

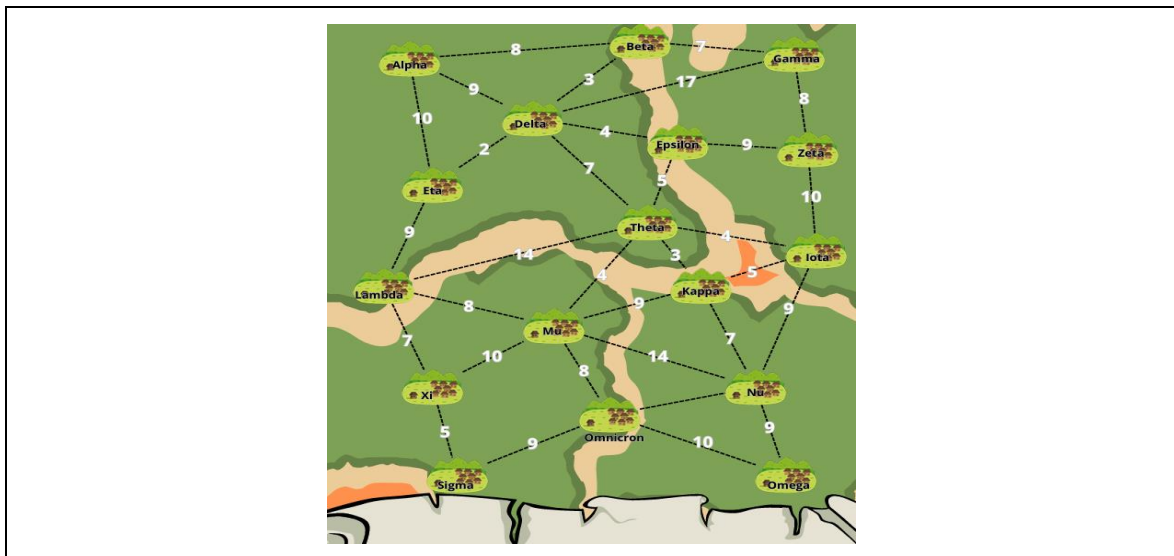
MODUL V

GRAPH

5.1 PERMASALAHAN

5.1.1 DESKRIPSI

Pak ophet adalah calon bupati dari kabupaten merembu, pak ophet berasal dari salah satu desa di kabupaten tersebut dalam rangka melakukan kampanye pak ophet menginginkan supaya bisa mengunjungi semua desa dengan rute tercepat yang dimana berarti kalian di minta untuk membuat list jalur dari desa pak ophet ke setiap desa lainnya dalam jalan tercepat Perlu di perhatikan bahwa jalan tercepat blum tentu berasal dari jalan langsung ke tempatnya, dan bisa saja ada jalan yang lebih pendek dengan mengitari desa lainnya Perlu di perhatikan bahwa juga untuk di urutkan desa terdekat terlebih dahulu karena pak ophet menginginkan untuk kampane di desa desa terdekatnya terlebih dahulu



