**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Проектирование и анализ алгоритмов»**

Тема: Потоки в сети

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Запевалов А.И. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2019

**Цель работы.**

Разработать программу для нахождения максимального 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).

**Основные теоретические положения.**

**Сеть** – ориентированный взвешенный граф, имеющий один исток и один сток.  
**Исток** – вершина, из которой рёбра только выходят.  
**Сток** – вершина, в которую рёбра только входят.  
**Поток** – абстрактное понятие, показывающее движение по графу.  
**Величина потока** – числовая характеристика движения по графу (сколько всего выходит из стока = сколько всего входит в сток).  
**Пропускная способность** – свойство ребра, показывающее, какая максимальная величина потока может пройти через это ребро.  
**Максимальный поток (максимальная величина потока)** – максимальная величина, которая может быть выпущена из стока, которая может пройти через все рёбра графа, не вызывая переполнения ни в одном ребре.  
**Фактическая величина потока в ребре** – значение, показывающее, сколько величины потока проходит через это ребро.

**Экспериментальные результаты.**

Входные данные:

7

a

d

a b 2

b c 3

c d 5

a e 7

e f 4

f c 8

b g 1

g d 6

Результат:

6

a b 2

a e 4

b c 1

b g 1

c d 5

e f 4

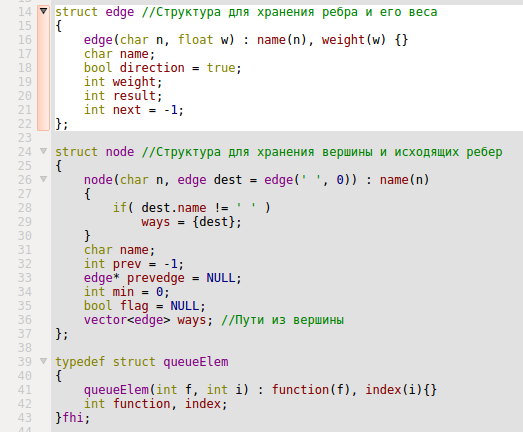
f c 4

g d 1

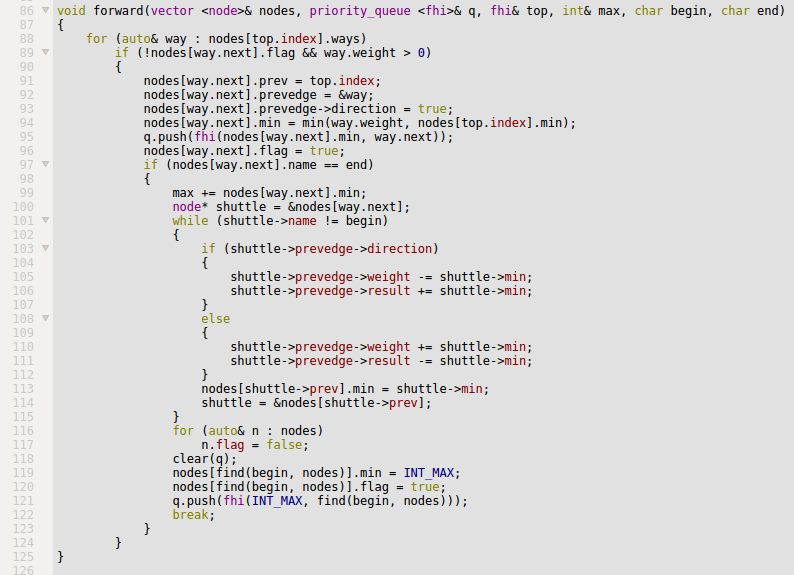
**Обработка результатов эксперимента.**

Подход к выполнению задачи.

