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

ОТЧЕТ

по практической работе №2
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритмы на графах

Студент гр. 7383 _____ Кирсанов А.Я.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Постановка задачи.

Цель работы.

Решение задачи поиска кратчайшего пути с помощью алгоритма A* в графе с двумя финишами (требуется найти путь до любого из них).

Реализация задачи.

Был создан класс **Matrix** со следующими полями:

unsigned int size – размер матрицы.

char begin – начало пути.

char end – первый финиш.

char end2 – второй финиш.

vector<vector<unsigned> > matrix – матрица 26x26, заполненная нулями.

vector<unsigned> camefrom – вектор, хранящий последовательность рассмотренных вершин.

vector<double> costsofar – вектор, хранящий длину пути до данной вершины.

vector<double> queue – вектор, хранящий эвристическую оценку вершин до первого финиша.

vector<double> queue2 – вектор, хранящий эвристическую оценку вершин до второго финиша.

vector<char> finalway – вектор, хранящий последовательность вершин, ведущих к одному из финишей кратчайшим путем.

Matrix (unsigned int N) – конструктор объекта класса. Инициализирует вектор **matrix** нулями.

~Matrix () – деструктор класса.

void setLeng(unsigned i, unsigned j, double leng) – ставит по координатам **i** и **j** длину пути между вершинами в матрицу.

void read() – функция, считывающая заданное количество вершин и весов соединяющих их ребер, заполняет матрицу и вызывает функцию поиска кратчайшего пути.

bool findtheway(unsigned start) – функция, описывающая алгоритм A*. На вход принимается начальная вершина. Пока не будет найдена одна из конечных вершин или пока очередь вершин не опустеет, берет из очереди вершину с минимальной эвристической оценкой и рассматривает её соседей, рассчитывает для них эвристические оценки до обоих финишей, первая оценка кладется в очередь под номером 1, вторая – во вторую. Каждому соседу ставится её предок. Если найден более короткий путь до соседа, эвристические оценки пересчитываются. Когда все соседи положены в очередь, вершина удаляется из рассмотрения.

double h(unsigned current)/ double h2(unsigned current) – функции эвристической оценки расстояний от данной вершины до конечных.

Функция **main()** создает объект класса **Matrix** и вызывает функцию **Matrix :: read()**.

Исходный код программы представлен в Приложении Б.

Исследование сложности алгоритма.

Функция **findtheway** в худшем случае пройдет по всем вершинам за $|V|$ итераций и для каждой вершины два раза задаст эвристическую оценку за $|V| + |E|$ итераций. Здесь $|V|$ - множество вершин, а $|E|$ - множество ребер. Таким образом количество итераций в худшем случае $2|V|^2 + 2|V||E|$, сложность равна $O(|V|^2 + |V||E|)$.

Тестирование.

Программа тестировалась в среде разработки Qt с помощью компилятора MinGW 5.3.0 в операционной системе Windows 10.

Тестовые случаи представлены в Приложении А.

Вывод.

В ходе выполнения задания был реализован алгоритм A*, находящий путь минимальной длины до одной из вершин графа. Также оценена сложность алгоритма.

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Выходные данные
4 a j k a e 1.01 a f 2 e j 2 f k 1	afk
6 a z k a b 1 a c 1 b d 1 c d 3 c k 5 d z 1	ack
6 a e z a b 1 a c 1 c d 3 d e 1 c z 5 b d 1	abde

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <limits>
#include <cmath>

using namespace std;

bool is_equal(double x, double y) {
    return fabs(x - y) < std::numeric_limits<double>::epsilon();
}

class Matrix{
private:
    unsigned size;
    char begin;
    char end;
    char end2;

    vector<vector<double> > matrix;
    vector<unsigned> camefrom;
    vector<double> costsofar;
    vector<double> queue;
    vector<double> queue2;
    vector<char> finalway;



public:
    Matrix() : size(26){
        for(unsigned int i = 0; i < 26; i++){
            vector<double> temp;
            for(unsigned int j = 0; j < 26; j++)
                temp.push_back(0);
```

```
            matrix.push_back(temp);

            camefrom.push_back(0);
            costsofar.push_back(0);
            queue.push_back(numeric_limits<double>::max());
            queue2.push_back(numeric_limits<double>::max());
        }
    }

    void setLeng(unsigned i, unsigned j, double leng){
        matrix[i][j] = leng;
    }

    ~Matrix(){
        for(unsigned int i = 0; i < size; i++){
            matrix[i].clear();
        }
        matrix.clear();
        camefrom.clear();
        costsofar.clear();
        queue.clear();
        queue2.clear();
        finalway.clear();
    }

    void read();
    bool findtheway(unsigned start);
    double h(unsigned current);
    double h2(unsigned current);
};

void Matrix :: read(){
    double leng;
    char i, j;
```

```
    int N;
    cin >> N;
    cin >> begin;
    cin >> end;
    cin >> end2;

    for (int k = 0; k < N; k++) {
        cin >> i;
        cin >> j;
        cin >> leng;
        setLeng(static_cast<unsigned>(i) - 97, static_cast<unsigned>(j) -
97, leng);
    }

    if(findtheway(static_cast<unsigned>(begin) - 97)){
        for (vector<char> :: iterator it = finalway.begin(); it !=
finalway.end(); ++it)
                cout << *it;
    }
    else cout << "no way";
}

double Matrix :: h(unsigned current){
    return abs(static_cast<double>(end) - 97 - current);
}

double Matrix :: h2(unsigned current){
    return abs(static_cast<double>(end2) - 97 - current);
}

bool Matrix :: findtheway(unsigned start){
    queue[start] = 0;
    costsofar[start] = 0;
    double min = numeric_limits<double>::max(), tmpF;
```

```
    unsigned k, current = start;
    while(queue.size() != 0){
        min = numeric_limits<double>::max();
        for (k = 0; k < queue.size(); k++) {
            if(queue[k] < min && queue[k] > 0){
                min = queue[k];
                current = k;
            }
        }
        for (k = 0; k < queue2.size(); k++) {
            if(queue2[k] < min && queue2[k] > 0){
                min = queue2[k];
                current = k;
            }
        }

        if(current == static_cast<unsigned>(end) - 97 || current ==
static_cast<unsigned>(end2) - 97){
            finalway.insert(finalway.begin(), static_cast<char>(current +
97));
            unsigned d = current;
            while(camefrom[d] != start){
                finalway.insert(finalway.begin(),
static_cast<char>(camefrom[d] + 97));
                d = camefrom[d];
            }
            finalway.insert(finalway.begin(), static_cast<char>(start +
97));
            return true;
        }

        if(current != start && is_equal(min,
numeric_limits<double>::max())){
            return false;
```

```
        }

        for (unsigned j = 0; j < matrix[current].size(); j++) {
            if(matrix[current][j] > 0){
                tmpF = costsofar[current] + matrix[current][j];

                if((is_equal(costsofar[j], 0) || tmpF < costsofar[j])){
                    costsofar[j] = tmpF;
                    queue[j] = tmpF + h(j);
                    queue2[j] = tmpF + h2(j);
                    camefrom[j] = current;
                }
            }
        }
        queue[current] = numeric_limits<double>::max();
        queue2[current] = numeric_limits<double>::max();
    }
    return false;
}

int main()
{
    Matrix mt;
    mt.read();
    return 0;
}
```