Gambar 5.1 Ilustrasi dari *Graph* Permasalahan

5.1 OUTPUT DARI DEMO PERMASALAHAN

Theta	Alpha	Mu
Theta => Kappa	Alpha => Beta	Mu => Theta
Theta => Iota	Alpha => Delta	Mu => Theta => Kappa
Theta => Mu	Alpha => Eta	Mu => Theta => Iota
Theta => Epsilon	Alpha => Delta => Epsilon	Mu => Lambda
Theta => Delta	Alpha => Beta => Gamma	Mu => Omicron
Theta => Delta => Eta	Alpha => Delta => Theta	Mu => Theta => Epsilon
Theta => Delta => Beta	Alpha => Eta => Lambda	Mu => Xi
Theta => Kappa => Nu	Alpha => Delta => Theta => Kappa	Mu => Theta => Delta
Theta => Mu => Lambda	Alpha => Delta => Theta => Iota	Mu => Omicron => Nu
Theta => Mu => Omicron	Alpha => Delta => Theta => Mu	Mu => Theta => Delta => Eta
Theta => Epsilon => Zeta	Alpha => Delta => Epsilon => Zeta	Mu => Theta => Delta => Beta
Theta => Mu => Xi	Alpha => Delta => Theta => Kappa => Nu	Mu => Xi => Sigma
Theta => Delta => Alpha	Alpha => Eta => Lambda => Xi	Mu => Theta => Epsilon => Zeta
Theta => Delta => Beta => Gamma	Alpha => Delta => Theta => Mu => Omicron	Mu => Omicron => Omega
Theta => Kappa => Nu => Omega	Alpha => Eta => Lambda => Xi => Sigma	Mu => Theta => Delta => Alpha
Theta => Mu => Xi => Sigma	Alpha => Delta => Theta => Kappa => Nu => Omega	Mu => Theta => Delta => Beta => Gamma

Gambar 5.2 Output Permasalahan

5.2 PROGRAM OPNET

1. Algoritma

- a. Program dimulai.
- b. Program menampilkan graph dari theta ke semua desa dengan memilih jalur terpendek.
- c. Program menampilkan graph dari Alpha ke semua desa dengan memilih jalur terpendek.
- d. Program menampilkan graph dari Mu ke semua desa dengan memilih jalur terpendek.
- e. Program selesai.

2. Source Code

a. Kelas Node

```
class Node {
    String nama, parent;
    int jarak;
    Node next;

    public Node(String nama, int jarak, String parent) {
        this.nama = nama;
        this.jarak = jarak;
        this.parent = parent;
        this.next = null;
    }
}

class Edge {
    String from, to;
    int jarak;
    Edge next;

    public Edge (String from, String to, int jarak) {
        this.from = from;
        this.to = to;
        this.jarak = jarak;
        this.next = null;
    }
}
```

b. Kelas Edge

```
class Edge {
    String from, to;
    int jarak;
    Edge next;

    public Edge(String from, String to, int jarak) {
        this.from = from;
        this.to = to;
        this.jarak = jarak;
        this.next = null;
    }
}
```

c. Kelas EdgeList

```

class EdgeList {
    Edge head;
    public void add(String from, String to, int jarak) {
        Edge newEdge = new Edge(from, to, jarak);
        if (head == null) {
            head = newEdge;
        } else {
            Edge temp = head;
            while (temp.next != null) {temp = temp.next;}
            temp.next = newEdge;
        }
    }

    public LinkedList getNeighbors(String nama) {
        LinkedList neighbors = new LinkedList();
        Edge temp = head;
        while (temp != null) {
            if (temp.from.equals(nama)) {neighbors.add(temp.to,
temp.jarak, nama);}
            else if (temp.to.equals(nama)) {neighbors.add(temp.from,
temp.jarak, nama);}
            temp = temp.next;
        }
        return neighbors;
    }

    public int getDistance(String from, String to) {
        Edge temp = head;
        while (temp != null) {
            if ((temp.from.equals(from) && temp.to.equals(to)) ||
(temp.from.equals(to) && temp.to.equals(from))) {
                return temp.jarak;
            }
            temp = temp.next;
        }
        return Integer.MAX_VALUE;
    }
}

```

d. Kelas LinkedList

```

class LinkedList {
    Node head;

    public void add(String nama, int jarak, String parent) {
        Node newNode = new Node(nama, jarak, parent);
        if (head == null) {head = newNode;}
        else {
            Node temp = head;
            while (temp.next != null) {temp = temp.next;}
            temp.next = newNode;
        }
    }

    public boolean contains(String nama) {
        Node temp = head;
        while (temp != null) {
            if (temp.nama.equals(nama)) {return true;}
            temp = temp.next;
        }
    }
}

