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: Алгоритмы на графах

Студент гр. 7383 _____ Левкович Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Содержание

Цель работы

Ознакомиться с алгоритмом поиска кратчайшего пути A*, написать программу, реализующую этот алгоритм.

Реализация задачи

Для решения задачи был написан класс Graph, который содержит поля для хранения самого графа в виде матрицы смежности, размера матрицы, начальную и конечную вершины, вес ребра и вектор с результатом. Метод New_Init считывает данные из консоли и добавляет все ребра в вектор, далее все вершины добавляются в вспомогательный вектор vector<type> для того, чтобы узнать количество различных вершин. Это необходимо для того, чтобы узнать, какой размер должен быть у матрицы смежности. Далее с помощью метода find_index находим позиции для матрицы смежности, в которых должны быть веса соответствующих ребер. Далее идет вызов метода A_Star.

Метод A_Star содержит вектор решений vector<A>, где A это структура, которая хранит вес ребра, текущий результат и приоритет. В качестве эвристической функции использовалась функция, которая вычисляла сумму разности конечной вершины и текущей с длиной пройденного пути. Далее начинаем поиск пути со стартового символа, для этого ищем соответствующую строку и ищем в ней все ребра, которые инцидентны вершине, все эти ребра добавляем в вектор решений vector<A> для последующего выбора лучшего решения. Как только рассмотрели все инцидентные ребра для вершины, сохраняем результат на данном шаге в type* finish. Переходим к следующей вершине и повторяем алгоритм уже для новой вершины. Если мы уже пришли в конечную вершину, сохраняем данное решение.

Исходный код программы представлен в Приложении А.

Тестирование

Программа собрана в операционной системе Windows с использованием компилятора g++. В других ОС и компиляторах тестирование

не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Сложность алгоритма

Было принято оценить сложность алгоритма. На рис. 1 представлен график зависимости количества итераций для алгоритма A* от количества ребер.

