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. 7383 ___ Сычевский Р.А.

Преподаватель ___ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Реализовать и исследовать алгоритм Форда-Фалкерсона поиска максимального потока в сети.

Постановка задачи

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона. Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).

Ход работы

Была написана программа на языке программирования C++. Код представлен в приложении А.

Для представления сети используется класс, представляющий из себя список смежности. В списке хранятся ребра и поток через эти ребра. Функция `build_vert` строит граф по входным данным. Далее вызывается функция `sort_vert`, которая сортирует элементы графа в алфавитном порядке. Функция `find_way` находит пусть в сети по правилу: каждый раз выполняется переход по ребру, ведущему в вершину, имя которой в алфавите ближайшее к началу алфавита. Эта функция так же уменьшает поток во всех ребрах найденного пути на наименьшую пропускную способность ребра из найденного пути. Если на каком-то этапе пропускная способность ребра равна нулю, то оно не учитывается. Алгоритм заканчивает работу, когда невозможно найти путь из истока в сток.

Тестирование

Тестирование проводилось в Windows 10. По результатам тестирования были выявлены ошибки в коде. Тестовые случаи представлены в приложении Б.

Исследование алгоритма

На каждом шаге алгоритм добавляет поток увеличивающего пути к уже имеющемуся потоку. Следовательно, на каждом шаге алгоритм увеличивает поток по крайней мере на единицу, следовательно, он сойдётся не более чем за O(f) шагов, где f — максимальный поток в графе. Можно выполнить каждый шаг за время O(E), где E — число рёбер в графе, тогда общее время работы алгоритма ограничено O(f|E|).

Выводы

Был изучен алгоритм Форда-Фалкерсона поиска максимального потока в сети. Была реализована версия алгоритма на языке C++, исследована сложность алгоритма, по результатам сложность равна O(f|E|).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

class vertex{
    public:
        char m_name;
        vector<vertex*> next;
        vector<int> m_length;
        vector<int> flow;
        int painted;
        int painted2;

        void add_vert(char name, int length){
            vertex* tmp = new vertex;
            tmp->m_name = name;
            tmp-> painted = 0;
            next.push_back(tmp);
            m_length.push_back(length);
            flow.push_back(length);
        }
        void  add_vert(vertex* tmp, int length){
            next.push_back(tmp);
            m_length.push_back(length);
            flow.push_back(length);
        }
};
class edge{
    public:
        char start;
        char end;
        int flow;
};
void clear(vertex* current){
    current->painted = 0;
    current->painted2 = 0;
    if(!current->next.empty())
        for(int i = 0; i < current->next.size(); i++){
            if(current->next[i]->painted == 1)
                clear(current->next[i]);
            else
                continue;
        }
    return;
}
vertex* find_vert(vertex* tmp, char name){
    vertex* find = NULL;
    tmp->painted2 = 1;
    if(tmp->m_name == name){
        return tmp;
```

```
    }
    else
        if(!tmp->next.empty())
            for(int i = 0; i < tmp->next.size(); i++){
                if(tmp->next[i]->painted2 != 1)
                    find = find_vert(tmp->next[i], name);
                else
                    continue;
                if(find != NULL)
                    break;
            }
    return find;
}
void build_vert(vertex* tmp, vertex* start, char old_v, char new_v, int length){
    if(tmp->m_name == old_v){
        for(int i = 0; i < tmp->next.size(); i++)
            if(tmp->next[i]->m_name == new_v)
                return;
        vertex* find = find_vert(start, new_v);
        if(find == NULL){
            tmp->add_vert(new_v, length);
        }else{
            tmp->add_vert(find, length);
        }
    }
    tmp->painted = 1;
    if(!tmp->next.empty()){
        for(int i = 0; i < tmp->next.size(); i++)
            if(tmp->next[i]->painted != 1)
                build_vert(tmp->next[i], start, old_v, new_v, length);
            else
                continue;
    }
}
int comp(vertex* a, vertex* b){
    return a->m_name < b->m_name;
}
void sort_vert(vertex* current){
    current->painted = 1;
    int flag = 1;
    vertex* tmp;
    int tmp_int = 0;
    if(!current->next.empty())
    while(flag){
        flag = 0;
        for(int i = 0; i < current->next.size()-1; i++){
            if(current->next[i]->m_name < current->next[i+1]->m_name){
                tmp = current->next[i+1];
                tmp_int = current->flow[i+1];
                current->next[i+1] = current->next[i];
                current->flow[i+1] = current->flow[i];
                current->m_length[i+1] = current->m_length[i];
                current->next[i] = tmp;
                current->flow[i] = tmp_int;
                current->m_length[i] = tmp_int;
                flag = 1;
```