```

```

    }
    return false;
}

public void remove (String nama) {
    if (head != null && head.nama.equals(nama)) {
        head = head.next;
        return;
    }
    Node temp = head;
    while (temp != null && temp.next != null) {
        if (temp.next.nama.equals(nama)) {
            temp.next = temp.next.next;
            return;
        }
        temp = temp.next;
    }
}

public Node getMinDistanceNode() {
    Node minNode = head;
    Node temp = head;
    while (temp != null) {
        if (temp.jarak < minNode.jarak) {minNode = temp;}
        temp = temp.next;
    }
    return minNode;
}

public void updateDistance(String nama, int newDistance, String
parent) {
    Node temp = head;
    while (temp != null) {
        if (temp.nama.equals(nama)) {
            temp.jarak = newDistance;
            temp.parent = parent;
            return;
        }
        temp = temp.next;
    }
}

public void clear() {head = null;}

public void sortByDistance() {
    if (head == null || head.next == null) {
        return;
    }

    boolean swapped;
    do {
        swapped = false;
        Node current = head;
        while (current.next != null) {
            if (current.jarak > current.next.jarak ||
                (current.jarak == current.next.jarak &&
current.nama.compareTo(current.next.nama) > 0)) {
                String tempNama = current.nama;
                int tempJarak = current.jarak;
                String tempParent = current.parent;

```

```

        current.nama = current.next.nama;
        current.jarak = current.next.jarak;
        current.parent = current.next.parent;

        current.next.nama = tempNama;
        current.next.jarak = tempJarak;
        current.next.parent = tempParent;

        swapped = true;
    }
    current = current.next;
}
} while (swapped);
}

@Override
public String toString() {
    Node temp = head;
    StringBuilder result = new StringBuilder();
    while (temp != null) {
        result.append(temp.nama).append("
").append(temp.jarak).append(" ");
        if (temp.next != null) result.append(" => ");
        temp = temp.next;
    } return result.toString();
}

public void addWithPriority(String nama, int jarak, String
parent) {
    Node newNode = new Node(nama, jarak, parent);
    if (head == null || head.jarak > jarak || (head.jarak ==
jarak && head.nama.compareTo(nama) > 0)) {
        newNode.next = head;
        head = newNode;
    } else {
        Node temp = head;
        while (temp.next != null &&
            (temp.next.jarak < jarak || (temp.next.jarak ==
jarak && temp.next.nama.compareTo(nama) <= 0))) {
            temp = temp.next;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
}

class EdgeList {
    Edge head;
    public void add(String from, String to, int jarak) {
        Edge newEdge = new Edge(from, to, jarak);
        if (head == null) {
            head = newEdge;
        } else {
            Edge temp = head;
            while (temp.next != null) {temp = temp.next;}
            temp.next = newEdge;
        }
    }

    public LinkedList getNeighbors(String nama) {

```

```

        LinkedList neighbors = new LinkedList();
        Edge temp = head;
        while (temp != null) {
            if (temp.from.equals(nama)) {neighbors.add(temp.to,
temp.jarak, nama);}
            else if (temp.to.equals(nama)) {neighbors.add(temp.from,
temp.jarak, nama);}
            temp = temp.next;
        }
        return neighbors;
    }

    public int getDistance(String from, String to) {
        Edge temp = head;
        while (temp != null) {
            if ((temp.from.equals(from) && temp.to.equals(to)) ||
(temp.from.equals(to) && temp.to.equals(from))) {
                return temp.jarak;
            }
            temp = temp.next;
        }
        return Integer.MAX_VALUE;
    }
}

