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

ОТЧЕТ

по лабораторной работе№2

по дисциплине «Построение и анализ алгоритмов»

Тема: Жадный алгоритм и A*.

Студент гр. 7304 _____ Субботин А.С.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы:

Изучить и реализовать на языке программирования C++ жадный алгоритм поиска пути в графе и алгоритм A* поиска кратчайшего пути в графе между двумя заданными вершинами.

Формулировка задачи:

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

Ход работы:

Жадный алгоритм:
По мере получения входных данных формируется вектор, элемент которого – структура, состоящая из вершины-истока, а также вектора, содержащего вершину-сток и вес ребра между ними. Каждый вектор в этой структуре сортируется по невозрастанию весов – для того, чтоб начинать просмотр с конца и при нахождении тупика удалять элемент методом pop_back(). Далее алгоритм следующий: до тех пор, пока не найден искомый путь, происходит создание копии вектора для текущей вершины, производится переход в вершину из конца вектора (гарантируется, что она имеет минимальный вес), при отсутствии путей из этой вершины происходит возврат с удалением последнего элемента вектора. Рекурсивная работа функции приводит к нахождению пути между заданными вершинами, или свидетельствует об отсутствии такого пути отсутствием выводимой информации.
Алгоритм A*:
Принцип поиска тот же, что и у жадного алгоритма, только сортировка производится в соответствии с положением элемента в таблице ASCII (в зависимости от его удалённости от начала таблицы) , и, аналогично, по невозрастанию весов для равных значений первого критерия, и поиск заканчивается не при первом нахождении конечной вершины, а при полном отсутствии возможных путей из исходной.

Результаты работы программы:

Исходные данные	Жадный алгоритм	A*
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	abcde	ade
a e a f 1 a p 2 a r 2 p d 7 p m 4 m z 2 d z 1 z k 1 k e 1 r m 4 r s 5 r t 3 r h 5 s y 9 s x 2 y n 3 n e 1 x e 1 t x 8 h c 15 c e 1	apmzke	arsxe
a d a b 1 b c 1 c e 1 a d 10	ad	ad
a g a p 10 a r 10 a s 10		

Выводы:

В ходе выполнения данной лабораторной работы был исследован и реализован жадный алгоритм, который оказался довольно прост по принципу и по реализации – на каждом шаге выбирается ребро с наименьшим весом, но цена за простоту – отсутствие гарантий того, что полученный путь будет иметь минимальный вес. Алгоритм A* незначительно сложнее, но за счёт полного перебора можно иметь уверенность в том, что путь будет иметь минимально возможный вес.

Приложение 1. Жадный алгоритм

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

struct Node{
    char tail;
    float price;
};

struct Graph{
    char head;
    vector <struct Node> node;
};

bool Compare(struct Node first, struct Node second){
    return first.price >= second.price;
}

void Fun(vector <struct Graph>& graph, char first, char final, vector <char>&
bestway, vector <char>& way, float& bestwaydata, float& waydata){
    if(!bestway.empty())
            return;
    for(auto it = graph.begin(); it != graph.end(); it++){
        if(it->head == first){
            vector <struct Node> node(it->node);
            while(!node.empty()){
                auto pointer = node.end();
                pointer--;
                if(bestwaydata >= waydata + pointer->price){
                    waydata += pointer->price;
                    way.push_back(pointer->tail);
                    if(pointer->tail == final){
                        if((bestwaydata == waydata && way.size() < bestway.size())
|| bestwaydata > waydata){
                            bestway = way;
                            bestwaydata = waydata;
                        }
                    }
                    else
                        Fun(graph, pointer->tail, final, bestway, way,
bestwaydata, waydata);
                    waydata -= pointer->price;
                    node.pop_back();
                    way.pop_back();
                }
                else
                    node.pop_back();
            }
        }
    }
}

int main()
{
    char first, final;
    cin >> first >> final;
    vector <struct Graph> graph;
    struct Node node;
    struct Graph newnode;
    char head;
```

```
    bool flag;
    while(cin >> head >> node.tail >> node.price){
        flag = false;
        for(auto it = graph.begin(); it != graph.end(); it++)
            if(head == it->head){
                (it->node).push_back(node);
                flag = true;
                break;
            }
        if(!flag){
            newnode.head = head;
            newnode.node.push_back(node);
            graph.push_back(newnode);
            newnode.node.pop_back();
        }
    }
    for(auto it = graph.begin(); it != graph.end(); it++)
        sort((it->node).begin(), (it->node).end(), Compare);
    vector <char> way;
    way.push_back(first);
    vector <char> bestway;
    float waydata = 0, bestwaydata = 9999;
    Fun(graph, first, final, bestway, way, bestwaydata, waydata);
    for(auto it = bestway.begin(); it != bestway.end(); it++)
        cout << *it;
    return 0;
}
```

Приложение 2. A*

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

struct Node{
    char tail;
    float price;
};

struct Graph{
    char head;
    vector <struct Node> node;
};

bool Compare(struct Node first, struct Node second){
    if(first.tail == second.tail) return first.tail > second.tail;
    return first.tail < second.tail;
}

void Fun(vector <struct Graph>& graph, char first, char final, vector <char>&
bestway, vector <char>& way, float& bestwaydata, float& waydata){
    for(auto it = graph.begin(); it != graph.end(); it++){
        if(it->head == first){
            vector <struct Node> node(it->node);
            while(!node.empty()){
                auto pointer = node.end();
                pointer--;
                if(bestwaydata >= waydata + pointer->price){
                    waydata += pointer->price;
                    way.push_back(pointer->tail);
                    if(pointer->tail == final){
                        if((bestwaydata == waydata && way.size() < bestway.size())
|| bestwaydata > waydata){
                            bestway = way;
                            bestwaydata = waydata;
                        }
                    }
                    else
                        Fun(graph, pointer->tail, final, bestway, way,
bestwaydata, waydata);
                    waydata -= pointer->price;
                    node.pop_back();
                    way.pop_back();
                }
                else
                    node.pop_back();
            }
        }
    }
}

int main()
{
    char first, final;
    cin >> first >> final;
    vector <struct Graph> graph;
    struct Node node;
    struct Graph newnode;
    char head;
    bool flag;
```

```
    while(cin >> head >> node.tail >> node.price){
        flag = false;
        for(auto it = graph.begin(); it != graph.end(); it++)
            if(head == it->head){
                (it->node).push_back(node);
                flag = true;
                break;
            }
        if(!flag){
            newnode.head = head;
            newnode.node.push_back(node);
            graph.push_back(newnode);
            newnode.node.pop_back();
        }
    }
    for(auto it = graph.begin(); it != graph.end(); it++)
        sort((it->node).begin(), (it->node).end(), Compare);
    vector <char> way;
    way.push_back(first);
    vector <char> bestway;
    float waydata = 0, bestwaydata = 9999;
    Fun(graph, first, final, bestway, way, bestwaydata, waydata);
    for(auto it = bestway.begin(); it != bestway.end(); it++)
        cout << *it;
    return 0;
}
```