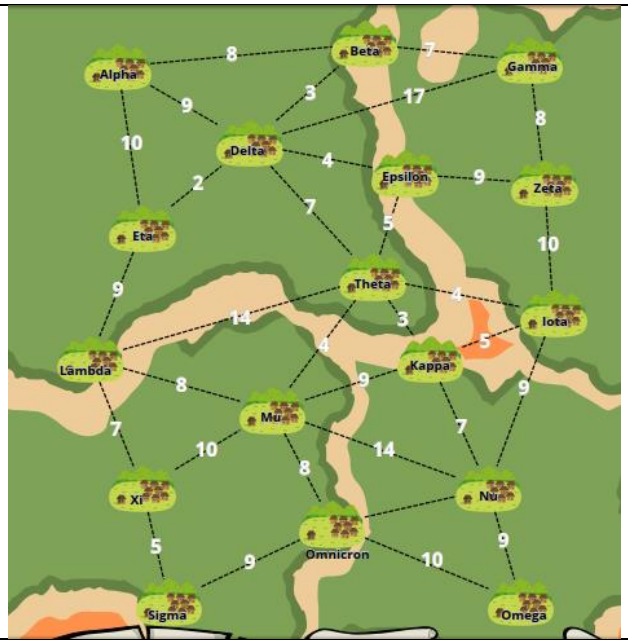
MODUL V

GRAPH

5.1 PERMASALAHAN

5.1.1 Pilkada Kabupaten Merembu



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 mengujungin 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 kempane di desa desa terdekatnya terlebih dahulu

1. Apabila rumah pak ophet di Theta

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

2. Apabila rumah pak ophet di Alpha

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

3. Apabila rumah pak ophet di Mu

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

Silahkan di coba untuk apabila pak ophet memiliki rumah di setiap desa kabupaten merembu

