Лабораторная работа № 9 по курсу дискретного анализа: графы

Выполнил студент группы М8О-308Б-20 МАИ *Зубко Дмитрий.*

Условие

Кратко описывается задача:

> 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.

Метод решения

1. Алгоритм Дейкстры не умеет работать с отрицательными ребрами. Поэтому мы избавляемся от таких ребер в графе, используя метод изменения веса. Суть этого метода заключается в том, что строится для заданного графа новая весовая функция, которая неотрицательна для всех ребер графа и сохраняющая кратчайшие пути. Для этого мы добавляем в граф фиктивную вершину S и строим из неё во все вершины ребра с весом 0.

2. Запускаем для вершины S алгоритм Беллмана-Форда, который возвращает кратчайшее расстояние от фиктивной вершины до каждой вершины графа. Также он обнаруживает негативный цикл и завершает в таком случае алгоритм. Его суть в том, что мы проходим V – 1 раз по всем ребрам и релаксируем их. Проверка на негативный цикл – запуска алгоритма V раз. Если происходит еще одна релаксация – есть отрицательный цикл. В конце перевзвешиваем ребра по формуле: $\boldsymbol{\omega(\varphi(u,v)) = \omega(u,v) + \varphi(u) - \varphi(v)}$.

3. Запускаем для перевзвешенного графа алгоритм Дейкстры. Он возвращает

кратчайшие расстояния между всеми парами вершин. В конце алгоритма возвращаем граф к первоначальному виду, применяя обратную потенциальную функцию: $\boldsymbol{\omega}(\boldsymbol{\varphi}(\boldsymbol{u},\boldsymbol{v})) = \boldsymbol{\omega}(\boldsymbol{u},\boldsymbol{v}) + \boldsymbol{\varphi}(\boldsymbol{v}) - \boldsymbol{\varphi}(\boldsymbol{u})$. Алгоритм Дейкстры заключается в том, что мы поддерживаем множество вершин, для которых уже вычислены кратчайшие пути до них из стартовой вершины. На каждой итерации выбираем вершину, которая не помечена посещенной и соответствует минимальному пути. Выбранная вершина добавляется в множество посещенных и происходит релаксация всех исходящих из неё ребер.

Описание программы

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

const long long INF = LONG_MAX;

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

struct Graph {
   std::vector<Edge> edges;
   long long vertices;

   explicit Graph(long long vertices) : vertices(vertices + 1) {}

   void input(long long m) {
      for (long long i = 0; i < m; ++i) {
         int v1;
         int v2;
         int w;
         std::cin >> v1 >> v2 >> w;
         this->edges.emplace_back(v1 - 1, v2 - 1, w);
      }
      for (long long i = 0; i < this->vertices; ++i) {
         this->edges.emplace_back(this->vertices, i, 0);
      }
   }
};
```

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

    for (auto &edge: graph->edges) { // ищем цикл
        if (distances[edge.start] < INF) {
            if (distances[edge.start] + edge.weight < distances[edge.end]) {
                return true;
            }
        }
    }
    for (auto &edge: graph->edges) {
        edge.weight = edge.weight + distances[edge.start] - distances[edge.end];
    }
    return false;
}

void DijkstraAlgorithm(Graph* graph, std::vector<long long>& distances) {
    for (long long vertice = 0; vertice < graph->vertices - 1; ++vertice) {
        std::vector<long long> distances_from_vertice(graph->vertices - 1, INF);
        distances_from_vertice[vertice] = 0;

        std::vector<bool> completed(graph->vertices - 1, false);

        for (long long i = 0; i < graph->vertices - 1; ++i) {
            long long start = -1;

            for (long long j = 0; j < graph->vertices - 1; ++j) { // поиск минимума
                if ((start == -1 or distances_from_vertice[j] < distances_from_vertice[start]) and
!completed[j]) {
                    start = j;
                }
            }

            if (distances_from_vertice[start] == INF) break;
```

```
            completed[start] = true;

            for (auto &edge: graph->edges) {
                if (edge.start == start) {
                    distances_from_vertice[edge.end] = std::min(distances_from_vertice[edge.end],
                                                        distances_from_vertice[edge.start] + edge.weight);
                }
            }
        }

        for (long long i = 0; i < graph->vertices - 1; ++i) {
            if (distances_from_vertice[i] == INF) {
                std::cout << "inf";
            } else {
                std::cout << distances_from_vertice[i] - distances[vertice] + distances[i];
            }
            std::cout << ' ';
        }

        std::cout << std::endl;
    }
}

int main() {
    long long n;
    long long m;
    std::cin >> n >> m;
    auto *graph = new Graph(n);
    graph->input(m);
    std::vector<long long> distances(graph->vertices);
    bool negative_cycle = BellmanFord(graph, distances);
    if (negative_cycle) {
        std::cout << "Negative cycle";
    } else {
        DijkstraAlgorithm(graph, distances);
    }
}
```

Дневник отладки

Были проблемы с изначальной инициализацией массивов, эту проблему удалось исправить достаточно быстро.

Тест производительности

Сложность алгоритма: $O(VD + VE)$, где $O(D)$ – время работы алгоритма Дейкстры.

1) V = 300, E = 300 => 0.035с

2) V = 1000, E= 2000 => 0,4с

3) V = 3000, E = 1000 => 1,5с

4) V = 5000, E = 10000 => 17,454с

Из проведенных тестов, можно заметить, что время работы программы соответствует заявленной сложности.

Недочёты

Программа работает корректно только для правильных входных данных.

Выводы

Благодаря данной лабораторной работе я вспомнил основы работы с графами. Повторил основные алгоритмы и изучил новые. Смог реализовать алгоритм Беллмана-Форда со сложностью $O(VE)$. С его помощью переделал входной граф в граф, где нет отрицательных весов на ребрах. Проверил граф, на негативный цикл. Затем применил алгоритм Джонсона для нахождения кратчайшего расстояния между всеми парами вершин в графе.