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*

Студентка гр. 1304 _____ Виноградова М.О.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2023

Цель работы.

Решить задачу построения пути в ориентированном графе при помощи жадного алгоритма и алгоритма A стар.

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.

Входные данные.

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

Выходные данные.

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

Выполнение работы.

Жадный алгоритм

Для хранения данных использовалась переменная **graph** типа map, где ключом являлась сама вершина, а значением – массив пар смежных с ней вершин и их весов.

way – переменная для хранения пути, типа vector<char>.

Функция **initGraph()** – осуществляет инициализацию графа. Пока пользователь вводит корректные данные формата char, char, float, они записываются в переменную graph.

Функция **printWay()** – выводит на экран итоговый путь между введёнными вершинами.

Функция **greedAlgorithm(char start)** – принимает на вход вершину от которой нужно найти дальнейший путь. Далее находит самое дешевое ребро из смежных и добавляет его в путь, удаляя из графа (так как вершина уже просмотрена). Если у вершины нет смежных с ней вершин, то она удаляется из пути.

Алгоритм будет работать до тех пор, пока в пути не окажется искомая вершина.

Код приведен в приложении А.

Задание.

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.

Входные данные.

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

Выходные данные.

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

Выполнение работы.

Алгоритм A*

Для хранения данных использовалась переменная **graph** типа map, где ключом являлась сама вершина, а значением – массив пар смежных с ней вершин и их весов.

way – переменная для хранения пути, типа vector<char>.

Также для хранения полного пути (с учетом эвристики) используется переменная **f** – типа map<char,float>, для хранения пути без эвристики используется переменная **g**(такого же типа). Для определения через какую вершину был улучшен путь для рассматриваемой вершины используется переменная **parent** – типа map<char,char>.

Функция **initGraph()** – осуществляет инициализацию графа. Пока пользователь вводит корректные данные формата char, char, float, они записываются в переменную graph. Также в словарь **g** по ключу каждой вершины записывается максимально возможное число INT_MAX(необходимо для работы алгоритма).

Функция **h(char start,char end)** – возвращает модуль разницы между двумя символами в ASCII таблице. Является эвристической функцией.

Функция **findMin(std::vector<char>Q)** – находит в переданном массиве вершину с минимальной оценкой пути (по словарю f) и возвращает ее.

Функция **Astar(char start, char goal)** – реализует алгоритм поиска кратчайшего пути между двумя вершинами в графе, с помощью эвристической функции (h). Для работы необходимо два массива вершин: **U** – уже просмотренные вершины, **Q** – вершины, которые необходимо рассмотреть. На первом шаге в Q необходимо добавить стартовую вершину, а также изменить оценку данной вершины в словаре g на ноль, тогда в полном пути будет учитываться только эвристика. Алгоритм продолжает свою работу пока массив Q не пуст или пока не найдено решение. На каждом шаге находиться лучшая (те с самой маленькой оценкой по f) вершина, далее она удаляется из Q и записывается в просмотренные вершины. Если найденная вершина не является

решением, тогда рассматриваются все смежные с ней вершины. Вычисляется путь через минимальную на данном шаге вершину (те значение из g по минимальной вершине плюс вес ребра между минимальной и смежной с ней). Если смежная вершина еще не была просмотрена или путь до нее может быть улучшен, то данные обновляются и фиксируется вершина, через которую были произведены улучшения. Если смежной вершины нет в Q, то она добавляется в данный массив.

Функция **initWay(char end)** – строит оптимальный путь по меткам полученным в результате работы функции Astar().

Функция **printWay(std::vector<char> way)** – инвертирует путь (так как изначально он строился от конечной вершины до стартовой) и выводит его на экран.

Код приведен в приложении А.

Выводы.

В результате работы реализованы два алгоритма для нахождения кратчайшего пути в графе: жадный алгоритм и алгоритм А стар. Жадный алгоритм на каждом шаге получает локально оптимальное решение. Алгоритм А стар с помощью эвристической функции и данными о уже найденных путях может улучшать текущий путь. В качестве эвристической оценки использовалась разница в ASCII кодах символов (текущем и финальным), которые являются вершинами в графе. Также были написаны необходимые для реализации работы алгоритмов функции. Выполнена задача лабораторной работы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include "math.h"
#include <algorithm>
#include <float.h>

// граф храниться в формате вершина -> массив смежных вершин в виде пары
(вершина, вес пути)
std::map <char, std::vector<std::pair<char,float>>> graph;
//переменная для хранения итогового пути
std::vector<char> way;

//функция инициализации графа
//пока пользователь вводит корректные данные, они записываются в граф
void initGraph(){
    char a,b;
    float f;
    while(std::cin>>a>>b>>f){
        std::pair<char,float> q(b,f);
        graph[a].push_back(q);
    }
}

//функция вывода в консоль пути
void printWay(){
    for(int i=0;i<way.size();i++){
        std::cout<<way[i];
    }
}

