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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритмы поиска пути в графах

Студент гр.8382 _____ Ефимова М.А.

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2020

Цель работы.

Научиться реализовывать алгоритмы поиска пути в графе.

Задание.

Разработайте программу, которая решает задачу построения кратчайшего пути в *ориентированно* 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. Пример входных данных

a e

a b 3.0

b c 1.0

c d 1.0

a d 5.0

d e 1.0

В первой строке через пробел указываются начальная и конечная вершины. Далее в каждой строке указываются рёбра графа и их вес.

В качестве выходных данных необходимо предоставить строку, в которой перечислены все вершины до конечной. Для приведённых в примере входных данных ответом будет

ade

Вар. 4.Модификация A* с двумя финишами (требуется найти путь до любого из двух

.

Описание алгоритмов.

1. Жадный алгоритм

Ввод данных происходит в функции *input.* Считав данные, вызывается функция *Sort_Weigth (nodes),* сортирующая рёбра, исходящие из вершины, по убыванию. После этого вызывается функция *Find_Way* в которой переход от вершины

происходит по минимальному ребру, исходящему из неё.

Алгоритм работает, пока не будет найдена конечная вершина.

Сложность.

Сложность алгоритма по числу операций — O(|E + V|), по памяти— O(|V|^2).В худшем случае проходим по каждому ребру. Проходим вершины тоже по одному разу.

2. Алгоритм A*

Ввод данных происходит в функции *input.* Считав данные, запускается функция *A**, определяющая минимальный путь между двумя вершинами. Первоначальная вершина кладётся в очередь с приоритетом, после чего начинается цикл *while,* работающий до тех пор, пока вершина кучи меньше расстояния от конечной вершины до начальной или пока очередь с приоритетом не станет пустой. На каждой итерации цикла из очереди достаётся вершина с минимальным значением. Функция *change_dist* просматривает все соседние вершины и если расстояние от начальной вершины до неё меньше расстояния от первоначальной вершины до этого соседа, то его расстояние изменяется, а сама вершина кладётся в очередь с приоритетом. Посчитав кратчайшее расстояние до конечной вершины, цикл завершается.

Сложность.

По времени:

Сложность по времени зависит от разбора отдельных случаев (эвристики).
В худшем случае число вершин растет по экспоненте (исследуемых алгоритмом) по сравнению с длиной оптимального пути.
Сложность становится полиномиальной, когда эвристика удовлетворяет условию:
|h(x) – h*(x)| <= O(log h*(x)), h* - точная оценка расстояния от вершины a1 в вершину a2 (конечная).
T.e delta не должна расти быстрее логарифма от оптимальной эвристики.
В худшем случае сложность по памяти экспоненциальная, т.к храним экспоненциальное количество узлов. В лучшем случае O(V^2).

В алгоритмах для хранения графа используется список смежности, реализованный с помощью стркутуры данных *vector<vector<pair<double,int>>> nodes*, в котором первый вектор обозначает список всех вершин графа, второй вектор список всех вершин, смежных для каждого элемента первого вектора.

Описание функций и структур данных.

void recovery _way(vector<int> &prev,int vertex,map<int,char> &convert_to_char) - функция для вывода списка вершин по кратчайшему пути. *vector<int> &prev* — вектор, хранящий в *prev[i]* номер предыдущей вершины для вершины с номером *i*, через которую проходит кратчайший путь, номер конечной вершины, *map<int,char> &convert_to_char)* — словарь соответствия номера вершины и её буквенного представления. Используется в алгоритме A*

v*oid change_dist(vector<vector<pair<int,double>>> &nodes, map<int,char> &convert_to_char,, priority_queue<pair<double,int>> &near_way,vector<double> &dist, vector<int> &prev, int top1,int top2)*

— функция пересчёта расстояний до начальной вершины

v*ector<vector<pair<int, double>>> &nodes* — струтура данных для списка смежности, *map<int,char> &convert_to_char* - словарь, переводящий номер вершины в её буквенное обозначение. *priority_queue<pair<double,int>> &near_way* — очередь с приоритетом, v*ector<double> &dist* — вектор расстояний до начальной вершины, *vector<int> &prev* — вектор, хранящий в *prev[i]* номер предыдущей вершины для вершины с номером *i*, через которую проходит кратчайший путь, *int top1*— номер текущей вершины, *int top2* — номер начальной вершины. Используется в алгоритме A*

void input(vector<vector<pair<int,double>>> &nodes,map<int,char> &convert_to_char,map<char,int> &convert_to_int) — функция для ввода исходных данных, *vector<vector<pair<int,double>>> &nodes* — структура данных для хранения списка смежности, *map<int,char> &convert_to_char* — словарь, переводящий номер вершины в её буквенное обозначение,

map<char,int> &convert_to_int — словарь для перевода буквенного обозначения вершины в её номер. Используется в первом и втором алгоритмах.

