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

ОТЧЕТ

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

Студент гр.8382 _____ Синельников М.Р

Преподаватель _____ Фирсов М.А

Санкт-Петербург
2020

Цель работы.

Научиться реализовывать алгоритм нахождения максимального потока в сети.

Задание.

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).

Входные данные:

N — количество ориентированных рёбер графа

v0 — исток

vn — сток

vi vj wij — ребро графа

vi vj wij — ребро графа

…

Выходные данные:

Pmax — величина максимального потока

vi vj wij — ребро графа с фактической величиной протекающего потока

vi vj wij — ребро графа с фактической величиной протекающего потока

…

В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Sample Input:

```
7
a
f
a b 7
a c 6
```

b d 6
c f 9
d e 3
d f 4
e c 2

Sample Output:

12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2

вариант 1 - поиск в ширину. Поочерёдная обработка вершин текущего фронта, перебор вершин в алфавитном порядке.

Описание алгоритма.

Ввод данных происходит в функции *input.* Считав данные, запускается функция *FordFulkerson,* реализующая основную идею алгоритма. Изначально поток для всех ребёр равен нулю, а для противоложных рёбер равняется пропускной способности исходного ребра. В цикле *while* ищутся все возможные пути, по которым можно пустить поток. Нахождение пути происходит с помощью обхода в ширину с обработкой вершин в алфавитном порядке, поэтому в данном случае применяется не обычная очередь, а очередь с приоритетом, где приоритет определяется номером символа вершины. Найдя путь, функция *computeMinCapacity* вычисляет значение минимальной пропускной способности на найденном пути, а в функции *changeFlow* значение потока для каждого ребра на найденном пути увеличивается на минимальную пропускную способность, в то время как для противоположного ему ребра уменьшается. Если поток по ребру становится равным пропускной способности этого ребра, то ребро помечается как посещённое, что означает, что в нахождении очередного пути данное ребро принимать участия не будет. Также увеличивается значение переменной *max_flow* на величину минимальной

пропускной способности. Цикл заканчивает работу, когда очередной путь от истока к стоку найти нельзя. После завершения цикла на экран печатается величина максимального потока по сети, а функция *formOutput* печатает найденный поток через каждое ребро.

Граф представлен в виде *vector<map<int, vector<Edge>>>,* где размер первого вектора равняется количеству вершин в графе, словарь хранит характеристики рёбер между вершиной i и всеми смежными с ней вершинами, а второй вектор необходим для случая, если между двумя вершинами более одного ребра. Характеристики ребра представлены структурой *Edge*. В данном алгоритме используются два графа — исходный и граф из обратных рёбер.

Так как для поиска пути используется поиск в ширину, то в данном случае мы имеем дело с алгоритмом Эдмондса-Карпа, который является частным случаем алгоритмом Форда-Фалкерсона. Сложность этого алгоритма по времени O(|V| * |E|^2). С учётом того, что в данном задании используется очередь с приоритетом, реализованная в виде двоичной кучи, для формирования вершин в алфовитном порядке, итоговая сложность алгоритма по времени равняется O(|V| * log|V| * |E|^2). Сложность алгоритма по памяти — O(|V| * |E|).

Описание функций и структур данных.

void foldFurkenson(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes, map<int,string> &convert_to_string,map<string,int> &convert_to_int,int source,int sink) — основная функция алгоритма. В ней реализован цикл, в котором ищутся все возможные пути. *vector<map<int,vector<Edge> >> nodes* — структура данных для хранения графа, *vector<map<int,vector<Edge> >> reversed_nodes* — структура данных для хранения графа с обратными рёбрами, *map<string, int> &convert_to_int* — словарь для перевода символьного обозначения вершины в её номер. *map<int,string> &convert_to_string* — словарь, переводящий номер вершины в её символьное обозначение.

void changeFlow(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes, vector<pair<pair<int,int>,int>> &prev,int sink,int min) — функция, в которой происходит изменение потока для каждой вершины на найденном пути. *vector<map<int,vector<Edge> >> nodes* — структура данных для хранения графа, *vector<map<int,vector<Edge> >> reversed_nodes* — структура данных для хранения графа с обратными рёбрами, *vector<pair<pair<int,int>, int>> &prev* — вектор для хранения предыдущей вершины.

