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. 7304 _____ Ажель И.В.

Преподаватель _____ Филатов А.Ю.

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.

Ход выполнения работы.

Жадный алгоритм:

Для удобства в начале работы жадного алгоритма поиска пути в ориентированном графе, список рёбер сортируется по неубыванию их весов. Алгоритм начинает поиск из заданной вершины. Текущая просматриваемая вершина добавляется в список просмотренных. В отсортированном списке рёбер выбирается первое (сортировка гарантирует, что это будет минимальное), которое начинается в просматриваемой вершине, если эта вершина не просмотрена, то текущей вершиной становится та, в которой заканчивается это ребро, если она уже просмотрена, то выбирается следующее ребро. Если в какой-то момент из текущей вершины нет путей, то происходит откат на шаг

назад, и в предыдущей вершине выбирается другое ребро, если это возможно. Алгоритм заканчивает свою работу, когда текущей вершиной становится искомая, или когда были просмотрены все рёбра, которые начинаются из исходной вершины.

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	
	Жадный алгоритм	A*
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	abcde	ade

a d a b 1.0 b c 1.0 c a 1.0 a d 5.0	ad	ad
a l a b 1 a f 3 b c 5 b g 3 f g 4 c d 6 d m 1 g e 4 e h 1 e n 1 n m 2 g i 5 i j 6 i k 1 j l 5 m j 3	abgenmjl	abgenmjl

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.

Жадный алгоритм.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct edge
{
    char beg;
    char end;
    double heft;
};

bool compare(edge first, edge second)
{
    return first.heft < second.heft;
}

class Graph
{
private:
    vector <edge> graph;
    vector <char> result;
    vector <char> viewingpoint;
    char source;
    char estuary;

public:
    Graph()
    {
         cin >> source >> estuary;
         char temp;
         while(cin >> temp)
         {
             edge element;
             element.beg = temp;
             if(!(cin >> element.end))
                 break;
             if(!(cin >> element.heft))
                 break;
             graph.push_back(element);
         }
         sort(graph.begin(), graph.end(), compare);
    }

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

    void initSearch()
    {
        if(source != estuary)
            Search(source);
```

```
    }

    bool Search(char value)
    {
        if(value == estuary)
        {
            result.push_back(value);
            return true;
        }
        viewingpoint.push_back(value);
        for(size_t i(0); i < graph.size(); i++)
        {
            if(value == graph.at(i).beg)
            {
                if(isViewing(graph.at(i).end))
                    continue;
                result.push_back(graph.at(i).beg);
                bool flag = Search(graph.at(i).end);
                if(flag)
                   return true;
                result.pop_back();
            }
        }
        return false;
    }

    void Print()
    {
        for(size_t i(0); i < result.size(); i++)
            cout << result.at(i);
    }
};

int main()
{
    Graph element;
    element.initSearch();
    element.Print();
    return 0;
}
```

A*.

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

using namespace std;

struc  edg      //ребро графа
t      e
{
    char beg;
    char end;
    double heft;
};

struct step     //возможные пути
{
    string path;
    double length;
};
```

```
class Graph
{
private:
    vector <edge> graph;
    vector <step> result;
    vector <char> viewingpoint;
    char source;
    char estuary;

public:
    Graph()
    {
         cin >> source >> estuary;
         char temp;
         while(cin >> temp)
         {
             edge element;
             element.beg = temp;
             cin >> element.end;
             cin >> element.heft;
             graph.push_back(element);
         }
         string buf = "";
         buf += source;
         for(size_t i(0); i < graph.size(); i++)
         {
             if(graph.at(i).beg == source)
             {
                 buf += graph.at(i).end;
                 result.push_back({buf,
                 graph.at(i).heft}); buf.resize(1);
             }
         }
         viewingpoint.push_back(source);
    }

    size_t MinSearch() //возвращает индекс минимального элемента
из непросмотренных
    {
        double min = 9999;
        size_t temp;
        for(size_t i(0); i < result.size(); i++)
        {
            if(result.at(i).length + abs(estuary - result.at(i).path.back()) <
min)
            {
                if(isViewing(result.at(i).path.back()))
                {
                    result.erase(result.begin() + i);
                }
                else
                {
                    min = result.at(i).length + abs(estuary
- result.at(i).path.back());
                    temp = i;
                }
            }
        }
        return temp;
    }

    bool isViewing(char value)
```

```
    {
        for(size_t i = 0; i < viewingpoint.size(); i++)
            if(viewingpoint.at(i) == value)
                return true;
        return false;
    }

    void Search()
    {
        while(true)
        {
            size_t min = MinSearch();
            if(result.at(min).path.back() ==
            estuary) {
                cout << result.at(min).path;
                return;
            }
            for(size_t i(0); i < graph.size(); i++)
            {
                if(graph.at(i).beg == result.at(min).path.back())
                {
                    string buf = result.at(min).path;
                    buf += graph.at(i).end;
                    result.push_back({buf, graph.at(i).heft +
result.at(min).length});
                }
            }
            viewingpoint.push_back(result.at(min).path.back())
            ; result.erase(result.begin() + min);
        }
    }
};

int main()
{
    Graph element;
    element.Search();
    return 0;
}
```