**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 поиска путей в графах»**

Студентка гр. 7382 Лящевская А. П.

Преподаватель Фирсов М. А.

Санкт-Петербург

2019

**Цель работы.**

Создать и реализовать на практике алгоритм А\*. Реализована визуализация алгоритмов, действий. Был получен опыт в подробной, понятной, с пояснениями демонстрации хода алгоритма.

**Задание.**

**Жадный алгоритм.**

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.

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 ребра графа и их вес

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет

abcde

**Алгоритм А\*.**

Разработайте программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом А\*. 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 ребра графа и их вес

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет

*ade*

**Вариант 1м.**

Матрица смежности. В A\* вершины именуются целыми числами (в т.   
ч. отрицательными).

**Описание алгоритма A\*.**

1. Создается 2 списка вершин — ожидающие рассмотрения и уже рассмотренные. В ожидающие добавляется точка старта, список рассмотренных пока пуст.
2. Для каждой точки рассчитывается F = G + H. G — расстояние от старта до точки, H — примерное расстояние от точки до цели. О расчете этой величины будет сказано позднее. Так же каждая точка хранит ссылку на точку, из которой в нее пришли.
3. Из списка точек на рассмотрение выбирается точка с наименьшим F. Обозначим ее X.
4. Если X — цель, то мы нашли маршрут.
5. Переносим X из списка ожидающих рассмотрения в список уже рассмотренных.
6. Для каждой из точек, соседних для X (обозначим эту соседнюю точку Y), делаем следующее:
7. Если Y уже находится в рассмотренных — пропускаем ее.
8. Если Y еще нет в списке на ожидание — добавляем ее туда, запомнив ссылку на X и рассчитав Y.G (это X.G + расстояние от X до Y) и Y.H.
9. Если же Y в списке на рассмотрение — проверяем, если X.G + расстояние от X до Y < Y.G, значит мы пришли в точку Y более коротким путем, заменяем Y.G на X.G + расстояние от X до Y, а точку, из которой пришли в Y на X.
10. Если список точек на рассмотрение пуст, а до цели мы так и не дошли — значит маршрут не существует.

**Функция примерной оценки расстояния до цели.**

Эта функция должна выполнять несколько условий:

* Функция никогда не переоценивает расстояние до цели.
* Для это функции расстояния выполняется неравенство треугольника. Поясню подробнее: предположим у нас есть три точки — A, B и C. Для путей A-B B-C и A-C должно быть верно следующее неравенство: A-B + B-C >= A-C.

**Сложность алгоритма.**

Временна́я сложность алгоритма A\* зависит от эвристики. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию:

, где h\* — оптимальная эвристика, то есть точная оценка расстояния из вершины x к цели. Другими словами, ошибка h(x) не должна расти быстрее, чем логарифм от оптимальной эвристики.

Сложность по операциям и по памяти составляет , где *d* – длина кратчайшего пути. В случае хорошей эвристики сложность по памяти: О(V^2), сложность по операциям: О(ЕlogЕ).

**Описание жадного алгоритма.**

Алгоритм, который на каждом шаге ищет локально-оптимальное решение, предполагая, что конечное общее решение, являющееся суперпозицией локальных, также будет оптимальным.

Говорят, что к оптимизационной задаче применим принцип жадного выбора, если последовательность локально оптимальных выборов даёт глобально оптимальное решение. В типичном случае доказательство оптимальности следует такой **схеме**:

1. Доказывается, что жадный выбор на первом шаге не закрывает пути к оптимальному решению: для всякого решения есть другое, согласованное с жадным выбором и не хуже первого.
2. Показывается, что подзадача, возникающая после жадного выбора на первом шаге, аналогична исходной.
3. Рассуждение завершается по индукции.