int computeMinCapacity(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,vector<pair<pair<int,int>,int>> &prev,int sink, unsigned int max_possible_flow) — функция возвращает мининимальную пропускную способность на найденном пути. *vector<map<int,vector<Edge> >> nodes* — структура данных для хранения графа, *vector<map<int,vector<Edge> >> reversed_nodes* — структура данных для хранения графа с обратными рёбрами, *vector<pair<pair<int,int>, int>> &prev* — вектор для хранения предыдущей вершины при формировании пути, *int sink* — номер стока, *unsigned int max_possible_flow* — значение максимального возможного потока через какое-либо ребро.

void emptyQueue(priority_queue<string,vector<string>,greater<string>> &neigbours) — функция для освобождения кучи. *priority_queue<string,vector<string>,greater<string>> &neigbours* — очередь с приоритетом для поиска в ширину.

void zeroInitialization(vector<bool> &visited) — функция для инициализации каждой вершины как ещё непосещённой. *vector<bool> &visited* — булевский массив для пометки посещённых вершин.

void input(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,map<int,string> &convert_to_string,map<string,int> &convert_to_int,int number_of_edges) — функция для ввода входных данных. *vector<map<int,vector<Edge> >> nodes* —

структура данных для хранения графа, *vector<map<int,vector<Edge> >> reversed_nodes* — структура данных для хранения графа с обратными рёбрами, *map<string, int> &convert_to_int* — словарь для перевода символьного обозначения вершины в её номер. *map<int,string> &convert_to_string* — словарь, переводящий номер вершины в её символьное обозначение, *int number_of_edges* — количество рёбер в графе.

unsigned int computeMaxFlow*(vector<map<int,vector<Edge>>> &nodes)* — возвращает максимальный возможный поток через какую-либо вершину. *vector<map<int,vector<Edge> >> nodes* — структура данных для хранения графа.

void formAdjacencList(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,map<string,int> &convert_to_int,vector<inputElement> &input_sequence) — функция для формирования списка смежности вершин по полученной входной последовательности. *vector<map<int,vector<Edge> >> nodes* — структура данных для хранения графа, *vector<map<int,vector<Edge> >> reversed_nodes* — структура данных для хранения графа с обратными рёбрами, *map<string, int> &convert_to_int* — словарь для перевода символьного обозначения вершины в её номер, *vector<inputElement> &input_sequence* — вектор для хранения последовательности входных рёбер.

void printCurrentWay(map<int,string> &convert_to_string,vector<pair<pair<int,int>,int>> &prev,int curr_vertex,bool flag,int source) - функция для вывода промежуточного построенного пути. *map<int,string> &convert_to_string* — словарь, переводящий номер вершины в её символьное обозначение, *vector<pair<pair<int,int>, int>> &prev* — вектор для хранения предыдущей вершины при формировании пути, *int curr_vertex* — номер текущей вершины, *bool flag* — значении *false* сигнализирует, что выводится промежуточный маршрут, *true* — что выводится полной путь от истока в сток.

bool sortbyalf(const inputElement &a, const inputElement &b) — функция для сортировки рёбер при выводе результата. Возвращает 1, если *a < b,* 0 в обратном случае, *const inputElement &a, const inputElement &b* — ссылки на рёбра.

void formOutput(vector<map<int,vector<Edge>>> &nodes,map<int,string> &convert_to_string) — функция для вывода потока через каждое ребро. *vector<map<int,vector<Edge> >> nodes* — структура данных для хранения графа, *map<int,string> &convert_to_string* — словарь, переводящий номер вершины в её символьное обозначение.

vector<map<int,vector<Edge> >> nodes — структура данных для хранения графа.

vector<map<int,vector<Edge> >> reversed_nodes — структура данных для хранения графа с обратными рёбрами.

map<string, int> &convert_to_int — словарь для перевода символьного обозначения вершины в её номер.

map<int,string> &convert_to_string — словарь, переводящий номер вершины в её символьное обозначение.

