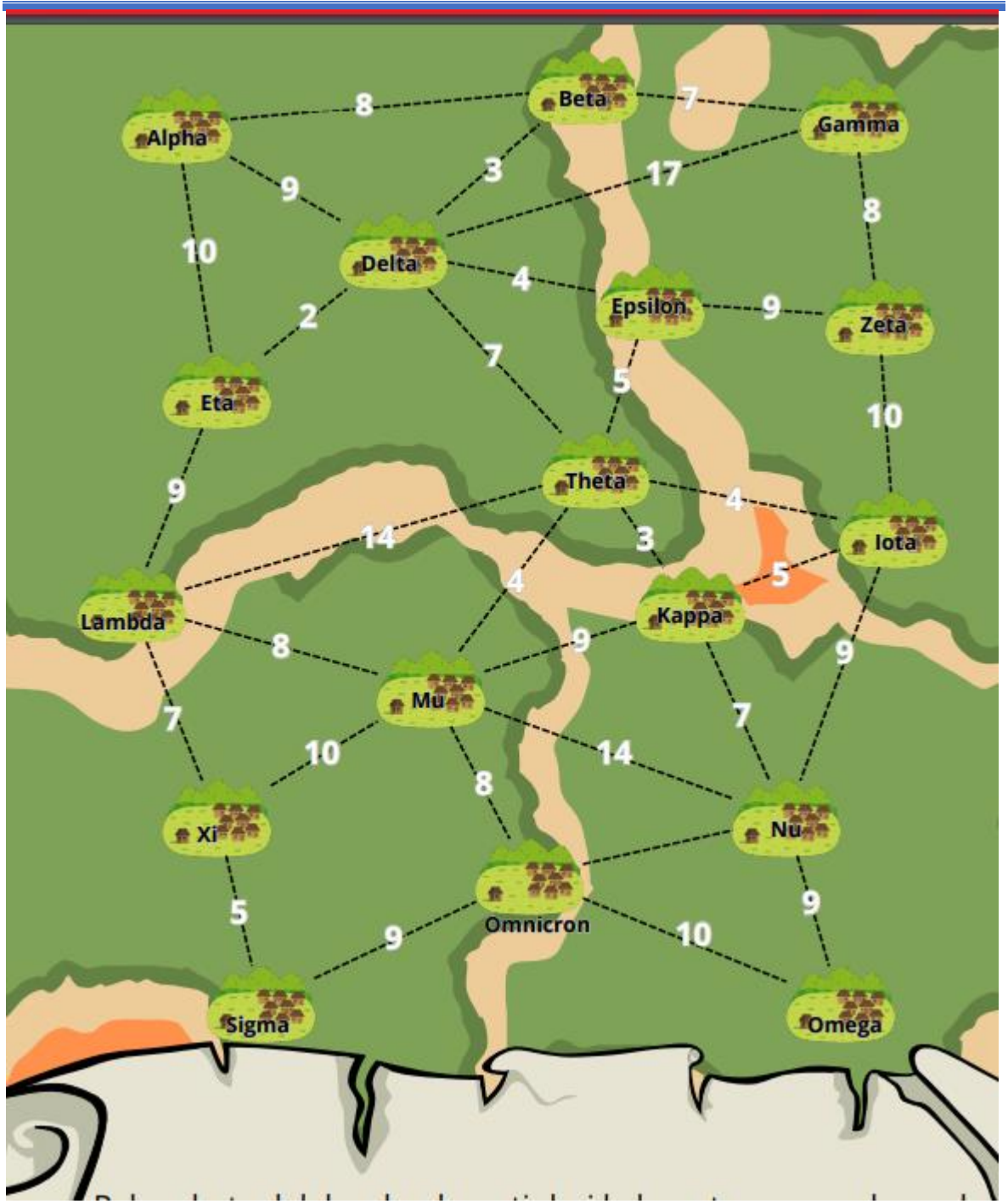
MODUL 5

GRAPH

4.1 PERMASALAHAN

4.3.1 Archon



Pak ophet adalah calon bupati dari kabupaten merembu, pak ophet berasal dari salah satu desa di kabupaten tersebut dalamrangka melakukan kampanye pak ophet menginginkan supaya bisa mengujungin semua desa dengan rute tercepat yang dimanaberarti 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 Theta Apabila rumah pak ophet di Theta Alpha Apabila rumah pak ophet di Alpha Mu Perlu di perhatikan bahwa juga untuk di urutkan desa terdekat terlebih dahulu karena pak ophet menginginkan untuk kampane di desa desa terdekatnya terlebih dahulu