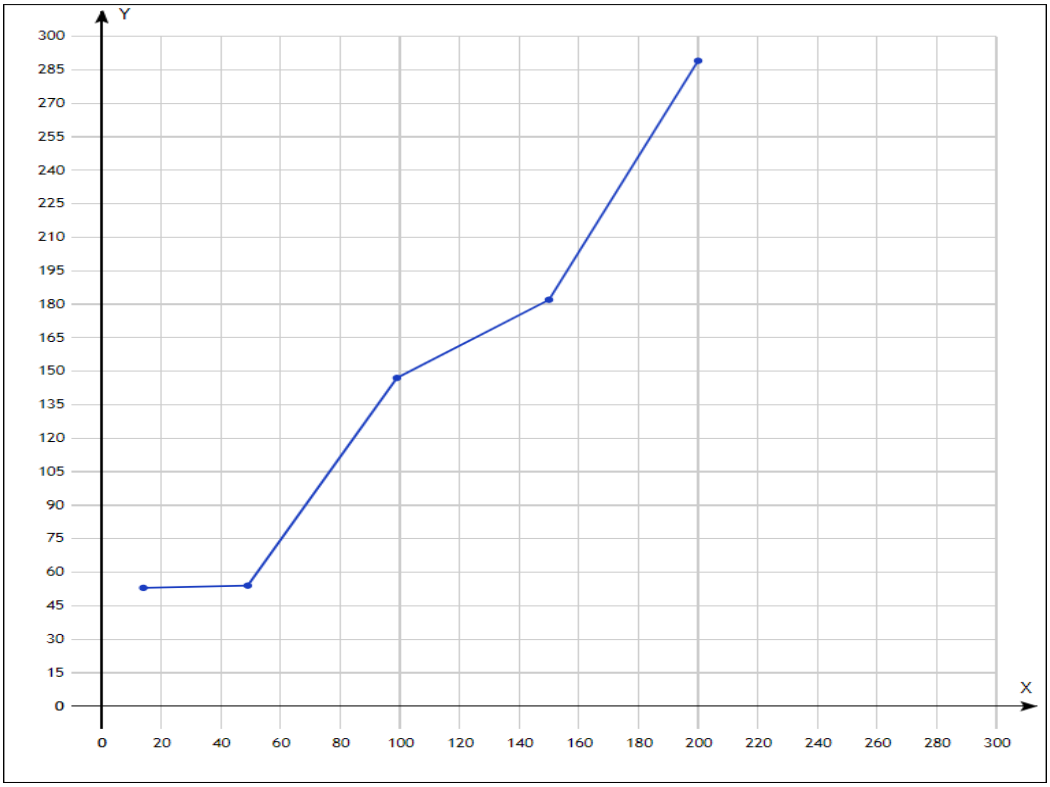

Рисунок 1 - График зависимости количества итераций от количества ребер в графе

Вывод

В ходе выполнения лабораторной работы был изучен алгоритм A*, была написана программа, реализующая этот алгоритм.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <ctype.h>
#include <cstring>
#include <algorithm>
using namespace std;
struct A{
    vector<char> resulte;
    double weight = 0;
    int prior = 0;
};
struct B{
    char a;
    char b;
    double weight;
};
template <typename T, typename type>
class Graph{
    vector<vector<T>> m;
    size_t size;
    type start;
    type finish;
    double weight;
    vector<type> result;

public:
    Graph(){
        size = 26;
    }

    bool find(vector<type> t, type c){
```

```
    if(t.size()==0){
        return false;
    }
    for(int i = 0;i<t.size();i++){
        if(t[i]==c){
            return true;
        }
    }
    return false;
}

int find_index(vector<type> t, type c){
    int index = 0;
    for(int i = 0;i<t.size();i++){
        if(c==t[i])
            return index;
        index++;
    }
}

void new_Init(){
    vector<B> edge;
    vector<type> a;
    type vertex1, vertex2;
    cin>>start>>finish;
    bool t = true;
    while(t){
        if(cin>>vertex1 && isalpha(vertex1)){
            cin>>vertex2>>weight;
            edge.push_back({vertex1, vertex2, weight});
            if(!find(a, vertex1))
                a.push_back(vertex1);
            if(!find(a, vertex2))
                a.push_back(vertex2);
```

```
            }
            else {
                t = false;
            }
        }
        sort(a.begin(), a.end());
        size = a.size();
        m.resize(size);
        for(int i = 0;i<size;i++){
            m[i].resize(size);
        }
        for(int i = 0;i<edge.size();i++){
            m[find_index(a, edge[i].a)][find_index(a, edge[i].b)] =
edge[i].weight;
        }
        A_Star(a);
    }

    void GreedIsGood(vector<type> a){
        result.resize(size);
        int min = 1000;
        int res = 0;
        result[0] = start;
        while (result[res] != this->finish)
        {
            min = 1000;
            res++;
            for (int i = 0; i < size; i++)
                if ((min > m[find_index(a, result[res-1])][i]) &&
(m[find_index(a, result[res-1])][i] != 0)){
                    min = m[find_index(a, result[res-1])][i];
                    result[res] = a[i];
                }
```

```
            if(min == 1000){
                m[find_index(a, result[res-1])][find_index(a, result[res-
1])] = 0;
                res-=2;
            }
            else {
                 m[find_index(a, result[res-1])][find_index(a,
result[res])] = 0;
            }
        }
        for (int i = 0;i<result.size();i++) {
            if(isalpha(result[i]))
                    cout<<result[i];
        }
    }

    int veclen(vector<type> a){
        int count = 0;
        for (int i = 0;i<a.size();i++) {
             count++;
            if(a[i]==0)
                return i;
        }
    }

    void A_Star(vector<type> a){
        std::vector<A> solution;//Вектор решений
        A *curr_sol;
        vector<type> str(size, 0);
        type *finish = nullptr;
        bool vertex_duplication;
        T weight_cur, fin_w;
        int sol = 0, current = 0;//Вспомогательные переменные
        current = find_index(a, this->start);
```