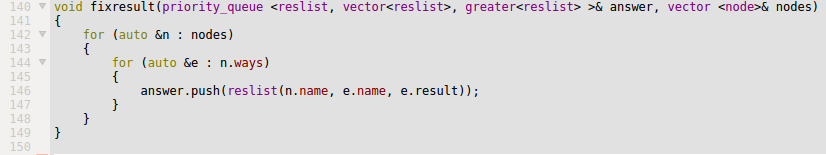
1. Создаются необходимые структуры:

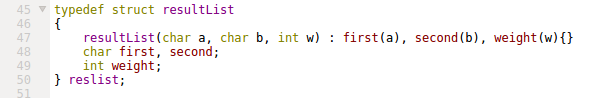
Программа использует граф, содержащий узлы, в которых хранятся списки исходящих ребер, индекс предыдущего узла в пути, указатель на предыдущий узел в списке, флаг для проверки, попадал ли этот путь уже в очередь, максимально возможный поток через путь, оканчивающийся данным узлом, а также имя узла. Каждое ребро содержит имя узла в который оно попадает, флаг направления в пути, доступный и занятый поток, а также индекс следующего узла. Очередь содержит индекс ребра и значение потока через него.

2. Основная функция и принцип ее работы.

На каждом шагу программа добавляет в очередь все доступные узлы. Это продолжается до тех пор, пока не будет добавлен конечный узел. Когда достигается конечный узел, по указателям на предыдущие узлы в ребрах пути происходит возврат в корень получившегося дерева. В процессе этого из доступного потока каждого ребра отнимается величина максимального потока в пути и добавляется в занятный поток. После этого очередь очищается и в нее заносится исток. Этот процесс продолжается до тех пор, пока можно найти путь от истока до стока.

3.Запись в результирующую структуру.



По завершении работы функции forward() в структуре, содержащей граф, остается информация о величине заполняющего потока в каждом ребре. Информация о ребрах и потоке в них записывается в результирующую структуру, после чего выводится.

**Выводы.**

Был реализован поиск потоков в графе методом Форда-Фалкерсона.

В процессе создания возникли проблемы с сохранением найденного пути, хранения исходного графа, упорядочивания узлов в очереди.

Созданы необходимые структуры, вмещающие граф, его узлы и исходящие из них ребра с различными значениями потока. Циклически обрабатываются узлы, добавляются в очередь узлы на концах исходящих ребер текущего узла, тем самым строится дерево с корнем в истоке. По достижении стока происходит запись потока на пути, который строится возвратом к истоку. Программа заполняет ребра до тех пор, пока можно найти путь в сток. В результате в ребрах остается максимальный суммарный поток из нескольких путей, не превышающий доступный для каждого ребра.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы.**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <fstream>

#include <queue>

#include "limits.h"

#include "math.h"

using namespace std;

struct node;

struct edge //Структура для хранения ребра и его веса

{

edge(char n, float w) : name(n), weight(w) {}

char name;

bool direction = true;

int weight;

int result;

int next = -1;

};

struct node //Структура для хранения вершины и исходящих ребер

{

node(char n, edge dest = edge(' ', 0)) : name(n)

{

if( dest.name != ' ' )

ways = {dest};

}

char name;

int prev = -1;

edge\* prevedge = NULL;

int min = 0;

bool flag = NULL;

vector<edge> ways; //Пути из вершины

};

typedef struct queueElem

{

queueElem(int f, int i) : function(f), index(i){}

int function, index;

}fhi;

typedef struct resultList

{

resultList(char a, char b, int w) : first(a), second(b), weight(w){}

char first, second;

int weight;

} reslist;

int operator < (fhi pair1, fhi pair2)

{

return pair1.function < pair2.function;

}

int operator > (reslist a, reslist b)

{

if (a.first == b.first)

return a.second > b.second;

return a.first > b.first;

}

void clear( std::priority\_queue <fhi> &q )

{

std::priority\_queue<fhi> empty;

std::swap( q, empty );

}

int find(char name, vector<node> nodes) //поиск узла с нужным именем

{

for(node& n : nodes)

if(name == n.name)

return &n - &nodes[0]; //возврат номера в массиве с нужным именем

return -1; //если такого нет, то -1

}

int findedge (char name, vector<edge> ways)

{

for (edge& n: ways)

if (name == n.name)

return &n - &ways[0]; //индекс в массиве

return -1;