5.2 HASIL PERCOBAAN

5.2.1 Pilkada Kabupaten Merembu

1. Algoritma
    a. Mulai
    b. Siapkan graf untuk desa dan jalur.
    c. Tambahkan desa ke graf.
    d. Tambahkan jalur antar desa dengan jarak.
    e. Tentukan desa awal.
    f. Cari desa dengan jarak terkecil.
    g. Perbarui jarak ke desa tetangga.
    h. Ulangi hingga semua desa diperiksa.
    i. Tampilkan hasil jarak dan jalur terpendek.
    j. Ulangi untuk semua desa sebagai titik awal.
    k. Selesai
2. *Source Code*

```
public class Node {
    String desa;
    int weight;
    Node next;
    LinkedList edges;

    public Node (String desa, int weight) {
        this.desa = desa;
        this.weight = weight;
        this.next = null;
    }
}

public class Vertex {
    String name;
    LinkedList edges;

    Vertex(String name) {
        this.name = name;
        this.edges = new LinkedList();
    }
}

public class LinkedList {
    Node head;

    public Node getHead() {
        return head;
    }

    void add(String desa, int weight) {
        Node newNode = new Node(desa, weight);
        if (head == null) {
```

```
                head = newNode;
            } else {
                Node current = head;
                while (current.next != null) {
                    current = current.next;
                }
                current.next = newNode;
            }
        }
    }

    public class Graph {
        LinkedList vertices;
        int numVertices;


        Graph() {
            vertices = new LinkedList();
            numVertices = 0;
        }

        void addVertex(String name) {
            vertices.add(name, 0);
            numVertices++;
        }

        void addEdge(String source, String tujuan, int weight) {
            Node current = vertices.getHead();
            while (current != null) {
                if (current.desa.equals(source)) {
                    current.next = new Node(tujuan, weight);
                    break;
                }
                current = current.next;
            }
        }

        void dijkstra(String startVertex) {
            LinkedList visited = new LinkedList();
            LinkedList distances = new LinkedList();
            LinkedList paths = new LinkedList();

            Node current = vertices.getHead();
            while (current != null) {
                distances.add(current.desa, Integer.MAX_VALUE);
                paths.add(current.desa, 0);
                current = current.next;
            }

            distances.add(startVertex, 0);

            while (visited.getHead() == null) {
                String u = minDistance(distances, visited);
                visited.add(u, 0);

                Node edgeNode = getVertexEdges(u);
                while (edgeNode != null) {
                    String v = edgeNode.desa;
                    int weight = edgeNode.weight;
```

```
                if (!isVisited(v, visited) && getDistance(v,
distances) > getDistance(u, distances) + weight) {
                    setDistance(v, distances, getDistance(u,
distances) + weight);
                    setPath(v, paths, u);
                }
                edgeNode = edgeNode.next;
            }
        }

        printPaths(paths, startVertex);
    }

    String minDistance(LinkedList distances, LinkedList
visited) {
        Node current = distances.getHead();
        String minVertex = null;
        int minDistance = Integer.MAX_VALUE;

        while (current != null) {
            if (!isVisited(current.desa, visited) &&
getDistance(current.desa, distances) < minDistance) {
                minDistance = getDistance(current.desa,
distances);
                minVertex = current.desa;
            }
            current = current.next;
        }
        return minVertex;
    }

    Node getVertexEdges(String name) {
        Node current = vertices.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                return current.edges.getHead();
            }
            current = current.next;
        }
        return null;
    }

    boolean isVisited(String name, LinkedList visited) {
        Node current = visited.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                return true;
            }
            current = current.next;
        }
        return false;
    }

    int getDistance(String name, LinkedList distances) {
        Node current = distances.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                return current.weight;
            }
            current = current.next;
        }
```

```
            return Integer.MAX_VALUE;
        }

        void setDistance(String name, LinkedList distances, int
    distance) {
            Node current = distances.getHead();
            while (current != null) {
                if (current.desa.equals(name)) {
                    current.weight = distance;
                    break;
                }
                current = current.next;
            }
        }

        void setPath(String name, LinkedList paths, String path) {
            Node current = paths.getHead();
            while (current != null) {
                if (current.desa.equals(name)) {
                    current.desa = path;
                    break;
                }
                current = current.next;
            }
        }

        void printPaths(LinkedList paths, String startVertex) {
            System.out.println("Paths from " + startVertex + ":");
            Node current = paths.getHead();

            while (current != null) {
                if (!current.desa.equals(startVertex)) {
                    System.out.println(startVertex + " => " +
    current.desa);
                }
                current = current.next;
            }
        }

    }

    public class Main {
        // soal readme : Tambahkan dengan mencoba untuk masing
    masing desa sebagai titik mulai (Total 17)
        public static void main(String[] args) {
            Graph graph = new Graph();

            // Adding vertices
            String[] villages = {"Alpha", "Beta", "Gamma",
    "Delta", "Epsilon", "Zeta", "Eta", "Theta", "Iota", "Kappa",
    "Lambda", "Mu", "Nu", "Xi", "Omnicron", "Sigma", "Omega"};
            for (String village : villages) {
                graph.addVertex(village);
            }

            // Adding edges based on the provided connections
            graph.addEdge("Theta", "Kappa", 3);
            graph.addEdge("Theta", "Iota", 4);
            graph.addEdge("Theta", "Epsilon", 5);
            graph.addEdge("Theta", "Delta", 7);
            graph.addEdge("Theta", "Mu", 4);
```

```
        graph.addEdge("Theta", "Lambda", 14);
        graph.addEdge("Theta", "Xi", 7);

        graph.addEdge("Iota", "Zeta", 10);
        graph.addEdge("Iota", "Theta", 4);
        graph.addEdge("Iota", "Kappa", 5);
        graph.addEdge("Iota", "Nu", 9);

        graph.addEdge("Kappa", "Theta", 3);
        graph.addEdge("Kappa", "Iota", 5);
        graph.addEdge("Kappa", "Mu", 9);
        graph.addEdge("Kappa", "Nu", 7);

        graph.addEdge("Mu", "Theta", 14);
        graph.addEdge("Mu", "Eta", 9);
        graph.addEdge("Mu", "Xi", 7);

        graph.addEdge("Lambda", "Theta", 14);
        graph.addEdge("Lambda", "Epsilon", 5);
        graph.addEdge("Lambda", "Mu", 8);

        graph.addEdge("Nu", "Iota", 9);
        graph.addEdge("Nu", "Mu", 14);
        graph.addEdge("Nu", "Kappa", 7);
        graph.addEdge("Nu", "Omega", 9);

        graph.addEdge("Omnicron", "Sigma", 9);
        graph.addEdge("Omnicron", "Mu", 8);
        graph.addEdge("Omnicron", "Omega", 10);

        graph.addEdge("Sigma", "Xi", 5);
        graph.addEdge("Sigma", "Omnicron", 9);

        graph.addEdge("Omega", "Omnicron", 10);
        graph.addEdge("Omega", "Nu", 9);

        // Running Dijkstra's algorithm from Theta
        graph.dijkstra("Theta");
    }
}
```

5.3 ANALISIS DATA

5.3.1 Little Blue World

```
public class Node {
    String desa;
    int weight;
    Node next;
    LinkedList edges;

    public Node (String desa, int weight) {
        this.desa = desa;
        this.weight = weight;
        this.next = null;
    }
}
```