```
        weight_cur = 0;
        while(1)
        {
            for (int i = 0; i < size; i++)
                if (m[find_index(a, (char)(current+97))][i] != 0)
                {
                    vertex_duplication = false;// флаг дублирования
вершины
                    for (int j = 0; j < veclen(str); j++)
                        if(a[i] == str[j]) {vertex_duplication = true;
break;}//поиск дублирующей вершины
                    if ((vertex_duplication == false) && (a[i] != this-
>start))
                    {
                        curr_sol = new A;//запись элемента в вектор
                        solution.push_back(*curr_sol);
                        solution.at(sol).resulte.resize(size);
                        for(int k =
0;k<solution.at(sol).resulte.size();k++)
                            solution.at(sol).resulte[k] = str[k];
                        solution.at(sol).resulte[veclen(str)]=a[i];
                        solution.at(sol).weight=m[find_index(a,
(char)(current+97))][i] + weight_cur;
                        solution.at(sol).prior=this->finish - (int)a[i];
                        delete curr_sol;
                        sol++;
                    }
                }
            sol--;
            std::sort(solution.begin(), solution.end(), [](const A& a,
const A& b) {//сортировка по убыванию
                      return (a.weight+a.prior) > (b.weight+b.prior);});
```

```
            if(finish != nullptr)//если есть уже какой-то результат, то
отсекаем решения, дающие вес больше имеющегося
            {
                while (sol >=0)
                    {
                        if(solution.at(sol).weight < fin_w) break;
                        else {solution.erase(solution.begin() + sol);
sol--;}
                    }
                if (sol == -1) break; //выход из главного цикла
            }
            if(solution.at(sol).resulte[veclen(solution.at(sol).resulte)
- 1] == this->finish)
            {
                if(finish == nullptr) //сохранение получившегося решения
                {
                    finish = new char(solution.at(sol).resulte.size());
                    for(int i = 0;i<solution.at(sol).resulte.size();i++){
                        finish[i] = solution.at(sol).resulte[i];
                    }
                    fin_w=solution.at(sol).weight;
                    solution.erase(solution.begin() + sol);
                }
                else {
                    if(solution.at(sol).weight < fin_w)
                        {
                            delete[] finish; finish = new
char(solution.at(sol).resulte.size());
                        for(int i =
0;i<solution.at(sol).resulte.size();i++){
                            finish[i] = solution.at(sol).resulte[i];
                        }
                            fin_w=solution.at(sol).weight;
                        }
```

```
                        solution.erase(solution.begin() + sol);
                    }
                sol--;
                while (sol >=0)//удаление решений, дающих больший вес
                {
                    if(solution.at(sol).weight < fin_w) break;
                    else {solution.erase(solution.begin() + sol); sol--;}
                }

                if (sol == -1) break;//выход из главного цикла
            }
            current =
(int)(solution.at(sol).resulte[veclen(solution.at(sol).resulte)-1]) - 97;
            str = solution.at(sol).resulte;
            weight_cur = solution.at(sol).weight;
        }
        std::cout<<this->start<<finish<<std::endl;
        delete [] finish;
        finish = nullptr;
    }
};

int main()
{
    Graph<double, char> s;
    s.new_Init();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.
ТЕСТОВЫЕ СЛУЧАИ

В качестве эвристической функции использовалась функция, которая высчитывает модуль разности конечной вершины и текущей.

Входные данные	Выходные данные
-1 -6 -1 -5 6 -5 -6 3 -1 -4 8 -4 -6 1	-1 -5 -6
1 -1 1 2 3 2 3 1 3 -7 1 1 -7 6 1 4 3 4 -1 4 -7 -1 1	1 2 3 -7 -1
-5 -8 -5 10 5 10 -20 8 -20 -8 2 -5 5 1 5 4 3 4 8 6	-5 10 -20 -8