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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «ПиАА»

Тема: Жадный алгоритм и A*

Студент(ка) гр. 0000 _____ Ивченко А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Ознакомится с алгоритма A* нахождения лучшего пути в ориентированном графе.

Формулировка задания.

1.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.

Вар. 2. В A* эвристическая функция для каждой вершины задаётся неотрицательным числом во входных данных.

Описание алгоритма.

Алгоритм принимает в качестве аргументов матрицу смежности графа и словарь map, которые определяются исходя из входных данных. По словарю можно по ключу названия вершины определить ее эвристическую функцию, заданную пользователем.

В начале работы алгоритма создается закрытый список элементов closedset (список пройденных вершин) и открытый список openset(список элементов, которых предстоит пройти) и словарь fromset, в котором по ключу названия вершины можно определить предшествующую для нее вершину. Вершина представляет собой структуру с полями: название вершины, значение стоимость от начала пути G(x) и оценочное значение F(x). Начальная вершина имеет G(x) = 0, а F(x) равняется значению пользовательской эвристики для данной вершины.

Основный цикл алгоритма продолжается пока список openset не пуст. В теле цикла находится элемент openset с минимальным значением F, который

впоследствии добавляется в closedset и удаляется из openset. В процессе прохода по соседям, каждая смежная вершина проверяется на наличие в closedset. Если она там есть, то эта вершина игнорируется. Далее проверяется есть ли этот сосед в openset: если да, то при условии, что путь через эту вершину наилучший, его характеристики обновляются в соответсвии с матрицей смежности и словарем map; если его нет в openset, то его характеристики обновляются тем же образом и вершина заносится в открытый список.

Временная сложность алгоритма: $O(e+v^2)$, где e – кол-во вершин, v – кол-во ребер

Тестирование

```
a e
a b 3 4
b c 1 3
c d 1 2
a d 5 4
d e 1 1
^Z
ade
```

```
a e
a b 3 4

b c 1 3
c d 1 2
a d 6 4
d e 1 1
^Z
abcde
```

Вывод.

В ходе лабораторной был разобран алгоритм A* нахождения лучшего пути в ориентированном графе, а также вычислена его оценочная сложность

Исходный код

```
	#include<iostream>
#include <string>
#include<vector>
#include<cstdlib>
#include<math.h>
#include<map>
#include <algorithm>
#include <stack>

using namespace std;

struct Vertex {
	int id;
	double G, F;
};


void RECONSTRUCT_PATH(std::map<char, char>& from, char start, char end){


	std::vector<char> pathset;
	char current = end;
	pathset.push_back(current);

	do{

		current = from[current];
		pathset.push_back(current);


	} while (from[current] != start);

	pathset.push_back(start);

	std::reverse(pathset.begin(), pathset.end());

	for (auto a : pathset)
		cout << a;

	std::cout << "\n";

}
void tentative_is_better(Vertex *neighbour, int t, std::map<char, int>& cmp) {//обновление
характеристик соседней вершины

	neighbour->G = t;
	neighbour->F = neighbour->G + cmp[neighbour->id];
	//cout << neighbour->F;

}

void A(Vertex start, Vertex end, int **m, int n, std::map<int, int> &cmp)
{
	vector <Vertex> closedset;
	vector <Vertex> openset;
	std::map<char, char> fromset;

	start.G = 0;
	start.F = start.G + cmp[start.id];
	openset.push_back(start);
```

```
	while (openset.size() != 0) {

		Vertex current;

		int index = 0;
		int min_F = 1000;
		for (int i = 0; i < openset.size(); i++) {// нахождение элемента openset с
мин. оценкой F;
			if (min_F > openset[i].F)
				min_F = openset[i].F;
		}
		for (int i = openset.size() - 1; i >= 0; i--) {
			if (min_F == openset[i].F) {
				index = i;
				break;
			}
		}

		current = openset[index];


		if (current.id == end.id) {
				RECONSTRUCT_PATH(fromset, start.id, end.id);
				return;

		}

		closedset.push_back(openset[index]);
		openset.erase(openset.begin() + index);

		for (int i = 0; i < n; i++) {//прохождение по соседям
			if (m[current.id - 97][i] != 0) {//в матрице смежности

				Vertex *neighbour;
				bool in_cs = 0;

				for (int j = 0; j < closedset.size(); j++)
					if (i + 97 == closedset[j].id)
						in_cs = 1;

				if(in_cs) continue;

				int tentative_g_score = m[current.id - 97][i] +
current.G;//предв оценка g эл (A*)

				int count = 0;

				for (int j = 0; j < openset.size(); j++) {

					if (i + 97 == openset[j].id) {

						neighbour = &openset[j];
						if (tentative_g_score < neighbour->G) {

							fromset[i + 97] = current.id;
							neighbour->G = tentative_g_score;
							neighbour->F = tentative_g_score + cmp[i +
97];


						}
						else break;
					}
					else count++;
```

```
                        }
                        if(count == openset.size()) {

                                fromset[i + 97] = current.id;
                                neighbour = new Vertex;

                                neighbour->id = i + 97;

                                neighbour->G = tentative_g_score;
                                neighbour->F = tentative_g_score + cmp[i+97];

                                openset.push_back(*neighbour);

                        }

                }
        }


    }
}

int main() {

    char start, end, a, b;
    double c, h;

    std::map<int,int> cmp;

    std::cin >> start >> end;
    double n = end - 96;

    int** m = new int*[n];
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
        for (int j = 0; j < n; j++)
            m[i][j] = 0;
    }
    do {

        std::cin >> a >> b >> c >> h;
        m[a - 97][b - 97] = c;
        int m = static_cast<int>(a);
        cmp[m] = h;

    }
    while (std::cin);

    Vertex v_start, v_end;

    v_start.id = static_cast<int>(start);
    v_end.id = static_cast<int>(end);

    A(v_start, v_end, m, n, cmp);

    delete[] m;
    }
```