов кратчайшие расстояния от заданной вершины до всех остальных
- Обозначим:
  s - начальная вершина (от которой ищем расстояние)
  w(ij) - Вес ребра ij
  D[i] - текущая кратчайшая длина пути из s в i
  (изначально D[i] $= \infty$ при i $\neq$ s, D[s] $=$ 0)
- M_0 — вершины, расстояние до которых уже вычислено (но, возможно, не окончательно)
  M_1 — вершины, расстояние до которых вычисляется (реализуем в виде дека)
  M_2 — вершины, расстояние до которых ещё не вычислено

Алгоритм Левита

- Изначально все вершины помещаются в множество M_2, кроме вершины s, которая помещается в множество M_1

Алгоритм Левита

- Изначально все вершины помещаются в множество M_2, кроме вершины s, которая помещается в множество M_1
- На каждом шаге алгоритма мы берём вершину достаём верхний элемент из M_1 Пусть v — это выбранная вершина. Переводим эту вершину во множество M_0. Затем просматриваем все рёбра, выходящие из этой вершины. Пусть t — это второй конец текущего ребра (то есть не равный v), а w(vt) — это длина текущего ребра. Тогда имеем 3 варианта:

Алгоритм Левита

- Изначально все вершины помещаются в множество M_2, кроме вершины s, которая помещается в множество M_1
- На каждом шаге алгоритма мы берём вершину достаём верхний элемент из M_1 Пусть v — это выбранная вершина. Переводим эту вершину во множество M_0. Затем просматриваем все рёбра, выходящие из этой вершины. Пусть t — это второй конец текущего ребра (то есть не равный v), а w(vt) — это длина текущего ребра. Тогда имеем 3 варианта:
  1) Если $t \in M_2$, то переносим t во множество M_1 в конец очереди. D[t] делаем равным D[v] + w(vt)

Алгоритм Левита

- Изначально все вершины помещаются в множество M_2, кроме вершины s, которая помещается в множество M_1
- На каждом шаге алгоритма мы берём вершину достаём верхний элемент из M_1 Пусть v — это выбранная вершина. Переводим эту вершину во множество M_0. Затем просматриваем все рёбра, выходящие из этой вершины. Пусть t — это второй конец текущего ребра (то есть не равный v), а w(vt) — это длина текущего ребра. Тогда имеем 3 варианта:

  1) Если $t \in M_2$, то переносим t во множество M_1 в конец очереди. D[t] делаем равным D[v] + w(vt)

  2) Если $t \in M_1$, то пытаемся улучшить значение D[t]: D[t] = min (D[t], D[v] + w(vt)). Сама вершина t никак не передвигается в очереди

Алгоритм Левита

- Изначально все вершины помещаются в множество M_2, кроме вершины s, которая помещается в множество M_1
- На каждом шаге алгоритма мы берём вершину достаём верхний элемент из M_1 Пусть v — это выбранная вершина. Переводим эту вершину во множество M_0. Затем просматриваем все рёбра, выходящие из этой вершины. Пусть t — это второй конец текущего ребра (то есть не равный v), а w(vt) — это длина текущего ребра. Тогда имеем 3 варианта:
  1) Если $t \in M_2$, то переносим t во множество M_1 в конец очереди. D[t] делаем равным D[v] + w(vt)
  2) Если $t \in M_1$, то пытаемся улучшить значение D[t]: D[t] = min (D[t], D[v] + w(vt)). Сама вершина t никак не передвигается в очереди
  3) Если $t \in M_0$, и если D[t] можно улучшить (D[t] > D[v] + w(vt)), то улучшаем D[t], а вершину t возвращаем в множество M_1, помещая её в начало очереди

