**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**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 8304 |  | Мельникова О.А. |
| Преподаватель |  | Размочаева Н. В. |

Санкт-Петербург

2020

**Цель работы.**

Реализовать алгоритм Форда-Фалкерсона, найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

**Вариант 3.** Поиск в глубину. Рекурсивная реализация.

**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 c 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).

**Описание алгоритма.**

На каждой итерации поиска в глубину пути от истока к стоку рассматриваются разные варианты прохода, рассматриваются не посещенные вершины. Если для дуги верно, что пропускная способность больше потока, то узел к которому она ведет можно рассматривать. Если алгоритм нашел цепь, и ребро не обратное, то уменьшаем пропускные способности всех ребер на максимальную величину, на которую можно увеличить поток по каждой дуге (наименьшее ребро), если ребро обратное, к ребрам пути от стока к истоку вес минимального ребра прибавляется (если такой вершины не существует, то она достраивается). Максимальный поток в графе, увеличивается на вес минимального ребра пути. Фактический поток через ребра определяется как разность между первоначальным ребром и ребром, после преобразований. В консоль выводится результат работы алгоритма и промежуточные результаты.

Сложность алгоритма по операциям: O (E \* F), E – число ребер в графе,

F – максимальный поток

Сложность алгоритма по памяти: O (N+E), N – количество вершин,

E – количество ребер

**Описание функций и структур данных.**

|  |  |
| --- | --- |
| struct **Edge** {  char end{}; //куда входит ребро  int bandwidth{}; //max пропуск. способность  int flow{}; //поток в сети  bool is\_reverse = false; //обратное ли  }; | Хранит данные о ребрах |
| struct Vertex {  std::vector<Edge> destinations;  }; | Хранит всех детей вершины |
| std::map<char, Vertex> dictOfVertex; | Словарь вершин |
| std::map<char, bool> visited; | Словарь посещенных вершин |
| void input(); | Функция считывания данных в dictOfVertex |
| int bfs(char currVertex /\*текущая вершина\*/, int c\_min /\*минимальный поток в сети\*/, std::string tab); | Рекурсивная функция поиска пути в графе. |