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

4.2 HASIL PERCOBAAN

4.4.1 *Little Blue World*

1. Algoritma
    a. Mulai
    b. Membuat struktur *node* untuk menyimpan data setiap elemen dalam *tree*, di mana setiap *node* memiliki atribut nama, *power*, dan referensi ke *child* kiri dan *child* kanan.
    c. Membuat struktur *queue* untuk mendukung proses traversal *level-order*, dengan setiap elemen antrian yang menyimpan data *node* dan referensi ke elemen berikutnya dalam antrian.
    d. Membuat struktur *counter* untuk menghitung jumlah langkah yang diperlukan dalam proses pencarian dengan cara menyimpan nilai integer yang dapat diperbarui.
    e. Membangun *tree* dengan menambahkan *node* - *node* ke dalam struktur *tree* secara manual, dimulai dari root dan menambahkan *child* kiri dan *child* kanan.
    f. Progam menampilkan elemen-elemen *tree* menggunakan traversal *level-order* dengan kondisi jika *tree* kosong maka berhenti, jika tida maka data ditampilkan.
    g. Program mencari node berdasarkan nama menggunakan DFS (*Depth-First Search*) secara rekursif.
    h. Program mencari *node* berdasarkan *power* menggunakan BFS (*Breadth-First Search*).
    i. Program menampilkan struktur *tree* dan hasil pencarian sesuai dengan operasi yang telah dilakukan.
    j. Selesai.
2. *Source Code*

```
public class Graph {
    private SimpulGaris hubungannya;
    private static class SimpulGaris {
        String kunci;
        DaftarGaris nilai;
        SimpulGaris next;

        SimpulGaris(String kunci, DaftarGaris nilai, SimpulGaris next) {
            this.kunci = kunci;
            this.nilai = nilai;
            this.next = next;
        }
    }
    private static class DaftarGaris {
        NodeGaris head;

        void tambah(Garis data) {
```

```
            if (head == null) {
                head = new NodeGaris(data, null);
                return;
            }
            NodeGaris saatIni = head;
            while (saatIni.next != null) {
                saatIni = saatIni.next;
            }
            saatIni.next = new NodeGaris(data, null);
        }
    }

    private static class NodeGaris {
        Garis data;
        NodeGaris next;

        NodeGaris(Garis data, NodeGaris next) {
            this.data = data;
            this.next = next;
        }
    }

    private static class Garis {
        String tujuan;
        int bobot;

        Garis(String tujuan, int bobot) {
            this.tujuan = tujuan;
            this.bobot = bobot;
        }
    }

    private DaftarGaris dapatkanDaftarGaris(String kunci) {
        SimpulGaris saatIni = hubungannya;
        while (saatIni != null) {
            if (saatIni.kunci.equals(kunci)) {
                return saatIni.nilai;
            }
            saatIni = saatIni.next;
        }
        return null;
    }
    private static class EntriInt {
        String kunci;
        int nilai;
        EntriInt next;

        EntriInt(String kunci, int nilai, EntriInt next) {
            this.kunci = kunci;
            this.nilai = nilai;
            this.next = next;
        }
    }
    private static class EntriStr {
        String kunci;
        String nilai;
        EntriStr next;

        EntriStr(String kunci, String nilai, EntriStr next) {
```

```
            this.kunci = kunci;
            this.nilai = nilai;
            this.next = next;
        }
    }
    private static class SimpulAntrianPrioritas {
        String nama;
        int jarak;
        SimpulAntrianPrioritas next;

