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:	А. А. Терво
Преподаватель:	А. А. Кухтичев
Группа:	М8О-207Б
Дата:	
Оценка:	
Подпись:	

Москва, 2022

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 длину кратчайшего пути из вершины с номером $start$ в вершину с номером $finish$ при помощи алгоритма Беллмана-Форда. 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, кратных ребер и циклов отрицательного веса.

Формат входных данных

В первой строке заданы $1 \leq n \leq 10^5$, $1 \leq m \leq 3 * 10^5$, $1 \leq start \leq n$ и $1 \leq finish \leq n$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от -10^9 до 10^9.

Формат результата

Необходимо вывести одно число – длину кратчайшего пути между указанными вершинами. Если пути между указанными вершинами не существует, следует вывести строку "No solution"(без кавычек).

1 Описание

Требуется написать реализацию алгоритма Беллмана–Форда. Это алгоритм поиска кратчайшего пути во взвешенном графе от одной вершины графа до всех остальных, работающий за время $O(|V|*|E|)$, где V – количество вершин графа, а E – количество рёбер.

Идея состоит в том, чтобы использовать принцип динамического программирования, а именно мемоизацию, то есть сохранение результатов вычислений для последующего их применения. За один шаг будем вычислять кратчайший путь из $start$ в любую другую точку, содержащий не более i рёбер. На каждом шаге будем добавлять к каждому пути по одному ребру и смотреть, короче ли он, чем уже существующий. Если да, то заменять путь более коротким. Таким образом, через $|V| - 1$ итераций мы гарантировано получим кратчайшие пути из вершины $start$ во все остальные.

2 Исходный код

Создадим вспомогательную структуру $TEdge$, в ней будут храниться данные о ребре: точка, в которую оно приходит и его вес.

Также я определелил константу для того, чтобы понимать, когда пути не существует, она будет равна максимально возможному числу типа `int`.

Этапы работы программы следующие:

1. Вводим с клавиатуры параметры
2. Создаём вектор рёбер и заполняем его в цикле
3. Далее следует сам алгоритм
4. Выводим результат

```
#include <iostream>
#include <vector>
#include <limits>

const int IMAX = std::numeric_limits<int>::max();

struct TEdge {
  int r;
  int weight;
  TEdge(int vertex, int w) {
    r = vertex;
    weight = w;
  }
  ~TEdge() {};
};

int main() {
  int n, m, start, finish;
  std::cin >> n >> m >> start >> finish;
  --start;
  --finish;

  std::vector<std::vector<TEdge>> vertices(n);

  int l, r, w;
  for (int i = 0; i < m; ++i) {
    std::cin >> l >> r >> w;
    vertices[l - 1].push_back(TEdge(r - 1, w));
  }

  //Bellman-Ford algorithm
```

```
  std::vector<int> lengths(n, IMAX);
  lengths[start] = 0;
  for (int j = 0; j < n; ++j) {
    for (int i = 0; i < n; ++i) {
      for (TEdge edge : vertices[i]) {
        if (lengths[i] + edge.weight < lengths[edge.r]) {
          lengths[edge.r] = lengths[i] + edge.weight;
        }
      }
    }
  }

  if (lengths[finish] == IMAX) {
    std::cout << "No solution\n";
  } else {
    std::cout << lengths[finish] << std::endl;
  }

  return 0;
}
```

3 Консоль

```
alext@alext-pc solution]$ ./solution
5 6 1 5
1 2 2
1 3 0
3 2 -1
2 4 1
3 4 4
4 5 5
5
[alext@alext-pc solution]$ ./solution
3 4 1 2
1 2 2
1 3 0
3 2 -1
2 1 1
-1
```

4 Тест производительности

Я буду сравнивать алгоритм Беллмана–Форда с алгоритмом Флойда–Уоршелла.

Тесты представляют собой графы с количеством вершин и рёбер 30 и 600 соответственно в первом случае и 100 и 8000 во втором случае.

```
[alext@alext-pc solution]$ ./a.out >test.txt
30 20
[alext@alext-pc solution]$ ./warshall <test.txt
9
Floyd-Warshall time: 149us
[alext@alext-pc solution]$ ./solution <test.txt
Bellman-Ford time: 25us
9
[alext@alext-pc solution]$ ./a.out >test.txt
100 80
[alext@alext-pc solution]$ ./warshall <test.txt
2
Floyd-Warshall time: 4400us
[alext@alext-pc solution]$ ./solution <test.txt
Bellman-Ford time: 269us
2
```

Алгоритм Беллмана–Форда показывает лучшее время на таких тестах, так как имеет меньшую временную сложность, равную $O(m*n)$ против $O(n^3)$ у Флойда–Уоршелла (здесь m – количество рёбер, n – количество вершин).

5 Выводы

При выполнении этой лабораторной работы я вспомнил алгоритм Флойда–Уоршелла и изучил алгоритм Форда–Беллмана, который использует в своей реализации также изученные ранее принципы динамического программирования.

Этот алгоритм достаточно быстр, а также позволяет использовать рёбра с отрицательным весом, чего не может, например, алгоритм Дейкстры.

Список литературы

[1] *Алгоритм Беллмана – Форда – Википедия.*
URL: `http://ru.wikipedia.org/wiki/Алгоритм_Беллмана_-_Форда` (дата обращения: 16.12.2022).