void from_list(vector<vector<pair<int,double>>> &nodes,map<char,int> &convert_to_int,vector<inputElement> &input_sequence)

—функция для формирования списка.

find_way(vector<vector<pair<int,double>>> &nodes,vector<int> &road,int top1,int top2,bool &check) — функция нахождения пути в графе для жадного алгоритма.

void sort_weigths(vector<vector<pair<int,double>>> &nodes) — функция сортировки ребёр в порядке увеличения веса для каждой вершины. Используется в жадном алгоритме.

vector<vector<pair<int,double >>> nodes — структура данных для хранения списка смежности.

map<char,int> &convert_to_int — словарь для перевода буквенного обозначения вершины в её номер.

map<int,char> &convert_to_char — словарь, переводящий номер вершины в её буквенное обозначение.

vector<int> &prev — вектор, хранящий в *prev[i]* номер предыдущей вершины для вершины с номером *i,* через которую проходит кратчайший путь в алгоритме *A**.

v*ector<double> &dist* — вектор расстояний от вершины с индексом i до начальной вершины в алгоритме *A**.

vector<int> &road — вектор вершин, через которые проходит минимальный путь в жадном алгоритме.

vector<input_El> &input_seq — вектор для хранения последовательности входных рёбер.

struct inputEl{

char top1;

char top2;

double weigth; }; - структура, описывающая входное ребро. Поле *char top1* - обозначение вершины, из которой выходит ребро, *char top2* — обозначение вершины, в которую входит ребро, *double weigth;* — расстояние между рёбрами.

Жадный алгоритм.

Исходный код алгоритма представлен в приложении А. struct

Input_El – структура, где инициализируются вводимые

элементы bool Sort – функция сортировки

void Sort_Weigth – функция сортировки весов

void Print_Output – функция для печати вывода

void Find_Way – функция нахождения пути

void Input – основная функция. Происходит считывание

вершин, конвертация в целочисленные значения и далее их

сравнение

Алгоритм A*.

Исходный код алгоритма представлен в приложении Б.
struct Input_El – структура, где инициализируются вводимые

элементы bool Sort – функция сортировки

int counting_sum_of_weigth – считает сумму всех

весов void recovery_way – функция восстановления

пути void change_dist – функция изменения расстояния

до соответствующей вершины

void Print_Output – функция для печати вывода

void AStar– основная функция. Происходит считывание вершин, также

возвращает найденный путь в виде списка вершин.

Тестирование.

Программа была протестирована на следующих исходных данных:

1) Жадный алгоритм

Входные данные	Результат
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	abcde
b d b a 2 b d 15 a g 4 g d 8	bagd
a c a b 5 b c 4 a c 8	abc

2) A* алгоритм

Входные данные	Результат
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ade
b d b a 2 b d 15	bagd

a g 4
g d 8

a c ac
a b 5
b c 4
a c 8

Выводы.

Были получены навыки работы с алгоритмами поиска путей в графе. В частности, были реализованы жадный алгоритм, находящий путь из одной вершины в другую, а также были реализованы алгоритмы A* и Дейкстры для нахождения кратчайших путей.

Приложения А. Исходный код жадного алгоритма

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

using namespace std;

//структура, описывающая входное ребро

struct Input_El { // структура вводимого элемента
	char top1;//из которой выходит ребро
	char top2;//в которую входит ребро
	double weigth;//расстояние между рёбрами
};
/*Это ссылка на функцию вида bool foo(const T& a, const T& b), где T — тип элементов
сортируемого вектора
Задача этой функции — проверить, упорядочены ли a и b по возрастанию.
Иными словами, не превышает ли значение a таковое у b (говоря математическим языком —
соблюдается ли условие a < b).*/

bool Sort(const pair<int, double >& a, const pair<int, double >& b) {// функция сортировки
	return (a.second < b.second);
}

