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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритмы на графах

Студент гр. 8304 _____ Ястребов И.М.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Разработать 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.

Вариант 6.

Реализация очереди с приоритетами, используемой в A*, через двоичную кучу.

Описание алгоритма.

В процессе работы алгоритма для вершин рассчитывается функция f(v) = g(v) + h(v), где

g(v) - наименьшая стоимость пути в v из стартовой вершины,

h(v) - эвристическое приближение стоимости пути от v до конечной цели.

Фактически, функция f(v) - длина пути до цели, которая складывается из пройденного расстояния g(v) и оставшегося расстояния h(v). Исходя из этого, чем меньше значение f(v), тем раньше мы откроем вершину v, так как через неё мы предположительно достигнем расстояние до цели быстрее всего. Открытые алгоритмом вершины хранятся в очереди с приоритетом по значению f(v).

Сложность алгоритма: O(|v|*|v| + |k|), где

v- множество вершин,

k - множество ребер.

Описание основных структур данных и функций.

```
std::map<char, std::vector<std::pair<char, int>>> desk;
```

- словарь, в котором хранится граф по принципу [вершина]-(все вершины, соединенные с данной ребрами и стоимости ребер)

```
std::priority_queue<std::pair<int, char>, std::vector<std::pair<int, char>>,
std::greater<std::pair<int, char>>> priorities;
```

- очередь с приоритетами. Хранит пары с вершиной для перехода и ее приоритетом. Для сравнивания используем std::greater, top() возвращал значение с минимальным приоритетом, а не максимальным.

```
std::map<char, char> path;
```

- словарь для хранения «предыдущих» вершин.

```
std::map<char, int> cost;
```

- словарь для хранения стоимости пути от старта до текущей вершины.

```
int heuristic(char& first, char& second);
```

- эвристическая функция, подсчитывающая близость символов, обозначающих вершины графа, в таблице ASCII.

```
void printResult(char& start, char& finish, std::map<char, char>& prev, int
menu);
```

- функция, выводящая конечный результат. В зависимости от значения menu происходит вывод либо в консоль, либо в файл.

Тестирование.

Ввод	**Вывод**
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	Result: ade
a e a b 3.0 b d 2.0 b e 3.0 d e 4.0 e a 1.0	Result: abe

e f 2.0 a g 8.0 f g 1.0	
b e a b 1.0 a e 2.0 a c 2.0 b d 7.0 a g 2.0 d e 4.0 g e 1.0	Result: bde

Вывод.

В ходе выполнения данной работы была написана программа, которая решает задачу построения кратчайшего пути в ориентированном графе методом A*.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include "pch.h"
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <queue>
#include <fstream>
#include <string>

int heuristic(char& first, char& second); // Эвристическая функция
void printResult(char& start, char& finish, std::map<char, char>& prev, int
menu); // Вывод результата

static std::string inPath("./input.txt");
static std::string outPath("./output.txt"); //Ввод и вывод из файла

int heuristic(char& first, char& second) {
	return std::abs(first - second);
}

void printResult(char& start, char& finish, std::map<char, char>& prev, int
menu) { //Вывод результата
	std::string result;

	char current = finish;

	result += current;

	while (current != start) { //Формируем строку результата
		current = prev[current];
		result += current;
	}

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

	if (menu == 1) { //Выводим результирующий путь
		std::cout << "Result: ";

		std::cout << result;
	}

	else { // То же самое, но с файлом, а не консолью
		std::ofstream file;

		file.open(outPath);

		if (!file.is_open()) {
			std::cout << "Can't open file!\n";
		}

		file << "Result: ";
```

```
			file << result;
		}
}

int main() {
		std::map<char, std::vector<std::pair<char, int>>> desk; // Храним граф в
виде словаря
		char start(0), destination(0);

		char edgeLeft(0), edgeRight(0);
		float edgeCost(0);

		std::cout << "Choose input format" << std::endl << std::endl
			<< "1 - console" << std::endl
			<< "2 - file" << std::endl; // Выбор варианта ввода

		int mode = 0;
		std::cin >> mode;

		if (mode == 1) {//Считывание с консоли
			std::cin >> start >> destination;

			while (std::cin >> edgeLeft >> edgeRight >> edgeCost) {
				if (edgeLeft == '-')
					break;

				desk[edgeLeft].push_back(std::make_pair(edgeRight,
edgeCost));
			}
		}

		else if (mode == 2) {//Считывание из файла
			std::ifstream file;

			file.open(inPath);

			if (!file.is_open()) {
				std::cout << "Can't open file!" << std::endl;
				return 0;
			}

			file >> start >> destination;

			while (file >> edgeLeft >> edgeRight >> edgeCost) {
				desk[edgeLeft].push_back(std::make_pair(edgeRight,
edgeCost));
			}
		}


		std::priority_queue<std::pair<int, char>,
					std::vector<std::pair<int, char>>,
					std::greater<std::pair<int, char>>>
priorities;
```

```
	priorities.push(std::make_pair(0, start));//Объявляем очередь, вносим
начальную вершину

	std::map<char, char> path;//Путь храним как множество его
последовательных ребер
	std::map<char, int> cost;//Стоимость пути до вершины

	path[start] = start;
	cost[start] = 0;

	while (!priorities.empty()) {//Пока не прошерстим всю очередь
		char current = priorities.top().second;//Выбираем вершину согласно
приоритету
		priorities.pop();

		std::cout << "Current node: " << current << std::endl;
		std::cout << "Current result path";
		printResult(start, current, path, 1); //Выводим промежуточные
данные в консоль
		std::cout << std::endl;

		if (current == destination)//Если дошли, куда хотели, то
заканчиваем обход
			break;

		for (auto& next : desk[current]) {//Обходим все возможные
продолжения пути
			int new_cost = cost[current] + next.second;//Высчитываем
стоимость пути в каждую новую вершину

			if (!cost.count(next.first) || new_cost < cost[next.first]) {
				cost[next.first] = new_cost; //Если новый путь короче
или мы тут впервые, обновляем стоимость

				int priority = new_cost + heuristic(next.first,
destination);//Пересчитываем приоритет

				priorities.push(std::make_pair(priority,
next.first));//Заносим в очередь

				path[next.first] = current;//Предыдущая для следующей -
это текущая
			}
		}
	}

	std::cout << std::endl << "Console or file output?" << std::endl <<
std::endl
		<< "1 - console" << std::endl//Выбираем вариант вывода
		<< "2 - file" << std::endl;

	std::cin >> mode;

	printResult(start, destination, path, mode);//Вызываем вывод результата
```

```
        return 0;
}
```