```

e. Kelas ShortPathFinder

```

class ShortestPathFinder {
    public static void findShortestPaths(String start, EdgeList
edges) {
        LinkedList visited = new LinkedList();
        LinkedList unvisited = new LinkedList();
        unvisited.add(start, 0, null);

        while (unvisited.head != null) {
            Node currentNode = unvisited.getMinDistanceNode();
            unvisited.remove(currentNode.nama);
            visited.add(currentNode.nama,      currentNode.jarak,
currentNode.parent);

            LinkedList neighbors =
edges.getNeighbors(currentNode.nama);
            Node temp = neighbors.head;
            while (temp != null) {
                if (!visited.contains(temp.nama)) {
                    int newDistance = currentNode.jarak +
edges.getDistance(currentNode.nama, temp.nama);
                    if (!unvisited.contains(temp.nama)) {
                        unvisited.add(temp.nama,      newDistance,
currentNode.nama);
                    } else {
                        unvisited.updateDistance(temp.nama,
newDistance, currentNode.nama);
                    }
                }
                temp = temp.next;
            }
        }
        visited.sortByDistance();
        Node temp = visited.head;
        while (temp != null) {
            printRoute(temp.nama, visited);
        }
    }
}

```

```

        temp = temp.next;
    }
}

public static void printRoute(String nama, LinkedList visited) {
    StringBuilder route = new StringBuilder();
    buildRoute(nama, visited, route);
    System.out.println(route);
}

public static void buildRoute(String nama, LinkedList visited,
    StringBuilder route) {
    Node temp = visited.head;
    while (temp != null) {
        if (temp.nama.equals(nama)) {
            if (temp.parent != null) {
                buildRoute(temp.parent, visited, route);
                route.append(" => ");
            }
            route.append(temp.nama);
            return;
        }
        temp = temp.next;
    }
}
}

```

f. Kelas Main

```

public class Main {
    private static final EdgeList edges1;
    private static final EdgeList edges2;
    private static final EdgeList edges3;

    static {
        edges1 = new EdgeList();
        edges1.add("Theta", "Kappa", 3);
        edges1.add("Theta", "Iota", 4);
        edges1.add("Theta", "Mu", 4);
        edges1.add("Theta", "Epsilon", 5);
        edges1.add("Theta", "Delta", 7);
        edges1.add("Delta", "Eta", 2);
        edges1.add("Delta", "Beta", 3);
        edges1.add("Kappa", "Nu", 7);
        edges1.add("Mu", "Lambda", 8);
        edges1.add("Mu", "Omicron", 8);
        edges1.add("Epsilon", "Zeta", 8);
        edges1.add("Mu", "Xi", 10);
        edges1.add("Delta", "Alpha", 9);
        edges1.add("Delta", "Gamma", 17);
        edges1.add("Nu", "Omega", 9);
        edges1.add("Beta", "Gamma", 7);
        edges1.add("Xi", "Sigma", 5);

        edges2 = new EdgeList();
        edges2.add("Alpha", "Beta", 8);
        edges2.add("Alpha", "Delta", 9);
        edges2.add("Alpha", "Eta", 10);
        edges2.add("Delta", "Epsilon", 4);
        edges2.add("Beta", "Gamma", 7);
        edges2.add("Delta", "Theta", 7);
        edges2.add("Eta", "Lambda", 9);
    }
}

```

```
edges2.add("Theta", "Kappa", 3);
edges2.add("Theta", "Iota", 4);
edges2.add("Theta", "Mu", 4);
edges2.add("Epsilon", "Zeta", 9);
edges2.add("Kappa", "Nu", 7);
edges2.add("Lambda", "Xi", 7);
edges2.add("Mu", "Omicron", 8);
edges2.add("Xi", "Sigma", 5);
edges2.add("Nu", "Omega", 9);

edges3 = new EdgeList();
edges3.add("Mu", "Theta", 4);
edges3.add("Theta", "Kappa", 3);
edges3.add("Theta", "Iota", 4);
edges3.add("Mu", "Lambda", 8);
edges3.add("Mu", "Omicron", 8);
edges3.add("Theta", "Epsilon", 5);
edges3.add("Mu", "Xi", 10);
edges3.add("Theta", "Delta", 7);
edges3.add("Beta", "Gamma", 7);
edges3.add("Omicron", "Nu", 3);
edges3.add("Delta", "Eta", 2);
edges3.add("Delta", "Beta", 3);
edges3.add("Xi", "Sigma", 5);
edges3.add("Epsilon", "Zeta", 9);
edges3.add("Omicron", "Omega", 10);
edges3.add("Alpha", "Delta", 9);
}

public static void main(String[] args) {
    ShortestPathFinder.findShortestPaths("Theta", edges1);
    System.out.println();
    ShortestPathFinder.findShortestPaths("Alpha", edges2);
    System.out.println();
    ShortestPathFinder.findShortestPaths("Mu", edges3);
}
}
```


