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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Построение и анализ алгоритмов»
Тема: Жадный алгоритм и A*

Студент гр. 7383 _____ Корякин М.П.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Изучить и реализовать на языке программирования C++ жадный алгоритм поиска пути в графе и алгоритм A* поиска кратчайшего пути в графе между двумя заданными вершинами.

Формулировка задания.

Разработайте программу, которая решает задачу построения пути в ориентированном графе при помощи жадного алгоритма. Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

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.

Дополнительное задание: Вариант 4c. Списки смежности. Модификация A* с двумя финишами (требуется найти путь до любого из двух).

Описание работы алгоритма.

A*:

Поиск начинается из исходной вершины. В текущие возможные пути добавляются все рёбра из начальной вершины. Происходит выбор минимального пути, где учитывается эвристическая близость вершины к искомой (в нашем случае это близость в таблице ASCII), если в выбранном пути последняя вершина уже была просмотрена, то этот путь удаляется из списка, и снова происходит выбор минимального пути. Выбираются из всех рёбер графа те, которые начинаются из последней вершины в этом пути. Эта вершина добавляется к этому пути, и новый путь заносится в список возможных путей, с увеличением стоимости, равной переходу по этому ребру.

Когда были выбраны все рёбра, которые начинаются из последней вершины в этом пути, то эта вершина добавляется в список просмотренных, а сам путь удаляется из списка возможных путей. Дальше снова происходит выбор минимального пути. Алгоритм заканчивает свою работу, когда достигается искомая вершина.

Результат работы программы.

Входные данные	Результат
a e b a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ab
a d c a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	abc
a e d a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ad
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ade (no modification A*)

Выводы.

В ходе выполнения данной лабораторной работы были изучены и реализованы два алгоритма. Первый – жадный алгоритм поиска пути в ориентированном графе. Этот алгоритм выбирает наименьший путь на каждом шаге – в этом заключается жадность, алгоритм достаточно прост, но за это платит своей надёжностью, так как он не гарантирует, что найденный путь будет минимальным возможным. Второй – алгоритм поиска минимального пути в ориентированном графе A*, который является модификацией алгоритма Дейкстры. Модификация состоит в том, что A* находит минимальные пути не до каждой вершины в графе, а для заданной. В ходе его работы при выборе пути учитывается не только вес ребра, но и эвристическая близость вершины к

искомой. A* гарантирует, что найденный путь будет минимальным возможным.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

using namespace std;

class OneFin     //структура ребра графа
{
public:
    char top;
    char last;
    double dist;
};

class ways    //возможные пути
{
public:
    string path;
    double lgth;
};

class graph
{
private:
    vector <OneFin> spisok;
    vector <ways> result;
    vector <char> watch;
    char source;
    char finish1;
    char finish2;
public:
    graph(){
         cin >> source >> finish1 >> finish2;
         char temp;
         while(cin >> temp)
         {
             OneFin element;
             element.top = temp;
             cin >> element.last;
             cin >> element.dist;
             spisok.push_back(element);
         }
         string buf = "";
         buf += source;
         for(size_t i=0; i < spisok.size(); i++)
         {
             if(spisok.at(i).top == source)
             {
```

```
                buf += spisok.at(i).last;
                result.push_back({buf, spisok.at(i).dist});
                buf.resize(1);
            }
        }
        watch.push_back(source);
    }

    size_t MinSearch() //возвращает минимальный индекс еще не
просмотренных вершин
    {
        double min = 9999;
        size_t temp;
        char val;
        for(size_t i=0; i < result.size(); i++)
        {
            if (abs(finish1- result.at(i).path.back()) >=
abs(finish1- result.at(i).path.back()))
                val = finish2;
            else
                val = finish1;

            if(result.at(i).lgth + abs(val -
result.at(i).path.back()) < min){
                if(Check(result.at(i).path.back())){
                    result.erase(result.begin() + i);
                }
                else
                {
                    min = result.at(i).lgth + abs(val -
result.at(i).path.back());
                    temp = i;
                }
            }
        }
        return temp;
    }

    bool Check(char value){
        for(size_t i = 0; i < watch.size(); i++)
            if(watch.at(i) == value)
                return true;
        return false;
    }

    void SearchPath(){
        while(true){
            size_t min = MinSearch();
            if(result.at(min).path.back() == finish1 ||
result.at(min).path.back() == finish2){
                cout << result.at(min).path;
                return;
```

```
            }
            for(size_t i=0; i < spisok.size(); i++){
                if(spisok.at(i).top == result.at(min).path.back()){
                    string buf = result.at(min).path;
                    buf += spisok.at(i).last;
                    result.push_back({buf, spisok.at(i).dist +
result.at(min).lgth});
                }
            }
            watch.push_back(result.at(min).path.back());
            result.erase(result.begin() + min);
        }
    }
};

int main(){
    graph element;
    element.SearchPath();
    return 0;
}
```