Действие алгоритма начинается со стартовой вершины. Просматриваются все ребра, идущие от нее. Выбирается ребро с наименьшим весом. Далее алгоритм продолжается по такой же схеме, пока не будет достигнута конечная вершина, либо не останется не пройденных из текущей вершины ребер. В таком случае, алгоритм откатывается назад, к предыдущей вершине и так далее, пока не встретится не пройденное ребро, либо пока мы не просмотрим все доступные ребра, а пути не найдем.

Так, к пути до конечной вершины добавляется точка, когда происходит переход в нее и отсекается в случае возврата к предыдущей точке.

Сложность жадного алгоритма по операциям *.* Обуславливается это тем, что в худшем случае алгоритм просматривает все вершины и ребра.

Сложность по памяти составляет так как для хранения графа используется матрица смежности.

**Описание структур**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название структуры** | **Тип** | **Объект** | **Параметры** | **Описание** |
| struct Vertex {} | - | type value; | - | Именование вершины |
| - | double weight; | - | Вес пути до этой вершины |
| - | Type from; | - | Именование вершины, из которой мы пришли в эту вершину |
| сlass Directed\_Graph | Private | Type \*list | - | Хранит список |
| Private | Size\_t size\_l = 0 | - | Размер list |
| Private | Double \*\*matrix | - | Хранит матрицу смежности вершин |
| Private | void print() | - | Распечатывает граф |
| Private | void push(const type key, const type value, const float weight) |  | Добавляет элемент |
| Private | size\_t find\_or\_add(const type litter) |  | Находит элемент в противном случае добавляет его |
| Private | void resize\_list() | - | Изменяет размер list |
| Private | bool in\_arr(type c, vector <Vertex> &arr) |  | Сообщает есть ли элемент в arr |
| Private | size\_t index\_in\_arr(type c, vector <Vertex> &arr) |  | Находит индекс элемента в arr |
|  | Private | size\_t min\_arr(vector <Vertex> &arr, size\_t finish) |  | Находит индекс минимального элемента в arr |
|  | Private | void find\_way(size\_t start, size\_t finish, stack <type> &way) |  | Находит минимальный путь в графе в соответствии с алгоритмом А\* |

**Тестирование.**

**A\*.**

|  |  |  |
| --- | --- | --- |
| Номер  теста | Тест: | Результат: |
| 1 | a e  a b 3.0  b c 1.0  c d 1.0  a d 5.0  d e 1.0 | ade |
| 2 | a f  a b 3.0  b c 1.0  c d 1.0  a d 5.0  d e 1.0 |  |
| 3 | a e  a b 3.0  b c 1.0  c d 1.0  a d 6.0  d e 1.0 | abcde |
| 4 | -2 2  -2 -1 3  -1 0 1  0 1 1  -2 1 5  1 2 1 | -212 |

**Жадный алгоритм.**

|  |  |  |
| --- | --- | --- |
| Номер  теста | Тест: | Результат: |
| 1 | a e  a b 3.0  b c 1.0  c d 1.0  a d 5.0  d e 1.0 | abcde |
| 2 | a f  a b 3.0  b c 1.0  c d 1.0  a d 5.0  d e 1.0 |  |
| 3 | a e  a b 3.0  b c 1.0  c d 1.0  a d 6.0  d e 1.0 | abcde |

**Вывод.**

В процессе выполнения лабораторной работы был продуман, создан и реализован на практике алгоритм А\*. Также была реализована визуализация алгоритмов, действий. Был получен опыт в подробной, понятной, с пояснениями демонстрации хода алгоритма.