//функция, которая реализует жадный алгоритм, проходится по всем смежным
вершинам и находит самый дешевый путь
//если пути равны, то сравниваются их ASCII символы
//просмотренная вершина удаляется из графа, если все смежные с переданной
вершины просмотрены, то удаляется верхняя вершина из пути
void greedAlgorithm(char start){

    float min =FLT_MAX;
    int min_i;
    int min_c='z';
    for(int i=0;i<graph[start].size();i++){//цикл по смежным вершинам
        if(min>graph[start][i].second){//лучший путь
            min = graph[start][i].second;
            min_i = i;
        }
        if(min==graph[start][i].second){//если есть два одинаковых пути
            if(min_c>graph[start][i].first){
                min_c=graph[start][i].first;
                min_i = i;
```

```
            }
        }
    }

    if (graph[start].size()==0){
        way.pop_back();
    }else{
        way.push_back(graph[start][min_i].first);
        std::remove(graph[start].begin(),
graph[start].end(),graph[start][min_i]);
        graph[start].resize(graph[start].size()-1);
    }
}


int main() {

    //получение стартовой вершины и конечной вершины
    char start,end;
    std::cin>>start>>end;

    //инициализация графа
    initGraph();

    //добавление стартовой вершины в путь
    way.push_back(start);

    //пока не достигнута конечная вершина, вызывается функция жадного по-
иска
    while(count(way.begin(),way.end(),end)==0){
        greedAlgorithm(way[way.size()-1]);
    }

    //вывод итогового пути
    printWay();

    return 0;
}
```

Название файла: star.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include "math.h"
#include <algorithm>
#include <float.h>
#include <limits.h>
//граф, который вводит пользователь
std::map <char, std::vector<std::pair<char,float>>> graph;
//словарь для оценки пути от вершины
std::map<char,float>g;
//словарь для оценки полного пути
std::map<char,float>f;
//словарь для определения вершины, через которую были произведены улучше-
ния
std::map<char,char>parent;
```

```
//функция инициализации графа
//пока пользователь вводит корректные данные, они записываются в граф
void initGraph(){
    char v1,v2;
    float len;
    while(std::cin>>v1>>v2>>len){
        std::pair<char,float> tmp_el(v2,len);
        //инициализация графа
        graph[v1].push_back(tmp_el);
        //инициализация словаря g
        g[v1] = INT_MAX;
        g[v2] = INT_MAX;
    }
}

//эврестическая функция
float h(char start,char end){
    return abs(int(end)-int(start));
}

//функция для нахождения минимальной вершине (по оценки пути)
char findMin(std::vector<char>Q){
    char current = Q[0];
    int min=INT_MAX;
    for(int i=0;i<Q.size();i++){
        if(min>=f[Q[i]]) {
            current = Q[i];
            min = f[Q[i]];
        }
    }
    return current;
}

//функция, которая реализует алгоритм А стар
//пока список вершин не пуст (или не найлено решение), находится лучшая
вершина из списка. Она удаляется из этого списка и добавляется в список
просмотренных вершин.
// Далее проверяется можно ли улучшить через данную вершину путь до сме-
ных с ней вершин
bool Astar(char start,char goal){
    std::vector<char> U;
    std::vector<char> Q;
    Q.push_back(start);
    g[start]=0;
    f[start] = g[start]+h(start,goal);

    while(Q.size() != 0){
        char current = findMin(Q);

        std::remove(Q.begin(), Q.end(),current);//удаление
        Q.resize(Q.size()-1);

        if (current == goal){
            return true;
        }

        U.push_back(current);
```

```
        for(int i=0;i<graph[current].size();i++){
            char v=graph[current][i].first;
            float score = g[current] + graph[current][i].second;

            if (count(U.begin(),U.end(),v)!=0 && score>=g[v]){
                continue;
            }

            if(count(U.begin(),U.end(),v)==0 || score<g[v]){
                if(score<g[v]){
                    parent[v] = current;
                    g[v]=score;

                }
                f[v] = g[v] + h(v,goal);

                if (count(Q.begin(),Q.end(),v)==0){
                    Q.push_back(v);
                }
            }
        }
    }
    return false;
}
//функция для нахождения оптимального пути через метки, по которым улуч-
шались вершины(словарь parent)
std::vector<char> initWay(char end){
    std::vector<char> way;
    way.push_back(end);
    char tmp=end;
    while(parent[tmp]){
        way.push_back(parent[tmp]);
        tmp = parent[tmp];
    }
    return way;
}

//функция для вывод в консоль найденного пути
void printWay(std::vector<char> way){
    for(int i=way.size()-1;i>=0;i--){
        std::cout<<way[i];
    }
}
int main() {
    //начальная и конечная вершины
    char start,end;

    //получение начальной и конечной вершин
    std::cin>>start>>end;

    //инициализация графа
    initGraph();

    //алгоритм А стар
    Astar(start,end);

    //вывод на экран
```

```
    printWay(initWay(end));

    return 0;
}
```