Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 9 по курсу дискретного анализа: графы

Студент: Н. С. Токарев
Преподаватель: Н. А. Зацепин
Группа: М8О-307Б-18
Дата:
Оценка:
Подпись:

Москва
2020

Условие

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.

Метод решения

Данное задание предполагает реализацию алгоритма Дейкстры по нахождению минимального пути.

1. Создаем метки(показатель минимального пути) для каждой вершины: инициализирую огромными числами. Вершину с которой начинаю путь инициализирую нулем.

2. Из текущей вершины считаем все возможные минимальные пути до следующих вершин, если путь минимален, то отмечаем вершину полученным минимальным путем.

3. Перехожу к выполнению действия 2 для следующих, еще не пройденных вершин.

4. Алгоритм завершается, когда все вершины пройдены.

Описание программы

Для решения поставленной задачи я создал класс полями которого являются значение максимальной вершины, объект, похожий на матрицу смежности графов. Таким образом для каждой вершины определен массив содержащий информацию, о ребрах данной вершины(вес ребра и вершина, к которой ребро направлено). Таким образом конструктор данного класса конструирует объект,схожий с матрицей смежности.
Алгоритм Дейкстры: Для начала я создаю массив,в котором отмечаю все вершины метками(минимальным числом по дефолту равным огромному и недостижимому числу). Также мне понадобится еще одна структура для хранения не пройденных вершин и их меток(в моем случае - это будет очередь, содержащая информацию о непройденной вершине и минимальном пути до этой вершины).
while(!peaks.empty()) { .
long long v = peaks.front().to; // получаю не пройденную вершину v
peaks.pop();
for(TData temp : matrix[v]) { // прохожу по строке матрицы смежности графов для полученной вершины v
if(distance[temp.to] > distance[v] + temp.weight) { // Если путь от вершины v до temp.to является минимальным, то изменяю метку temp.to

distance[temp.to] = distance[v] + temp.weight;
peaks.push(TData(temp.to,distance[temp.to])); // Добавляю вершину temp.to в очередь с непройденными вершинами
}
}
}
Таким образом, данный алгоритм будет работать, так как, когда для каждой вершины будет подсчитана правильная метка и условие: if(distance[temp.to] > distance[v] + temp.weight) не будет выполняться и соответсвенно в очередь peaks не будут добавляться новые вершины и она опустеет. Сложность данного алгоритма колеблется от O(n) до O(n * n),где n - количество вершин.

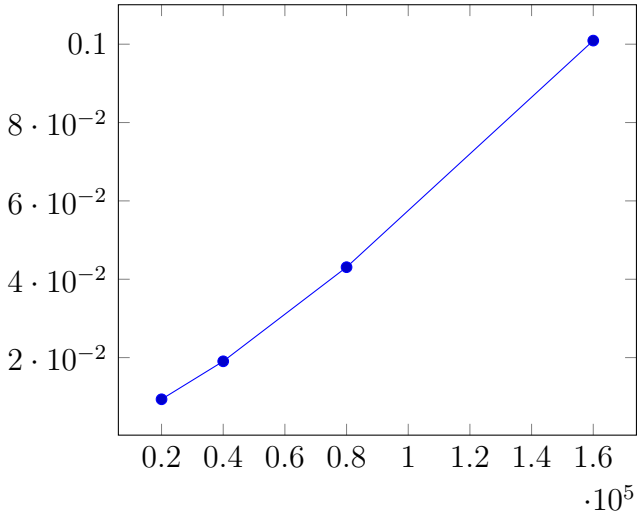
Исходный код

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

class TGraph {
public:
    TGraph(std::istream& in, const long long& count, const long long& lines) {
        matrix.resize(count);
        size = count;
        matrix.resize(size);
        long long from, to, weight;
        for(long long i = 0; i < lines; i++) {
            in >> from >> to >> weight;
            TData temp;
            temp.to = from - 1;
            temp.weight = weight;
            matrix[to - 1].push_back(temp);
            temp.to = to - 1;
            matrix[from - 1].push_back(temp);
        }
    }


    long long Find(long long& start, long long& finish) {
        std::vector<long long> distance(size, std::numeric_limits<long long int>::max())
        std::queue<TData> peaks;
        peaks.push(TData(start - 1, 0));
        distance[start - 1] = 0;
```

```
        while(!peaks.empty()) {
            long long v = peaks.front().to;
            peaks.pop();
            for(TData temp : matrix[v]) {
                if(distance[temp.to] > distance[v] + temp.weight) {
                    distance[temp.to] = distance[v] + temp.weight;
                    peaks.push(TData(temp.to,distance[temp.to]));
                }
            }
        }
        return distance[finish - 1];
    }

private:
    struct TData {
        long long to;
        long long weight;
        TData(long long nto, long long nweight): to(nto), weight(nweight)
        {}
        TData(): to(0), weight(0)
        {}
    };
    long long size;
    std::vector<std::vector<TData>> matrix;
};




int main() {
    long long max, count, start, finish;
    std::cin >> max >> count >> start >> finish;
    TGraph Our(std::cin, max, count);
    long long result;
    result = Our.Find(start,finish);
    if(result == std::numeric_limits<long long int>::max()) {
        std::cout << "No solution" << std::endl;
    } else {
        std::cout << result << std::endl;
    }
    return 0;
}
```

Тест производительности

Тесты сгенерированы таким образом, что для последующих точек количество вершин и ребер увеличивается вдвое. На графике по оси x колчество ребер, где количество вершин графа для точек = количество ребер / 2.



Пояснения к графику: Ось y - время в секундах. Ось x - количество ребер.

Выводы

Таким образом для взвешенного неориентированного графа, не имеющего петель и кратных ребер был реализован алгоритм Дейкстры. Для графа, данного по заданию, было довольно не сложно реализовать алгоритм Дейкстры. Для графа, имеющего кратные вершины или ориенированные ребра работа моей программы стоит под вопросом. Теория графов позволяет решать различные задачи в большей степени, связанные с логистикой или различными предвижениями. На данный день существует огромное количество различных алгоритмов, а также приемов, которые используются в теории графов. Сложность реализованного алгоритма колеблется от O(n) до O(n * n),где n - количество вершин.