```
            }
        }
    }
    if(!current->next.empty())
        for(int i = 0; i < current->next.size(); i++)
            if(current->next[i]->painted != 1)
                sort_vert(current->next[i]);
            else
                continue;
}
void pr_vert(vertex* tmp, int depth){
    cout << tmp->m_name << "   ";
    if(!tmp->next.empty()){
        cout << tmp->flow[0];
        pr_vert(tmp->next[0], depth+1);
    }
    if(tmp->next.size() > 1)
        for(int i = 1; i < tmp->next.size(); i++){
            for(int j = 0; j <= depth; j++)
            pr_vert(tmp->next[i], depth+1);
        }
}
int find_way(vertex* current, char last, int min){
    if(current->m_name == last)
        return min;
    if(current->next.empty())
        return 0;
    int tmp_min = min;
    int weight = 0;
    current->painted = 1;
    for(int i = 0; i < current->next.size(); i++){
        if(current->next[i]->m_name == last)
            if(current->flow[i] != 0){
                if(current->flow[i] < min || min == 0)
                    tmp_min = current->flow[i];
                weight = find_way(current->next[i], last, tmp_min);
                current->flow[i] = current->flow[i] - weight;
                return weight;
            }
    }
    for(int i = 0; i < current->next.size(); i++){
        tmp_min = min;
        if(current->flow[i] == 0)
            continue;
        if(current->next[i]->painted == 1)
            continue;
        if(current->flow[i] < min || min == 0)
            tmp_min = current->flow[i];
        weight = find_way(current->next[i], last, tmp_min);
        if(weight == 0)
            continue;
        current->flow[i] = current->flow[i] - weight;
        return weight;
    }
    if(weight == 0)
        current->painted = 0;
```

```
    return weight;
}
int check_arr(vector<edge> &arr, edge tmp){
    int flag = 0;
    for(int i = 0; i < arr.size(); i++){
        if(arr[i].start == tmp.start && arr[i].end == tmp.end && arr[i].flow ==
tmp.flow)
            flag = 1;
    }
    return flag;
}
void full_edge(vertex* current, vector<edge> &arr){
    for(int i = 0; i < current->next.size(); i++){
        edge tmp;
        tmp.start = current->m_name;
        tmp.end = current->next[i]->m_name;
        tmp.flow = current->m_length[i]-current->flow[i];
        if(!arr.empty()){
            if(check_arr(arr, tmp))
                continue;
        }
        arr.push_back(tmp);
        full_edge(current->next[i], arr);
    }
}
void pr_edge(vector<edge> &arr){
    for(int i = 0; i < arr.size(); i++)
        cout << arr[i].start << ' ' << arr[i].end << ' ' << arr[i].flow << endl;
}
int edge_comp(edge a, edge b){
    if(a.start == b.start)
        return a.end < b.end;
    else
        return a.start < b.start;
}
void del_vertex(vertex* tmp){
    tmp->painted = 1;
    if(tmp->next.size()){
        for(int i = 0; i < tmp->next.size(); i++){
            if(tmp->next[i]->painted == 0)
                del_vertex(tmp->next[i]);
        }
        for(int i = tmp->next.size(); i > 0; i--){
            delete tmp->next[i];
            tmp->next.pop_back();
        }
    }
}
int main(){
    int count;
    cin >> count;
    if(count == 0)
        return 0;
    char first;
    cin >> first;
    char last;
```

```
    cin >> last;
    vertex* start = new vertex;
    start->m_name = first;
    start->painted = 0;
    char f;
    char l;
    int len;
    vector<edge> arr1;
    edge tmp_edge;
    for(int i = 0; i < count; i++){
        cin >> f >> l >> len;
        tmp_edge.start = f;
        tmp_edge.end = l;
        tmp_edge.flow = len;
        arr1.push_back(tmp_edge);
    }
    sort(arr1.begin(), arr1.end(), edge_comp);
    for(int i = 0; i < count; i++){
        build_vert(start, start, arr1[i].start, arr1[i].end, arr1[i].flow);
        clear(start);
    }
    sort_vert(start);
    clear(start);
    int flag = 1;
    int count_flow = 0;
    while(flag){
        flag = find_way(start, last, 0);
        count_flow += flag;
        clear(start);
    }
    cout << count_flow << endl;
    vector<edge> arr;
    full_edge(start, arr);
    sort(arr.begin(), arr.end(), edge_comp);
    pr_edge(arr);
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Результат
7 a f a b 8 b c 4 c d 3 d e 2 e b 1 e f 1 b f 6	7 a f a b 7 b c 1 c d 6 d e 1 e b 1 e f 0 b f 1
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
16 a e a b 2 b a 2 a d 1 d a 1 a c 3 c a 3 b c 4 c b 4 c d 1	6 a b 2 a c 3 a d 1 b a 0 b c 0 b e 3 c a 0 c b 1 c d 0 c e 2 d a 0

d c 1 c e 2 e c 2 b e 3 e b 3 d e 1 e d 1	d c 0 d e 1 e b 0 e c 0 e d 0