vector<pair<pair<int,int>, int>> &prev — вектор, хранящий в *prev[i].first.first* номер предыдущей вершины для вершины с номером *i,* через которую проходит путь,в поле *prev[i].first.second* индекс ребра, по которому был проложен путь, а в поле *prev[i].second* 1 или 0, в зависимости от того, принадлежит ребро исходному графу или графу с обратными рёбрами.

priority_queue<string,vector<string>, greater<string>> neigbours — очередь с приоритетом для обработки вершин в алфавитном порядке.

vector<inputElement> &input_sequence — вектор для хранения последовательности входных рёбер.

```
struct inputElement{
string vertex1;
string vertex2;
int weight;
```

}; - структура для хранения элемента входной последовательности: *string vertex1, string vertex2* — обозначения вершин, *int weight* — поток через ребро между этими вершинами.

```
struct Edge{
    int capacity;
    int flow = 0;
    bool visited = false;
```

}; - структура для хранения показателей ребра: *int capacity* — текущая проводимость ребра, *int flow* — текущий поток через ребро, *bool visited* — состояние о том, можно ли пройти по этому ребру или нет.

Тестирование.

Программа была протестирована на следующих исходных данных:

Входные данные	Результат
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
7 1 5 1 2 2 2 5 5 1 3 6 3 4 2 4 5 1 3 2 3 2 4 1	6 1 2 2 1 3 4 2 4 0 2 5 5 3 2 3 3 4 1 4 5 1
5 one five one five 2 one two 2 two three 3 three four 2	4 four five 2 one five 2 one two 2 three four 2 two three 2

four five 2	
5 1 4 1 2 10000 1 3 10000 2 3 1 3 4 10000 2 4 10000	20000 1 2 10000 1 3 10000 2 3 0 2 4 10000 3 4 10000
5 1 2 1 1 10000 1 2 1 1 2 4 1 2 100 2 1 900	Ищем очередной путь: Положили в очередь вершину 1 - исток Достали из очереди вершину 1 Положили в очередь вершину 2 Дошли до стока Найденный путь: 1 2 Текущий возможный поток через сеть - 1 Ищем очередной путь: Положили в очередь вершину 1 - исток Достали из очереди вершину 1 Положили в очередь вершину 2 Дошли до стока Найденный путь: 1 2 Текущий возможный поток через сеть - 5 Ищем очередной путь: Положили в очередь вершину 1 - исток Достали из очереди вершину 1 Положили в очередь вершину 2 Дошли до стока Найденный путь: 1 2 Текущий возможный поток через сеть - 105 Ищем очередной путь: Положили в очередь вершину 1 - исток Достали из очереди вершину 1 Больше путей не найти Максимальный поток: 105 Потоки по всем рёбрам: 1 1 0 1 2 1 1 2 4 1 2 100 2 1 0

Выводы.

Были получены навыки работы с алгоритмами поиска максимального потока в графу. В частности, был реализован модифицированный алгоритм

Эдмонда-Карпа с приоритетом обработки вершин .в лексиграфическом порядке, который, в свою очередь, является частным случаем алгоритма Форда-Фалкерсона.

Приложения А. Исходный код программы

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <queue>
#include <stack>
#include <string>

using std::cin;
using std::cout;
using std::vector;
using std::map;
using std::sort;
using std::queue;
using std::pair;
using std::make_pair;
using std::priority_queue;
using std::stack;
using std::string;
using std::greater;

struct inputElement{
    string vertex1;
    string vertex2;
    int weight;
};

struct Edge{
    int capacity;
    int flow = 0;
    bool visited = false;
};


bool sortbyalf(const inputElement &a, const inputElement &b){//сортировака вершин для вывода в алфавитном порядке
    if(a.vertex1 != b.vertex1)
        return (a.vertex1 < b.vertex1);
```

