Московский Авиационный Институт

(Национальный Исследовательский Университет)

**Факультет «Компьютерные науки и прикладная математика»**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №9 по курсу**

**«Дискретный анализ»**

**Тема работы**

**“Графы”**

Студент : Личковаха А.С.

Группа : М8О-310Б-21

Преподаватель : Н.К. Макаров

Оценка : \_\_\_\_\_\_\_\_\_\_\_

Дата : \_\_\_\_\_\_\_\_\_\_\_

Подпись : \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

1. **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<= n <= 10^5 и 1<= m <= 10^5 , 1<= start <= n и 1<= finish <= m. 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).

1. **Описание**

В [ориентированном](https://neerc.ifmo.ru/wiki/index.php?title=Ориентированный_граф) взвешенном [графе](https://neerc.ifmo.ru/wiki/index.php?title=Основные_определения:_граф,_ребро,_вершина,_степень,_петля,_путь,_цикл) , вес [рёбер](https://neerc.ifmo.ru/wiki/index.php?title=Основные_определения:_граф,_ребро,_вершина,_степень,_петля,_путь,_цикл) которого неотрицателен и определяется весовой функцией, алгоритм Дейкстры находит длины кратчайших [путей](https://neerc.ifmo.ru/wiki/index.php?title=Основные_определения:_граф,_ребро,_вершина,_степень,_петля,_путь,_цикл) из заданной [вершины](https://neerc.ifmo.ru/wiki/index.php?title=Основные_определения:_граф,_ребро,_вершина,_степень,_петля,_путь,_цикл) до всех остальных.  
В алгоритме поддерживается множество посещенных вершин , для которых уже вычислены длины кратчайших путей до них из начальной. На каждой итерации основного цикла выбирается вершина , которой на текущий момент соответствует минимальная оценка кратчайшего пути. Вершина добавляется в множество проверенных и производится релаксация всех исходящих из неё рёбер.

Сложность алгоритма составляетO(m\*log(n))по времени и памяти, так как мы n раз решаем линейную задачу.

1. **Исходный код**

#include <string>

#include <iostream>

#include <vector>

#include <set>

#include <fstream>

#include <algorithm>

using namespace std;

struct edge {

int node1;

int node2;

int weight;

};

long long minTrail( int nodes, int edges, int start, int finish, std::vector<edge> graph) {

std::vector< long long> result(nodes + 1, -1); // точно не будет

std::vector<bool> checked(nodes + 1, 0);// здесь отмечаем посещённые вершины

set<pair<long long , int>> queue;

queue.insert({ 0,start });

result[start] = 0;

long long fullTrail;

while (!queue.empty()){

auto firstElement = queue.begin();

int id = (\*firstElement).second;

queue.erase(firstElement);

if (checked[id] == true) continue;

checked[id] = true;

for ( int i = 0; i < edges; i++) {

if (graph[i].node1 == id) {

fullTrail = result[id] + graph[i].weight;

if (fullTrail < result[graph[i].node2] || result[graph[i].node2] == -1) {

result[graph[i].node2] = fullTrail;

queue.insert({ fullTrail,graph[i].node2 });

}

}

else if (graph[i].node2 == id) {

fullTrail = result[id] + graph[i].weight;

if (fullTrail < result[graph[i].node1] || result[graph[i].node1] == -1) {

result[graph[i].node1] = fullTrail;

queue.insert({ fullTrail,graph[i].node1});

}

}

}

}

return result[finish];

}

int main() {

int nodes, edges;

int start, finish;

std::cin >> nodes >> edges >> start >> finish;

std::vector<edge> graph;

int node1, node2;

int edgeWeight;

for ( int k = 0; k < edges; k++) {

std::cin >> node1 >> node2 >> edgeWeight;

graph.push\_back({ node1, node2, edgeWeight });

}

long long trail = minTrail(nodes,edges, start, finish, graph);

if (trail == -1) {

std::cout << "No solution" << std::endl;

}

else {

std::cout << trail << std::endl;

}

}

1. **Демонстрация работы программы**

5 6 1 5

1 2 2

1 3 0

3 2 10

4 2 1

3 4 4

4 5 5

8

1. **Выводы**

Для выполнение данной лабораторной работ я применил метод Дейкстры. Этот способ решения помогает решать задачу кратчайшего пути от начальной вершины до всех остальных. При этом работая за логарифм он позволяет решать данную задачу более эффективно и оптимально, нежели наивным путём.