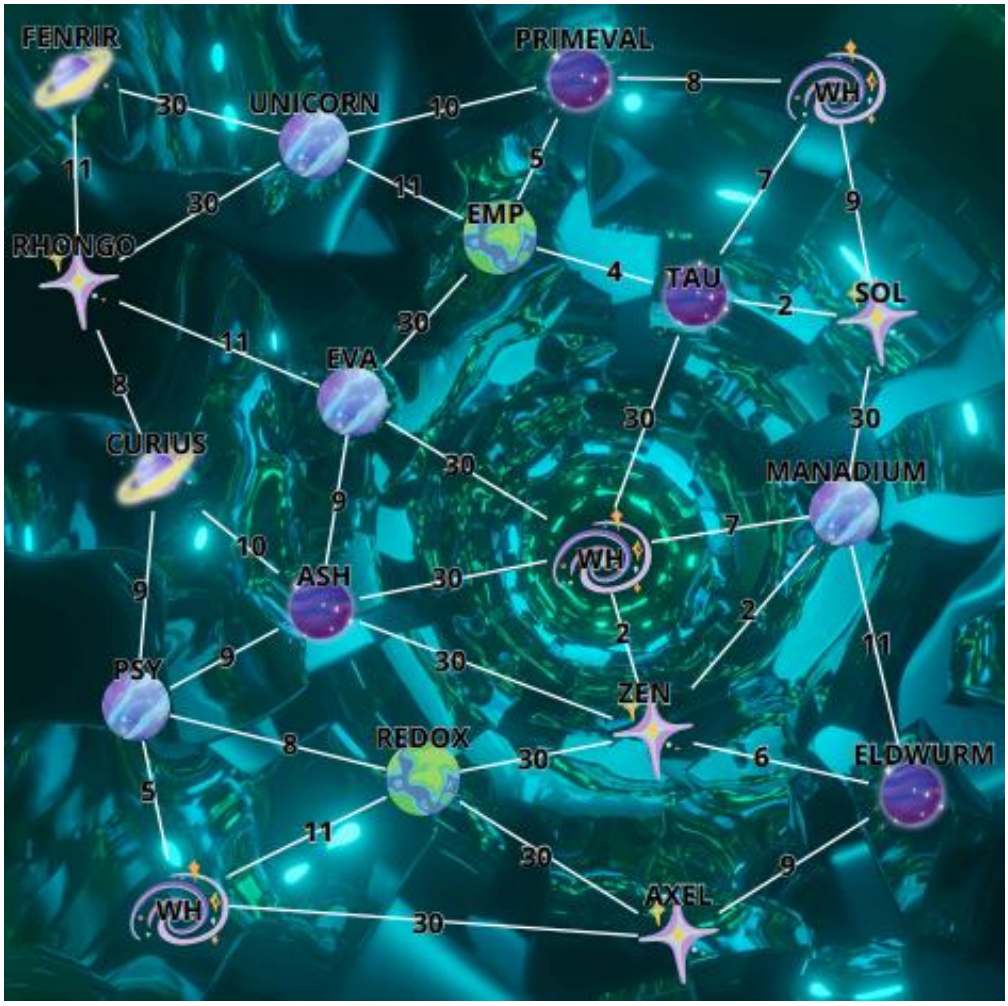
MODUL V

GRAPH

5.1 PERMASALAHAN

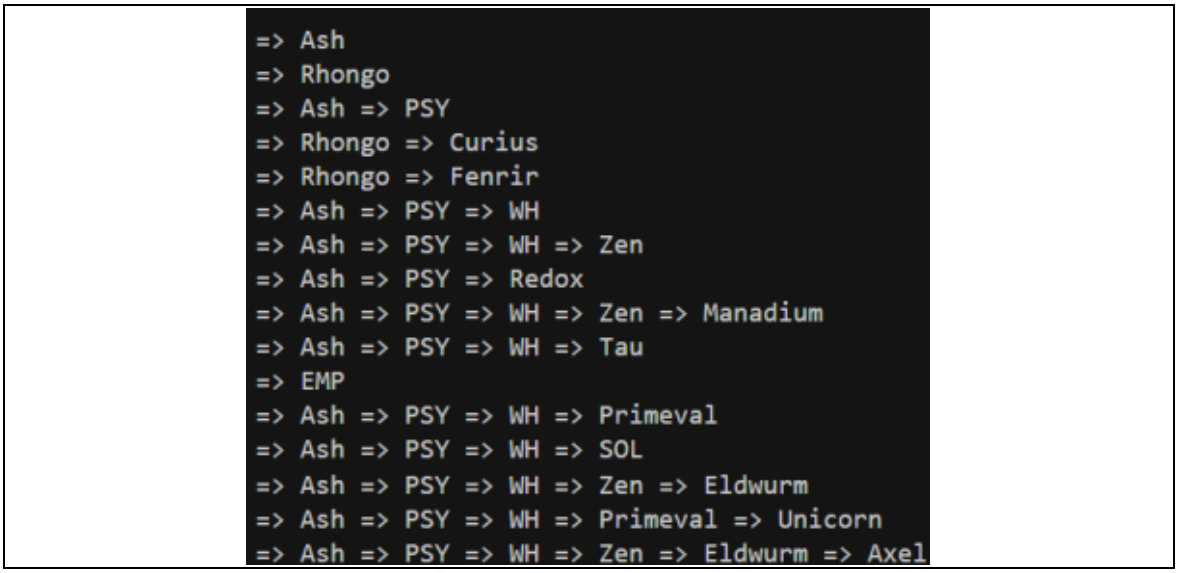
5.1.1 Verda Kalanta

Verda Kalanta adalah sesosok mahluk inter-stelar yang dimana memiliki kekuatan yang sangat dahsyat sehingga dengan mudahmenempuh perjalanan antar bintang. Disebuah galaxy verda sedang melakukan expedisi dalam rangkamengenali galaxy yang baru saja ia temukan, galaxy ini beberapahal unik dan baru pertama kali ia temukan