**Приложение 1. Код программы.**

* **2.1**[**.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <cmath>

#include <iostream>

#include <functional>

#include <algorithm>

#include <vector>

#include <stack>

#define FINISH -1

#define WAS 1

#define WASNT 0

#define type char

#define DEMO

#define SIZE 10

using namespace std;

class Directed\_Graph{

private:

type \*list;

bool \*was;

size\_t size\_l = 0;

double \*\*matrix;

public:

Directed\_Graph()

: list(new type[SIZE]), was(new bool[SIZE]())

{

matrix = new double\*[SIZE];

for(int i = 0; i < SIZE; i++)

matrix[i] = new double[SIZE]();

}

void print()

{

for(int i = 0; i < size\_l; i++)

{

cout << list[i] << " : ";

for(int j = 0; j < size\_l; j++)

if(matrix[i][j] != 0)

cout << list[j] << "(" << matrix[i][j] << ") ";

cout << endl;

}

}

void push(const type key, const type value, const float weight)

{

size\_t k = find\_or\_add(key);

size\_t v = find\_or\_add(value);

matrix[k][v] = weight;

}

size\_t find\_or\_add(const type litter)

{

for(int i = 0; i < size\_l; i++)

if(list[i] == litter)

return i;

resize\_list();

list[size\_l] = litter;

return size\_l++;

}

void resize\_list()

{

if(size\_l%SIZE != 0 || !size\_l) return;

type \*new\_list = new type[size\_l + SIZE];

for(int i = 0; i < size\_l; i++)

new\_list[i] = list[i];

delete [] list;

list = new\_list;

bool \*new\_was = new bool[size\_l + SIZE]();

for(int i = 0; i < size\_l; i++)

new\_was[i] = was[i];

delete [] was;

was = new\_was;

double \*\*new\_matrix = new double\*[size\_l + SIZE];

for(int i = 0; i < size\_l + SIZE; i++)

new\_matrix[i] = new double[size\_l + SIZE]();

for(int i = 0; i < size\_l; i++)

{

for(int j = 0; j < size\_l; j++)

new\_matrix[i][j] = matrix[i][j];

delete [] matrix[i];

}

delete [] matrix;

matrix = new\_matrix;

};

bool find\_min(const size\_t ver, size\_t &min)

{

min = -1;

for(int i = 0; i < size\_l; i++)

if(matrix[ver][i] && !was[i]){

min = i;

break;

}

cout << "Поиск минимального ребра для текущей вершины (" << list[ver] <<"):" << endl;

if(min == -1)

{

cout << "Доступных вершин из текущей больше нет." << endl;

return 0;

}

cout << "Список доступных:" << endl;

for(int i = 0; i < size\_l; i++)

{

if(matrix[ver][i] && !was[i])

cout << list[i] << " (" << matrix[ver][i] << ") ";

if(matrix[ver][i] && !was[i] && matrix[ver][i] < matrix[ver][min] && matrix[ver][i] != matrix[ver][min])

min = i;

}

cout << endl << "Минимальная из всех - " << list[min] << " (" << matrix[ver][min] << ")" << endl;

was[min] = 1;

return 1;

}

int find\_way(const size\_t start, const size\_t finish, deque <char> &way)

{

cout << "Добавляем в путь вершину :" << list[start] << endl;

way.push\_back(list[start]);

if(start == finish)

{

cout << "Путь найден." << endl;

return FINISH;

}

size\_t min;

int result;

while(find\_min(start, min))

{

result = find\_way(min, finish, way);

if(result == FINISH)

return FINISH;

way.pop\_back();

}

return 0;

};

};

int main(){

setlocale(LC\_ALL, "rus");

char start, finish;

cin >> start >> finish;

Directed\_Graph graf;

char from, to;

float weight;

while(cin >> from >> to >> weight)

{

if(from == '^')

break;

graf.push(from, to, weight);

}

cout << "Полученный граф :" << endl;

graf.print();

deque <char> way;

cout << "Поиск пути :" << endl;

if(!graf.find\_way(graf.find\_or\_add(start), graf.find\_or\_add(finish), way))

return 0;

cout << endl;

cout << "Найденный путь :" << endl;

while(!way.empty())

{

cout << way.front();

way.pop\_front();

}

return 0;

}

* **2.2.**[**c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <cmath>

#include <iostream>

#include <functional>

#include <algorithm>

#include <vector>

#include <stack>

#define FINISH 0

#define WAS 1

#define WASNT 0

#define type char

#define DEMO

#define SIZE 10

using namespace std;

struct Vertex //Структура вершины

{

type value;

double weight;

type from;