        SimpulAntrianPrioritas(String nama, int jarak,
SimpulAntrianPrioritas next) {
            this.nama = nama;
            this.jarak = jarak;
            this.next = next;
        }
    }
    private static class SimpulSet {
        String data;
        SimpulSet next;

        SimpulSet(String data, SimpulSet next) {
            this.data = data;
            this.next = next;
        }
    }
    private static class SimpulTumpukan {
        String data;
        SimpulTumpukan next;

        SimpulTumpukan(String data, SimpulTumpukan next) {
            this.data = data;
            this.next = next;
        }
    }
    private static class SimpulJalur {
        Jalur data;
        SimpulJalur next;

        SimpulJalur(Jalur data, SimpulJalur next) {
            this.data = data;
            this.next = next;
        }
    }

    private static class Jalur {
        String rute;
        int jarak;

        Jalur(String rute, int jarak) {
            this.rute = rute;
            this.jarak = jarak;
        }
    }

    private boolean berisiDalamSet(SimpulSet set, String data) {
        SimpulSet saatIni = set;
        while (saatIni != null) {
            if (saatIni.data.equals(data)) return true;
```

```
            saatIni = saatIni.next;
        }
        return false;
    }

    private int dapatkanJarak(EntriInt jarak, String kunci) {
        EntriInt saatIni = jarak;
        while (saatIni != null) {
            if (saatIni.kunci.equals(kunci)) return saatIni.nilai;
            saatIni = saatIni.next;
        }
        return Integer.MAX_VALUE;
    }

    private EntriInt masukkanJarak(EntriInt jarak, String kunci, int nilai)
{
        EntriInt saatIni = jarak;
        while (saatIni != null) {
            if (saatIni.kunci.equals(kunci)) {
                saatIni.nilai = nilai;
                return jarak;
            }
            saatIni = saatIni.next;
        }
        jarak = new EntriInt(kunci, nilai, jarak);
        return jarak;
    }

    private EntriStr masukkanSebelumnya(EntriStr sebelumnya, String kunci,
String nilai) {
        EntriStr saatIni = sebelumnya;
        while (saatIni != null) {
            if (saatIni.kunci.equals(kunci)) {
                saatIni.nilai = nilai;
                return sebelumnya;
            }
            saatIni = saatIni.next;
        }
        sebelumnya = new EntriStr(kunci, nilai, sebelumnya);
        return sebelumnya;
    }

    private SimpulAntrianPrioritas
tambahkanKeAntrianPrioritas(SimpulAntrianPrioritas antrian, String nama,
int jarak) {
        if (antrian == null || jarak < antrian.jarak) {
            return new SimpulAntrianPrioritas(nama, jarak, antrian);
        }
        SimpulAntrianPrioritas saatIni = antrian;
        while (saatIni.next != null && saatIni.next.jarak <= jarak) {
            saatIni = saatIni.next;
        }
        saatIni.next = new SimpulAntrianPrioritas(nama, jarak,
saatIni.next);
        return antrian;
    }

    private Jalur bangunJalur(String tujuan, EntriStr sebelumnya, String
mulai, int jarak) {
```

```
        SimpulTumpukan tumpukanJalur = null;
        String saatIni = tujuan;
        while (saatIni != null && !saatIni.equals(mulai)) {
            tumpukanJalur = new SimpulTumpukan(saatIni, tumpukanJalur);
            saatIni = dapatkanSebelumnya(sebelumnya, saatIni);
        }
        if (saatIni == null) return null;
        StringBuilder rute = new StringBuilder(mulai);
        while (tumpukanJalur != null) {
            rute.append(" => ").append(tumpukanJalur.data);