```
    return (a.vertex2 < b.vertex2);
}

void formAdjacencList(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,map<string,int> &convert_to_int,vector<inputElement> &input_sequence){//формируем список смежности
    int vertex,dest;
    int cap;
    Edge edge;
    for(int i = 0;i < input_sequence.size();i++){//проходим по всем заданным рёбрам
        vertex = convert_to_int[input_sequence[i].vertex1];//вершину1 перевод из буквенного обозначения в числовое
        dest = convert_to_int[input_sequence[i].vertex2];//вершину2 перевод из буквенного обозначения в числовое
        cap  = input_sequence[i].weight;
        edge.capacity = cap;
        edge.visited = false;
        edge.flow = 0;
       //добавляе  вершину2 в список вершины1
        nodes[vertex][dest].push_back(edge);
        edge.flow = edge.capacity;
        edge.capacity = 0;
        edge.visited = true;
        reversed_nodes[dest][vertex].push_back(edge);
    }
}

void printCurrentWay(map<int,string> &convert_to_char,vector<pair<pair<int,int>,int>> &prev,int curr_vertex,bool flag,int source){//выводим текущий маршрут
    stack<int> way;
    int stack_size;
    int i = curr_vertex;
    if(flag)
        cout << "Найденный путь: \n";
    else
        cout << "Один из текущих маршрутов: \n";
    while (prev[i].first.first != -1){
        way.push(i);
        i = prev[i].first.first;
    }
    way.push(source);
    stack_size = way.size();
    for(int i = 0;i < stack_size;i++){//печатаем найденный путь
        cout << convert_to_char[way.top()] << " ";
```

```
        way.pop();
    }
    cout << "\n";
}

void formOutput(vector<map<int,vector<Edge>>> &nodes,map<int,string> &convert_to_string){
    cout << "Потоки по всем рёбрам: \n";
    vector<inputElement> output;
    inputElement element;
    for(int i = 0;i < nodes.size();i++){
        for (auto it = nodes[i].begin(); it != nodes[i].end(); ++it){//каждое ребро формируем как элемент структуры
            for(int j = 0;j < nodes[i][it->first].size();j++) {
                element.vertex1 = convert_to_string[i];
                element.vertex2 = convert_to_string[it->first];
                element.weight = nodes[i][it->first][j].flow;
                output.push_back(element);
            }
        }
    }
    sort(output.begin(),output.end(),sortbyalf);
    for(int i = 0;i < output.size();i++)//выводим поток через каждое ребро
        cout << output[i].vertex1 << " " << output[i].vertex2 << " " << output[i].weight << "\n";
}

unsigned int computeMaxFlow(vector<map<int,vector<Edge>>> &nodes){//вычисление значение максимального потока через какое-либо ребро
    unsigned int max = 0;
    for (int i = 0; i < nodes.size() ; ++i) {
        for (auto it = nodes[i].begin(); it != nodes[i].end(); ++it) {
            for(int j = 0;j < nodes[i][it->first].size();j++)
                if (nodes[i][it->first][j].capacity > max)
                    max = nodes[i][it->first][j].capacity;
        }
    }
    return max;
}

void input(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,map<int,string> &convert_to_string,map<string,int> &convert_to_int,int number_of_edges){//ввод входных данных
    vector<inputElement> input_sequnece;//вектор рёбер графа
    inputElement elem;
```

```
    string vertex1,vertex2;
    vertex1 = ' ';
    int capacity;
    int counter = 0;
    int i = 0;
    while (i < number_of_edges){//цикл для ввода рёбер графа
        cin >> vertex1;
        cin >> vertex2;
        cin >> capacity;
        elem.vertex1 = vertex1;
        elem.vertex2 = vertex2;
        elem.weight = capacity;
        input_sequnece.push_back(elem);
         if(convert_to_int.find(vertex1) == convert_to_int.end()) {//если вершина ещё не встречалась, записываем её в
map convert_to_int.find
            convert_to_int[vertex1] = counter;
            convert_to_string[counter] = vertex1;
            counter++;
        }
        if(convert_to_int.find(vertex2) == convert_to_int.end()) {
            convert_to_int[vertex2] = counter;
            convert_to_string[counter] = vertex2;
            counter++;
        }
        i++;
    }
    nodes.resize(counter);//задаеём размер векттора
    reversed_nodes.resize(counter);//задаём размер вектора
     formAdjacencList(nodes,reversed_nodes,convert_to_int,input_sequnece);//вызываем функцию для формирования
списка смежности
}

void zeroInitialization(vector<bool> &visited){//инициализируем вершины как ещё не посещённые
    for(int i = 0;i < visited.size();i++)
        visited[i] = false;
}

void emptyQueue(priority_queue<string,vector<string>,greater<string>> &neigbours){//опустошаем кучу
    while (!neigbours.empty())
        neigbours.pop();
}
```

