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 Нам Ё Себ

Преподаватель Размочаева Н.В.

Санкт-Петербург

2020

Вариант 4.

Цель работы.

Построение и анализ алгоритма A* на основе на решения задачи о нахождении минимального пути в графе.

Основные теоретические положения.

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.

Описание алгоритма.

Для решения поставленной задачи был реализован алгоритм A*. В качестве эвристической функции была использована функция heuristic(q1, q2), возвращающая расстояние между двумя символами. Очередь с приоритетами была реализована на основе массиве. В начале каждой итерации в массиве ищется элемент приоритет, которого минимален, он удаляется из очереди, и начинается осмотр всех ребер выходящих из выбранного элемента. Если нашлась вершина путь до которой был больше чем найденный, то данный путь заменяется на найденный. Для хранения значений имен узлов и ребер выходящих из низ был использован словарь. И структура Elem, хранящая ребра выходящие из текущей вершины, имя вершины из которой был найден минимальный путь и длина до начальной позиции. Сложность алгоритма: O(|V|*|V| + |E|), где V – множество вершин, а E – множество ребер.

Вывод промежуточной информации.

Во время основной части работы алгоритма происходит вывод промежуточной информации а именно, выбранная на данном вершина (вершина с меньшим приоритетом), вершина путь до которой был изменен при помощи ребра выходящего из выбранной вершины. Также выводится скорость работы алгоритма и его сложность.

Тестирование.

Таблица 1 – Результаты тестирование

Ввод	**Вывод**
a l m a b 1.000000 a f 3.000000 b c 5.000000 b g 3.000000 f g 4.000000 c d 6.000000 d m 1.000000 g e 4.000000 e h 1.000000 e n 1.000000 n m 2.000000 g i 5.000000 i j 6.000000 i k 1.000000 j l 5.000000 m j 3.000000	For first_end: abgenmjl For second_end: abgenm
g j m a b 1.000000 a f 3.000000 b c 5.000000 b g 3.000000 f g 4.000000 c d 6.000000 d m 1.000000 g e 4.000000 e h 1.000000 e n 1.000000 n m 2.000000 g i 5.000000 i j 6.000000	For first_end: genmj For second_end: genm

i k 1.000000 j l 5.000000 m j 3.000000	
a f i a b 0 a c 0 b d 0 c e 0 c s 0 d f 0 d s 0 s i 0 e f 0	For first_end: acef For second_end: acsi
a e b a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	For first_end: ade For second_end: ab

Вывод.

В ходе работы был построен и анализирован алгоритм A* на основе решения задачи о нахождении минимального пути в графе. Исходный код программы представлен в приложении 1.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <queue>
#include <algorithm>
#include <ctime>

struct Elem{
	char prev;
	std::vector<std::pair<char, int>> paths;
	int lenght_to_start = std::numeric_limits<int>::max();
};

int heuristic(char q1, char first_end){
	return std::abs(q1 - first_end);
}

void write(char end, char start, std::map<char, Elem>& elem){
	for (auto& i : elem)
	{
		if (i.first == end && i.second.lenght_to_start ==
std::numeric_limits<int>::max())
		{
			std::cout << "no path\n";
			return;
		}
	}

	std::string answer(1, end);

	while (true)
	{
		if (answer.back() == start)
			break;
		answer += elem[answer.back()].prev;
	}

	std::reverse(answer.begin(), answer.end());
	std::cout << answer;
}

int main(){
	std::map<char, Elem> elem;
	char start = 0;
	char first_end = 0;
	char second_end = 0;

	std::cout << "Input start, first end, second end: ";
	std::cin >> start >> first_end >> second_end;
```

```
	char start_vertex = 0;
	char end_vertex = 0;
	float lenght = 0;
	while (std::cin >> start_vertex >> end_vertex >> lenght)
	{
		if (lenght == -1)
			break;

		elem[start_vertex].paths.push_back(std::make_pair(end_vertex, lenght));
		if (start_vertex == start)
			elem[start_vertex].lenght_to_start = 0;
	}

	std::vector<char> q;

	for (auto& i : elem)
		q.push_back(i.first);


	auto time_start = clock();
	while (!q.empty())
	{
		char current;
		size_t erase_index;
		int min_priority = -1;
		for (size_t i = 0; i < q.size(); ++i)
		{
			if (elem[q[i]].lenght_to_start ==
std::numeric_limits<int>::max())
				continue;

			size_t current_priority = elem[q[i]].lenght_to_start +
heuristic(q[i], first_end);
			if (current_priority < min_priority || min_priority == -1)
			{
				min_priority = elem[q[i]].lenght_to_start + heuristic(q[i],
first_end);
				erase_index = i;
				current = q[i];
			}
		}
		if (min_priority == -1)
			break;

		q.erase(q.begin() + erase_index);

		for (auto& next : elem[current].paths)
		{
			int old_value = elem[next.first].lenght_to_start;
			int new_value = elem[current].lenght_to_start + next.second;
			if (old_value > new_value)
			{
				elem[next.first].lenght_to_start = new_value;
				elem[next.first].prev = current;
			}
```

```
            }
        }
        auto time_end = clock();

        std::cout << "\nTime: ";
        std::cout << (double)(time_end - time_start) / CLOCKS_PER_SEC << "\n";
        std::cout << "\nFor first_end: ";
        write(first_end, start, elem);
        std::cout << "\nFor second_end: ";
        write(second_end, start, elem);
        std::cout << "\n\nСложность алгоритма: O(|V|*|V| + |E|) V - мн-во вершин, E -
мн-во ребер\n";

        return 0;
}
```