            tumpukanJalur = tumpukanJalur.next;
        }
        return new Jalur(rute.toString(), jarak);
    }

    private String dapatkanSebelumnya(EntriStr sebelumnya, String kunci) {
        EntriStr saatIni = sebelumnya;
        while (saatIni != null) {
            if (saatIni.kunci.equals(kunci)) return saatIni.nilai;
            saatIni = saatIni.next;
        }
        return null;
    }

    private SimpulJalur sisipkanJalurTerurut(SimpulJalur jalurTerurut,
Jalur jalurBaru) {
        if (jalurTerurut == null || jalurBaru.jarak <
jalurTerurut.data.jarak) {
            return new SimpulJalur(jalurBaru, jalurTerurut);
        }
        SimpulJalur saatIni = jalurTerurut;
        while (saatIni.next != null && saatIni.next.data.jarak <=
jalurBaru.jarak) {
            saatIni = saatIni.next;
        }
        saatIni.next = new SimpulJalur(jalurBaru, saatIni.next);
        return jalurTerurut;
    }

    public void tampilkanJalur(String mulai) {
        EntriInt jarak = null;
        EntriStr sebelumnya = null;
        SimpulAntrianPrioritas antrianPrioritas = null;
        SimpulSet dikunjungi = null;

        jarak = new EntriInt(mulai, 0, jarak);
        antrianPrioritas = new SimpulAntrianPrioritas(mulai, 0, null);

        while (antrianPrioritas != null) {
            SimpulAntrianPrioritas simpulSaatIni = antrianPrioritas;
            antrianPrioritas = antrianPrioritas.next;
            String namaSaatIni = simpulSaatIni.nama;

            if (berisiDalamSet(dikunjungi, namaSaatIni)) continue;
            dikunjungi = new SimpulSet(namaSaatIni, dikunjungi);

            DaftarGaris garis = dapatkanDaftarGaris(namaSaatIni);
            if (garis != null) {
                NodeGaris nodeGaris = garis.head;
```

```
                while (nodeGaris != null) {
                    Garis garisSaatIni = nodeGaris.data;
                    int jarakBaru = dapatkanJarak(jarak, namaSaatIni) +
garisSaatIni.bobot;
                    int jarakAda = dapatkanJarak(jarak,
garisSaatIni.tujuan);
                    if (jarakBaru < jarakAda) {
                        jarak = masukkanJarak(jarak, garisSaatIni.tujuan,
jarakBaru);
                        sebelumnya = masukkanSebelumnya(sebelumnya,
garisSaatIni.tujuan, namaSaatIni);
                        antrianPrioritas =
tambahkanKeAntrianPrioritas(antrianPrioritas, garisSaatIni.tujuan,
jarakBaru);
                    }
                    nodeGaris = nodeGaris.next;
                }
            }
        }

        SimpulJalur semuaJalur = null;
        EntriInt entriJarak = jarak;
        while (entriJarak != null) {
            String tujuan = entriJarak.kunci;
            if (!tujuan.equals(mulai)) {
                Jalur jalur = bangunJalur(tujuan, sebelumnya, mulai,
entriJarak.nilai);
                if (jalur != null) {
                    semuaJalur = new SimpulJalur(jalur, semuaJalur);
                }
            }
            entriJarak = entriJarak.next;
        }
        SimpulJalur jalurTerurut = null;
        SimpulJalur nodeJalur = semuaJalur;
        while (nodeJalur != null) {
            jalurTerurut = sisipkanJalurTerurut(jalurTerurut,
nodeJalur.data);
            nodeJalur = nodeJalur.next;
        }
        System.out.println(mulai);
        SimpulJalur simpulJalurSaatIni = jalurTerurut;
        while (simpulJalurSaatIni != null) {
            System.out.println(simpulJalurSaatIni.data.rute);
            simpulJalurSaatIni = simpulJalurSaatIni.next;
        }