**Тестирование.**

|  |  |
| --- | --- |
| Входные данные:  7  a  f  a b 7  a c 6  b d 6  c f 9  d e 3  d f 4  e c 2 | Результат работы программы:  Максимальный поток 0  Текущая вершина a  Цикл по детям a  Ребро ведет в c; П/С пути 6; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина c  Цикл по детям c  Ребро ведет в f; П/С пути 9; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина f  Текущая вершина - сток  Возвращаемое значение 6  Возвращаемое значение 6  Максимальный поток 6  Текущая вершина a  Цикл по детям a  Ребро ведет в c; П/С пути 6; поток через ребро 6  Ребро не посещенное  Ребро ведет в b; П/С пути 7; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина b  Цикл по детям b  Ребро ведет в d; П/С пути 6; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина d  Цикл по детям d  Ребро ведет в e; П/С пути 3; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина e  Цикл по детям e  Ребро ведет в c; П/С пути 2; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина c  Цикл по детям c  Ребро ведет в f; П/С пути 9; поток через ребро 6  Ребро не посещенное  Ребро не обратное  Текущая вершина f  Текущая вершина - сток  Возвращаемое значение 2  Возвращаемое значение 2  Возвращаемое значение 2  Возвращаемое значение 2  Возвращаемое значение 2  Максимальный поток 8  Текущая вершина a  Цикл по детям a  Ребро ведет в c; П/С пути 6; поток через ребро 6  Ребро не посещенное  Ребро ведет в b; П/С пути 7; поток через ребро 2  Ребро не посещенное  Ребро не обратное  Текущая вершина b  Цикл по детям b  Ребро ведет в d; П/С пути 6; поток через ребро 2  Ребро не посещенное  Ребро не обратное  Текущая вершина d  Цикл по детям d  Ребро ведет в e; П/С пути 3; поток через ребро 2  Ребро не посещенное  Ребро не обратное  Текущая вершина e  Цикл по детям e  Ребро ведет в c; П/С пути 2; поток через ребро 2  Ребро не посещенное  Ребро ведет в e; П/С пути 2; поток через ребро 0  Ребро посещенное  Ребро ведет в d; П/С пути 3; поток через ребро 0  Ребро посещенное  Ребро ведет в f; П/С пути 4; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина f  Текущая вершина - сток  Возвращаемое значение 4  Возвращаемое значение 4  Возвращаемое значение 4  Максимальный поток 12  Текущая вершина a  Цикл по детям a  Ребро ведет в c; П/С пути 6; поток через ребро 6  Ребро не посещенное  Ребро ведет в b; П/С пути 7; поток через ребро 6  Ребро не посещенное  Ребро не обратное  Текущая вершина b  Цикл по детям b  Ребро ведет в d; П/С пути 6; поток через ребро 6  Ребро не посещенное  12  a b 6  a c 6  b d 6  c f 8  d e 2  d f 4  e c 2 |
| Входные данные:  8  a  h  a b 5  a c 4  a d 1  b g 1  c e 2  c f 3  d e 6  e h 4  f h 4  g h 8 | Результат работы программы:  Максимальный поток 0  Текущая вершина a  Цикл по детям a  Ребро ведет в d; П/С пути 1; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина d  Цикл по детям d  Ребро ведет в e; П/С пути 6; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина e  Цикл по детям e  Ребро ведет в c; П/С пути 2; поток через ребро 0  Ребро не посещенное  Ребро ведет в h; П/С пути 4; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина h  Текущая вершина - сток  Возвращаемое значение 1  Возвращаемое значение 1  Возвращаемое значение 1  Максимальный поток 1  Текущая вершина a  Цикл по детям a  Ребро ведет в d; П/С пути 1; поток через ребро 1  Ребро не посещенное  Ребро ведет в c; П/С пути 4; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина c  Цикл по детям c  Ребро ведет в e; П/С пути 2; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина e  Цикл по детям e  Ребро ведет в c; П/С пути 2; поток через ребро 0  Ребро посещенное  Ребро ведет в h; П/С пути 4; поток через ребро 1  Ребро не посещенное  Ребро не обратное  Текущая вершина h  Текущая вершина - сток  Возвращаемое значение 2  Возвращаемое значение 2  Возвращаемое значение 2  Максимальный поток 3  Текущая вершина a  Цикл по детям a  Ребро ведет в d; П/С пути 1; поток через ребро 1  Ребро не посещенное  Ребро ведет в c; П/С пути 4; поток через ребро 2  Ребро не посещенное  Ребро не обратное  Текущая вершина c  Цикл по детям c  Ребро ведет в e; П/С пути 2; поток через ребро 2  Ребро не посещенное  Ребро ведет в f; П/С пути 3; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина f  Цикл по детям f  Ребро ведет в c; П/С пути 3; поток через ребро 0  Ребро посещенное  Ребро ведет в b; П/С пути 5; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина b  Цикл по детям b  Ребро ведет в g; П/С пути 1; поток через ребро 0  Ребро не посещенное  Ребро не обратное  Текущая вершина g  Цикл по детям g  Ребро ведет в b; П/С пути 1; поток через ребро 0  Ребро посещенное  3  a b 0  a c 2  a d 1  b g 0  c e 2  c f 0  d e 1  e h 3 |

**Выводы.**

В ходе выполнения лабораторной работы был реализован алгоритм Форда-Фалкерсона, который находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

**Приложение A.  
Исходный код**

#include <iostream>

#include <map>

#include <vector>

#include <algorithm>

#include <fstream>

char begin, end;

struct **Edge** {

char end{}; //куда входит ребро

int bandwidth{}; //max пропуск. способность

int flow{}; //поток в сети

bool is\_reverse = false; //обратное ли

};

struct Vertex { //здесь все дети вершины

std::vector<Edge> destinations;