}

void forward(vector <node>& nodes, priority\_queue <fhi>& q, fhi& top, int& max, char begin, char end)

{

for (auto& way : nodes[top.index].ways)

if (!nodes[way.next].flag && way.weight > 0)

{

nodes[way.next].prev = top.index;

nodes[way.next].prevedge = &way;

nodes[way.next].prevedge->direction = true;

nodes[way.next].min = min(way.weight, nodes[top.index].min);

q.push(fhi(nodes[way.next].min, way.next));

nodes[way.next].flag = true;

if (nodes[way.next].name == end)

{

max += nodes[way.next].min;

node\* shuttle = &nodes[way.next];

while (shuttle->name != begin)

{

if (shuttle->prevedge->direction)

{

shuttle->prevedge->weight -= shuttle->min;

shuttle->prevedge->result += shuttle->min;

}

else

{

shuttle->prevedge->weight += shuttle->min;

shuttle->prevedge->result -= shuttle->min;

}

nodes[shuttle->prev].min = shuttle->min;

shuttle = &nodes[shuttle->prev];

}

for (auto& n : nodes)

n.flag = false;

clear(q);

nodes[find(begin, nodes)].min = INT\_MAX;

nodes[find(begin, nodes)].flag = true;

q.push(fhi(INT\_MAX, find(begin, nodes)));

break;

}

}

}

/\*void backward(vector <node>& nodes, priority\_queue <fhi>& q, fhi& top)

{

if (nodes[top.index].prev != -1 && !nodes[nodes[top.index].prev].flag)

{

nodes[nodes[top.index].prev].prev = top.index;

nodes[nodes[top.index].prev].prevedge = &nodes[nodes[top.index].prev].ways[findedge(nodes[top.index].name, nodes[nodes[top.index].prev].ways)];

nodes[nodes[top.index].prev].prevedge->direction = false;

nodes[nodes[top.index].prev].min = min(nodes[nodes[top.index].prev].prevedge->result, nodes[top.index].min);

q.push(fhi(nodes[nodes[top.index].prev].min, nodes[top.index].prev));

nodes[nodes[top.index].prev].flag = true;

}

}\*/

void fixresult(priority\_queue <reslist, vector<reslist>, greater<reslist> >& answer, vector <node>& nodes)

{

for (auto &n : nodes)

{

for (auto &e : n.ways)

{

answer.push(reslist(n.name, e.name, e.result));

}

}

}

int main() //ввод

{

ifstream fin("input.txt");

char begin, end;

int n;

int max = 0;

fin >> n >> begin >> end;

//cin >> n >> begin >> end;

char dest, source;

float weight;

vector<node> nodes;

int index = 0;

while( fin >> source >> dest >> weight )

{

// while( cin >> source >> dest >> weight )

// {

index = find(source, nodes);

if( index != -1 )

nodes[index].ways.push\_back(edge(dest, weight));

else

{

nodes.push\_back(node(source, edge(dest,weight)));

index = &nodes.back() - &nodes[0];

}

if( find(dest, nodes) == -1 )

{

nodes.push\_back(node(dest));

nodes[index].ways.back().next = nodes.size() - 1;

}

else

nodes[index].ways.back().next = find(dest, nodes);

}

fin.close();

priority\_queue <fhi> q;

nodes[find(begin, nodes)].min = INT\_MAX;

nodes[find(begin, nodes)].flag = true;

q.push(fhi(INT\_MAX, find(begin, nodes)));

while (!q.empty())

{

fhi top = q.top();

q.pop();

forward(nodes, q, top, max, begin, end);

/\*if (q.top().index != 0)

backward(nodes, q, top);\*/

}

priority\_queue <reslist, std::vector<reslist>, std::greater<reslist>> answer;

fixresult(answer, nodes);

cout << max << endl;

while(!answer.empty())

{

cout << answer.top().first << ' ' << answer.top().second << ' ' << answer.top().weight << endl;

answer.pop();

}

return 0;

}