        System.out.println();
    }

    public void tambahGaris(String sumber, String tujuan, int bobot) {
        DaftarGaris garis = dapatkanDaftarGaris(sumber);
        if (garis == null) {
            garis = new DaftarGaris();
            hubungannya = new SimpulGaris(sumber, garis, hubungannya);
        }
        garis.tambah(new Garis(tujuan, bobot));
    }
public class Main {
    public static void main(String[] args) {
```

```
Graph graf = new Graph();

graf.tambahGaris("Alpha", "Bela", 7);
graf.tambahGaris("Alpha", "Delta", 9);
graf.tambahGaris("Alpha", "Eta", 10);
graf.tambahGaris("Bela", "Alpha", 8);
graf.tambahGaris("Bela", "Gamma", 7);
graf.tambahGaris("Bela", "Delta", 3);
graf.tambahGaris("Gamma", "Bela", 7);
graf.tambahGaris("Gamma", "Delta", 17);
graf.tambahGaris("Gamma", "Zeta", 8);
graf.tambahGaris("Delta", "Alpha", 9);
graf.tambahGaris("Delta", "Bela", 3);
graf.tambahGaris("Delta", "Eta", 2);
graf.tambahGaris("Delta", "Theta", 7);
graf.tambahGaris("Delta", "Epsilon", 4);
graf.tambahGaris("Delta", "Gamma", 17);
graf.tambahGaris("Epsilon", "Delta", 4);
graf.tambahGaris("Epsilon", "Zeta", 9);
graf.tambahGaris("Epsilon", "Theta", 5);
graf.tambahGaris("Zeta", "Gamma", 8);
graf.tambahGaris("Zeta", "Epsilon", 9);
graf.tambahGaris("Zeta", "Iota", 10);
graf.tambahGaris("Eta", "Alpha", 10);
graf.tambahGaris("Eta", "Delta", 2);
graf.tambahGaris("Eta", "Lambda", 9);
graf.tambahGaris("Theta", "Delta", 7);
graf.tambahGaris("Theta", "Epsilon", 5);
graf.tambahGaris("Theta", "Kappa", 3);
graf.tambahGaris("Theta", "Mu", 4);
graf.tambahGaris("Theta", "Iota", 4);
graf.tambahGaris("Theta", "Lambda", 14);
graf.tambahGaris("Iota", "Zeta", 10);
graf.tambahGaris("Iota", "Theta", 4);
graf.tambahGaris("Iota", "Kappa", 5);
graf.tambahGaris("Iota", "Nu", 9);
graf.tambahGaris("Lambda", "Eta", 9);
graf.tambahGaris("Lambda", "Theta", 14);
graf.tambahGaris("Lambda", "Mu", 8);
graf.tambahGaris("Lambda", "Xi", 7);
graf.tambahGaris("Kappa", "Theta", 3);
graf.tambahGaris("Kappa", "Mu", 9);
graf.tambahGaris("Kappa", "Nu", 7);
graf.tambahGaris("Kappa", "Iota", 5);
graf.tambahGaris("Mu", "Theta", 4);
graf.tambahGaris("Mu", "Lambda", 8);
graf.tambahGaris("Mu", "Xi", 10);
graf.tambahGaris("Mu", "Omnicron", 8);
graf.tambahGaris("Mu", "Nu", 14);
graf.tambahGaris("Mu", "Kappa", 9);
graf.tambahGaris("Xi", "Lambda", 7);
graf.tambahGaris("Xi", "Mu", 10);
graf.tambahGaris("Xi", "Sigma", 5);
graf.tambahGaris("Nu", "Iota", 9);
graf.tambahGaris("Nu", "Kappa", 7);
graf.tambahGaris("Nu", "Mu", 14);
graf.tambahGaris("Nu", "Omega", 9);
graf.tambahGaris("Omnicron", "Omega", 10);
```

```
        graf.tambahGaris("Omnicron", "Mu", 8);
        graf.tambahGaris("Omnicron", "Sigma", 9);
        graf.tambahGaris("Sigma", "Xi", 5);
        graf.tambahGaris("Sigma", "Omnicron", 9);
        graf.tambahGaris("Omega", "Nu", 9);
        graf.tambahGaris("Omega", "Omnicron", 10);
        graf.tampilkanJalur("Alpha");
        graf.tampilkanJalur("Beta");
        graf.tampilkanJalur("Gamma");
        graf.tampilkanJalur("Delta");
        graf.tampilkanJalur("Epsilon");
        graf.tampilkanJalur("Zeta");
        graf.tampilkanJalur("Eta");
        graf.tampilkanJalur("Theta");
        graf.tampilkanJalur("Iota");
        graf.tampilkanJalur("Lambda");
        graf.tampilkanJalur("Kappa");
        graf.tampilkanJalur("Mu");
        graf.tampilkanJalur("Xi");
        graf.tampilkanJalur("Nu");
        graf.tampilkanJalur("Omnicron");
        graf.tampilkanJalur("Sigma");
        graf.tampilkanJalur("Omega");
    }
}


}
```