/*функция сортировки ребёр в порядке увеличения веса для каждой вершины.
 используется  список смежности, реализованный с помощью стркутуры данных
vector<vector<pair<double,int>>> nodes,
 в котором
 первый вектор - список всех вершин графа,
 второй вектор - список всех вершин, смежных для каждого элемента первого вектора
*/
void Sort_Weigth(vector<vector<pair<int, double>>>& nodes) {//функция сортировки ребёр в
порядке увеличения веса для каждой вершины.
	int len = nodes.size();//размер весов
	for (int i = 0; i < len; i++) {
		sort(nodes[i].begin(), nodes[i].end(), Sort);
	}
}
/* map<int,char> &convert_to_char — словарь, переводящий номер вершины в её буквенное
обозначение,
   vector<int> &road — вектор вершин, через которые проходит минимальный путь
  */
void Print_Output(map<int, char>& convert_to_char, vector<int>& road) {//перевод в char и
вывод
	for (int i = 0; i < road.size(); i++) {
		cout << convert_to_char[road[i]];
	}
}
/* функция для формирования списка
   vector<vector<pair<int,double >>> nodes — структура данных для хранения списка
смежности.
   map<char,int> &convert_to_int — словарь для перевода буквенного обозначения вершины в её
номер
   vector<inputElement> &input_seq — вектор для хранения последовательности входных рёбер.
   input_seq 'состоит' из struct Input_El
*/
void From_List(vector<vector<pair<int, double>>>& nodes, map<char, int>& convert_to_int,
vector<Input_El>& input_seq) {
	int top1;//из которой выходит ребро
	int top2;//в которую входит ребро
	double weigth;//расстояние между рёбрами.
	for (int i = 0; i < input_seq.size(); i++) {
		top1 = convert_to_int[input_seq[i].top1];//берем i-тый элемент из вектора
input_seq и затем берем у него поле top1
		top2 = convert_to_int[input_seq[i].top2];
		weigth = input_seq[i].weigth;
```

```
			nodes[top1].push_back(make_pair(top2, weigth));//make_pair создает пару из
top2 и weight и  вставляет в массив nodes под индексом top1
		}
}
/* функция нахождения пути в графе для жадного алгоритма
   vector<vector<pair<int,double >>> nodes – структура данных для хранения списка
смежности.
   vector<int> &road – вектор вершин, через которые проходит минимальный путь в жадном
алгоритме.
   переход от вершины происходит по минимальному ребру, исходящему из неё.
*/
void Find_Way(vector<vector<pair<int, double>>>& nodes, vector<int>& road, int top1, int
top2, bool& check) {// функция нахождения пути в графе для жадного алгоритма
		road.push_back(top1);//Для добавления элементов в вектор применяется функция
push_back(), в который передается добавляемый элемент
		if (top1 == top2) {
			check = true;//проверка
			return;
		}
		for (int i = 0; i < nodes[top1].size(); i++) {
			Find_Way(nodes, road, nodes[top1][i].first, top2, check);//вызов функции с
проверкой
			if (check)
				return;
			road.pop_back();//pop_back() все делает наоборот – удаляет одну ячейку в конце
вектора.
		}
}
/* vector<vector<pair<int,double>>> &nodes – структура данных для хранения списка
смежности,
 map<int,char> &convert_to_char – словарь, переводящий номер вершины в её буквенное
обозначение,
 map<char,int> &convert_to_int – словарь для перевода буквенного обозначения вершины в её
номер.
*/
void Input(vector<vector<pair<int, double>>>& nodes, map<int, char>& convert_to_char,
map<char, int>& convert_to_int) {//функция для ввода исходных данных
		vector<Input_El> input_seq;//вводимая последовательность
		Input_El elem;
		char top1;
		char top2;
		top1 = ' ';
		double weigth;
		int k = 0;  //счетчик
		while (cin >> top1) {
			if (!top1)
				break;
			cin >> top2;
			cin >> weigth;
			elem.top1 = top1;
			elem.top2 = top2;
			elem.weigth = weigth;
			input_seq.push_back(elem);
			if (convert_to_int.find(top1) == convert_to_int.end()) {
				convert_to_int[top1] = k;
				convert_to_char[k] = top1;
				k++;
			}
			if (convert_to_int.find(top2) == convert_to_int.end()) {
				convert_to_int[top2] = k;
				convert_to_char[k] = top2;
				k++;
			}
		}
		nodes.resize(k);//resize () позволяет изменить количество символов, добавляет /
удаляет элементы в зависимости от заданного им размера.
		From_List(nodes, convert_to_int, input_seq);
}