Алгоритм Левита

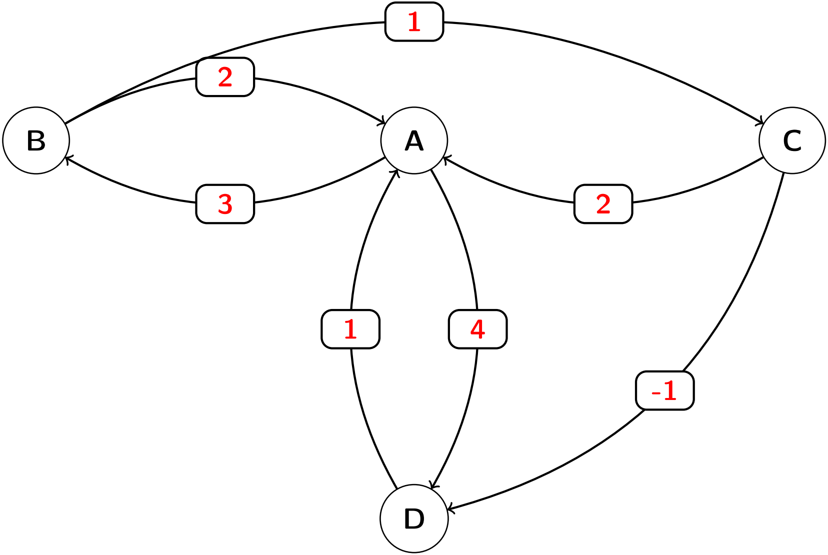
- Изначально все вершины помещаются в множество M_2, кроме вершины s, которая помещается в множество M_1
- На каждом шаге алгоритма мы берём вершину достаём верхний элемент из M_1 Пусть v — это выбранная вершина. Переводим эту вершину во множество M_0. Затем просматриваем все рёбра, выходящие из этой вершины. Пусть t — это второй конец текущего ребра (то есть не равный v), а w(vt) — это длина текущего ребра. Тогда имеем 3 варианта:

  1) Если $t \in M_2$, то переносим t во множество M_1 в конец очереди. D[t] делаем равным D[v] + w(vt)

  2) Если $t \in M_1$, то пытаемся улучшить значение D[t]: D[t] = min (D[t], D[v] + w(vt)). Сама вершина t никак не передвигается в очереди

  3) Если $t \in M_0$, и если D[t] можно улучшить (D[t] > D[v] + w(vt)), то улучшаем D[t], а вершину t возвращаем в множество M_1, помещая её в начало очереди

Рассмотрим граф



Проверка

- В каких множествах M_i и сколько раз окажется вершина s (начальная вершина) во время действия алгоритма?

Проверка

- В каких множествах M_i и сколько раз окажется вершина s (начальная вершина) во время действия алгоритма?
  -Изначально s находится в M_1, первого шага перемещается в M_0 и находится там до конца

Проверка

- В каких множествах M_i и сколько раз окажется вершина s (начальная вершина) во время действия алгоритма?
  -Изначально s находится в M_1, первого шага перемещается в M_0 и находится там до конца
- Что будет если какая-то вершина t не достижима из s?

Проверка

- В каких множествах M_i и сколько раз окажется вершина s (начальная вершина) во время действия алгоритма?
  -Изначально s находится в M_1, первого шага перемещается в M_0 и находится там до конца
- Что будет если какая-то вершина t не достижима из s?
  -Вершина t будет находиться все время в M_2 с расстоянием равным ∞

Проверка

- В каких множествах M_i и сколько раз окажется вершина s (начальная вершина) во время действия алгоритма?
  -Изначально s находится в M_1, первого шага перемещается в M_0 и находится там до конца
- Что будет если какая-то вершина t не достижима из s?
  -Вершина t будет находиться все время в M_2 с расстоянием равным ∞
- Что делать, если помимо расстояния хотим узнать сам путь?

Проверка

- В каких множествах M_i и сколько раз окажется вершина s (начальная вершина) во время действия алгоритма?
  -Изначально s находится в M_1, первого шага перемещается в M_0 и находится там до конца
- Что будет если какая-то вершина t не достижима из s?
  -Вершина t будет находиться все время в M_2 с расстоянием равным ∞
- Что делать, если помимо расстояния хотим узнать сам путь?
  -Создадим массив P[1...V(G)], содержащий в P[i] вершину, предшествующую вершине i в кратчайшем пути. (Значение P[i] обновляется на v при успешной релаксации ребра vi)