```
int computeMinCapacity(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,vector<pair<pair<int,int>,int>> &prev,int sink, unsigned int max_possible_flow){//вычисляем минималбную пропускную способность
  int min,i;
  min = max_possible_flow;
  i = sink;
  while (1){//проходим найденный путь от стока к истоку
    if(prev[i].first.first == -1)//выходим, когда дошли до стока
      break;
     if((prev[i].second && nodes[prev[i].first.first][i][prev[i].first.second].capacity < min) || (!prev[i].second && reversed_nodes[prev[i].first.first][i][prev[i].first.second].capacity < min))
      if(prev[i].second)
        min = nodes[prev[i].first.first][i][prev[i].first.second].capacity;
      else
        min = reversed_nodes[prev[i].first.first][i][prev[i].first.second].capacity;
    i = prev[i].first.first;//переходим к следующей вершине на пути
  }
  return min;//возвращаем мин пропускную способность
}

void changeFlow(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes, vector<pair<pair<int,int>,int>> &prev,int sink,int min){//меняем поток для рёбер на найденном пути
  int i;
  i = sink;
  while (1){
    if(prev[i].first.first == -1)//выходим, когда дошли до стока
      break;
    if(prev[i].second) {// обрабатываем случай, когда прошли по обычному ребру
      nodes[prev[i].first.first][i][prev[i].first.second].flow += min;
      nodes[prev[i].first.first][i][prev[i].first.second].capacity -= min;
      reversed_nodes[i][prev[i].first.first][prev[i].first.second].flow -= min;
      reversed_nodes[i][prev[i].first.first][prev[i].first.second].capacity += min;
      if(!nodes[prev[i].first.first][i][prev[i].first.second].capacity)
        nodes[prev[i].first.first][i][prev[i].first.second].visited = true;
      if(reversed_nodes[i][prev[i].first.first][prev[i].first.second].capacity)
        reversed_nodes[i][prev[i].first.first][prev[i].first.second].visited = false;
    } else{//обрабатываем случай, когда по обратному ребру
      reversed_nodes[prev[i].first.first][i][prev[i].first.second].flow += min;
      reversed_nodes[prev[i].first.first][i][prev[i].first.second].capacity -= min;
      nodes[i][prev[i].first.first][prev[i].first.second].flow -= min;
      nodes[i][prev[i].first.first][prev[i].first.second].capacity += min;
      if(!reversed_nodes[prev[i].first.first][i][prev[i].first.second].capacity)
```

```
            reversed_nodes[prev[i].first.first][i][prev[i].first.second].visited = true;
        if(nodes[i][prev[i].first.first][prev[i].first.second].capacity)
            nodes[i][prev[i].first.first][prev[i].first.second].visited = false;
    }
    i = prev[i].first.first;//переходим к следующей вершине на пути
  }
}