4.3 ANALISIS DATA

4.5.1 Rijal CUP

```
public class Graph {
    private SimpulGaris hubungannya;
    private static class SimpulGaris {
        String kunci;
        DaftarGaris nilai;
        SimpulGaris next;

        SimpulGaris(String kunci, DaftarGaris nilai, SimpulGaris next) {
            this.kunci = kunci;
            this.nilai = nilai;
            this.next = next;
        }
    }
```

Kode tersebut mendeklarasikan kelas Graph yang digunakan untuk merepresentasikan sebuah graf. Graf ini menggunakan struktur data yang lebih sederhana dengan dua komponen utama: simpul (node) dan hubungan antar simpul (edges). Setiap simpul dalam graf ini diwakili oleh kelas SimpulGaris, yang memiliki atribut kunci untuk menyimpan nama atau identifikasi simpul, nilai yang berfungsi untuk menyimpan daftar hubungan atau edge yang keluar dari simpul tersebut, dan next yang menunjuk ke simpul berikutnya dalam daftar tertaut.

```
    private static class DaftarGaris {
        NodeGaris head;

        void tambah(Garis data) {
            if (head == null) {
                head = new NodeGaris(data, null);
                return;
            }
```

Script di atas digunakan untuk membuat representasi struktur data node dalam antrean yang digunakan untuk menyimpan elemen-elemen dari tipe data "`Node`", kemudian class "`QueueNode`" terdiri dari dua atribut yaitu "`data`", yang menyimpan nilai atau referensi objek dari tipe "`Node`", dan "`next`" yang merupakan referensi ke node berikutnya dalam antrean.

```
    private static class NodeGaris {
        Garis data;
        NodeGaris next;

        NodeGaris(Garis data, NodeGaris next) {
            this.data = data;
            this.next = next;
        }
    }
}
```

```
private static class Garis {
        String tujuan;
        int bobot;

        Garis(String tujuan, int bobot) {
            this.tujuan = tujuan;
            this.bobot = bobot;
        }
    }

    private DaftarGaris dapatkanDaftarGaris(String kunci) {
        SimpulGaris saatIni = hubungannya;
        while (saatIni != null) {
            if (saatIni.kunci.equals(kunci)) {
                return saatIni.nilai;
            }
            saatIni = saatIni.next;
        }
        return null;
    }
```

Script di atas digunakan untuk melakukan deklarasi dari *class* "`Queue`", kemudian dalam *class* tersebut menggunakan implementasi *queue* berbasis *linked list*, yang digunakan untuk mengelola elemen-elemen dalam antrean dengan prinsip FIFO (*First In*, *First Out*). Kelas ini memiliki dua atribut yaitu "`front`" dan "`rear`", yang masing-masing menunjuk pada node pertama dan terakhir dalam antrean.

```
class Counter {
 int value;

 public Counter(int value) {
  this.value = value;
 }
}
```

Script di atas digunakan untuk melakukan deklarasi *class* "`Counter`" yang memiliki atribut berupa variabel "`value`" dengan tipe data *int*. *Class* ini dilengkapi dengan sebuah *constructor* yang menerima satu parameter, yaitu "`value`", untuk menginisialisasi atribut "`value`" pada saat objek dari *class* ini dibuat.

```
public class Main {
 public static void printLevelOrder(Node root) {
  if (root == null) return;

  Queue queue = new Queue();
  queue.enqueue(root);

  System.out.println("Menampilkan Elemen Tree Dengan Urutan Level-
Order:");
  while (queue.front != null) {
   Node current = queue.dequeue();

   String leftNama = (current.left != null) ? current.left.nama : "-";
   String rightNama = (current.right != null) ? current.right.nama : "-
";

   System.out.println("[ Name: " + current.nama + ", Power: " +
current.power + " ] => " +
      "Left: " + leftNama + ", Right: " + rightNama);

   if (current.left != null) queue.enqueue(current.left);
   if (current.right != null) queue.enqueue(current.right);
  }
```

```
 }
```

