Лабораторная работа № 9 по курсу дискретного анализа: графы

Выполнил студент группы М8О-308Б-20 МАИ *Борисов Ян*.

Условие

Кратко описывается задача:

1. Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от *1* до *n*. Необходимо найти длины кратчайших путей между всеми парами вершин при помощи алгоритма Джонсона. Длина пути равна сумме весов ребер на этом пути. Обратите внимание, что в данном варианте веса ребер могут быть отрицательными, поскольку алгоритм умеет с ними работать. Граф не содержит петель и кратных ребер.

2. 6

Метод решения

1. Так как алгоритм Дейкстры не умеет работать с отрицательными ребрами, необходимо на время избавиться от них в нашем графе. Для этого мы добавляем в граф фиктивную вершину S и строим из нее ребра с весом 0 в каждую вершину исходного графа.
2. Для нового графа запускаем алгоритм Беллмана – Форда, который либо обнаруживает наличие отрицательного цикла в графе и завершает алгоритм, либо возвращает кратчайшие расстояния от фиктивной вершины S до каждой вершины исходного графа. Суть алгоритма заключается в том, что мы V – 1 раз проходим по всем ребрам и релаксируем их если $\boldsymbol{d[v] > d[u] + w(u,w)}$. Если на V-ой итерации происходит еще одна релаксация, то в графе имеется отрицательный цикл. С помощью этих кратчайших расстояний мы перевзвешиваем ребра по следующей формуле:

   $$\boldsymbol{\omega\varphi(u,v) = \omega(u,v) + \varphi(u) - \varphi(v).}$$

   Удаляем фиктивную вершину и запускаем алгоритм Дейкстры для каждой вершины графа, который возвращает кратчайшие расстояния до каждой другой вершины графа. Для преобразования этих расстояний к изначальному графу необходимо применить обратную формулу перевзвешивания

$$\omega\varphi(u,v) = \omega(u,v) - \varphi(u) + \varphi(v).$$

3. Суть алгоритма Дейкстры заключается в том, что в алгоритме поддерживается множество вершин, для которых уже вычислены длины кратчайших путей до них из s. На каждой итерации основного цикла выбирается вершина, не помеченная посещенной, которой на текущий момент соответствует минимальная оценка кратчайшего пути. Вершина u добавляется в множество посещенных и производится релаксация всех исходящих из неё рёбер.

Описание программы

В моей программе один файл main.cpp:

```
#include <iostream>
#include <vector>
#include <climits>

const long INF = LONG_MAX;

struct Edge{
    int start, end;
    long long weight;
    Edge(int start, int end, long long weight);
};

Edge::Edge(int start, int end, long long weight) {
    this->start = start, this->end = end, this->weight = weight;
}

struct Graph{
    std::vector<Edge> edges;
    int vertices;
    Graph(int N);
};

Graph::Graph(int N) {
    this->vertices = N + 1;
}


std::pair<bool,long long*> BellmanFord(Graph* graph) {
    auto* dist = new long long[graph->vertices];
    dist[graph->vertices - 1] = 0;
    for(int i = 0; i < graph->vertices - 1; ++i) {
        dist[i] = INF;
    }
    for(int i = 0; i < graph->vertices - 1; ++i) {
        for (auto & edge : graph->edges) {
            if (dist[edge.start] < INF) {
                dist[edge.end] = std::min(dist[edge.start] + edge.weight,
dist[edge.end]);
```

```
            }
        }
    }
    for (auto & edge : graph->edges) {
        if (dist[edge.start] < INF) {
            if (dist[edge.start] + edge.weight < dist[edge.end]) {
                return std::pair(false, nullptr);
            }
        }
    }
    for (auto & edge : graph->edges) {
        edge.weight = edge.weight + dist[edge.start] - dist[edge.end];
    }
    return std::pair(true, dist);
}

void Dijkstra(Graph* graph, long long* dist1) {
    for(int vertice = 0; vertice < graph->vertices - 1; ++vertice) {
        long long dist[graph->vertices - 1];
        dist[vertice] = 0;
        for (int i = 0; i < graph->vertices - 1; ++i) {
            if (i != vertice) {
                dist[i] = INF;
            }
        }
        bool visited[graph->vertices - 1];
        for (int i = 0; i < graph->vertices - 1; ++i) {
            visited[i] = false;
        }
        for (int i = 0; i < graph->vertices - 1; ++i) {
            int start = -1;
            for (int j = 0; j < graph->vertices - 1; ++j) {
                if ((start == -1 || dist[j] < dist[start]) && !visited[j])
{
                    start = j;
                }
            }
            if(dist[start] == INF)
                break;
            visited[start] = true;
            for (auto &edge: graph->edges) {
                if (edge.start == start) {
                    dist[edge.end] = std::min(dist[edge.end],
                                              dist[edge.start] +
edge.weight);
                }
            }
        }
        for(int i = 0; i < graph->vertices - 1; ++i) {
            if(dist[i] == INF) {
                std::cout << "inf" << ' ';
            }
            else {
                std::cout << dist[i] - dist1[vertice] + dist1[i]<< ' ';
```

```
            }
        }
        std::cout << '\n';
    }
}

int main() {
    int N, M;
    std::cin >> N >> M;
    auto* graph = new Graph(N);
    int vertice1, vertice2, weigth;
    for(int i = 0; i < M; ++i) {
        std::cin >> vertice1 >> vertice2 >> weigth;
        graph->edges.emplace_back(vertice1 - 1, vertice2 - 1, weigth);
    }
    for(int i = 0; i < N; ++i) {
        graph->edges.emplace_back(N, i, 0);
    }
    std::pair<bool, long long*> bf = BellmanFord(graph);
    if(!bf.first) {
        std::cout << "Negative cycle";
    }
    else {
       Dijkstra(graph, bf.second);
    }
    return 0;
}
```

Дневник отладки

Длительное время отлаживал проблему с переводом полученных алгоритмом Дейкстры расстояний к изначальному состоянию графа.

Тест производительности

Померить время работы кода лабораторной и теста производительности на разных объёмах входных данных. Сравнить результаты. Проверить, что рост времени работы приувеличении объема входных данных согласуется с заявленной сложностью.

Сложность алгоритма: $O(V^3 + VE)$

1) V = 100, E = 100 => 0.005c

2) V = 1000, E = 1000 => 2.268c

3) V = 10000, E = 10000 => 2268c

Из проведенных тестов, можно заметить, что при увеличении V и E в 10 раз, время работы программы тоже увеличивается примерно в 1000 раз. Асимптотическая оценка подтвердилась.

Недочёты

Не были обнаружены.

Выводы

При выполнении данной лабораторной работы я ознакомился с алгоритмом Джонсона и смог решить задачу поиска кратчайшего расстояния между всеми парами вершин в графе.