};

class Directed\_Graph{ //Класс графа

private:

type \*list; //Список вершин

size\_t size\_l = 0; //Размер этого списка

double \*\*matrix; //Матрица смежности

public:

Directed\_Graph() //Инициализация графа

: list(new type[SIZE])

{

matrix = new double\*[SIZE];

for(int i = 0; i < SIZE; i++)

matrix[i] = new double[SIZE]();

}

void print() //Распечатка графа

{

for(int i = 0; i < size\_l; i++)

{

cout << list[i] << " : ";

for(int j = 0; j < size\_l; j++)

if(matrix[i][j] != 0)

cout << list[j] << "(" << matrix[i][j] << ") ";

cout << endl;

}

}

void push(const type key, const type value, const float weight) //Добавление вершины

{

size\_t k = find\_or\_add(key);

size\_t v = find\_or\_add(value);

matrix[k][v] = weight;

}

size\_t find\_or\_add(const type litter)//Нахождение или добавление индекса вершины

{

for(int i = 0; i < size\_l; i++)

if(list[i] == litter)

return i;

resize\_list();

list[size\_l] = litter;

return size\_l++;

}

void resize\_list() //Изменение размера списка вершин

{

if(size\_l%SIZE != 0 || !size\_l) return;

type \*new\_list = new type[size\_l + SIZE];

for(int i = 0; i < size\_l; i++)

new\_list[i] = list[i];

delete [] list;

list = new\_list;

double \*\*new\_matrix = new double\*[size\_l + SIZE];

for(int i = 0; i < size\_l + SIZE; i++)

new\_matrix[i] = new double[size\_l + SIZE]();

for(int i = 0; i < size\_l; i++)

{

for(int j = 0; j < size\_l; j++)

new\_matrix[i][j] = matrix[i][j];

delete [] matrix[i];

}

delete [] matrix;

matrix = new\_matrix;

};

bool in\_arr(type c, vector <Vertex> &arr) //Возвращает информацию о нахождении символа в массиве

{

for(int i = 0; i < arr.size(); i++)

if(arr[i].value == c)

return 1;

return 0;

};

size\_t index\_in\_arr(type c, vector <Vertex> &arr) //Возвращает индекс нахождения символа в массиве

{

for(size\_t i = 0; i < arr.size(); i++)

if(arr[i].value == c)

return i;

return 0;

};

size\_t min\_arr(vector <Vertex> &arr, size\_t finish) //Возвращает минимальный символ в массиве в соответствии в эвристикой

{

size\_t min = 0;

for(int i = 0; i < arr.size(); i++)

{

if(arr[i].weight + abs(arr[i].value - list[finish]) < arr[min].weight + abs(arr[min].value - list[finish]))

min = i;

if(arr[i].weight + abs(arr[i].value - list[finish]) == arr[min].weight + abs(arr[min].value - list[finish]))

if(abs(arr[i].value - list[finish]) < abs(arr[min].value - list[finish]))

min = i;

}

return min;

}

void find\_way(size\_t start, size\_t finish, stack <type> &way) //Находиждение путь алгоритмом А\*

{

#ifdef DEMO

cout << "В ожидающие добавляется точка старта (" << list[start] << "), список рассмотренных вершин пока пуст." << endl;

#endif

vector <Vertex> arr\_wait;//Ожидаемые вершины

vector <Vertex> arr\_past;//Просмотренные вершины

arr\_wait.push\_back({list[start], 0, 0});//Добавляем первую вершину

Vertex X, Y;

while(!arr\_wait.empty())//Пока есть вершины в списке ожидания

{

vector <Vertex>::iterator it = arr\_wait.begin();

size\_t min = min\_arr(arr\_wait, finish);//Достаем минимальный элемент

advance(it, min);

