МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №3
по дисциплине «Исследование операций»
Вариант 11

Бобовоза Владислава
Сергеевича
студента 3 курса, 6 группы
специальность «прикладная
математика»

Минск, 2024

Формализация линейной оптимизационной задачи

Условие задачи 11, «Второй по длине маршрут»

Задано N городов с номерами от 1 до N и сеть из M дорог с односторонним движением между ними. Каждая дорога определяется тройкой (i, j, k) натуральных чисел, где i — номер города, в котором дорога начинается, j — номер города, в котором дорога заканчивается, а k — её длина. Дороги друг с другом могут пересекаться только в конечных городах. Все ориентированные маршруты между двумя указанными городами A и B можно упорядочить в список по неубыванию их длин. Необходимо найти один из маршрутов, который является вторым в этом упорядоченном списке (между городами A и B существует по крайней мере два ориентированных маршрута). Вывести его длину L и города, через которые он проходит (искомый маршрут по дугам может проходить несколько раз).

Формат входных данных:

Первая строка содержит два целых числа N и M ($1 \le N \le 10000$, $1 \le M \le 100000$). Затем идут M строк файла по три числа в каждой: i, j, k. Последняя строка содержит номера городов A и B. Формат выходных данных Первая строка должна содержать длину второго по длине маршрута между городами A и B. Вторая строка — это номера городов, через которые проходит второй по длине маршрут. В случае если вторых по длине маршрутов несколько, выдать в выходной файл информацию о любом из них.

Пример:

входной файл	выходной
3 3 1 2 1 2 1 1 2 3 1 1 3	4 1 2 1 2 3
3 4 1 3 10 2 1 10 2 3 20 3 2 5 2 3	20 2 3

Небольшое пояснение к задаче:

Поскольку по условию задачи допускается, чтобы искомый маршрут по некоторым дугам проходил несколько раз, то для решения задачи можно применить алгоритм Дейкстры с использованием приоритетной очереди. В

данном алгоритме вершину необходимо считать просмотренной лишь после её второго удаления из приоритетной очереди.

