**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. 6381 |  | Поляков Н. С. |
| Преподаватель |  | Филатов А. Ю. |

Санкт-Петербург

2018

**Цель работы.**

Изучение жадного алгоритма для поиска пути в графе и алгоритма А\* для поиска кратчайшего пути в графе.

**Условия задания.**

1. 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.
2. 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.

**Входные данные.**

Первый строка: два символа через пробел – начальная и конечная вершины. Каждая последующая строка в формате вершина, пробел, вершина, пробел вес соответствует ребру в графе.

**Выходные данные.**

Строка – найденный путь.

**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

**Соответствующие выходные данных.**

1. Жадный алгоритм: abcde
2. A\*: ade

**Описание структур данных.**

|  |  |  |
| --- | --- | --- |
| Класс | Имя | Назначение |
| Edge | double weight | Вес ребра |
| Vertex \*end | Вершина, в которую ребро входит |
| Edge \*next | Следующее ребро в списке |
| void setWeight(double) | Устанавливает поле weight |
| void setEnd(Vertex\*) | Устанавливает поле end |
| void setNext(Edge\*) | Устанавливает поле next |
| double getWeight() | Возвращает значение поля weight |
| Vertex\* getEnd() | Возвращает значение поля end |
| Edge(double,Vertex\*) | Конструктор |
| Edge\*& getNext() | Возвращает значение поля next |
| void addToList(Priority\_list&,Vertex\*) | Добавляет вершину end в список приоритетов |
| Graph | Vertex \*head | Голова списка вершин |
| int number | Количество вершин в графе |
| Vertex\* addVertex(char) | Добавляет вершину к графу |
| void addEdge(char,char,double) | Добавляет ребро в граф |
| std::string AStar(char,char) | Находит путь в графе алгоритмом A\* |
| Vertex\* findVertex(char) | Находит вершину по ее имени |
| Graph() | Конструктор |
| item | item \*next | Следующий элемент списка |
| Vertex \*ver | Вкршина |
| double priority | Приоритет |
| Priority\_list | item \*head | Голова списка приоритетов |
| char end | Имя конечной вершины |
| void push(Vertex\*,double) | Добавляет вершину в список с приоритетом |
| Vertex\* pop() | Удаляет первый элемент списка |
| void pop(Vertex\*) | Удаляет вершину из списка |
| Priority\_list(char) | Конструктор |
| Vertex | char name | Имя вершины |
| bool visited | Индикатор посещенности |
| double path | Длина пути до этой вершины |
| Vertex \*prevVisited | Предыдущая посещенная вершина |
| Vertex \*next | Следующая вершина в списке |
| Vertex \*prev | Предыдущая вершина в списке |
| Edge \*trail | Список инцидентных ребер |
| char getName() | Возвращает значение поля name |
| bool getVisited() | Возвращает значение поля visited |
| void addToVisited(Priority\_list&) | Добавляет вершину к посещенным |
| void addEdge(double,Vertex\*) | Добавляет инцидентное ребро |
| Vertex(char) | Конструктор |
| void setNext(Vertex\*) | Устанавливает поле next |
| void setPrev(Vertex\*) | Устанавливает поле prev |
| void setPrevVisited(Vertex\*) | Устанавливает поле prevVisited |
| void setPath(double) | Устанавливает поле path |
| void setVisited(bool) | Устанавливает поле visited |
| void setName(char) | Устанавливает поле name |
| double getPath() | Возвращает значение поля path |
| Vertex\* getPrevVisited() | Возвращает значение поля prevVisited |
| Vertex\* getPrev() | Возвращает значение поля prev |
| Vertex\* getNext() | Возвращает значение поля next |

**Описание алгоритма.**

1. Жадный алгоритм.

На каждом шаге алгоритм выбирает ребро с наименьшим весом. Посещенная вершина сохраняется в строку с результатом и производится переход к следующей вершине по этому ребру и следующему шагу. При возникновении тупиковой ситуации, производится переход к предыдущей вершине с удалением последнего символа из строки с результатом.

1. A\*.

На каждом шаге алгоритма все вершины, в которые можно попасть из данной, добавляются в очередь с приоритетами с приоритетом, равным сумме величины пути до данной вершины и эвристичской оценки. Следующая выбирается из очереди с приоритетами – вершина, имеющая наименьший приоритет и производится переход к следующему шагу. При достижении конечной вершины, происходит проход получившегося пути от конца к началу с записью вершин в строку с результатом.

**Тестирование.**

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод | Рисунок |
| a e  a b 3.0  b c 1.0  c d 1.0  a d 5.0  d e 1.0 | Жадный алгоритм: abcde  A\*: ade |  |
| a e  a b 3.0  a d 5.0  a c 1.1  a f 0.5  b c 1.0  b g 0.9  c d 1.0  d e 5.1  d g 3.3  f e 1.7  f b 1.6  g e 4.2 | Жадный алгоритм: afbgde  A\*: afe |  |

**Вывод.**

В ходе выполнения данной лабораторной работы были изучены методы хранения графов в памяти компьютера, изучен жадный алгоритм поиска пути в графе и алгоритм А\*.