**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 А\*.

**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). Сложность по памяти экспоненциальная, временная сложность в худшем случае экспоненциальная, в лучшем полиномиальная.

**Описание функций и структур данных.**

Граф хранится в словареstd::map<**char**, elem>, где элем это структура, хранящая информацию о соседях и значение функции G.  
**struct** elem {  
 std::vector<std::pair<**char**, **int**>> ways;  
 **int** length;  
};

Функции из описания алгоритма:

**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);

Функция ищет для какой из открытых вершин будет минимальным путь.

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

Функция возвращает понятный для пользователя путь в графе.

std::string RECONSTRUCT\_PATH(std::map<**char**,**char**> & from,**char** start, **char** where);

Функция ищущая алгоритмом A\* минимальный путь.

**void** findWay(**char** start, **char** end1, **char** end2, std::map<**char**, elem>& my\_map);

**Выводы.**

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

**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;  
}