Листинг программы на языке C++:

```
#include <fstream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

const long long INF = 1e18;

// Структура представляет собой ребро графа
struct Edge {
    int adj_vertex; // Смежная вершина
    long long length; // Длина ребра
     bool in_short_path; // Флаг, указывающий, находится ли ребро в кратчайшем пути
   Edge(int vertex, long long len, bool flag = false) : adj_vertex(vertex), length(len), in_short_path(flag) {};
};

// Структура представляет информацию о пути до вершины
struct Path {
    long long dist; // Расстояние до вершины
    int from; // Предыдущая вершина пути
    int edge_id; // Индекс ребра на пути
   Path(long long distance = INF, int v = -1, int e = -1) : dist(distance), from(v), edge_id(e) {}

    // Перегрузка оператора < для приоритетной очереди
    bool operator<(const Path& other) const {
        return dist < other.dist;
    }
};

// Класс для нахождения кратчайших путей в графе
class ShortestPath {
public:
```

```
    ShortestPath(vector<vector<Edge>> &g) : graph(g) {
        num_of_vertex = graph.size();
    }

    // Второй кратчайший путь в графе
    void findSecondPath(int start_vertex, int finish_vertex) {
        // Первый кратчайший путь в графе
       vector<Path> first = findShortestPath(start_vertex, finish_vertex);

        // Отмечает ребра первого кратчайшего пути
        Path cur = first[finish_vertex];
        while (cur.from != -1) {
            graph[cur.from][cur.edge_id].in_short_path = true;
            cur = first[cur.from];
        }
         // Добавляем дополнительные вершины и ребра в граф для поиска
второго пути
        buildSecondLayer();

        // Второй кратчайший путь
              vector<Path>  second  =  findShortestPath(start_vertex  +
num_of_vertex, finish_vertex);
        second_path_len = second[finish_vertex].dist;
        second_path = getPath(second, finish_vertex);
    }

    // Возвращает длину второго кратчайшего пути
    long long getSecondLength() const {
        return second_path_len;
    }

    // Возвращает второй кратчайший путь
    vector<int> getSecondPath() const {
        return second_path;
    }

private:
    // Находит кратчайший путь в графе
    vector<Path> findShortestPath(int start_v, int finish_v) {
```

```
        vector<Path> path(graph.size(), Path()); // Инициализируем пути до каждой вершины как бесконечность
        priority_queue<Path> q; // Приоритетная очередь для выбора вершин с наименьшим расстоянием
        path[start_v] = Path(0); // Расстояние до стартовой вершины равно 0
        q.push(Path(0, start_v)); // Добавляем стартовую вершину в очередь

        // Применяем алгоритм Дейкстры
        while (!q.empty()) {
              int vertex = q.top().from; // Берем вершину с наименьшим расстоянием из очереди
            long long dist = q.top().dist;
            q.pop();

               if (dist > path[vertex].dist) continue; // Если текущее расстояние до вершины больше, чем уже известное, пропускаем

            // Обновляем расстояния до всех смежных вершин
            for (int i = 0; i < graph[vertex].size(); i++) {
                int to = graph[vertex][i].adj_vertex;
                long long new_dist = graph[vertex][i].length + dist;
                if (path[to].dist > new_dist) {
                    path[to] = Path(new_dist, vertex, i);
                     q.push(Path(new_dist, to)); // Добавляем обновленную вершину в очередь
                }
            }
        }

        return path; // Возвращаем массив кратчайших путей
    }

    // Добавляет дополнительные вершины и ребра в граф для поиска второго кратчайшего пути
    void buildSecondLayer() {
        for (int i = 0; i < num_of_vertex; i++) {
            vector<Edge> new_vertex = graph[i];
            for (int j = 0; j < graph[i].size(); j++) {
                int ind = j;
```

```
                if (!graph[i][j].in_short_path) {
                    new_vertex.push_back(new_vertex[j]);
                    ind = new_vertex.size() - 1;
                }
                new_vertex[ind].adj_vertex += num_of_vertex;
            }
            graph.push_back(move(new_vertex));
        }
    }

    // Восстанавливает путь из предыдущих вершин
    vector<int> getPath(const vector<Path> &prev, int finish) {
        vector<int> path;
        int v = finish;
        while (v != -1) {
            int ind = (v >= num_of_vertex) ? v - num_of_vertex : v;
            path.push_back(ind + 1); // Добавляет вершину в путь
            v = prev[v].from; // Переходим к предыдущей вершине
        }
        reverse(path.begin(), path.end()); // Переворачиваем путь
        return path;
    }

    int num_of_vertex; // Количество вершин в исходном графе
    long long second_path_len; // Длина второго кратчайшего пути
    vector<int> second_path; // Второй кратчайший путь
    vector<vector<Edge>> graph; // Граф
};

int main()
{
    ifstream fin("input.in");
    ofstream fout("output.out");
    ios_base::sync_with_stdio(false); fin.tie(NULL); fout.tie(NULL);

    int num_of_vertex, num_of_edges;
    fin >> num_of_vertex >> num_of_edges;

    int u, v;
    long long dist;
```

```
    vector<vector<Edge>> graph(num_of_vertex);
    for (int i = 0; i < num_of_edges; i++) {
        fin >> v >> u >> dist;
        graph[--v].push_back(Edge(--u, dist));
    }

    ShortestPath shortest_path(graph);

    int start_vertex, finish_vertex;
    fin >> start_vertex >> finish_vertex;

    shortest_path.findSecondPath(start_vertex - 1, finish_vertex - 1);

    fout << shortest_path.getSecondLength() << "\n";
    vector<int> path = shortest_path.getSecondPath();
    for (int i = 0; i < path.size(); i++) {
        fout << path[i];
        if (i < path.size() - 1) {
            fout << " ";
        }
    }

    return 0;
}
```

Проверка программы на известных входных данных:

входной файл	выходной
3 3 1 2 1 2 1 1 2 3 1 1 3	4 1 2 1 2 3
3 4 1 3 10 2 1 10 2 3 20 3 2 5 2 3	20 2 1 3

Если сверить пример и результаты моей программы, то можно увидеть что выходные файлы совпали, за исключением второго. Но это не ошибка. В примере 1, маршрут по дуге (1,2) пройдет дважды. В примере 2, маршрут 2->1->3 также является верным.