Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент: М. Ю. Курносов
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №9

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.

Формат ввода: В первой строке заданы $1 \leq n \leq 2000 1 \leq m \leq 4000$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от -10^9 до 10^9.

Формат вывода: Если граф содержит цикл отрицательного веса, следует вывести строку "Negative cycle"(без кавычек). В противном случае следует вывести матрицу из n строк и n столбцов, где j-е число в i-й строке равно длине кратчайшего пути из вершины i в вершину j. Если такого пути не существует, на соответствующей позиции должно стоять слово "inf"(без кавычек). Элементы матрицы в одной строке разделяются пробелом.

Примеры:

Входные данные:

5 4
1 2 -1
2 3 2
1 4 -5
3 1 1

Результат работы:

0 -1 1 -5 inf
3 0 2 -2 inf
1 0 0 -4 inf
inf inf inf 0 inf
inf inf inf inf 0

1 Описание

В ходе лабораторной работы был реализован и исследован алгоритм Джонсона для нахождения кратчайших путей между всеми парами вершин в ориентированном взвешенном графе. Алгоритм успешно протестирован на графах различных размеров и плотностей, включая графы с отрицательными весами ребер.

2 Исходный код

main.cpp:

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

const long long inf = std::numeric_limits<long long>::max();

struct edge{
    edge(int a, int b, long long k): first(a), second(b), weight(k){};
    int first, second;
    long long weight;
};

void Printmatr(std::vector<std::vector<long long>>& res, int n);
std::vector<std::vector<long long>> findPath(std::vector<edge>& e, int n, int m);

std::vector<long long> BF(std::vector<edge>& e, int n);
void SetWeight(std::vector<edge>& e, std::vector<long long>& res, int m);
std::vector<long long> Dijkstr(std::vector<edge>& plusEdges, int n, int m, std::vector
    <edge>& e, int s);



std::vector<long long> BF(std::vector<edge>& e, int n){
    std::vector<long long> res(n+1, inf);
    res[0] = 0;
    long long a, b, w;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < e.size(); j++){
            a = e[j].first;
            b = e[j].second;
            w = e[j].weight;
            if(res[a] != inf && res[a] + w < res[b]){
                res[b] = res[a] + w;
            }
        }
    }

    for(int i = 0; i < e.size(); i++){
        a = e[i].first;
        b = e[i].second;
        w = e[i].weight;
        if(res[a] != inf && res[a] + w < res[b]){
            res.clear();
            break;
        }
```

```
46      }
47      return res;
48  }
49
50  void SetWeight(std::vector<edge>& e, std::vector<long long>& res, int m){
51      e.erase(e.begin()+m, e.end());
52
53      long long a, b;
54      for(int i = 0; i < e.size(); i++){
55          a = e[i].first;
56          b = e[i].second;
57          e[i].weight += res[a] - res[b];
58      }
59  }
60
61
62  std::vector<long long> Dijkstr(std::vector<edge>& plusEdges, int n, int m, std::vector
        <edge>& e, int s){
63      std::vector<std::pair<long long, bool>> cur(n, std::make_pair(inf, false));
64      std::vector<long long> res(n, inf);
65      cur[s].first = 0;
66      res[s] = 0;
67
68      long long w, wres;
69      int a, b;
70      long long weight;
71      int idx;
72
73      for(int i = 0; i < n; i++){
74          idx = -1;
75          w = inf;
76          for(int j = 0; j < n; j++){
77              if(!cur[j].second && cur[j].first < w){
78                  w = cur[j].first;
79                  wres = res[j];
80                  idx = j+1;
81              }
82          }
83
84          if(idx == -1){
85              break;
86          }
87
88          cur[idx-1].second = true;
89          for(int k = 0; k < m; k++){
90              a = plusEdges[k].first;
91              if(a != idx){
92                  continue;
93              }
```

```
            b = plusEdges[k].second;
            weight = plusEdges[k].weight;
            if(w + weight < cur[b-1].first){
                cur[b-1].first = w + weight;
                res[b-1] = wres + e[k].weight;
            }
        }
    }

    return res;
}

void Printmatr(std::vector<std::vector<long long>>& res, int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(res[i][j] == inf){
                std::cout << "inf";
            }
            else{
                std::cout << res[i][j];
            }
            std::cout << " ";
        }
        std::cout << std::endl;
    }
}

std::vector<std::vector<long long>> findPath(std::vector<edge>& e, int n, int m)
{
    std::vector<long long> resBF;
    std::vector<edge> plusEdges = e;

    for(int i = 1; i < n+1; i++){
        plusEdges.push_back(edge(0, i, 0));
    }

    resBF = BF(plusEdges, n);
    std::vector<std::vector<long long>> res;

    if(resBF.empty()){
        std::cout << "Negative cycle\n";
        return res;
    }

    SetWeight(plusEdges, resBF, m);

    for(int i = 0; i < n; i++){
        res.push_back(Dijkstr(plusEdges, n, m, e, i));
    }
```

