

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Построение и анализ алгоритмов»
Тема: Потоки в сети

Студент гр. 8304

Николаева М. А.

Преподаватель

Размочаева Н. В.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с алгоритмом поиска максимального потока в сети Форда-Фалкерсона, научиться оценивать временную сложность алгоритма и применять его для решения задач.

Постановка задачи.

Разработать программу, которая решает задачу нахождения максимального потока в сети, а также фактической величины потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона (для поиска пути в алгоритме используется алгоритм BFS, соответствует индивидуализации работы №1).

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).

Выходные рёбра требуется отсортировать в лексикографическом порядке по первой вершине, потом по второй (в результатах работы программы должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Описание алгоритма.

Изначально рассматриваем нулевой максимальный поток, начальный граф с ориентированными ребрами и вершину-исток, вершину-сток. Далее, пока существует путь из вершины-истока в вершину-сток:

- 1) Проходим по найденному пути с конца и находим минимальное значение потока в графе. Запоминаем его.
- 2) Проходим по найденному пути в обратную сторону и для каждого ребра изменяем значение потока на найденное ранее значение (пункт 1)).
- 3) Добавляем к значению максимального потока найденное ранее минимальное значение потока в графе.

Когда путей больше не найдено, алгоритм заканчивает свою работу и выводит значение максимального потока в сети, а также все ребра со значением фактического потока в них.

Для поиска пути используется алгоритм поиска в ширину. Поочередно обрабатываются вершины текущего фронта, вершины перебираются в алфавитном порядке.

Анализ алгоритма.

Временная сложность: $O(E * F)$, E – число ребер в графе, F – максимальный поток

Затраты памяти: $O(V+E)$, V – количество вершин, E – количество ребер

Описание функций и структур данных.

Для хранения графа используется структура `Graph`, которая является контейнером `map`, содержащим `<char, map<char, int>>`. Ключ – первая вершина ориентированного ребра, соответственно. По нему получаем доступ к паре `<вторая вершина ребра, значение пропускной способности>`.

```
using Graph = map<char, map<char, int>>;
```

```
bool breadthFirstSearch (Graph& graph, char start, char end, map<char, char>& path)
```

 – функция поиска пути для алгоритма Форда-Фалкерсона. Реализует поиск в ширину. Принимает граф, стартовую вершину, конечную вершину и ссылку на путь. Возвращает `true`, в случае, когда путь был обнаружен.

```
void FordFulkersonAlgorithm (Graph& graph, char start, char end)
```

 – функция, реализующая алгоритм поиска максимального потока в сети Форда-Фалкерсона. Возвращаемое значение отсутствует. Принимает граф, стартовую и конечную вершину. Находит значение максимального потока и выводит результат работы на экран.

Тестирование программы.

Результаты тестирования см. в табл. 1.

Таблица 1 - Результаты работы программы

Ввод	Вывод алгоритма
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	Network Max Flow Value: 12 Actual flow through a - b :6 Actual flow through a - c :6 Actual flow through b - d :6 Actual flow through c - f :8 Actual flow through d - e :2 Actual flow through d - f :4 Actual flow through e - c :2
5 a d a b 2 a c 3 c b 1 b d 3 c d 2	Network Max Flow Value: 5 Actual flow through a - b :2 Actual flow through a - c :3 Actual flow through b - d :3 Actual flow through c - b :1 Actual flow through c - d :2
5 a d b d 3 c d 1 a b 2 a c 3 c b 1	Network Max Flow Value: 4 Actual flow through a - b :2 Actual flow through a - c :2 Actual flow through b - d :3 Actual flow through c - b :1 Actual flow through c - d :1

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм поиска максимального потока в сети Форда-Фалкерсона (для поиска пути использован алгоритм поиска в ширину), дана оценка времени работы алгоритма.

ПРИЛОЖЕНИЕ А.

КОД РАБОТЫ.

main.cpp:

```
#include <limits.h>
#include <iostream>
#include <map>
#include <queue>

using namespace std;

using Graph = map<char, map<char, int>>>;
bool breadthFirstSearch(Graph& graph, char start, char end, map<char, char>& path);
void FordFulkersonAlgorithm(Graph& graph, char start, char end);

void FordFulkersonAlgorithm(Graph& graph, char start, char end) {
    Graph flowGraph = graph;
    char u, v;

    map<char, char> path;
    int maxFlow = 0;

    while (breadthFirstSearch(flowGraph, start, end, path)) {
        int delta = INT_MAX;
        for (v = end; v != start; v = path[v]) {
            u = path[v];
            delta = min(delta, flowGraph[u][v]);
        }
        for (v = end; v != start; v = path[v]) {
            u = path[v];
            flowGraph[u][v] -= delta;
            flowGraph[v][u] += delta;
        }
        maxFlow += delta;
    }

    cout << "Network Max Flow Value: " << maxFlow << endl;
    int flow;
    for (auto& vertex : graph) {
        char u = vertex.first;

        for (auto neighbor : graph[u]) {
            char v = neighbor.first;
            int throughput = neighbor.second;

            if (throughput - flowGraph[u][v] < 0) {
                flow = 0;
            }
            else {
                flow = throughput - flowGraph[u][v];
            }
            cout << "Actual flow through "<< u << " - " << v << " : " << flow
<< endl;
        }
    }
}
```

```

bool breadthFirstSearch(Graph& graph, char start, char end, map<char, char>& path)
{
    queue<char> queue;
    queue.push(start);

    map<char, bool> visited;
    visited[start] = true;

    while (!queue.empty()) {
        char vertex = queue.front();
        queue.pop();

        for (auto neighbor : graph[vertex]) {
            char v = neighbor.first;
            int throughput = neighbor.second;

            if (!(visited[v]) && throughput > 0) {
                queue.push(v);
                visited[v] = true;
                path[v] = vertex;
            }
        }
    }

    return visited[end];
}

int main() {
    Graph graph;
    char start, end, u, v;
    int throughput, vertexCount;

    cout << "Enter the number of edges" << endl;
    cin >> vertexCount;
    cout << "source" << endl;
    cin >> start;
    cout << "stock" << endl;
    cin >> end;

    cout << "Enter the edges of the graph" << endl;
    for (int i = 0; i < vertexCount; ++i) {
        cin >> u >> v >> throughput;
        graph[u][v] = throughput;
    }

    FordFulkersonAlgorithm(graph, start, end);

    return 0;
}

```