Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа по курсу "Дискретный анализ" №9

Студент: Клименко В. М.

Преподаватель: Макаров Н. К.

Группа: М8О-203Б-22

Дата: ___________

Оценка: ___________

Подпись: ___________

Содержание

Постановка задачи

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 ввода

В первой строке заданы $1 \leq n \leq 10^5$ и $1 \leq m \leq 10^5$, $1 \leq start \leq n$ 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 вывода

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

Алгоритм решения

Основная идея алгоритма Дейкстры заключается в постоянном обновлении расстояния вершин, смежных ближайшей к изначальной, пока ближайшая не станет равна конечной.

Реализуется эта идея при помощи очереди с приоритетом – попавшие в нее вершины отсортированы по возрастанию расстояния от изначальной вершины. Положим, что расстояние до всех вершин из изначальной – бесконечность, а до нее самой – 0, добавим ее в очередь.

Каждый раз вытаскиваем первую вершину из очереди. Если она равна искомой – конец алгоритма, так как пути ближе точно нет, иначе – пересчитываем все расстояния до смежных вершин вытащенной вершины как расстояние до вытащенной вершины плюс длина ребра от вытащенной вершины до смежной.

Если новое расстояние меньше сохраненного – сохраняем новое и добавляем вершину в очередь.

Временная сложность – $O(m + n \log n)$, пространственная сложность – $O(mn)$.

Исходный код

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


struct Edge {
	size_t to;
	uint32_t weight;

	Edge(size_t to, uint32_t weight) : to(to), weight(weight) {}
};


std::optional<uint64_t> dijkstra(const size_t vertexCount, const size_t start,
const size_t finish, const std::vector<std::vector<Edge>> &graph) {
	// distances to the starting vertex
	std::vector<uint64_t> distances(vertexCount,
std::numeric_limits<uint64_t>::max());
	std::priority_queue<std::pair<uint64_t, size_t>,
std::vector<std::pair<uint64_t, size_t>>, std::greater<>> distanceToVertex;

	distances[start] = 0;
	distanceToVertex.emplace(0, start);

	while (!distanceToVertex.empty()) {
	const auto [distance, vertex] = distanceToVertex.top();
	if (vertex == finish) { break; }

	distanceToVertex.pop();

	if (distance > distances[vertex]) { continue; }

	for (const Edge &edge : graph[vertex]) {
		size_t adjacentVertex = edge.to;
```

```
            uint64_t newDistance = distance + edge.weight;

            if (newDistance < distances[adjacentVertex]) {
                distances[adjacentVertex] = newDistance;
                distanceToVertex.emplace(newDistance, adjacentVertex);
            }
        }
        }

        if (distances[finish] == std::numeric_limits<uint64_t>::max()) {
            return std::nullopt;
        } else {
            return distances[finish];
        }
}

int main() {
        uint32_t vertexCount, edgeCount;
        size_t start, finish;

        std::cin >> vertexCount >> edgeCount >> start >> finish;
        --start; --finish;

        std::vector<std::vector<Edge>> graph(vertexCount);

        for (uint32_t i = 0; i < edgeCount; ++i) {
            size_t from, to;
            uint32_t weight;
            std::cin >> from >> to >> weight;
            --from; --to;
            graph[from].emplace_back(to, weight);
            graph[to].emplace_back(from, weight);
        }

        if (std::optional<uint64_t> answ = dijkstra(vertexCount, start, finish,
graph)) {
            std::cout << *answ << '\n';
        } else {
            std::cout << "No solution\n";
        }
}
```

Тесты

Входные данные:

```
4 5 1 4
1 2 2
1 3 5
2 3 1
2 4 7
3 4 2
```

Выходные данные:

```
5
```

Входные данные:

```
5 6 1 5
1 2 4
1 3 2
2 3 1
2 4 5
3 4 8
4 5 3
```

Выходные данные:

```
11
```

Вывод

В ходе лабораторной работы я изучил алгоритм Дейкстры нахождения минимального расстояния для двух вершин в ациклическом ненаправленном графе без отрицательных ребер.