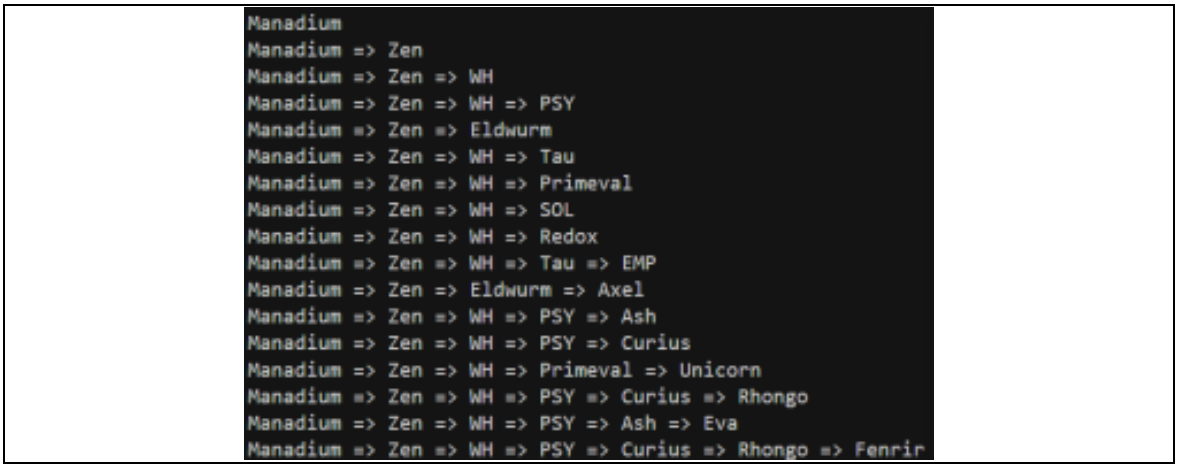


Gambar 5.1 Galaxy Verda Kalanta

Gambar di atas terdapat white hole, dimana white hole yang menyambung antara white hole lain sehingga bisa membuat verda langsung berpindah secara instan antar white hole lainnya hal ini dapat memperpenda waktu perjalanan verda dalam mengaruhi alam semesta Bantulah verda melakukan mapping terhadap galaxy baru, dia menginginkan sebuah list dari satu tempat ke tempat lainnya untuk semua tempat