Доказательство корректности алгоритма

Лемма 1

Алгоритм работает за конечное время

Доказательство корректности алгоритма

Лемма 1

Алгоритм работает за конечное время

$\lhd$ Алгоритм завершит работу, когда все вершины окажутся в M_0 и в M_2. Так как в исходном графе нет отрицательных циклов, то для каждой вершины либо нет пути, либо существует кратчайший путь. Тогда расстояние до каждой вершины может уменьшится только конечное число раз и, как следствие, вершина будет переведена из M_0 в M_1 тоже конечное число раз. С другой стороны, на каждом шаге текущая вершина гарантированно помещается в M_0. Тогда за конечное число шагов все вершины окажутся в M_0 и в M_2. $\rhd$

Доказательство корректности алгоритма

Лемма 2

После выполнения алгоритма ни одна релаксация ребра не будет успешной

Доказательство корректности алгоритма

Лемма 2

После выполнения алгоритма ни одна релаксация ребра не будет успешной

◁ Пойдем от противного. Пусть релаксация ребра uv успешна. Тогда имеем 2 варианта:

1. Вершина u попала в M_0 позже v. Тогда релаксация ребра uv была неуспешной. Значит, такого варианта не может быть
2. Вершина u попала в M_0 раньше v. Заметим, что с момента последнего попадания u в M_0 расстояние до нее не менялось (иначе, вершина была бы извлечена из M_0). Вес ребра uv тоже не меняется. Значит, и релаксация ребра uv будет неуспешной ▷

Улучщение реализации

Заменим дек M_1 на две очереди:

- M_1' - основная очередь. Помещаем в нее вершины из M_2
- M''_1 - срочная очередь. Помещаем в нее вершины $v \in M_0$, в случае удачной релаксации какого-то ребра uv

Алгоритм левита

Data: s $\in$ V(G) - нач. вершина, w(ij) - веса ребер, D[i] $\leftarrow \infty$, D[s] $\leftarrow$ 0
Result: D[i] - массив длин кратчайшийх si-путей

```
while M'_1 ≠ ∅ and M''_1 ≠ ∅ do
    u ← (M''_1 ≠ ∅ ? M'_1.pop() : M''_1.pop())
    for v : uv ∈ E(G) do
        if v ∈ M_2 then
            M'1.push(v); M2.remove(v); D[v] = D[u] + w(uv);
        end
        else if v ∈ M'_1 or v ∈ M''_1 then
            D[v]= min(D[v], D[u] + w(uv))
        end
        else if v ∈ M_0 and D[v] > D[u] + w(uv) then
            M''_1.push(v); M_0.remove(v); D[v] = D[u] + w(uv);
        end
        M_0.add(u)
    end
end
```

Сложность

Рассмотрим взвешенный граф на клике K_n с:

- w(1, n) = 0
- w(1, i) = w(1, i+1) + i - 1
- 1 < i < j <= n w(j, i) = j - i - 1

Чему равна сложность?

Сложность

Рассмотрим взвешенный граф на клике K_n с:

- w(1, n) = 0
- w(1, i) = w(1, i+1) + i - 1
- $1 < i < j <= n$ w(j, i) = j - i - 1

Чему равна сложность? Сложность O(n^3)

Алгоритм Дейкстры

- $D[i] = \infty$ при $i \neq s$, $\quad D[s] = 0$
- U - вершины, для которых уже вычислены длины кратчайших путей из s (Изначально $U = \{s\}$)
- На каждой итерации основного цикла выбирается вершина $u \notin U$, которой на текущий момент соответствует минимальная оценка кратчайшего пути. Вершина u добавляется в множество U и производится релаксация всех исходящих из неё рёбер

Алгоритм Беллмана-Форда

- Пусть d[k][u] — количество путей длины k рёбер, заканчивающихся в вершине u. Тогда d[k][u] $= \sum_{v:vu \in E(G)} d[k-1][v]$
- Аналогично посчитаем пути кратчайшей длины. Пусть s — стартовая вершина. Тогда d[k][u] $= min_{v:vu \in E(G)}$ d[k$-$1][v] $+$ w(u,v), при этом d[0][s]=0, а d[0][u]=∞