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. 7382 _____ Гаврилов А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Задание.

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

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.

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

Вар. 3с. Списки смежности. Написать функцию, проверяющую эвристику на допустимость и монотонность.

Пояснение задания.

На вход программе подается граф. Программа находит наименьший путь при помощи жадного алгоритма или алгоритма A*.

Описание алгоритмов.

Жадный алгоритм:

Берется начальная вершина и просматриваются все инцидентные ей вершины, берется в качестве следующей та, путь до которой наименьшей. Если текущая

ветка не достигает результата, то последняя вершина помещается в список тупиков (вершины из этого списка больше не просматриваются) и алгоритм возвращается на шаг назад. Алгоритм заканчивается, когда будет найдена конечная вершина. Сложность алгоритма по операциям O(VlogV + E) и по памяти O(E+V).

Алгоритм A*.

Берется начальная вершина, опускается в очередь с приоритетом, затем вытаскивается и ищутся оценки стоимости путей до инцидентных вершин, которые опускаются в эту очередь(очередь выдает вершину, путь до которой от начала наименьший). Затем вытаскивается следующая вершина и делается тоже самое, пока список не будет пуст или не будет достигнут конец. Сложность алгоритма по операциям O(ElogE + E), по памяти O(E+V).

Описание структур

Класс Graph.

Используется список смежности, реализованный с помощью контейнера map, хранит ключ — вершину и еще map, который для данного ключа показывает пути к инцидентным вершинам, то есть хранит пары {ключ, цена}.

Описание функций

Код программ приведен в приложении А.

В программе с жадным алгоритмом использовались:

bool find_point(std::list<char>& points_list, char point)

Функция ищет вершину в списке и возвращет true, если она есть, иначе false.

list<char> points_list- список вершин.

char point — искомая вершина.

bool has_any_way(std::list<char> &points_hist, std::list<char> &deadlocks, char point)

Проверка существует ли путь в вершину, в которой мы еще не были и не ведущая в тупик.

std::list<char> &points_hist – лист с вершинами текущего пути.

std::list<char> &points_hist – лист с вершинами ведущими в тупик.

В программе с алгоритмом A* использовались:

int heuristic(char a,char b)

Эвристическая функция.

char a,b – вершины для которых ищется разница по таблице ASCII.

void print_q(std::priority_queue <std::pair<double, char>,

std::vector < std::pair<double, char> >,

std::greater< std::pair<double, char>>> q)

std::priority_queue <std::pair<double, char>,

std::vector < std::pair<double, char> >,

std::greater< std::pair<double, char>>> q – очередь с приоритетом.

Метод печатает элементы в очереди.

Общие методы и функции для обеих программ:

void add_way(char from, char to, double length)

Добавление пути в список.
char from, to – вершины куда и откуда.
double length – стоимость пути до вершины to.

double find_way_cost(char from, char to)

char from, to – вершина «из» и вершина «в» соответственно.

Нахождение веса ребра между двумя вершинами.

void find_min_way(char from, char to)

char from, to – вершина «из» и вершина «в» соответственно.

Метод нахождения минимального пути в графе, и выписывания его в терминал.

int main()

Главная функция программы. Считывает входные данные, запускает алгоритм поиска минимального пути.

Тестирование.

Вводимые данные	Результат:
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ade
a m a l 3 a b 5 l m 1 l b 22 c l 15 b c 11 m k 4 k j 6 j o 8 n o 9 n m 1 c n 7 d c 6 d n 7 o d 1	anjiodefghiodclm

j o 8 d e 1 e f 7 f g 8 g h 3 h i 16 o h 1 o g 5 f o 9 j i 1 i o 1 a n 1 a d 29 n k 6 j k 4 n j 1	
a k a l 3 a b 5 l m 1 l b 22 c l 15 b c 11 m k 4 k j 6 j o 8 n o 9 n m 1	anjiodefghiodclmk

c n 7 d c 6 d n 7 o d 1 j o 8 d e 1 e f 7 f g 8 g h 3 h i 16 o h 1 o g 5 f o 9 j i 1 i o 1 a n 1 a d 29 n k 6 j k 4 n j 1	
a o a l 3 a b 5 l m 1 l b 22 c l 15 b c 11 m k 4	anjio

k j 6 j o 8 n o 9 n m 1 c n 7 d c 6 d n 7 o d 1 j o 8 d e 1 e f 7 f g 8 g h 3 h i 16 o h 1 o g 5 f o 9 j i 1 i o 1 a n 1 a d 29 n k 6 j k 4 n j 1	

Вывод.

В процессе выполнения лабораторной работы были изучены алгоритм A* и жадный алгоритм. Реализована структура данных – список смежности и реализованы методы, которые позволяют добавлять вершины в граф и искать наименьший путь до вершины. Исследованы сложности алгоритмов и исследована эвристическая функция на допустимость и монотонность в алгоритме A*.

Приложение А.