int main() {
		char c1;
		char c2;
```

```
        cin >> c1;
        cin >> c2;
        int top1;
        int top2;
        bool check = false;
        vector<vector<pair<int, double >>> nodes;
        vector<int> road;
        map<char, int> convert_to_int;
        map<int, char> convert_to_char;
        Input(nodes, convert_to_char, convert_to_int);
        Sort_Weigth(nodes);
        top1 = convert_to_int[c1];
        top2 = convert_to_int[c2];
        Find_Way(nodes, road, top1, top2, check);
        Print_Output(convert_to_char, road);
        return 0;
}
```

Приложения В. Исходный код алгоритма A*

```
#include <iostream>
#include <vector>
#include <stack>
#include <queue>
#include <map>
#include <math.h>
using namespace std;

struct InputEl {
    char top1; //первая вершина
    char top2; //вторая вершина
    double weigth;
};

/*
vector<vector<pair<double,int>>> nodes,
 первый вектор - список всех вершин графа,
 второй вектор - список всех вершин, смежных для каждого элемента первого вектора.

*/
void fromNeighbor_list(vector<vector<pair<int, double>>>& nodes, map<char, int>&
convert_to_int, vector<InputEl>& input_consistency) {
    //pair<int,double> через конструктор инициализируем пару целого типа и двойной точности
    double weigth;
    int top1, top2;
    for (int i = 0; i < input_consistency.size(); i++) { //меньше размера
последовательности
        top1 = convert_to_int[input_consistency[i].top1];
        top2 = convert_to_int[input_consistency[i].top2];
        weigth = input_consistency[i].weigth;
        nodes[top1].push_back(make_pair(top2, weigth)); //добавление в конец элемент и
обеспечивает присвоение знач.полям top2, weigth
    }
}

int counting_sum_of_weigth(vector<vector<pair<int, double>>>& nodes) {
    int sum = 0;
    for (int i = 0; i < nodes.size(); i++) {
        for (int j = 0; j < nodes[i].size(); j++) {
            sum += nodes[i][j].second;//second - второй элемент пары sec структуры pair
        }
    }
    return sum + 1;
}
/*
функция для вывода списка вершин по кратчайшему пути.
vector<int> &prev – вектор, хранящий в prev[i] номер предыдущей вершины,
для вершины с номером i, через которую проходит кратчайший путь.
```

```
map<int,char> &convert_to_char) — словарь соответствия номера вершины и её буквенного
представления
int top — номер конечной вершины
*/
void recovery_way(vector<int>& prev, map<int, char>& convert_to_char, int top) {
    stack<int> steck;//создаем стек
    while (prev[top] != -1) {
        steck.push(top);//добавляем top
        top = prev[top];
    }
    steck.push(top);//добавляем top
    int way = steck.size();//размер стека
    for (int i = 0; i < way; i++) {
        cout << convert_to_char[steck.top()];
        steck.pop();//удаляем верхний элемент стека
    }
}
/*
Ввод данных происходит в функции input.Считав данные, запускается функция A*.
vector<vector<pair<int,double>>> &nodes — структура данных для хранения списка смежности
map<int,char> &convert_to_char — словарь, переводящий номер вершины в её буквенное
обозначение
map<char,int> &convert_to_int — словарь для перевода буквенного обозначения вершины в её
номер
*/

void input(vector<vector<pair<int, double>>>& nodes, map<int, char>& convert_to_char,
map<char, int>& convert_to_int) {
    char top1;
    char top2;
    int i = 0;
    double weigth;
    top1 = ' ';
    int k = 0; //счетчик
    cout << "Input n: ";
    int n;
    cin >> n;
    vector<InputEl> input_consistency;//объявление массива структуры InputEl
    InputEl elem;//элемент структуры

