**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №**2

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

**Тема: Алгоритмы на графах**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Самакаев Д.И. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2019

**Вариант 3.**

## Цель работы.

Написать функцию, проверяющую эвристику на допустимость и монотонность.

## Основные теоретические положения.

Разработайте 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 следует взять близость символов, обозначающих вершины графа, в таблице ASCII.

## Описание алгоритма.

Для решения поставленной задачи был реализован алгоритм A\*. В качестве эвристической функции была использована функция evristic(c1, c2), возвращающая расстояние между двумя символами. В начале каждой итерации в массиве ищется элемент приоритет, которого минимален, он удаляется из списка и начинается осмотр всех ребер выходящих из выбранного элемента. Если нашлась вершина путь до которой был больше чем найденный, то данный путь заменяется на найденный. Для хранения значений имен узлов и ребер выходящих из них был использован словарь и структура elem, хранящая ребра выходящие из текущей вершины, имя вершины из которой был найден минимальный путь и длина до начальной позиции. Сложность алгоритма: О(|V|\*|V| + |E|), где V – множество вершин, а Е – множество ребер.

**Тестирование.**

Таблица 1 – Результаты тестирование

|  |  |
| --- | --- |
| **Ввод** | **Вывод** |
| a e  a b 3  b c 1  c d 1  a d 5  d e 1 | Allowable monotone ade |
| b m  a b 1  a f 3  b c 5  b g 3  f g 4  c d 6  d m 1 | Not allowable not monotone  bcdm |
| a e  a b 0  b c 1  c d0  a d 5  d e 1 | Not allowable Not monotone  abcde |

# Вывод.

В ходе работы был построен и анализирован алгоритм A\* на основе решения задачи о нахождении минимального пути в графе. Исходный код программы представлен в приложении 1.

**ПРИЛОЖЕНИЕ А.  
ИСХОДНЫЙ КОД**

#include <iostream>

#include <map>

#include <vector>

#include <algorithm>

#include <set>

struct elem {

std::vector<std::pair<char, int>> ways;

int length;

};

bool cmp(const std::pair<char, int>& a, const std::pair<char, int>& b) {

if (a.second == b.second) return a.first > b.first;

return a.second < b.second;

}

int get\_length(char a, std::map<char, elem>& my\_map) { return my\_map[a].length; }

int evristic(char a, char where) { return abs(a - where); }

int total(char a, char where, std::map<char, elem>& my\_map) { return get\_length(a, my\_map) + evristic(a, where); }

char MIN\_F(std::set<char>& open, char where, std::map<char, elem>& my\_map) {

int min = std::numeric\_limits<int>::max();

char curr = 0;

for (char i : open)

if (total(i, where, my\_map) < min) {

min = total(i, where, my\_map);

curr = i;

}

return curr;

}

std::string make\_path(std::map<char, char>& from, char start, char where) {

std::string path = { where };

char curr = where;

while (curr != start) {

curr = from[curr];

path += curr;

}

std::reverse(path.begin(), path.end());

return path;

}

void findWay(char start, char end, std::map<char, elem>& my\_map) {

char curr = start;

std::set<char> closed\_set;

std::set<char> open\_set = { start };

std::map<char, char> path\_syms;

while (!open\_set.empty()) {

curr = MIN\_F(open\_set, end, my\_map);

if (curr == end) {

std::cout << make\_path(path\_syms, start, end);

return;

}

open\_set.erase(curr);

closed\_set.insert(curr);

for (auto neighbour : my\_map[curr].ways) {

bool tentative\_is\_better;

if (closed\_set.find(neighbour.first) != closed\_set.end()) continue;

int tentative\_g\_score = get\_length(curr, my\_map) + neighbour.second;

if (open\_set.find(neighbour.first) == open\_set.end()) {

open\_set.insert(neighbour.first);

tentative\_is\_better = true;

}

else {

tentative\_is\_better = tentative\_g\_score < get\_length(neighbour.first, my\_map);

}

if (tentative\_is\_better) {

path\_syms[neighbour.first] = curr;

my\_map[neighbour.first].length = tentative\_g\_score;

}

}

}

return;

}

size\_t shortest\_way(std::map<char, elem>& my\_map,char start\_ch, char end\_ch, size\_t min\_length, size\_t current\_length, size\_t buff) {

if (start\_ch == end\_ch) {

return current\_length;

}

for (size\_t i = 0; i < my\_map[start\_ch].ways.size(); i++) {

buff = shortest\_way(my\_map, my\_map[start\_ch].ways[i].first, end\_ch, min\_length, current\_length + my\_map[start\_ch].ways[i].second, buff);

if (min\_length > buff)

min\_length = buff;

}

return min\_length;

}

bool check\_monotony(std::map<char, elem>& my\_map, char end\_ch) {

for (auto it = my\_map.begin(); it != my\_map.end(); ++it) {

for (size\_t i = 0; i < it->second.ways.size(); i++) {

if (evristic(it->first, end\_ch) - evristic(it->second.ways[i].first, end\_ch) > it->second.ways[i].second)

return false;

}

}

return true;

}

bool check\_ambissibility(std::map<char, elem>& my\_map, char end\_ch) {

if (check\_monotony(my\_map, end\_ch))

return true;

for (auto it = my\_map.begin(); it != my\_map.end(); ++it) {

if (evristic(it->first, end\_ch) > shortest\_way(my\_map, it->first, end\_ch, std::numeric\_limits<size\_t>::max(), 0, 0))

return false;

}

return true;

}

int main() {

char start, end;

std::cin >> start >> end;

char a, b;

float c = 0;

std::map<char, elem> my\_map;

while (std::cin >> a >> b >> c) {

if (c == -1) break;

my\_map[a].ways.push\_back({ b,c });

std::sort(my\_map[a].ways.begin(), my\_map[a].ways.end(), cmp);

}

if (check\_ambissibility(my\_map, end))

std::cout << "allowable\n";

else std::cout << "not allowable\n";

if(check\_monotony(my\_map, end))

std::cout << "monotone\n";

else std::cout << "not monotone\n";

findWay(start, end, my\_map);

return 0;

}