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:	К. А. Калугин
Преподаватель:	А. А. Кухтичев
Группа:	М8О-307Б
Дата:	
Оценка:	
Подпись:	

Москва, 2021

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти величину максимального потока в графе при помощи алгоритма Форда-Фалкерсона. Для достижения приемлемой производительности в алгоритме рекомендуется использовать поиск в ширину, а не в глубину. Истоком является вершина с номером 1, стоком – вершина с номером n. Вес ребра равен его пропускной способности. Граф не содержит петель и кратных ребер.

Формат входных данных: В первой строке заданы $1 \leqslant n \leqslant 2000$ и $1 \leqslant m \leqslant 10000$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от 0 до 1000000000.

Формат результата: Необходимо вывести одно число – искомую величину максимального потока. Если пути из истока в сток не существует, данная величина равна нулю.

1 Описание

Основная идея алгоритма Форда-Фалкерсона заключается в упрощении изначального графа, путем уменьшения весов его ребер в течение подсчета максимального потока.

2 Исходный код

Для представления графа будем использовать матрицу всех ребер и их весов. После этого ищем путь от истока до стока (используя поиск в ширину). При нахожении - находим ребро с наименьшим весом в этом пути и вычитаем этот вес из весов всех ребер пути. Таким образом, каждый раз находя путь, мы избавляемся от одного из ребер графа. В какой-то момент становится невозможно найти путь. Тогда мы выводим максимальный поток, равный сумме весов всех убранных ребер.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
using namespace std;

const int MAX_EDGE_COST = 1000000000; //
long long EK(const vector <vector <int>> & edges, vector <vector <int>> & weights) {//

    long long answer = 0;//
    bool f = true;//
    int n = edges.size () - 1;//
    while (f) {
        f = false;//
        queue <int> q;//
        vector <int> visited (n + 1);//
        q.push (1);//
        while (!q.empty ()) {//
            int t = q.front ();//
            q.pop (); //
            if (t == n) {//   -
                int mini = MAX_EDGE_COST;//
                int curr = n; //
                while (curr != 1) { //
                    mini = min (mini, weights [visited [curr]][curr]);//
                    curr = visited [curr];//
                }
                curr = n;
                while (curr != 1) {//
                    weights [visited [curr]][curr] -= mini;//
                    weights [curr][visited [curr]] += mini;//
                    curr = visited [curr];//
                }
                answer += mini;//
                f = true;//
                break;
            }
            for (int i = 0; i < edges [t].size (); ++ i) {//   ,
```

```
                if ((weights [t] [edges [t][i]] != 0) && (visited [edges [t][i]] == 0)) {//   ( )
                    visited [edges [t][i]] = t;//
                    q.push (edges [t][i]);// ,       .      -    .
                }
            }
        }
    }
    return answer;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector <vector <int>> edges (n + 1);//
    vector <vector <int>> weights (n + 1, vector <int> (n + 1));//
    for (int i = 0; i < m; ++ i) {//
        int v, u, cost; //
        cin >> v >> u >> cost; //
        edges [v].push_back(u);//
        edges [u].push_back(v);//
        weights [v][u] = cost;//   (   )
    }
    long long answer = EK (edges, weights);
    cout << answer << endl;
    return 0;
}
```

3 Консоль

```
PS C:\VSC\DA>.\a.exe
5 6
1 2 4
1 3 3
1 4 1
2 5 3
3 5 3
4 5 10
7
```

4 Тест производительности

Тест производительности представляет из себя сравнение времени работы алгоритма Эдмондса-Карпа с помощью поиска в ширину и алгоритма Форда-Фалкерсона с использованием поиска в глубину соответственно. Все тесты производились на полных графах с числом рёбер $m = n^2$, веса рёбер определялись случайно и имели значения от 1 до 10^9.

Число вершин полного графа n	Время работы алгоритма Эдмондса-Карпа	Время работы алгоритма Форда-Фалкерсона
5	53 мкс.	38 мкс.
10	220 мкс.	316 мкс.
20	1806 мкс.	10082 мкс.
40	30940 мкс.	246109 мкс.
80	123838 мкс.	4256952 мкс.

Как видно, время работы алгоритма Эдмондса-Карпа заметно быстрее алгоритма Форда-Фалкерсона с использованием поиска в глубину. Это связано с тем, что в ходе выбора увеличивающего пути алгоритм Форд-Фалкерсона может выбрать неоптимальный, который не является кратчайшим и поэтому может содержать использовавшиеся ранее обратные ребра с низкой стоимостью, отчего насыщение потока будет происходить очень медленно.

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ» я освежил в памяти знания, полученные на дискретно математике и узнал о способах работы с графами с точки зрения программирования.

Список литературы

[1] *Алгоритм Форда — Фалкерсона.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Форда_-_Фалкерсона` (дата обращения: 24.11.2021).