};

std::map<char, Vertex> dictOfVertex; //словарь вершин

std::map<char, bool> visited;

void **input**(){

int N;

std::cin >> N; //кол-во ориентированных ребер графа

std::cin >> begin; //исток

std::cin >> end; //сток

Vertex tmp = Vertex();

char startVertex, endVertex;

int length;

Edge edge{};

for(int i = 0; i < N; ++i){

std::cin >> startVertex >> endVertex >> length;

bool f = false;

for(auto& k : dictOfVertex[startVertex].destinations) { //прохождение по всем детям вершины

if(k.end == endVertex ){ //если уже есть конец

k = {endVertex, length, 0, false};

f = true;

break;

}

}

if(f) //все заполнено

continue;

if(dictOfVertex.find(startVertex) == dictOfVertex.end()) { //если нет начала

edge = {endVertex, length, 0};

tmp.destinations.push\_back(edge); //доб. ребро в массив детей вершины

dictOfVertex[startVertex] = tmp; // вершину доб. в словарь вершин

visited[startVertex] = false;

}

else { //если есть начало

edge = {endVertex, length, 0};

dictOfVertex[startVertex].destinations.push\_back(edge); //добавляем ребро к уже существ.

}

tmp.destinations.clear();

if(startVertex != begin) { //если не исток

if(dictOfVertex.find(endVertex) == dictOfVertex.end()) { //если нет конца

edge = {startVertex, length, 0, true};

tmp.destinations.push\_back(edge); //доб. ребро в массив детей вершины

dictOfVertex[endVertex] = tmp; // вершину доб. в словарь вершин

visited[startVertex] = false;

}

else{ //если конец есть

edge = {startVertex, length, 0, true};

dictOfVertex[startVertex].destinations.push\_back(edge); //добавляем ребро к уже существ.

}

tmp.destinations.clear();

}

}

for(auto& i : dictOfVertex) {

std::sort(i.second.destinations.begin(), i.second.destinations.end(), [](Edge e1, Edge e2){return e1.bandwidth < e2.bandwidth;});

}

}

void input(std::string argv) {

int N;

std::ifstream fin;

fin.open(argv);

fin >> N; //кол-во ориентированных ребер графа

fin >> begin; //исток

fin >> end; //сток

Vertex tmp = Vertex();

char startVertex, endVertex;

int length;

Edge edge{};

for(int i = 0; i < N; ++i){

fin >> startVertex >> endVertex >> length;

bool f = false;

for(auto& k : dictOfVertex[startVertex].destinations) { //прохождение по всем детям вершины

if(k.end == endVertex ){ //если уже есть конец

k = {endVertex, length, 0, false};

f = true;

break;

}

}

if(f) //все заполнено

continue;

if(dictOfVertex.find(startVertex) == dictOfVertex.end()) { //если нет начала

edge = {endVertex, length, 0};

tmp.destinations.push\_back(edge); //доб. ребро в массив детей вершины

dictOfVertex[startVertex] = tmp; // вершину доб. в словарь вершин

visited[startVertex] = false;

}

else { //если есть начало

edge = {endVertex, length, 0};

dictOfVertex[startVertex].destinations.push\_back(edge); //добавляем ребро к уже существ.

}

tmp.destinations.clear();

if(startVertex != begin) { //если не исток

if(dictOfVertex.find(endVertex) == dictOfVertex.end()) { //если нет конца

edge = {startVertex, length, 0, true};

tmp.destinations.push\_back(edge); //доб. ребро в массив детей вершины

dictOfVertex[endVertex] = tmp; // вершину доб. в словарь вершин

visited[startVertex] = false;

}

else{ //если конец есть

edge = {startVertex, length, 0, true};

dictOfVertex[startVertex].destinations.push\_back(edge); //добавляем ребро к уже существ.