5.3 ANALISIS DATA

5.3.1 Program Ophet

```
class Node {
    String nama, parent;
    int jarak;
    Node next;

    public Node(String nama, int jarak, String parent) {
        this.nama = nama;
        this.jarak = jarak;
        this.parent = parent;
        this.next = null;
    }
}

class Edge {
    String from, to;
    int jarak;
    Edge next;

    public Edge (String from, String to, int jarak) {
        this.from = from;
        this.to = to;
        this.jarak = jarak;
        this.next = null;
    }
}
```

Script di atas

```
class Edge {
    String from, to;
    int jarak;
    Edge next;

    public Edge(String from, String to, int jarak) {
        this.from = from;
        this.to = to;
        this.jarak = jarak;
        this.next = null;
    }
}
```

Script di atas

```
class EdgeList {
    Edge head;
    public void add(String from, String to, int jarak) {
        Edge newEdge = new Edge(from, to, jarak);
        if (head == null) {
            head = newEdge;
        } else {
            Edge temp = head;
            while (temp.next != null) {temp = temp.next;}
            temp.next = newEdge;
        }
    }

    public LinkedList getNeighbors(String nama) {
        LinkedList neighbors = new LinkedList();
        Edge temp = head;
        while (temp != null) {
```

```

        if (temp.from.equals(nama)) {neighbors.add(temp.to,
temp.jarak, nama);}
        else if (temp.to.equals(nama)) {neighbors.add(temp.from,
temp.jarak, nama);}
        temp = temp.next;
    }
    return neighbors;
}

public int getDistance(String from, String to) {
    Edge temp = head;
    while (temp != null) {
        if ((temp.from.equals(from) && temp.to.equals(to)) ||
(temp.from.equals(to) && temp.to.equals(from))) {
            return temp.jarak;
        }
        temp = temp.next;
    }
    return Integer.MAX_VALUE;
}
}

```

Script di atas

```

class LinkedList {
    Node head;

    public void add(String nama, int jarak, String parent) {
        Node newNode = new Node(nama, jarak, parent);
        if (head == null) {head = newNode;}
        else {
            Node temp = head;
            while (temp.next != null) {temp = temp.next;}
            temp.next = newNode;
        }
    }

    public boolean contains(String nama) {
        Node temp = head;
        while (temp != null) {
            if (temp.nama.equals(nama)) {return true;}
            temp = temp.next;
        }
        return false;
    }

    public void remove (String nama) {
        if (head != null && head.nama.equals(nama)) {
            head = head.next;
            return;
        }
        Node temp = head;
        while (temp != null && temp.next != null) {
            if (temp.next.nama.equals(nama)) {
                temp.next = temp.next.next;
                return;
            }
            temp = temp.next;
        }
    }
}

