

**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: Жадный алгоритм и  $A^*$**

Студент гр. 7304

\_\_\_\_\_

Давыдов А.А.

Преподаватель

\_\_\_\_\_

Филатов А.Ю.

Санкт-Петербург

2019

## Цель работы.

Реализовать жадный алгоритм и астар поиска минимального пути в графе.

### Задача1

Разработайте 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.

Пример входных данных

```
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0
```

В первой строке через пробел указываются начальная и конечная вершины. Далее в каждой строке указываются ребра графа и их вес.

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет

```
Abcde
```

### Задача2

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.

Пример входных данных

```
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
```

d e 1.0

В первой строке через пробел указываются начальная и конечная вершины  
Далее в каждой строке указываются ребра графа и их вес

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет

Ade

### Экспериментальные результаты.

- 1) Была реализована функция `greedy_algorithm`, реализующая жадный алгоритм и возвращающая строку – итоговый путь

```
int compare_weights(pair<char,double>& p1,pair<char, double>& p2)
{
    return p1.second < p2.second;
}

//function return summary path to destination vertex
string greedy_algorithm(map<char, vector<pair<char, double>>> &graph, char
fromVertex, char toVertex) {
    map<char,bool> visitedVertex;
    map<char,vector<pair<char,double>>>::iterator it_for_graph;
    vector<char> cur_path;

    for(it_for_graph = graph.begin(); it_for_graph!= graph.end();
it_for_graph++)
        visitedVertex[it_for_graph->first]=false;
    cur_path.push_back(fromVertex); //push start vertex

    for(int i=0; i < graph.size(); ++i) {
        if(cur_path[cur_path.size()-1]==toVertex)
            break;

        visitedVertex[cur_path[cur_path.size() - 1]]=true; //mark last added
vertex as visited
        vector<pair<char, double>> incident_vertexs;

        //simplify cycle with iterator
        for(pair<char,double> incident_vertex:
graph.find(cur_path[cur_path.size()-1])->second)
            if(!visitedVertex[incident_vertex.first])
                incident_vertexs.push_back(incident_vertex);

        //back at 1 step
        if(incident_vertexs.size()==0)
        {
            cur_path.pop_back();
            continue;
        }

        pair<char,double> min = *min_element(incident_vertexs.begin(),
incident_vertexs.end() ,compare_weights);
        cur_path.push_back(min.first);
    }
}
```

```

    string path_to_vertex = "";

    //concatenate summary path
    for(int i = 0; i < cur_path.size(); i++){
        path_to_vertex = path_to_vertex + cur_path[i];
    }

    return path_to_vertex;
}

```

2) Была реализована функция AStar, реализующая поиск кратчайшего пути алгоритмом A\* и возвращающая строку - итоговый путь.

```

string AStar(map<char, vector<pair<char, double>>> &graph, char fromVertex,
char toVertex)
{
    map<char, bool> visitedVertex;
    map<char, vector<pair<char, double>>>::iterator it_for_graph;
    map<char, char> came_from;
    map<char, double> cost_from_start; //queue for choose next minmal element
    cost_from_start[fromVertex] = 0;

    for(it_for_graph = graph.begin(); it_for_graph!= graph.end();
    ++it_for_graph)
        visitedVertex[it_for_graph->first] = false;

    for(int i = 0; i < graph.size(); ++i)
    {
        //choise vertex whay to which most smaller than to other
        //using lamda function for give access to variable toVertex
        map<char, double>::iterator cur = min_element(cost_from_start.begin(),
cost_from_start.end(), [toVertex] (pair<char, double> v1, pair<char, double> v2) {
            return (v1.second+abs(v1.first-
toVertex)) < (v2.second+abs(v2.first-toVertex)); });

        if(cur->first == toVertex)
            break;

        visitedVertex[cur->first] = true;

        for(pair<char, double> incident_vertex: graph[cur->first])
        {
            if(!visitedVertex[incident_vertex.first])
            {
                double new_cost = cost_from_start[cur->first] +
incident_vertex.second;

                //element doesnt contains in cost_from_start or has greater
weight
                if(cost_from_start.find(incident_vertex.first) ==
cost_from_start.end() || cost_from_start[incident_vertex.first] > new_cost)
                {
                    cost_from_start[incident_vertex.first] = new_cost;
                    came_from[incident_vertex.first] = cur->first;
                }
            }
        }

        cost_from_start.erase(cur); //delete cur from queue
    }
}

```

```

    string path_to_vertex = "";

    for(char cur_v = toVertex; path_to_vertex[0] != fromVertex; cur_v =
came_from.find(cur_v)->second)
        path_to_vertex = cur_v + path_to_vertex;

    return path_to_vertex;
}

```

### 3) Код функции main

```

int main() {
    char A; //из точки A
    char B; // в точку B
    cin >> A >> B;
    map<char, vector<pair<char, double>>> graph;
    char v1;
    char v2;
    double weight;

    //cin return 0 if didn't read symbols
    while ((cin >> v1) && (cin >> v2) && (cin >> weight))
    {
        graph[v1].push_back(pair<char, double>(v2, weight));
        graph[v2]; //insert vertex v2 for avoid situation when vertex doesn't
has edges
    }

    //cout << greedy_algorithm(graph, A, B);
    cout << AStar(graph, A, B);
    return 0;
}

```

### Выводы.

Были изучены и реализованы алгоритмы A\* и жадный поиск в соответствии с условием заданий.