X = arr\_wait.at(min);//Помещаем в Х

#ifdef DEMO

cout << "Список на рассмотрение :" << endl;

for(int i = 0; i < arr\_wait.size(); i++)

cout << arr\_wait[i].value << " - F(" << arr\_wait[i].weight + abs(arr\_wait[i].value - list[finish]) << "), ";

cout << endl;

#endif

arr\_wait.erase(it); //Извлекаем элемент Х из массива вершин на ожидание

#ifdef DEMO

cout << "Наименьшая точка из списка на рассмотрение: " << X.value << endl;

cout << "Перейдем к ней." << endl << endl;

#endif

arr\_past.push\_back(X); //Помещаем элемент Х в массив просмотренных вершин

if(X.value == list[finish]) //Если дошли до финиша, взвращаемся

break;

#ifdef DEMO

cout << "Инцидентные вершины для (" << X.value << "):" << endl;

#endif

for(int i = 0; i < size\_l; i++)//Перебираем инцидентные вершины для Х

{

size\_t index\_x = find\_or\_add(X.value);//Находим индекс Х

if(matrix[index\_x][i] != 0) //Если существует путь

{

#ifdef DEMO

cout << list[i] << " - G(" << matrix[index\_x][i] + X.weight << ") H(" << abs(list[i] - list[finish]) << ") F(";

cout << matrix[index\_x][i] + X.weight + abs(list[i] - list[finish]) << ") - ";

#endif

Y = {list[i], matrix[index\_x][i] + X.weight, X.value}; //Создаем элемент для массива

if(in\_arr(Y.value, arr\_past)) //Если вершина уже в пройденном списке, то продолжаем

{

#ifdef DEMO

cout << "Вершина уже находится в рассмотренном списке, пропускаем ее." << endl;

#endif

continue;

}

if(!in\_arr(Y.value, arr\_wait)) //Если вершины нет в списке на ожидание, то добавляем ее туда

{

#ifdef DEMO

cout << "Вершины нет рассмотренном списке и в списке ожидания. Добавляем ее в список на рассмотрение." << endl;

#endif

arr\_wait.push\_back(Y);

}

else //Иначе обновляем У в списке на ожидание

{

#ifdef DEMO

cout << "Вершины нет рассмотренном списке но есть списке ожидания." << endl;

#endif

Vertex &old\_Y = arr\_wait[index\_in\_arr(Y.value, arr\_wait)];

if(Y.weight < old\_Y.weight)

{

#ifdef DEMO

cout << "Новый путь до вершины лучше предыдущего. Заменяем предыдущий им." << endl;

#endif

old\_Y.weight = Y.weight;

old\_Y.from = Y.from;

}

#ifdef DEMO

else

cout << "Новый путь до вершины не лучше предыдущего, ничего не делаем." << endl;

#endif

}

}

}

}

if(!in\_arr(list[finish], arr\_past)) //Если не дошли до финиша, то заканчиваем

return;

#ifdef DEMO

cout << "Маршрут найден." << endl;

#endif

Vertex \*ptr = &X;

while(ptr->value != list[start])//Находим путь

{

way.push(ptr->value);

ptr = &arr\_past[index\_in\_arr(ptr->from, arr\_past)];

}

way.push(ptr->value);

}

};

//Вывести стоимости пути в алг.

int main(){

setlocale(LC\_ALL, "rus");

type start, finish;

cin >> start >> finish;

Directed\_Graph graf;

type from, to;

double weight;

while(cin >> from >> to >> weight)//Считыние данных

{

if(from == '^')

break;

graf.push(from, to, weight);

}

#ifdef DEMO

cout << "Полученный граф :" << endl;

graf.print();

cout << endl;

#endif

stack <type> way;

graf.find\_way(graf.find\_or\_add(start), graf.find\_or\_add(finish), way);

#ifdef DEMO

cout << "Найденный путь от " << start << " до " << finish << " : ";

#endif

while(!way.empty())

{

cout << way.top() <<" ";

way.pop();

}

cout << endl;

return 0;

}