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 _____ Бочаров Ф.Д.

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>>> card;
```

- словарь для хранения графа. Для каждой вершины хранится вектор с парами, в которых хранится вершина, в которую можно перейти, и расстояние до нее.

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

- очередь с приоритетами через двоичную кучу. Хранит пары с названием

вершины для перехода и ее приоритетом. В качестве перегрузки используем std::greater, чтобы функцией top() возвращалось значение с минимальным приоритетом, а не максимальным.

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

- словарь для хранения вершины, из которой мы пришли в текущую.

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

- словарь для хранения расстояния от старта до текущей вершины.

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

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

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

- 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 g a b 3.0 a c 1.0 b d 2.0 b e 3.0 d e 4.0 e a 1.0 e f 2.0 a g 8.0	Result: ag

f g 1.0	
a f a c 1.0 a b 1.0 c d 2.0 b e 2.0 d f 3.0 e f 3.0	Result: abef
a d a b 1.0 b c 9.0 c d 3.0 a d 9.0 a e 1.0 e d 3.0	Result: aed
b e a b 1.0 a c 2.0 b d 7.0 b e 8.0 a g 2.0 b g 6.0 c e 4.0 d e 4.0 g e 1.0	Result: bge

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 <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <queue>
#include <fstream>
#include <string>


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

void print(char& start, char& finish, std::map<char, char>& prev) {
	std::vector<char> result;
	char current = finish;
	result.push_back(current);
	while (current != start) {
		current = prev[current];
		result.push_back(current);
	}
		std::cout << "Result: ";
		for (unsigned long int i = 0; i < result.size(); ++i) {
			std::cout << result[result.size() - i - 1];
		}

}

int main() {

	char start, finish;
	std::map<char, std::vector<std::pair<char, int>>> card;// граф
	char first, second;
	float len;

			std::cin >> start >> finish;
			while (std::cin >> first >> second >> len) {
				if (len == -1)
					break;
				card[first].push_back(std::make_pair(second, len));
			}


	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));//объявление очереди через двоичную кучу,
вносим начальную вершину
	std::map<char, char> prev;//откуда пришли в вершину
	std::map<char, int> cost;//стоимость всего пути до вершины
	prev[start] = start;
	cost[start] = 0;
	while (!priorities.empty()) {//пока очередь не станет пустой
		char current = priorities.top().second;//берем вершину с наименьшим
приоритетом
		priorities.pop();
		std::cout << "Visiting: " << current << std::endl;
		std::cout << "Current ";
		print(start, current, prev);
		std::cout << std::endl;
		if (current == finish)//если доходим до конца
```

```
                    break;//то завершаем цикл
                for (auto& next : card[current]) {//для текущей вершины прогоняем все
возможные пути
                    int new_cost = cost[current] + next.second;//считаем стоимость пути в
каждую из новых вершин
                    if (!cost.count(next.first) || new_cost < cost[next.first]) {//если
впервые в этой вершине
                        cost[next.first] = new_cost;//или новый возможный путь короче,
то обновляем стоимость
                        int priority = new_cost + heuristic(next.first,
finish);//считаем приоритет
                        priorities.push(std::make_pair(priority, next.first));//заносим
в очередь
                        prev[next.first] = current;//в следующую вершину пришли из
текущей
                    }
                }
        }

        print(start, finish, prev);
        return 0;
}
```