Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент:	М. А. Волков
Преподаватель:	А. А. Кухтичев
Группа:	М8О-207Б-19
Дата:	20 мая 2021 г.
Оценка:	
Подпись:	

Москва, 2021

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

Задан взвешенный ориентированный граф, состоящий из **n** вершин и **m** ребер. Вершины пронумерованы целыми числами от **1 до n**. Необходимо найти длины кратчайших путей между всеми парами вершин при помощи алгоритма Джонсона. Длина пути равна сумме весов ребер на этом пути. Обратите внимание, что в данном варианте веса ребер могут быть отрицательными, поскольку алгоритм умеет с ними работать. Граф не содержит петель и кратных ребер.

1 Описание

Требуется вывести кротчайшие пути для всех пар вершин графа, применяя алгоритм Джонсона.

Если верить статье [1], то данный алгоритм использует все преимущества алгоритма Дейкстры, но при этом использует некоторую модификацию, которая может работать с отрицательными весами графа.

Данная модификация использует, так называемый, метод **изменения веса** (reweighting) [2]. Данную переразвесовку можно делать различными способами: либо при помощи алгоритма Флойда-Уоршела, либо при помощи алгоритма Форда-Беллмана.

Выбор, очевидно, пал на второй алгоритм, так как он ассимптотически быстрее работает, что уменьшает ассимптотику всего алгоритма.

Мой алгоритм имеет сложность $O(NM\,logN)$, так как алгоритм Дейстры реализован через очередь с приоритетом.

2 Исходный код

Данный код демонстрирует 2 функции:

1. Алгоритм Дейкстры

2. Алгоритм Форда-Беллмана

Далее в фунции *main* происходит соединение данных алгоритмов.

```
#include <queue>
#include <iostream>
#include <vector>


const long long INF = 92233720368547758;

struct TEdge {
  long long to;
  long long w;
};

struct TEdgeF {
  long long from, to;
  long long w;
};

using graph = std::vector<std::vector<TEdge>>;

struct TPriority {
  long long dist;
  long long vertex;
};

bool operator<(const TPriority &lhs, const TPriority &rhs) {
  return lhs.dist > rhs.dist;
}

std::vector<long long> Deykstra(const int &n, const graph &gr, const int &start) {
  std::vector<long long> distanses(n, INF);
  std::vector<short> relaxed(n, 0);
  distanses[start] = 0;

  std::priority_queue<TPriority> pq;
  pq.push({0, start});

  while (!pq.empty()) {
    TPriority u = pq.top();
    pq.pop();
```

```cpp

    if (relaxed[u.vertex]) {
      continue;
    }
    relaxed[u.vertex] = 1;

    for (const TEdge &e : gr[u.vertex]) {
      if (distanses[e.to] > distanses[u.vertex] + e.w) {
        distanses[e.to] = distanses[u.vertex] + e.w;
        pq.push({distanses[e.to], e.to});
      }
    }
  }

  return distanses;
}


bool FordBellma(const long long &n, const long long &m, const graph &gr,
                std::vector<long long> &distanses) {//std::vector<long long> distanses(n+1,INF);
  std::vector<TEdgeF> grr(n);
  for (int i = 0; i < n; ++i) {
    grr[i].from = 0;
    grr[i].to = i + 1;
    grr[i].w = 0;
  }

  for (int i = 0; i < gr.size(); ++i) {
    for (const TEdge &edge : gr[i]) {
      grr.push_back({i + 1, edge.to + 1, edge.w});
    }
  }

  distanses[0] = 0;
  bool changed = false;

  for (int i = 0; i < grr.size(); ++i) {
    changed = false;
    for (const TEdgeF &e : grr) {
      if (distanses[e.to] > distanses[e.from] + e.w) {
        changed = true;
        distanses[e.to] = distanses[e.from] + e.w;
      }
    }
    if (!changed) {
      break;
    }
  }
```

```

  if (changed) {
    return false;
  }

  return true;
}


int main() {
  int n, m;
  std::cin >> n >> m;
  graph gr(n);

  for (int i = 0; i < m; ++i) {
    long long from, to;
    long long w;
    std::cin >> from >> to >> w;
    gr[from - 1].push_back({to - 1, w});
  }

  std::vector<long long> sigma(n + 1, INF);

  if (!FordBellma(n, m, gr, sigma)) {
    std::cout << "Negative cycle" << std::endl;
    return 0;
  }

  for (int i = 0; i < n; ++i) {
    for (TEdge &e : gr[i]) {
      e.w = e.w + sigma[i + 1] - sigma[e.to + 1];
    }
  }

  for (int i = 0; i < n; ++i) {
    std::vector<long long> res = Deykstra(n, gr, i);
    for (int j = 0; j < res.size(); ++j) {
      if (res[j] == INF) {
        std::cout << "inf ";
      }
      else {
        std::cout << res[j] - sigma[i + 1] + sigma[j + 1] << " ";
      }
    }
    std::cout << std::endl;
  }

  return 0;
}
```

3 Тест производительности

Сравнивание будет происходить с алгоритмом Форда-Беллмана, так как данный алгоритм присутствует в алгоритме Джонсона. Очень интересно сравнить их.

Тесты будут для одинаковых весов, так как они никак не влияют на ассимптотику. Главное, чтобы там не было отрицательных циклов.

Тесты:

1. $N = 10^4; M = 500$
2. $N = 2 * 10^4; M = 4000$

Джонсона:

```
1. Johnson 0.5 ms
2. Johnson 0.668 ms
```

Форда-Беллмана:

```
1. Ford-Bellman 0.34 ms
2. Ford-Bellman 0.358 ms
```

Видно, что ограничения задачи не позволяют полностью раскрыться алгоритму. Именно поэтому простенький алгоритм, работающий за $O(N * M)$ работает быстрее.

4 Выводы

В результате выполнения лабораторной работы я изучил несколько алгоритмов, использующиеся в поиске кратчайших путей в графе. Также узнал что такое граф и где они используются в повседневной жизни.

Список литературы

[1] *Алгоритм Джонсона на орграфе с отрицательными дугами – habr.com*
URL: https://habr.com/ru/company/otus/blog/510942/
(дата обращения: 20 мая 2021 г.)

[2] *Алгоритм Джонсона – ifmo.ru*
URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Джонсона
(дата обращения: 20 мая 2021 г.)