```
143     return res;
144 }
145 
146 int main(){
147     std::vector<edge> e;
148     int n, m;
149     int a, b;
150     long long w;
151     char x = 0, x1 = 0;
152 
153     std::cin >> n >> m;
154     for(int i = 0; i < m; i++){
155         std::cin >> a >> b >> w;
156         e.push_back(edge(a, b, w));
157     }
158     std::vector<std::vector<long long>> res;
159     res = findPath(e, n, m);
160     if(!res.empty()){
161         Printmatr(res, n);
162     }
163 }
```

main.cpp	
void Printmatr(std::vector<std::vector<long long»& res, int n)	Процедура для вывода содержимого матрицы(Ответа)
std::vector<std::vector<long long> > findPath (std::vector<edge>& e, int n, int m)	Функция реализующая Алгоритм Джонсона
std::vector<long long> BF(std::vector<edge>& e, int n)	Функция реализующая Алгоритм Белмана-Форда
void SetWeight(std::vector<edge>& e, std::vector<long long>& res, int m)	Функция для пересчета весов ребер
std::vector<long long> Dijkstr(std::vector<edge>& plusEdges, int n, int m, std::vector<edge>& e, int s)	Функция реализующая Алгоритм Дейкстры
int main()	Основная функция работы программы

3 Консоль

```
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/9lab$ g++ -o c main.cpp
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/9lab$ ./c
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
0 -1 1 -5 inf
3 0 2 -2 inf
1 0 0 -4 inf
inf inf inf 0 inf
inf inf inf inf 0
```

4 Тест производительности

Для оценки эффективности реализованного алгоритма Джонсона был проведен сравнительный тест производительности с наивным алгоритмом последовательного применения алгоритма Дейкстры из каждой вершины. Тестирование проводилось на серии случайно сгенерированных графов различного размера (от 5 до 50 вершин) и плотности (от разреженных до плотных), включая графы с отрицательными весами ребер. В ходе теста измерялось время выполнения обоих алгоритмов в микросекундах, проверялась корректность результатов их работы и вычислялся коэффициент ускорения.

```
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/9lab$ ./p
Тест #1: Очень маленький граф
Вершин: 5,Ребер: 8
Сгенерировано 7 ребер
Запуск алгоритма Джонсона...
Джонсон выполнен за: 26 мкс
Запуск наивного алгоритма...
Наивный алгоритм выполнен за: 17 мкс
Результаты совпадают: ДА
Отношение производительности (Наивный/Джонсон): 0.65x
Наивный быстрее в 1.53 раз
----------------------------------------

Тест #2: Маленький граф
Вершин: 10,Ребер: 15
Сгенерировано 12 ребер
Запуск алгоритма Джонсона...
Джонсон выполнен за: 97 мкс
Запуск наивного алгоритма...
Наивный алгоритм выполнен за: 60 мкс
Результаты совпадают: ДА
Отношение производительности (Наивный/Джонсон): 0.62x
Наивный быстрее в 1.62 раз
----------------------------------------

Тест #3: Средний граф
Вершин: 30,Ребер: 100
Сгенерировано 97 ребер
Запуск алгоритма Джонсона...
Джонсон выполнен за: 5209 мкс
```

```
Запуск наивного алгоритма...
Наивный алгоритм выполнен за: 3966 мкс
Результаты совпадают: ДА
Отношение производительности (Наивный/Джонсон): 0.76x
Наивный быстрее в 1.31 раз
----------------------------------------

Тест #4: Разреженный граф
Вершин: 200,Ребер: 2000
Сгенерировано 1980 ребер
Запуск алгоритма Джонсона...
Джонсон выполнен за: 1337436 мкс
Запуск наивного алгоритма...
Наивный алгоритм выполнен за: 1389512 мкс
Результаты совпадают: ДА
Отношение производительности (Наивный/Джонсон): 1.04x
Джонсон быстрее в 1.04 раз
----------------------------------------

Тест #5: Плотный граф
Вершин: 800,Ребер: 3000
Сгенерировано 2994 ребер
Запуск алгоритма Джонсона...
Джонсон выполнен за: 49482448 мкс
Запуск наивного алгоритма...
Наивный алгоритм выполнен за: 50405121 мкс
Результаты совпадают: ДА
Отношение производительности (Наивный/Джонсон): 1.02x
Джонсон быстрее в 1.02 раз
----------------------------------------

=== Тест с отрицательными весами (без циклов) ===

Граф с 10 вершинами,16 ребрами и отрицательными весами
Джонсон выполнен за: 84 мкс
Результаты выглядят корректно (саморасстояния = 0)
```

5 Выводы

В ходе выполнения работы был реализован алгоритм Джонсона, который является эффективным решением для нахождения кратчайших путей между всеми парами вершин, особенно для плотных графов и графов с отрицательными весами. Хотя на малых и разреженных графах он может уступать в производительности наивному алгоритму, его способность корректно обрабатывать отрицательные веса и хорошая масштабируемость на плотных графах делают его ценным инструментом для решения задач анализа графов. Алгоритм успешно прошел все тесты, подтвердив свою корректность, надежность и практическую применимость для решения реальных задач поиска кратчайших путей в графах со сложной структурой весов.

Список литературы

[1] *Гасфилд Дэн. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В.Романовского. — СПб.: Невский Диалект; БХВ Петербург, 2003. — 654 с.: ил.*

[2] Фундаментальные алгоритмы на C++ Роберт Седжвик Г7Г/ПИ/ торгово-иадатмьскнй дом 1Ж DiaSoft Москва • Санкт-Петербург •2002 (дата обращения: 10.10.2025).

[3] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е издание. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))