}

tmp.destinations.clear();

}

}

for(auto& i : dictOfVertex) {

std::sort(i.second.destinations.begin(), i.second.destinations.end(), [](Edge e1, Edge e2){return e1.bandwidth < e2.bandwidth;});

}

}

int bfs(char currVertex /\*текущая вершина\*/, int c\_min /\*минимальный поток в сети\*/, std::string tab) {

std::cout<<tab<<"Текущая вершина "<<currVertex<<"\n";

if(currVertex == end){ //конец

std::cout<<tab<<"Текущая вершина - сток\n";

return c\_min;

}

visited[currVertex] = true; //обознач. посещенной

std::cout<<tab<<"Цикл по детям "<<currVertex<<"\n";

for(auto& edge : dictOfVertex[currVertex].destinations) {

std::cout<<tab<<"Ребро ведет в "<<edge.end<<"; П/С пути "<< edge.bandwidth << "; поток через ребро "<< edge.flow <<"\n";

if(!visited[edge.end]){

std::cout<<tab<<"Ребро не посещенное\n";

}else{

std::cout<<tab<<"Ребро посещенное\n";

}

if (edge.end != currVertex) { //не цикл

if (!visited[edge.end] && (edge.flow < edge.bandwidth && !edge.is\_reverse)) {

//если не посещенная и поток в сети меньше макс. пропуск. способности и не обратное

std::cout<<tab<<"Ребро не обратное\n";

int flow = bfs(edge.end, std::min(c\_min, edge.bandwidth - edge.flow), tab+" ");

//запуск. функц. с этим ребенком и мин. из тек. мин. потока в сети и разностью между макс. пропуск. способ. и потоком.

if (flow > 0) {

edge.flow += flow;

std::cout<<tab<<"Возвращаемое значение "<< flow <<"\n";

return flow;

}

} else if (!visited[edge.end] && (edge.is\_reverse && edge.flow > 0)) {

//если не посещенная, обратная и поток больше 0

std::cout<<tab<<"Ребро обратное\n";

int flow = bfs(edge.end, std::min(c\_min, edge.flow), tab+" ");

//запуск. функц. с этим ребенком и минимумом из тек. мин. потока в сети и потоком.

if (flow > 0) {

edge.flow -= flow;

std::cout<<tab<<"Возвращаемое значение "<< flow <<"\n";

return flow;

}

}

}

}

return 0;

}

void **output**(int flow){

std::cout << flow << std::endl;

for(auto& i : dictOfVertex) {

for(auto& j : i.second.destinations) {

if(!j.is\_reverse)

std::cout << i.first << " " << j.end << " " << j.flow << std::endl;

}

}

}

void output(int flow, std::string argv){

std::ofstream fout;

fout.open(argv);

fout << flow << std::endl;

for(auto& i : dictOfVertex) {

for(auto& j : i.second.destinations) {

if(!j.is\_reverse)

fout << i.first << " " << j.end << " " << j.flow << std::endl;

}

}

}

int **main**(int argc, char \*argv[]) {

if(argv[1]!=NULL && (std::string(argv[1]) != "res.txt")){

input(std::string(argv[1])); //из файла

}else if(argv[1]==NULL || (std::string(argv[1]) == "res.txt")){

input(); //из консоли

}

int flow = 0;

int iterationResult = 0;

std::cout<<"Максимальный поток " <<flow<< "\n";

while (true) {

iterationResult = bfs(begin, 1000, " ");

if(iterationResult <= 0)

break;

for (auto& i : visited)

i.second = false;

flow += iterationResult;

std::cout<<"Максимальный поток " <<flow<< "\n";

}

for(auto& i : dictOfVertex) {

std::sort(i.second.destinations.begin(), i.second.destinations.end(), [](Edge e1, Edge e2){return e1.end < e2.end;});

}

if(argv[1]==NULL){

output(flow);

}

else if(std::string(argv[1]) == "res.txt"){

output(flow, std::string(argv[1]));

}else{

if(argv[2]==NULL){

output(flow);

}

if(argv[2]!=NULL){

output(flow, std::string(argv[2]));

}

}

return 0;

}