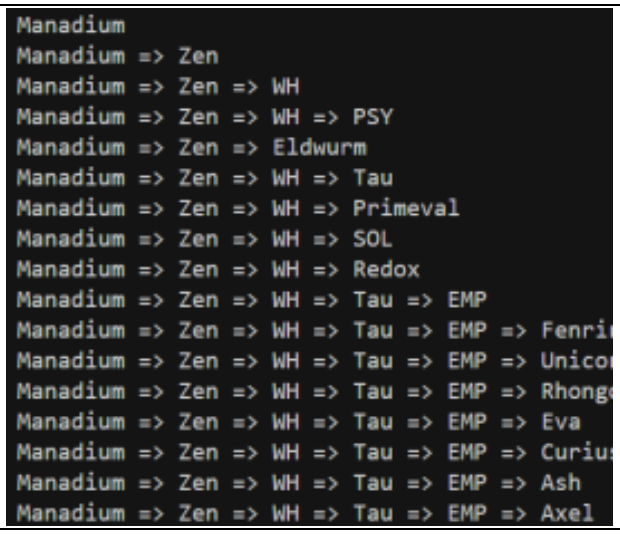
Gambar 5.2 Alur Planet Eva ke Semua Planet



Gambar 5.3 Alur Planet Manadium ke Semua Planet

Setelah berdiam beberapa juta tahun di galaxy ini verda menemukan sebuah trobosan baru yaitu sebuah teleportation device yang dia taruh di sebuah planet yang bernama fenrir, teleportation device ini memungkinkan fenrir untuk berpindah tempat di seluruh galxy secara instan, dengan teknologi baru ini bantukan kembali verda kalanta untuk melakukan mapping.

Gambar 5.4 Alur Planet Eva ke Semua Planet



Gambar 5.4 Alur Planet Manadium ke Semua Planet

5.2 HASIL PERCOBAAN

2.2.1 Program Antrian Pemusing Kepala

1. Algoritma
    a. Kelas Vertex
        i. *Start.*
        ii. Mendefinisikan nama, tetangga, dikunjungi, jarak, next, jalur.
        iii. Konstruksi Vertex dengan simpan nilai nama, next, tetangga, dikunjungi, jarak, dan jalur berdasarkan input dari parameter.
        iv. *Stop.*
    b. Kelas Tetangga
        i. *Start.*
        ii. Definisikan struktur data tetangga dengan atribut simpul, next dan bobot.
        iii. Inisialisasi tetangga dengan konstruktur, simpan referensi simpul tujuan ke atribut, simpan nilai bobot hubungan ke atribut lalu atur referensi next ke null.
        iv. Gunakan tetangga dalam struktur graf dengan tetangga akan menjadi bagian dari atribut dalam simpul.
        v. *Stop*.
    c. Kelas *Graph*
        i. *Start*.
        ii. Definisikan Vertex dengan nama simpul, daftar tetangga, jarak simpul awal, penanda simpul dan status serta Tetangga dengan simpul tujuan, bobot hubungan.
        iii. Inisialisasi Graf dengan membuat kelas Graph dengan simpul awal head disiapkan kosong.
        iv. Menambahkan simpul dengan membuat simpul baru berdasarkan nama dan tambahkan simpul ke akhir daftar simpul graf.
        v. Menambahkan hubungan pada fungsi addEgde dengan mencari simpul awal dan tujuan berdasarkan nama, jika tidak ditemukan, tambahkan hubungan tetangga dengan bobot tertentu.
        vi. Menelusuri dengan mengatur jarak simpul awal menjadi 0, ulangi proses memilih simpul dengan jarak terkecil yang belum di kunjungi lalu perbaharui jarak ke setiap tetangga jika lebih pendek, lalu tandai simpul sebagai telah dikunjungi. Ulangi proses hingga simpul tujuan ditemukan atau semua simpul selesai.

vii. Cetak hasil dengan jika jarak simpul tujuan masih sangat besar, tampilkan pesan bahwa jalur tidak ada. Jika ditemukan, cetak jalur dari simpul awal ke simpul tujuan.

viii. *Stop*

d. Kelas Main

i. *Start*

ii. Membuat sebuah objek baru dari kelas Graph sebagai graphEva.

iii. Membuat sebuah objek baru dari kelas Graph sebagai graphManadium.

iv. Melakukan input berbagai kebutuhan soal.

v. Print berbagai hasil yang ada berdasarkan planet yang digunakan sebagai awalan dan print yang menggunakan teleportation device.

vi. *Stop*

2. *Source Code*

a. