**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Жадный алгоритм и A\***

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Алтухов А.Д. |
| Преподаватель |  | Размочаева Н. В. |

Санкт-Петербург

2020

## Цель работы.

Построение и анализ жадного и эвристического алгоритмов нахождения кратчайшего пути в графе.

**Вариант 3.**

Написать функцию, проверяющую эвристику на допустимость и монотонность.

## Основные теоретические положения.

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 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.

## Описание алгоритма.

1. Жадный алгоритм.

В каждой вершине просматриваются все возможные пути и выбирается имеющий наименьший вес. Пройденный путь более не учитывается. Так продолжается до тех пор, пока не будет достигнута конечная вершина или не закончатся вершины для обработки.

1. Алгоритм А\*

Используется учет посещенных и непосещенных вершин. При обработке вершины все смежные с ней вносятся в набор непосещенных, обновляются их метки, рассчитанные на основе метки обрабатываемой вершины, веса пути и расстояния до конечной вершины. Обработанная вершина вносится в набор посещенных с учетом времени внесения. Следующая вершина для обработки выбирается по минимальной метке. При одинаковых метках приоритет имеет вершина, которая находится ближе к конечной вершине. Алгоритм заканчивает работу, когда начинается обработка конечной вершины или закончились доступные для обработки вершины.

После этого начинается восстановление пути. Для этого исследуются вершины, смежные с конечной. Если метка конечной это сумма метки смежной и веса соответствующего ребра, то искомый путь проходил через эту смежную метку и уже она принимается за конечную, после чего происходит очередная итерация.

Временная сложность алгоритмов: O(E + V\*V) = O(V2), где V — количество вершин, а E — количество ребер. Оценка справедлива для худшего случая, в котором будет произведен обход всех вершин и поиск смежных им по массиву (V2), и обход всех ребер.

Требуемая память: O(V2), так как для хранения связей используется матрица смежности.

## Описание основных структур данных и функций.

class Graph — класс, представляющий собой граф и методы работы с ним.

void greedySearch(int start, int end) — функция, запускающая жадный алгоритм.

int next() — поиск следующей вершины для жадного алгоритма.

void heuristicSearch(int start, int end) — функция, запускающая эвристический алгоритм.

int minNode() — поиск следующей вершины для эвристического алгоритма.

bool checkMonotony() — функция, проверяющая эвристику на монотонность. В данном случае под монотонностью понимается путь, в ходе обхода которого имена вершин возрастают.

Эвристика на допустимость проверяется самим фактом завершения эвристического алгоритма. Задача считается допустимой, если находится решение.

**Тестирование.**

Таблица 1 – Результаты тестирования жадного алгоритма.

|  |  |
| --- | --- |
| **Ввод** | **Вывод** |
| a g  a b 3.0  a c 1.0  b d 2.0  b e 3.0  d e 4.0  e a 1.0  e f 2.0  a g 8.0  f g 1.0 | abdeag |
| a d  a b 1.0  b c 1.0  c a 1.0  a d 8.0 | abcad |

Таблица 2 – Результаты тестирования эвристического алгоритма.

|  |  |
| --- | --- |
| **Ввод** | **Вывод** |
| 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 |
| a e  a b 3.0  b c 1.0  c d 1.0  a d 5.0  d e 1.0 | ade |

# Вывод.

В ходе работы были построены жадный и эвристический алгоритмы поиска кратчайшего пути. Также написаны функции, проверяющие дополнительные эвристики.

**ПРИЛОЖЕНИЕ А.  
ИСХОДНЫЙ КОД**

#include <iostream>

#include <map>

#include <vector>

#include <algorithm>

#include <stack>

#include <utility>

#include <queue>

#include <set>

#include <string>

#include <locale>

class Graph {

std::vector<std::vector<double>> matrix;

std::vector<int> distance;

std::vector<int> distanceWithHeuristic; //добавляется расстояние между символами

std::vector<int> visited; //хранит время посещения

std::set<int> notVisited;

std::vector<std::pair<int, double>> path;

std::queue<int> queue;

int visitTime;

std::string answer;

public:

Graph(int start, int end) {

visitTime = 1;

for (int i = 0; i < std::max(start - 97 + 1, end - 97 + 1); i++) {

distance.push\_back((i == start - 97) ? 0 : 10000);

visited.push\_back(0);

matrix.push\_back(std::vector<double>());

for (int j = 0; j < std::max(start - 97 + 1, end - 97 + 1); j++) {

matrix[i].push\_back(-1);

}

}

}

void expandMatrix(int maxSize) {

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

for (int j = matrix.size(); j < maxSize; j++) {

matrix[i].push\_back(-1);

}

}

for (int i = matrix.size(); i < maxSize; i++) {

distance.push\_back(10000);

visited.push\_back(0);

matrix.push\_back(std::vector<double>());

for (int j = 0; j < maxSize; j++) {

matrix[i].push\_back(-1);