```

```

public Node getMinDistanceNode() {
    Node minNode = head;
    Node temp = head;
    while (temp != null) {
        if (temp.jarak < minNode.jarak) {minNode = temp;}
        temp = temp.next;
    }
    return minNode;
}

public void updateDistance(String nama, int newDistance, String
parent) {
    Node temp = head;
    while (temp != null) {
        if (temp.nama.equals(nama)) {
            temp.jarak = newDistance;
            temp.parent = parent;
            return;
        }
        temp = temp.next;
    }
}

public void clear() {head = null;}

public void sortByDistance() {
    if (head == null || head.next == null) {
        return;
    }

    boolean swapped;
    do {
        swapped = false;
        Node current = head;
        while (current.next != null) {
            if (current.jarak > current.next.jarak ||
                (current.jarak == current.next.jarak &&
current.nama.compareTo(current.next.nama) > 0)) {
                String tempNama = current.nama;
                int tempJarak = current.jarak;
                String tempParent = current.parent;

                current.nama = current.next.nama;
                current.jarak = current.next.jarak;
                current.parent = current.next.parent;

                current.next.nama = tempNama;
                current.next.jarak = tempJarak;
                current.next.parent = tempParent;

                swapped = true;
            }
            current = current.next;
        }
    } while (swapped);
}

@Override
public String toString() {
    Node temp = head;
    StringBuilder result = new StringBuilder();

```

```

        while (temp != null) {
            result.append(temp.nama).append("
").append(temp.jarak).append("
");
            if (temp.next != null) result.append(" => ");
            temp = temp.next;
        } return result.toString();
    }

    public void addWithPriority(String nama, int jarak, String parent)
    {
        Node newNode = new Node(nama, jarak, parent);
        if (head == null || head.jarak > jarak || (head.jarak == jarak
&& head.nama.compareTo(nama) > 0)) {
            newNode.next = head;
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null &&
                (temp.next.jarak < jarak || (temp.next.jarak == jarak
&& temp.next.nama.compareTo(nama) <= 0))) {
                temp = temp.next;
            }
            newNode.next = temp.next;
            temp.next = newNode;
        }
    }
}

class EdgeList {
    Edge head;
    public void add(String from, String to, int jarak) {
        Edge newEdge = new Edge(from, to, jarak);
        if (head == null) {
            head = newEdge;
        } else {
            Edge temp = head;
            while (temp.next != null) {temp = temp.next;}
            temp.next = newEdge;
        }
    }

    public LinkedList getNeighbors(String nama) {
        LinkedList neighbors = new LinkedList();
        Edge temp = head;
        while (temp != null) {
            if (temp.from.equals(nama)) {neighbors.add(temp.to,
temp.jarak, nama);}
            else if (temp.to.equals(nama)) {neighbors.add(temp.from,
temp.jarak, nama);}
            temp = temp.next;
        }
        return neighbors;
    }

    public int getDistance(String from, String to) {
        Edge temp = head;
        while (temp != null) {
            if ((temp.from.equals(from) && temp.to.equals(to)) ||
(temp.from.equals(to) && temp.to.equals(from))) {
                return temp.jarak;
            }
        }
    }
}

```

```

        temp = temp.next;
    }
    return Integer.MAX_VALUE;
}
}

```

Script di atas

```

class ShortestPathFinder {
    public static void findShortestPaths(String start, EdgeList edges)
    {
        LinkedList visited = new LinkedList();
        LinkedList unvisited = new LinkedList();
        unvisited.add(start, 0, null);

        while (unvisited.head != null) {
            Node currentNode = unvisited.getMinDistanceNode();
            unvisited.remove(currentNode.nama);
            visited.add(currentNode.nama,                currentNode.jarak,
currentNode.parent);

            LinkedList neighbors =
edges.getNeighbors(currentNode.nama);
            Node temp = neighbors.head;
            while (temp != null) {
                if (!visited.contains(temp.nama)) {
                    int newDistance = currentNode.jarak +
edges.getDistance(currentNode.nama, temp.nama);
                    if (!unvisited.contains(temp.nama)) {
                        unvisited.add(temp.nama,                newDistance,
currentNode.nama);
                    } else {
                        unvisited.updateDistance(temp.nama,
newDistance, currentNode.nama);
                    }
                }
                temp = temp.next;
            }
        }
        visited.sortByDistance();
        Node temp = visited.head;
        while (temp != null) {
            printRoute(temp.nama, visited);
            temp = temp.next;
        }
    }

    public static void printRoute(String nama, LinkedList visited) {
        StringBuilder route = new StringBuilder();
        buildRoute(nama, visited, route);
        System.out.println(route);
    }

    public static void buildRoute(String nama, LinkedList visited,
StringBuilder route) {
        Node temp = visited.head;
        while (temp != null) {
            if (temp.nama.equals(nama)) {
                if (temp.parent != null) {
                    buildRoute(temp.parent, visited, route);
                    route.append(" => ");
                }
            }
        }
    }
}

