**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

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

Тема: Алгоритмы на графах

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

Санкт-Петербург

2019

**Цель работы.**

Разработать программу, которая решает задачу построения пути в *ориентированном* графе при помощи **жадного алгоритма**. Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

Разработать программу, которая решает задачу построения кратчайшего пути в *ориентированном* графе **методом А\***. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции необходимо взять близость символов, обозначающих вершины графа, в таблице ASCII.

**Основные теоретические положения.**

**Жадный алгоритм** — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

Если глобальная оптимальность алгоритма имеет место практически всегда, его обычно предпочитают другим методам оптимизации, таким как динамическое программирование

**A\*** — это модификация алгоритма Дейкстры, оптимизированная для единственной конечной точки. Алгоритм Дейкстры может находить пути ко всем точкам, A\* находит путь к одной точке. Он отдаёт приоритет путям, которые ведут ближе к цели.

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как *f(x)*). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (*x*) из начальной (обычно обозначается как *g(x)* и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как *h(x)*).

Функция *h(x)* должна быть **допустимой эвристической оценкой**, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации *h(x)* может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

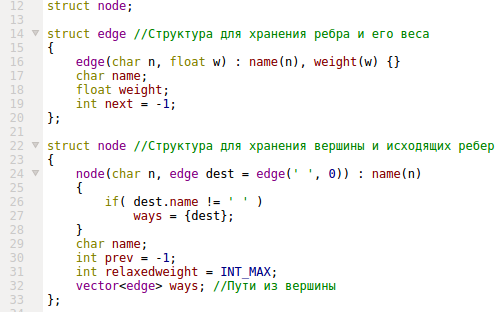
**Экспериментальные результаты.**



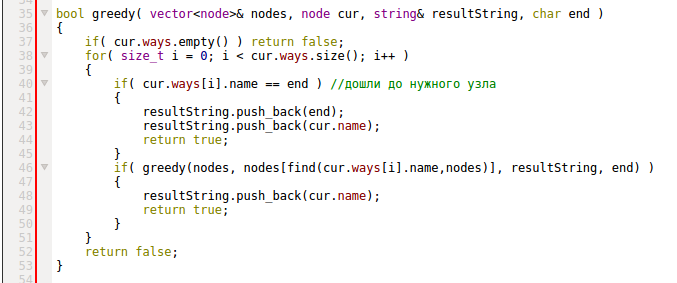
**Обработка результатов эксперимента.**

Подход к выполнению задачи.

1. Создаются необходимые структуры:

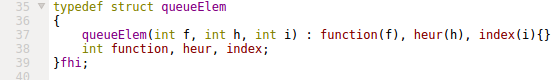
В обоих случаях программа использует граф, в котором каждый узел содержит список исходящих узлов и индекс родительского узла.

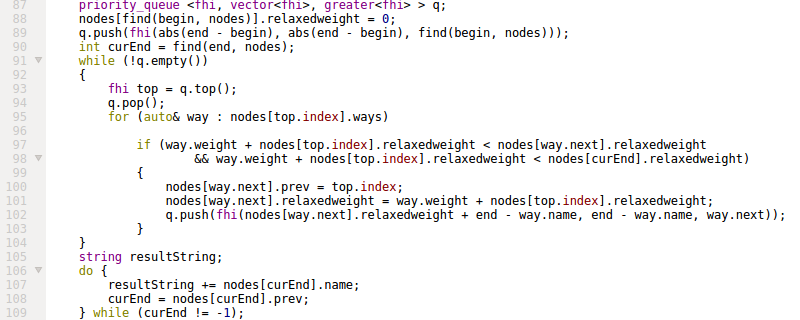
2. Основная функция жадного поиска и принцип ее работы.



При использовании жадного поиска функция вызывается для каждого исходящего ребра, причем сначала для ребер с меньшим весом. Когда в процессе найден путь до стока, то функция для оставшихся вариантов не вызывается. В результирующую строку записываются имена всех узлов включенных в путь.

3.Поиск А\*



Для реализации этого метода используется очередь с приоритетом (по возрастанию) для упорядочивания значений f(x) = g(x)+h(x), где g(x) - расстояние от истока до узла, а h(x) — эвристическая функция. На каждом этапе в очередь добавляются исходящие узлы. Из них впоследствии выбирается первый (имеющий наименьшее значение f(x)), и, если это не сток, операция повторяется. Лучший путь можно восстановить по предыдущим узлам, начиная от стока.

**Выводы.**

Были реализованы два метода нахождения пути в ориентированном графе. При этом использовались структуры для хранения ориентированных графов. В реализации поиска А\* для хранения узлов использовалась очередь с приоритетом. В обоих случаях программа нашла оптимальный вариант пути.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы(жадный поиск).**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <fstream>

using namespace std;

struct edge //Структура для хранения ребра и его веса

{

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

char name;

float weight;

};

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;

vector<edge> ways; //Пути из вершины

};

int find( char name, std::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

}

bool greedy( vector<node>& nodes, node cur, string& resultString, char end )

{

if( cur.ways.empty() ) return false;

for( size\_t i = 0; i < cur.ways.size(); i++ )

{

if( cur.ways[i].name == end ) //дошли до нужного узла

{

resultString.push\_back(end);

resultString.push\_back(cur.name);

return true;

}

if( greedy(nodes, nodes[find(cur.ways[i].name,nodes)], resultString, end) )

{

resultString.push\_back(cur.name);

return true;

}

}

return false;

}

int main(void) //ввод

{

//ifstream fin("input.txt");

char begin, end;

//fin >> begin >> end;

cin >> begin >> end;

char dest, source;

float weight;

vector<node> nodes;

int index = 0;

/\*while( fin >> source >> dest >> weight )

{\*/

while( std::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)));

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

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

}

//fin.close();

for( auto& n : nodes ) //сортировка по возрастанию веса

std::sort(n.ways.begin(), n.ways.end(),[](edge& a, edge& b)

{

return a.weight < b.weight;

});

string resultString;

greedy(nodes, nodes[find(begin, nodes)], resultString, end);

reverse(resultString.begin(), resultString.end());

cout << resultString;

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**Исходный код программы(поиск А\*)**

#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;

float weight;

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;

int relaxedweight = INT\_MAX;

vector<edge> ways; //Пути из вершины

};

typedef struct queueElem

{

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

int function, heur, index;

}fhi;

int operator > (fhi pair1, fhi pair2)

{

if (pair1.function == pair2.function)

return pair1.heur > pair2.heur;

return pair1.function > pair2.function;

}

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 main(void) //ввод

{

ifstream fin("input.txt");

char begin, end;

fin >> begin >> end;

//cin >> 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, vector<fhi>, greater<fhi> > q;

nodes[find(begin, nodes)].relaxedweight = 0;

q.push(fhi(abs(end - begin), abs(end - begin), find(begin, nodes)));

int curEnd = find(end, nodes);

while (!q.empty())

{

fhi top = q.top();

q.pop();

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

if (way.weight + nodes[top.index].relaxedweight < nodes[way.next].relaxedweight

&& way.weight + nodes[top.index].relaxedweight < nodes[curEnd].relaxedweight)

{

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

nodes[way.next].relaxedweight = way.weight + nodes[top.index].relaxedweight;

q.push(fhi(nodes[way.next].relaxedweight + end - way.name, end - way.name, way.next));

}

}

string resultString;

do {

resultString += nodes[curEnd].name;

curEnd = nodes[curEnd].prev;

} while (curEnd != -1);

reverse(resultString.begin(), resultString.end());

cout << resultString << endl;

return 0;

}