Script tersebut digunakan untuk merepresentasikan sebuah simpul (node) dalam struktur data LinkedList, yang berisi informasi tentang sebuah desa, bobot atau jarak dari desa tersebut (weight), simpul berikutnya dalam daftar (next), dan daftar jalur atau hubungan ke desa lain (edges). Konstruktor pada kelas ini digunakan untuk menginisialisasi node baru dengan nama desa dan bobot tertentu, sementara `next` diatur sebagai null, menandakan bahwa node ini belum terhubung ke node berikutnya saat dibuat.

```
public class Vertex {
    String name;
    LinkedList edges;

    Vertex(String name) {
        this.name = name;
        this.edges = new LinkedList();
    }
}
```

Script tersebut digunakan untuk merepresentasikan sebuah simpul utama (vertex) dalam graf, di mana setiap vertex memiliki nama unik (`name`) dan daftar jalur (edges) yang terhubung ke vertex lain. Konstruktor pada kelas ini digunakan untuk menginisialisasi vertex baru dengan nama tertentu, sekaligus membuat daftar kosong untuk menyimpan hubungan atau jalur yang akan ditambahkan nantinya.

```
public class LinkedList {
    Node head;

    public Node getHead() {
        return head;
    }

    void add(String desa, int weight) {
        Node newNode = new Node(desa, weight);
        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) {
```

```
                current = current.next;
            }
            current.next = newNode;
        }
    }
}
```

Script tersebut digunakan untuk membuat struktur data LinkedList, yang berfungsi menyimpan kumpulan node secara terurut. Kelas ini memiliki atribut head sebagai penanda node pertama dalam daftar. Metode getHead() digunakan untuk mengambil node pertama, sedangkan metode add() digunakan untuk menambahkan node baru ke dalam daftar. Jika daftar masih kosong, node baru akan menjadi node pertama (head). Jika tidak, metode ini akan menelusuri daftar hingga menemukan node terakhir, lalu menambahkan node baru di akhir daftar dengan menghubungkannya menggunakan next.

```
public class Graph {
    LinkedList vertices;
    int numVertices;


    Graph() {
        vertices = new LinkedList();
        numVertices = 0;
    }

    void addVertex(String name) {
        vertices.add(name, 0);
        numVertices++;
    }

    void addEdge(String source, String tujuan, int weight) {
        Node current = vertices.getHead();
        while (current != null) {
            if (current.desa.equals(source)) {
                current.next = new Node(tujuan, weight);
                break;
            }
            current = current.next;
        }
    }

    void dijkstra(String startVertex) {
        LinkedList visited = new LinkedList();
        LinkedList distances = new LinkedList();
        LinkedList paths = new LinkedList();

        Node current = vertices.getHead();
        while (current != null) {
            distances.add(current.desa, Integer.MAX_VALUE);
            paths.add(current.desa, 0);
            current = current.next;
        }

        distances.add(startVertex, 0);

        while (visited.getHead() == null) {
```

```
            String u = minDistance(distances, visited);
            visited.add(u, 0);

            Node edgeNode = getVertexEdges(u);
            while (edgeNode != null) {
                String v = edgeNode.desa;
                int weight = edgeNode.weight;

                if (!isVisited(v, visited) && getDistance(v, distances)
> getDistance(u, distances) + weight) {
                    setDistance(v, distances, getDistance(u, distances)
+ weight);
                    setPath(v, paths, u);
                }
                edgeNode = edgeNode.next;
            }
        }

        printPaths(paths, startVertex);
    }

    String minDistance(LinkedList distances, LinkedList visited) {
        Node current = distances.getHead();
        String minVertex = null;
        int minDistance = Integer.MAX_VALUE;

        while (current != null) {
            if        (!isVisited(current.desa,        visited)        &&
getDistance(current.desa, distances) < minDistance) {
                minDistance = getDistance(current.desa, distances);
                minVertex = current.desa;
            }
            current = current.next;
        }
        return minVertex;
    }

    Node getVertexEdges(String name) {
        Node current = vertices.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                return current.edges.getHead();
            }
            current = current.next;
        }
        return null;
    }

    boolean isVisited(String name, LinkedList visited) {
        Node current = visited.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                return true;
            }
            current = current.next;
        }
        return false;
    }

    int getDistance(String name, LinkedList distances) {
        Node current = distances.getHead();
```

```
        while (current != null) {
            if (current.desa.equals(name)) {
                return current.weight;
            }
            current = current.next;
        }
        return Integer.MAX_VALUE;
    }

    void setDistance(String name, LinkedList distances, int distance) {
        Node current = distances.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                current.weight = distance;
                break;
            }
            current = current.next;
        }
    }

    void setPath(String name, LinkedList paths, String path) {
        Node current = paths.getHead();
        while (current != null) {
            if (current.desa.equals(name)) {
                current.desa = path;
                break;
            }
            current = current.next;
        }
    }

    void printPaths(LinkedList paths, String startVertex) {
        System.out.println("Paths from " + startVertex + ":");
        Node current = paths.getHead();

        while (current != null) {
            if (!current.desa.equals(startVertex)) {
                System.out.println(startVertex + " => " + current.desa);
            }
            current = current.next;
        }
    }

}
```