    while (i < n) {
        cin >> top1;
        if (!top1) {
            break;
        }
        cin >> top2;
        cin >> weigth;
        elem.top1 = top1;
        elem.top2 = top2;
        elem.weigth = weigth;
        input_consistency.push_back(elem);//добавление в конец элемента
        if (convert_to_int.find(top1) == convert_to_int.end()) {//если найденный элемент в
контейнере с первой вершиной = указателю на конец контейнера
            convert_to_int[top1] = k; //a -> 1
            convert_to_char[k] = top1;// 1-> a
            k++;//2
        }
        if (convert_to_int.find(top2) == convert_to_int.end()) {//если найденный элемент в
контейнере со второй вершиной = указателю на конец контейнера
            convert_to_int[top2] = k;
            convert_to_char[k] = top2;
            k++;
        }
        i++;
    }
    nodes.resize(k); //устанавливаем размер счетчика для вектора
    fromNeighbor_list(nodes, convert_to_int, input_consistency);
}

/*
```

просматривает все соседние вершины и если расстояние от начальной вершины до неё меньше расстояния от первоначальной вершины до этого соседа,
то его расстояние изменяется, а сама вершина кладётся в очередь с приоритетом.
Посчитав кратчайшее расстояние до конечной вершины, цикл завершается.

vector<vector<pair<int, double>>> &nodes – струтура данных для списка смежности
map<int,char> &convert_to_char - словарь, переводящий номер вершины в её буквенное обозначение
priority_queue<pair<double,int>> &near_way – очередь с приоритетом
<double> &dist – вектор расстояний до начальной вершины
vector<int> &prev – вектор, хранящий в prev[i] номер предыдущей вершины для вершины с номером i, через которую проходит кратчайший путь
int top1– номер текущей вершины,
int top2 – номер начальной вершины
*/
```
void change_dist(vector<vector<pair<int, double>>>& nodes, priority_queue<pair<double,
int>>& near_ways, vector<double>& dist, vector<int>& prev, map<int, char>& convert_to_char,
int top, int top2) {
    for (int i = 0; i < nodes[top].size(); i++) {
        if (dist[nodes[top][i].first] > dist[top] + nodes[top][i].second) {
            dist[nodes[top][i].first] = dist[top] + nodes[top][i].second;
            prev[nodes[top][i].first] = top; //предыдущему даем знаение top
            near_ways.push(make_pair(-(dist[nodes[top][i].first] +
(int)convert_to_char[top2] - (int)convert_to_char[nodes[top][i].first]),
nodes[top][i].first));
        }
    }
}
/*
функция A*, определяющая минимальный путь между двумя вершинами.
Первоначальная вершина кладётся в очередь с приоритетом, после чего начинается цикл while, работающий до тех пор, пока вершина кучи меньше расстояния от конечной вершины до начальной или пока очередь с приоритетом не станет пустой.
На каждой итерации цикла из очереди достаётся вершина с минимальным значением
*/
void AStar(vector<vector<pair<int, double >>>& nodes, map<int, char>& convert_to_char, int
top1, int top2) {
    int current_top;
    int min;
    vector<double > dist(nodes.size());
    vector<int> prev(nodes.size());
    vector<bool> visited(nodes.size());
    priority_queue<pair<double, int>> near_ways;
    double max_dist = counting_sum_of_weigth(nodes);
    for (int i = 0; i < nodes.size(); i++)
        dist[i] = max_dist;
    dist[top1] = 0;
    prev[top1] = -1;
    near_ways.push(make_pair(-(dist[top1] + top2 - top1), top1));
    while (!near_ways.empty()) {
        while (1) {
            if (near_ways.empty())
                break;
            pair<int, int> current_min = near_ways.top();
            near_ways.pop();
            current_top = current_min.second;
            min = -current_min.first;
            if (!visited[current_top])
                break;
        }
        if (dist[top2] < min)
            break;
        visited[current_top] = true;
        change_dist(nodes, near_ways, dist, prev, convert_to_char, current_top, top2);
    }
    if (dist[top2] == max_dist)
        cout << -1;
    else {
        recovery_way(prev, convert_to_char, top2);
    }
```

```
}

int main() {
    char c1;
    char c2;
    cin >> c1;
    cin >> c2;
    int top1;
    int top2;
    map<char, int> convert_to_int;
    map<int, char> convert_to_char;
    vector<vector<pair<int, double >>> nodes;
    input(nodes, convert_to_char, convert_to_int);
    top1 = convert_to_int[c1];
    top2 = convert_to_int[c2];
    AStar(nodes, convert_to_char, top1, top2);
    return 0;
}
```