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:	Е. А. Суханов
Преподаватель:	А. А. Кухтичев
Группа:	М8О-306Б
Дата:	
Оценка:	
Подпись:	

Москва, 2021

Лабораторная работа №9

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

Задан взвешенный неориентированный 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 Дейкстры. Длина пути равна сумме весов ребер на этом пути. Граф не содержит петель и кратных ребер.

Формат входных данных В первой строке заданы $1 \leq n \leq 10^5$, $1 \leq m \leq 10^5$, $1 \leq start \leq n$ 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 0 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

Алгоритм дейкстры находит кратчайший путь от начальной вершины до остальных. Он является жадным: на каждом этапе берется вершина с минимальным значением пути, а затем происходит обновление кратчайших путей до ее соседей. Если путь от этой вершины до соседа короче уже найденного пути, то обновляем его. Сложность алгоритма складывается из двух операций:

1. Поиск вершины с минимальной длиной пути;
2. Обновление длин путей;

Первая операция в простейшей реализации занимает $O(n^2)$ времени, где n - кол-во вершин (линейный поиск для каждой вершины). Вторая операция работает за $O(m)$, где m - кол-во ребер. В итоге получаем $O(n^2+m)$, что медленно для решения нашей задачи.

Для получения более простой сложности мы можем использовать структуру данных «set». Она основана на красно-черном дереве и позволяет выполнять обе операции за логарифмическое время. Следовательно получаем $O(nlogn + mlogn)$, а при $n \approx m$ получаем $O(mlogn)$ и $O(n)$ по памяти.

2 Исходный код

Заголовочный файл solution.hpp:

```
#pragma once
#include <vector>
#include <set>
#include <limits>

namespace NSolution {
    using TGraph = std::vector<std::vector<std::pair<int, int>>>;
    long long FindShortestPath(const TGraph& g, int s, int f);
}
```

Реализация solution:

```cpp
#include "solution.hpp"
const long long INF = std::numeric_limits<long long>::max();
namespace NSolution {
    long long FindShortestPath(const TGraph& g, int s, int f) {
        int n = g.size();
        std::vector<long long> dp(n, INF);
        dp[s] = 0;

        std::set<std::pair<int, int>> q;
        q.insert(std::make_pair(dp[s], s));
        while(!q.empty()) {
            int v = q.begin()->second;
            q.erase(q.begin());

            for(int j = 0; j < (int)g[v].size(); ++j) {
                int to = g[v][j].first;
                long long len = g[v][j].second;
                if (dp[v] + len < dp[to]) {
                    q.erase(std::make_pair(dp[to], to));
                    dp[to] = dp[v] + len;
                    q.insert(std::make_pair(dp[to], to));
                }
            }
        }

        return dp[f] == INF ? -1 : dp[f];
    }
}
```

Файл main.cpp:

```cpp
#include <iostream>
#include "solution.hpp"

void ReadGraph(NSolution::TGraph& g, int& s, int& f) {
    int n, m;
    std::cin >> n >> m >> s >> f;
    s--;
    f--;
    g.resize(n);
    for(int i = 0; i < m; ++i) {
        int v, e, w;
        std::cin >> v >> e >> w;
        v--;
        e--;
        g[v].push_back({e, w});
        g[e].push_back({v, w});
    }
}
```

```

int main() {
    NSolution::TGraph g;

    int s, f;
    ReadGraph(g, s, f);

    long long ans = NSolution::FindShortestPath(g, s, f);
    if(ans == -1)
        std::cout << "No solution";
    else
        std::cout << ans;
    std::cout << std::endl;
    return 0;
}
```

3 Консоль

```
$ make solution
g++ -c -Wall -pedantic -std=c++14 -O2 main.cpp -o main.o
g++ -c -Wall -pedantic -std=c++14 -O2 solution.cpp -o solution.o
g++ -O2 main.o solution.o -o solution
$ ./solution
5 6 1 5
1 2 2
1 3 0
3 2 10
4 2 1
3 4 4
4 5 5
8
```

4 Тест производительности

Алгоритм Дейкстры я буду сравнивать с наивным алгоритмом, который перебирает все пути. Тестирование производится на сгенерированных тестах. 100 вершин и 100 ребер:

```
$ python3 ../generated_tests/generator.py && ./benchmark <rtest.txt
259
My Solution : 385us
259
Default Solution : 627us
```

130 вершин и 130 ребер:

```
$ python3 ../generated_tests/generator.py && ./benchmark <rtest.txt
262
My Solution : 373us
262
Default Solution : 31935us
```

100000 вершин и 100000 ребер

```
$ python3 ../generated_tests/generator.py && ./benchmark <rtest.txt
519
My Solution : 21178us
```

Алгоритм Дейкстры работает куда быстрее наивного алгоритма: $O(mlogn)$ и $O(m!)$.

5 Выводы

Данный алгоритм имеет очень хорошую сложность. Но данный алгоритм работает только на положительно взвешенных графах.

Список литературы

[1] *Дейкстра простая реализация*
URL: `https://e-maxx.ru/algo/dijkstra` (дата обращения: 29.08.2021)

[2] *Дейкстра с использованием множества*
URL: `https://e-maxx.ru/algo/dijkstra_sparse` (дата обращения: 29.08.2021)