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

отчет

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритмы на графах

Студент гр.8304 _____ Холковский К.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Задание.

Вариант 4

Модификация A* с двумя финишами (требуется найти путь до любого из двух).

Цель работы.

Разработать программу, которая решает задачу построения кратчайшего пути в ориентированном графе **методом A*.**

Описание алгоритма.

В процессе работы алгоритма для вершин рассчитывается функция f(v) = g(v) + h(v), где

g(v) - наименьшая стоимость пути в vv из стартовой вершины,

h(v) - эвристическое приближение стоимости пути от vv до конечной цели.

Фактически, функция f(v) - длина пути до цели, которая складывается из пройденного расстояния g(v) и оставшегося расстояния h(v). Исходя из этого, чем меньше значение f(v), тем раньше мы откроем вершину v, так как через неё мы предположительно достигнем расстояние до цели быстрее всего. Открытые алгоритмом вершины можно хранить в очереди с приоритетом по значению f(v). A* действует подобно алгоритму Дейкстры и просматривает среди всех маршрутов ведущих к цели сначала те, которые благодаря имеющейся информации (эвристическая функция) в данный момент являются наилучшими. Сложность по памяти экспоненциальная, временная сложность в худшем случае экспоненциальная, в лучшем полиномиальная.

Описание функций и структур данных.

```
struct elem {
    std::vector<std::pair<char, int>> ways;
    int length;
};
```

Структура хранит информацию о соседях в векторе пар и значение функции G, а сама хранится в словаре и по имени вершины графа можно получить всю нужную информацию.

```
int G(char a, std::map<char, elem> & my_map);
int H(char a, char where);
int F(char a, char where, std::map<char, elem>& my_map);
```

Уже были описаны ранее (см. описание алогритма).

```
char MIN_F(std::set<char> & open, char where, std::map<char, elem>& my_map);
```

Функция ищет для какой из открытых вершин будет минимальным путь.

```
std::string RECONSTRUCT_PATH(std::map<char,char> & from,char start, char where);
```

Функция возвращает понятный для пользователя путь в графе.

```
void findWay(char start, char end1, char end2, std::map<char, elem>& my_map);
```

Функция ищущая алгоритмом A* минимальный путь.

Выводы.

В ходе выполнения данной работы была написана программа, которая решает задачу построения кратчайшего пути в ориентированном графе **методом A***

Тестирование

input	**output**
a b c **a e 0** **a c 0** **e b 0**	**aeb** **ac**
a e c **a b 3.0** **b c 1.0** **c d 1.0** **a d 5.0** **d e 1.0**	**ade** **abc**
a b c **a d 3.0** **a c 2.1** **d b 1.0**	**adb** **ac**

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 G(char a, std::map<char, elem> & my_map) {return my_map[a].length;}

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

int F(char a, char where, std::map<char, elem>& my_map){ return G(a, my_map) + H(a,
where);}

char MIN_F(std::set<char> & open, char where, std::map<char, elem>& my_map) {
    int min = 2147483647;
    char curr = 0;
    for(char i: open)
        if(F(i, where, my_map) < min){
            min = F(i, where, my_map);
            curr = i;
        }
    return curr;
}

std::string RECONSTRUCT_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 end1, char end2, std::map<char, elem>& my_map) {
    char curr = start;
    std::set<char> closed;
    std::set<char> open = {start};
    std::map<char,char> from;
    while(!open.empty()) {
        curr = MIN_F(open,end1,my_map);

        if(curr == end1){
            std::cout << RECONSTRUCT_PATH(from, start, end1) << std::endl;

            if(from.find(end2) != from.end())
                std::cout << RECONSTRUCT_PATH(from, start, end2) << std::endl;
```

```
            return;
        }
        open.erase(curr);
        closed.insert(curr);

        for(auto neighbour : my_map[curr].ways) {
            bool tentative_is_better;
            if(closed.find(neighbour.first) != closed.end()) continue;
            int tentative_g_score = G(curr, my_map) + neighbour.second;
            if(open.find(neighbour.first) == open.end()){
                open.insert(neighbour.first);
                tentative_is_better = true;
            }
            else {
                tentative_is_better = tentative_g_score < G(neighbour.first, my_map);
            }

            if(tentative_is_better) {
                from[neighbour.first] = curr;
                my_map[neighbour.first].length = tentative_g_score;
            }
        }
    }
    std::cout << "No way" << std::endl;
}

int main() {
    char start, end1, end2;
    std::cin >> start >> end1 >> end2;
    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);
    }
    findWay(start, end1, end2, my_map);
    return 0;
}
```