Script di atas digunakan untuk mengimplementasikan struktur data Graph dengan menggunakan linked list untuk merepresentasikan simpul (vertices) dan sisi (edges). Kelas ini memiliki beberapa operasi, seperti menambahkan simpul dengan addVertex() dan menambahkan sisi antar simpul dengan addEdge(). Selain itu, terdapat implementasi algoritma Dijkstra melalui metode dijkstra() untuk mencari jalur terpendek dari simpul awal ke semua simpul lainnya. Fungsi tambahan seperti minDistance() digunakan untuk menemukan simpul dengan jarak terkecil yang belum dikunjungi, sementara metode seperti getDistance(), setDistance(), dan setPath() mengelola jarak dan jalur pada graf. Kelas ini juga memiliki fungsi printPaths() untuk mencetak jalur terpendek dari simpul awal ke tujuan.

```
public class Main {
    // soal readme : Tambahkan dengan mencoba untuk masing masing desa
sebagai titik mulai (Total 17)
    public static void main(String[] args) {
        Graph graph = new Graph();

        // Adding vertices
        String[] villages = {"Alpha", "Beta", "Gamma", "Delta",
"Epsilon", "Zeta", "Eta", "Theta", "Iota", "Kappa", "Lambda", "Mu", "Nu",
"Xi", "Omnicron", "Sigma", "Omega"};
        for (String village : villages) {
            graph.addVertex(village);
        }

        // Adding edges based on the provided connections
        graph.addEdge("Theta", "Kappa", 3);
        graph.addEdge("Theta", "Iota", 4);
        graph.addEdge("Theta", "Epsilon", 5);
        graph.addEdge("Theta", "Delta", 7);
        graph.addEdge("Theta", "Mu", 4);
        graph.addEdge("Theta", "Lambda", 14);
        graph.addEdge("Theta", "Xi", 7);

        graph.addEdge("Iota", "Zeta", 10);
        graph.addEdge("Iota", "Theta", 4);
        graph.addEdge("Iota", "Kappa", 5);
        graph.addEdge("Iota", "Nu", 9);

        graph.addEdge("Kappa", "Theta", 3);
        graph.addEdge("Kappa", "Iota", 5);
        graph.addEdge("Kappa", "Mu", 9);
        graph.addEdge("Kappa", "Nu", 7);

        graph.addEdge("Mu", "Theta", 14);
        graph.addEdge("Mu", "Eta", 9);
        graph.addEdge("Mu", "Xi", 7);

        graph.addEdge("Lambda", "Theta", 14);
        graph.addEdge("Lambda", "Epsilon", 5);
        graph.addEdge("Lambda", "Mu", 8);

        graph.addEdge("Nu", "Iota", 9);
        graph.addEdge("Nu", "Mu", 14);
        graph.addEdge("Nu", "Kappa", 7);
        graph.addEdge("Nu", "Omega", 9);

        graph.addEdge("Omnicron", "Sigma", 9);
        graph.addEdge("Omnicron", "Mu", 8);
        graph.addEdge("Omnicron", "Omega", 10);

        graph.addEdge("Sigma", "Xi", 5);
        graph.addEdge("Sigma", "Omnicron", 9);

        graph.addEdge("Omega", "Omnicron", 10);
        graph.addEdge("Omega", "Nu", 9);


        graph.dijkstra("Theta");
        graph.dijkstra("Alpha");
        graph.dijkstra("Mu");
        graph.dijkstra("Beta");
```

```
        graph.dijkstra("Gamma");
        graph.dijkstra("Delta");
        graph.dijkstra("Epsilon");
        graph.dijkstra("Zeta");
        graph.dijkstra("Eta");
        graph.dijkstra("Iota");
        graph.dijkstra("Kappa");
        graph.dijkstra("Lambda");
        graph.dijkstra("Nu");
        graph.dijkstra("Omega");
        graph.dijkstra("Xi");
        graph.dijkstra("Omnicron");
        graph.dijkstra("Sigma");

    }
}
```

Script tersebut digunakan untuk menguji implementasi graf dengan menerapkan algoritma Dijkstra pada berbagai titik awal di dalam graf. Pertama-tama membuat objek Graph, lalu menambahkan simpul (vertices) yang merepresentasikan desa-desa dengan nama unik, seperti “Alpha”, “Beta” hingga “Omega”. Setelah itu, hubungan antar desa (edges) beserta bobotnya ditambahkan ke dalam graf menggunakan metode addEdge(). Koneksi tersebut menggambarkan jarak antar desa. Kemudian, algoritma Dijkstra dipanggil untuk setiap desa sebagai titik awal, untuk menghitung dan mencetak jalur terpendek dari desa tersebut ke seluruh desa lainnya.