}

}

}

void setNode(int from, int whereto, double weight) {

if (std::max(from - 97, whereto - 97) >= matrix.size()) {

expandMatrix(std::max(from - 97 + 1, whereto - 97 + 1));

}

matrix[from-97][whereto-97] = weight;// a = 97

}

void printMatrix() {

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

for (int j = 0; j < matrix[i].size(); j++) {

std::cout << (char)(i + 97) << " " << (char)(j + 97) << " " << matrix[i][j] << "\n"; // a = 97

}

}

}

int next(int current) { //для жадного алгоритма: выбирает следующую вершину по наименьшему пути

int minPath = 10000;

int minIndex = -1;

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

if ((matrix[current][i] > -1) && (matrix[current][i] < minPath)) {

minPath = matrix[current][i];

minIndex = i;

}

}

return minIndex;

}

void greedySearch(int start, int end) {

path.push\_back({ start, 0 });

while (start != end) {

int index = next(start);

if (index > -1) {

path.push\_back({ index, matrix[start][index] });

matrix[start][index] = -1;

start = index;

}

else { //если больше путей нет возвращаемся на шаг назад

path.pop\_back();

start = path[path.size() - 1].first;

}

}

printResult();

}

int minNode() { //выбор следующей вершины для обработки по наименьшей метке с учетом расстояния между символами

int min = -1;

for (auto i : notVisited) {

if (min < 0) {

min = i;

}

else if (distanceWithHeuristic[min] > distanceWithHeuristic[i]) {

min = i;

}

else if (distanceWithHeuristic[min] == distanceWithHeuristic[i]) {

if (min < i) {

min = i;

}

}

}

return min;

}

bool heuristicSearch(int start, int end) {

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

distanceWithHeuristic.push\_back(10000);

}

notVisited.insert(start);

distance[start] = 0;//abs(end-start);

distanceWithHeuristic[start] = abs(end - start);

while (!notVisited.empty()) {

int curMinNode = minNode();

std::cout << "Обрабатываемая вершина: " << (char)(curMinNode + 97) << "\n";

if (curMinNode == end) {

printResultHeuristic(start, end);

return true;

}

notVisited.erase(curMinNode);

visited[curMinNode] = visitTime++;

for (int i = 0; i < matrix.size(); i++) { //обработка всех смежных вершин и обновление меток

if ((matrix[curMinNode][i] > -1) && (visited[curMinNode] != 0)) {

int newDistance = distance[curMinNode] + matrix[curMinNode][i]; //+ abs(i - end);

if ((!visited[i]) || (distance[i] > newDistance)) {

std::cout << "Обновление вершины: " << (char)(i + 97) << ". Новая метка: "<< newDistance <<"\n";

distance[i] = newDistance;

distanceWithHeuristic[i] = distance[i] + abs(i - end);

visited[i] = 0;

if (notVisited.find(i) == notVisited.end()) {

notVisited.insert(i);

}

}

}

}

std::cout << "\n\n";

}

return false;

}

void printResult() {

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

std::cout << (char)(path[i].first + 97); // a = 97

}

std::cout << "\n";

}

void printResultHeuristic(int start, int end) {

int trueEnd = end;

int cur = end;

std::stack<int> sequence;

sequence.push(end);

std::vector<int> options;

while (end != start) {

int withMinTime = -1;

int minIndex = -1;

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

if (matrix[i][end] > -1) {

if ((distance[end] - matrix[i][end]) == distance[i]) { //если есть два возможных перехода, то выбираем исходя из времени посещения

if (visited[i] > withMinTime) {

withMinTime = visited[i];

minIndex = i;

}

}

}

}

end = minIndex;

sequence.push(end);

}

std::cout << "Ответ: ";

while (!sequence.empty()) {

answer += (char)(sequence.top() + 97);

std::cout << (char)(sequence.top()+97);

sequence.pop();

}

std::cout << "\n";

}

bool checkMonotony() {

if (answer.size() == 0)

return false;

for (int i = 0; i < (answer.size()-1); i++) {

if (answer[i] > answer[i + 1]) {

return false;

}

}

return true;

}

};

int main(){

setlocale(LC\_ALL, "Russian");

char start = '\0';

char end = '\0';

std::cin >> start >> end;

char from = '\0';

char whereto = '\0';

double weight = 0;

Graph graph(start, end);

std::cin >> from >> whereto >> weight;

while (weight != -1) {

//while (!std::cin.eof()){

//std::cin >> from >> whereto >> weight;

graph.setNode(from, whereto, weight);

std::cin >> from >> whereto >> weight;

}

std::cout << "============================\n";

//graph.greedySearch(start - 97, end - 97);

if (graph.heuristicSearch(start - 97, end - 97))

std::cout << "Задача допустимая\n";

else

std::cout << "Задача не допустимая\n";

if (graph.checkMonotony())

std::cout << "Задача монотонная\n";

else

std::cout << "Задача не монотонная\n";

std::cout << "============================\n";

return 0;

}