Код программы с жадным алгоритмом:

```
#include <iostream>
#include <map>
#include <list>
#include <cmath>
#include <string>
#include <algorithm>

#define TEST
class Graph{
    bool find_point(std::list<char>& points_list, char point){                    //
функция поиска вершины в листе
    for(auto i=points_list.begin();i!=points_list.end();i++){
            if(*i==point)
                    return true;
    }
    return false;
    }
    std::map<char,std::map<char,double>> list;
    //список смежности
public:
    void add_way(char from, char to, double length){
    //добавляем путь
#ifdef TEST
    std::cout<<"Добавляем путь из: "<<from<<" в "<<to<<' '<<length<<std::endl;
#endif
    if(from==to)
            return;
    std::pair<char,double> way(to,length);
    if(list.find(from)==list.end()){
    //если вершины нет в графе, то добавляем ее
            std::map<char,double> l;
            l.insert(way);
            list.insert(std::pair<char,std::map<char,double>>(from,l));
```

```
            //std::cout<<list.begin()->first<<' '<<(list[from]).begin()-
>first<<std::endl;
            return;
    }
    list[from].insert(way);
                //если вершина уже есть, то добавляем только путь
    }
    double find_way_cost(char from, char to){
            //функция поиска длины пути до инцидентной вершины
    if(list.find(from)==list.end())
            return -1.0;
    auto i=list[from].find(to);
    if(i==list[from].end())
            return -1.0;
    return i->second;
    }
    bool has_any_way(std::list<char> &points_hist, std::list<char> &deadlocks,
char point){            //проверяет наличие путей из вершины, ведущих не в
посещенную вершину и не в тупик
    for(auto i=list[point].begin();i!=list[point].end();i++){
            if(!find_point(points_hist,i->first) && !find_point(deadlocks,i->first))
                return true;
    }
    return false;
    }
    void find_min_way(char from, char to){
    //функция поиска наименьшего пути
    std::list<char> points_hist, deadlocks;
    //список с текущим путем и список тупиков
    char cur=from,next;
    double min;
    points_hist.push_front(cur);
#ifdef TEST
    std::cout<<"Начинаем поиск!\n";
#endif
```

```
    while(cur!=to){
            //пока не найден конец
#ifdef TEST
            std::cout<<"Ищем пути из: "<<cur<<std::endl;
#endif
            bool flag=true;
            if(!has_any_way(points_hist,deadlocks,cur)){
    //если есть путь, то продолжаем, иначе возвращаемся в предыдущую вершину и ищем другие пути
                    //std::cout<<cur<<" тупик\n";
#ifdef TEST
                    std::cout<<"Найден тупик, возвращаемся на шаг назад и добавляем вершину в список тупиков!\n";
#endif
                    deadlocks.push_back(cur);
                    points_hist.pop_back();
                    cur=points_hist.back();
                    continue;
            }
            for(auto i=list[cur].begin();i!=list[cur].end();i++){
            //просматриваем инцидентные вершины
#ifdef TEST
                    std::cout<<"Просматриваем путь: "<<cur<<"->"<<i->first<<" за "<<i->second<<std::endl;
#endif
                    if(flag && !find_point(points_hist,i->first) && !find_point(deadlocks,i->first)){              //устанавливаем минимум, равный первому доступному пути
                            min=i->second;
                            next=i->first;
                            flag=false;
                            continue;
                    }
                    if(i->second<min && !find_point(points_hist,i->first) && !find_point(deadlocks,i->first)){ //если найден довое минимальное ребро
#ifdef TEST
```

```
                    std::cout<<"Следующая вершина: "<<i->first<<std::endl;
#endif
                            next=i->first;
                    }
            }
            cur=next;
            points_hist.push_back(cur);
#ifdef TEST
            std::cout<<"Текущий путь: ";
    for(auto i=points_hist.begin();i!=points_hist.end();i++){
                    std::cout<<*i;
    }
    std::cout<<std::endl<<std::endl;

#endif
    }
    std::cout<< "Результат: ";
    for(auto i=points_hist.begin();i!=points_hist.end();i++){                    //печать
результата
            std::cout<<(*i);
    }
    std::cout<<std::endl;
    }



};

int main(){
    Graph a;
    char start,end,from,to;
    std::cin>>start>>end;
    double length;
    while(std::cin>>from>>to>>length){
    a.add_way(from, to, length);
    }
    a.find_min_way(start,end);
```

```
    return 0;
}
```

Код программы для алгоритма A*:

```
#include <iostream>
#include <map>
#include <queue>
#include <cmath>
#include <string>
#include <algorithm>
#define TEST

class Graph{
    std::map<char,std::map<char,double>> list;
    //список смежности
public:
    void add_way(char from, char to, double length){                                    //
функция добавления пути и вершины в список
#ifdef TEST
    std::cout<<"Добавляем путь из: "<<from<<" в "<<to<<' '<<length<<std::endl;
#endif
    if(from==to)
            return;
    std::pair<char,double> way(to,length);
    if(list.find(from)==list.end()){
    //добавляем вершину "из", если ее нет в графе
            std::map<char,double> l;
            l.insert(way);
            list.insert(std::pair<char,std::map<char,double>>(from,l));
            return;
    }
    list[from].insert(way);
    //добавляем путь из вершины, если эта вершина уже есть в списке
    }
```

```
    double heuristic(char from, char to)
    //функция подсчета эвристики
    {
       return abs(static_cast<long>(from) - static_cast<long>(to));
    }

    double find_way_cost(char from, char to){
    //функция, возвращающая длину пути до инцидентной вершины
    if(list.find(from)==list.end())
            return 0.0;
    auto i=list[from].find(to);
    if(i==list[from].end())
            return 0.0;
    return i->second;
    }
    void print_q(std::priority_queue <std::pair<double, char>,
    std::vector < std::pair<double, char> >,
    std::greater< std::pair<double, char>>> q){                                    //
функция печати очереди с приоритетом
    std::cout<<"Queue:\n";
    while(!q.empty()){
            std::cout<<"Вершина: "<<q.top().second<<", эвристика + цена =
"<<q.top().first<<std::endl;
            q.pop();
    }
    std::cout<<std::endl;
    }
    double get_cost(std::string way){
    //возвращает длину пути без учета эвристики
    double sum=0.0;
    for(auto i=0;i<way.length()-1;i++){
            sum+=find_way_cost(way[i],way[i+1]);
    }
    return sum;
    }
```

```
    void is_adm(std::string way){
    //функция проверяет на допустимость и монотонность
    bool is_admis=true,is_mon=true;
    char to=*way.rbegin();
    for(auto i=0;i<way.length()-1 && is_admis;i++){
            auto cost=get_cost(way.substr(i));
            if(heuristic(way[i],to)>cost){                                  //если не
допустима
                    is_admis=false;
                    break;
            }
            if(heuristic(way[i],to)>find_way_cost(way[i],way[i+1])
+heuristic(way[i+1],to))  //если не монотонна
                    is_mon=false;
    }
    if(is_admis){
            std::cout<<"Функция допустима!\n";
            if(is_mon){
                    std::cout<<"Функция монотонна!\n";
            }
            else{
                    std::cout<<"Функция не монотонна!\n";
            }
    }
    else
            std::cout<<"Функция не допустима и не монотонна!\n";
    }

    void find_min_way(char from, char to){
    //функция поиска наименьшего пути
    std::priority_queue <std::pair<double, char>,
    std::vector < std::pair<double, char> >,
    std::greater< std::pair<double, char>>> q;                  //очередь с
приоритетом
    q.push({0,from});
#ifdef TEST
```

```
    std::cout<<"Начинаем поиск!\n";
#endif
    std::map<char, double> way_cost;                    //список пар с
минимально ценой до просмотренных вершин
    std::map<char,char> point_hist;                     //список путей
текущего результирующего путя
    while(!q.empty()){                                  //пока
список не пуст
            auto cur = q.top();
            q.pop();
            if(cur.second==to){
    //если найден конец
#ifdef TEST
                    std::cout<<"Конец достигнут!\n";
#endif
                    break;
            }
#ifdef TEST
            std::cout<<"Ищем пути из: "<<cur.second<<std::endl;
#endif
            for(auto i=list[cur.second].begin();i!=list[cur.second].end();i++){
                                    //просматриваем все инцидентные вершины
#ifdef TEST
                    std::cout<<"Просматриваем путь: "<<cur.second<<"->"<<i-
>first<<" за "<<i->second<<std::endl;
#endif
                    double new_cost=way_cost[cur.second]
+find_way_cost(cur.second,i->first);                    //считаем
оценку до вершины
                    if(!way_cost.count(i->first) || new_cost<way_cost[i->first]){
                                    //если в вершину еще не ходили, либо
найден путь меньше предыдущего
#ifdef TEST
                            std::cout<<"Добавляем вершину в очередь!\n";
#endif
                            way_cost[i->first]=new_cost;
```

```
                        std::cout<<cur.second<<"->"<<i->first<<' '<<
new_cost<<std::endl;
                        q.push({new_cost+heuristic(i->first,to),i->first});
                                        //опускаем в очередь
                        point_hist[i->first]=cur.second;
                }
        }
#ifdef TEST
                print_q(q);
#endif

    }
    std::string answer;
    answer.push_back(to);
                                                //печать ответа
    while (answer.back() != from)
        answer.push_back(point_hist[answer.back()]);
    std::reverse(answer.begin(), answer.end());
    is_adm(answer);
    std::cout<<"Результат: \n";
    for (auto e : answer)
        std::cout << e;
    std::cout<<std::endl;
    }

};

int main(){
    Graph a;
    char start,end,from,to;
    std::cin>>start>>end;
    double length;
    while(std::cin>>from){
    std::cin>>to>>length;
    a.add_way(from, to, length);
    }
```

```
    a.find_min_way(start,end);
    return 0;
}
```