```

```

        route.append(temp.nama);
        return;
    }
    temp = temp.next;
}
}
}

```

Script di atas

```

public class Main {
    private static final EdgeList edges1;
    private static final EdgeList edges2;
    private static final EdgeList edges3;

    static {
        edges1 = new EdgeList();
        edges1.add("Theta", "Kappa", 3);
        edges1.add("Theta", "Iota", 4);
        edges1.add("Theta", "Mu", 4);
        edges1.add("Theta", "Epsilon", 5);
        edges1.add("Theta", "Delta", 7);
        edges1.add("Delta", "Eta", 2);
        edges1.add("Delta", "Beta", 3);
        edges1.add("Kappa", "Nu", 7);
        edges1.add("Mu", "Lambda", 8);
        edges1.add("Mu", "Omicron", 8);
        edges1.add("Epsilon", "Zeta", 8);
        edges1.add("Mu", "Xi", 10);
        edges1.add("Delta", "Alpha", 9);
        edges1.add("Delta", "Gamma", 17);
        edges1.add("Nu", "Omega", 9);
        edges1.add("Beta", "Gamma", 7);
        edges1.add("Xi", "Sigma", 5);

        edges2 = new EdgeList();
        edges2.add("Alpha", "Beta", 8);
        edges2.add("Alpha", "Delta", 9);
        edges2.add("Alpha", "Eta", 10);
        edges2.add("Delta", "Epsilon", 4);
        edges2.add("Beta", "Gamma", 7);
        edges2.add("Delta", "Theta", 7);
        edges2.add("Eta", "Lambda", 9);
        edges2.add("Theta", "Kappa", 3);
        edges2.add("Theta", "Iota", 4);
        edges2.add("Theta", "Mu", 4);
        edges2.add("Epsilon", "Zeta", 9);
        edges2.add("Kappa", "Nu", 7);
        edges2.add("Lambda", "Xi", 7);
        edges2.add("Mu", "Omicron", 8);
        edges2.add("Xi", "Sigma", 5);
        edges2.add("Nu", "Omega", 9);

        edges3 = new EdgeList();
        edges3.add("Mu", "Theta", 4);
        edges3.add("Theta", "Kappa", 3);
        edges3.add("Theta", "Iota", 4);
        edges3.add("Mu", "Lambda", 8);
        edges3.add("Mu", "Omicron", 8);
        edges3.add("Theta", "Epsilon", 5);
        edges3.add("Mu", "Xi", 10);
        edges3.add("Theta", "Delta", 7);
    }
}

```

```
edges3.add("Beta", "Gamma", 7);
edges3.add("Omicron", "Nu", 3);
edges3.add("Delta", "Eta", 2);
edges3.add("Delta", "Beta", 3);
edges3.add("Xi", "Sigma", 5);
edges3.add("Epsilon", "Zeta", 9);
edges3.add("Omicron", "Omega", 10);
edges3.add("Alpha", "Delta", 9);
}

public static void main(String[] args) {
    ShortestPathFinder.findShortestPaths("Theta", edges1);
    System.out.println();
    ShortestPathFinder.findShortestPaths("Alpha", edges2);
    System.out.println();
    ShortestPathFinder.findShortestPaths("Mu", edges3);
}
}
```

Script di atas