Script di atas digunakan untuk mendeklarasikan sebuah *class* bernama "`Main`" yang berisi metode "`printLevelOrder`" untuk menampilkan elemen-elemen dari sebuah struktur data *tree* dengan urutan *level-order traversal*. Metode ini menggunakan *queue* untuk menyimpan *node-node* yang akan diproses secara berurutan dari level atas ke bawah, mulai dari root. Jika node root bernilai *null*, metode langsung keluar tanpa melakukan apa-apa. Setiap node yang diproses akan dikeluarkan dari antrian dan dicetak dengan format yang menunjukkan atribut nama dan *power*, serta informasi tentang child node kiri dan kanan. Selanjutnya, jika *node* tersebut memiliki *child* kiri atau kanan, *child* tersebut akan dimasukkan ke dalam antrian untuk diproses pada iterasi berikutnya hingga semua *node* selesai ditelusuri.

```
public static boolean cariNama(Node root, String nama, Counter langkah)
{
  if (root == null) return false;
  langkah.value++;
  if (root.nama.equals(nama)) {
   System.out.println(nama + " ditemukan setelah " + langkah.value + "
langkah berdasarkan DFS");
   return true;
  }

  return cariNama(root.left, nama, langkah) || cariNama(root.right, nama,
langkah);
 }

 public static void cariNama(Node root, String nama) {
  System.out.println("Mencari " + nama + " Berdasarkan Nama Dengan DFS");
  Counter langkah = new Counter(0);
  if (!cariNama(root, nama, langkah)) {
   System.out.println(nama + " tidak ditemukan setelah " + langkah.value
+ " langkah berdasarkan DFS");
  }
 }

 public static void cariPower(Node root, int jumlahPower) {
  if (root == null) {
   System.out.println("Tree kosong, tidak ada elemen untuk dicari!");
   return;
  }

  Queue queue = new Queue();
  queue.enqueue(root);
  int langkah = 0;

  System.out.println("Mencari Hopper Berdasarkan Power Dengan BFS");
  while (queue.front != null) {
   Node current = queue.dequeue();
   langkah++;

   if (current.power == jumlahPower) {
```

```
    System.out.println("Hopper dengan power " + jumlahPower + " ditemukan
setelah " + langkah + " langkah berdasarkan BFS");
    return;
   }

   if (current.left != null) queue.enqueue(current.left);
   if (current.right != null) queue.enqueue(current.right);
  }
  System.out.println("Hopper dengan power " + jumlahPower + " tidak
ditemukan setelah " + langkah + " langkah berdasarkan BFS");
 }
```

Script di atas digunakan untuk mendeklarasikan fungsi yaitu "`cariNama()`" dan "`cariPower()`", fungsi "`cariNama()`" menggunakan pendekatan rekursif DFS untuk mencari *node* berdasarkan atribut nama. Dengan bantuan objek "`Counter`", fungsi ini dapat menghitung jumlah langkah yang diperlukan selama pencarian. Sementara itu, fungsi "`cariPower()`" menggunakan algoritma BFS untuk mencari *node* berdasarkan atribut *power*. Algoritma ini menggunakan *queue* untuk *traversal level-order*, menelusuri setiap *node* satu per satu dari atas ke bawah dan kiri ke kanan

```
public static void main(String[] args) {
  Node root = new Node("Tomas Wulfhart", 57);
  root.left = new Node("Marco Rosso", 24);
  root.right = new Node("Lucius", 30);
  root.left.left = new Node("Salazar Mateo Sanchez", 40);
  root.left.right = new Node("Mirabelle", 117);
  root.right.left = new Node("Ludwig van Noordijk", 302);
  root.right.right = new Node("Charles-Pierre Blanc", 16);
  root.left.left.left = new Node("Zaheer Ar-Ramluni", 9);
  root.left.left.right = new Node("Azhkun Salinci", 68);
  root.right.left.left = new Node("Iason Argos", 75);
  root.right.left.right = new Node("Igor Svarocic", 75);
  root.right.right.left = new Node("The Stag Ranger", 88);
  root.right.right.right = new Node("Blue Whiskey", 99);

  printLevelOrder(root);
  cariNama(root, "Azhkun Salinci");
  cariPower(root, 88);
 }
}
```

Script di atas adalah deklarasi dari *class* "`Main`" yang merupakan fungsi utama untuk mengatur masukkan dari pengguna, kemudian dalam *class* tersebut dilakukan penambahan data ke dalam *tree* yang posisinya diatur dengan menggunakan "`left`" dan "`right`".