void foldFurkenson(vector<map<int,vector<Edge>>> &nodes,vector<map<int,vector<Edge>>> &reversed_nodes,
map<int,string> &convert_to_string,map<string,int> &convert_to_int,int source,int sink){
  unsigned int max_possible_flow = computeMaxFlow(nodes);
  unsigned int max_flow = 0;
  int min_capacity;
  bool check;
  priority_queue<string,vector<string>, greater<string>> neigbours;
  vector<pair<pair<int,int>,int>> prev(nodes.size());
  vector<bool> visited(nodes.size());
  unsigned long size;
  int curr_vertex;
  prev[source] = make_pair(make_pair(-1,0),0);
  while (1) {//пока можно найти путь
    cout << "Ищем очередной путь: \n";
    neigbours.push(convert_to_string[source]);
    cout << "Положили в очередь вершину " << convert_to_string[source] << " - исток\n";
    check = false;
    zeroInitialization(visited);
    visited[source] = true;
    while (!neigbours.empty()) {
      size = neigbours.size();
      for (int i = 0; i < size; ++i) {
        curr_vertex = convert_to_int[neigbours.top()];
        cout << "Достали из очереди вершину " << convert_to_string[curr_vertex] << "\n";
        neigbours.pop();
          for (auto it = nodes[curr_vertex].begin(); it != nodes[curr_vertex].end(); ++it){//обрабатываем смежные
вершины с текущей по исходному графу
            for(int j = 0;j < nodes[curr_vertex][it->first].size();j++) {
              if (!visited[it->first] && !nodes[curr_vertex][it->first][j].visited) {//проверка на то, можно ли пройти
по ребру и посещена ли уже вершина
                neigbours.push(convert_to_string[it->first]);
                cout << "Положили в очередь вершину " << convert_to_string[it->first] << "\n";
                prev[it->first] = make_pair(make_pair(curr_vertex,j), 1);
                visited[it->first] = true;
```

```
                    if (it->first == sink) {//если сток, заканчиваем поиск пути
                        cout << "Дошли до стока\n";
                        printCurrentWay(convert_to_string, prev, sink, true, source);
                        check = true;
                        break;
                    } else
                        printCurrentWay(convert_to_string, prev, it->first, false, source);
                }
            }
        }
        if (check)
            break;
                for (auto it = reversed_nodes[curr_vertex].begin(); it != reversed_nodes[curr_vertex].end(); ++it) {//обрабатываем смежные вершины с текущей по обратному графу
            for(int j = 0;j < reversed_nodes[curr_vertex][it->first].size();j++) {
                if (!visited[it->first] && !reversed_nodes[curr_vertex][it->first][j].visited) {//проверка на то, можно ли пройти по ребру и посещена ли уже вершина
                    neigbours.push(convert_to_string[it->first]);
                    cout << "Положили в очередь вершину " << convert_to_string[it->first] << "\n";
                    prev[it->first] = make_pair(make_pair(curr_vertex,j), 0);
                    visited[it->first] = true;
                    if (it->first == sink) {//если сток, заканчиваем поиск пути
                        cout << "Дошли до стока\n";
                        printCurrentWay(convert_to_string, prev, sink, true, source);
                        check = true;
                        break;
                    } else
                        printCurrentWay(convert_to_string, prev, it->first, false, source);
                }
            }
        }
        if (check)
            break;
    }
    if (check)
        break;
}
if(neigbours.empty())
    break;
        min_capacity = computeMinCapacity(nodes,reversed_nodes,prev,sink,max_possible_flow);//вычисляем минимальную пропускную способность
changeFlow(nodes,reversed_nodes,prev,sink,min_capacity);//меняем потоки по рёбрам в найденном пути
```

```
        max_flow += min_capacity;//увеличиваем максимальный поток
        cout << "Текущий возможный поток через сеть; " << max_flow << "\n";
        emptyQueue(neigbours);
    }
    cout << "Больше путей не найти\n";
    cout << "Максимальный поток: " << max_flow << "\n";
    formOutput(nodes,convert_to_string);//формируем выход
}

int main() {
    string v1,v2;
    int number_of_edges;
    int source,sink;
    vector<map<int,vector<Edge>>> nodes;
    vector<map<int,vector<Edge>>> reversed_nodes;
    map<int,string> convert_to_string;
    map<string,int> convert_to_int;
    cout << "Введите количество рёбер: ";
    cin >> number_of_edges;
    cout << "Введите исток и сток: ";
    cin >> v1  >> v2;
    if(!number_of_edges)
        cout << "Максимальный поток: " << 0;
    else {
        cout << "Вводите рёбра: ";
        input(nodes, reversed_nodes, convert_to_string, convert_to_int, number_of_edges);
        source = convert_to_int[v1];
        sink = convert_to_int[v2];
        foldFurkenson(nodes, reversed_nodes, convert_to_string, convert_to_int, source, sink);
    }
    return 0;
}
```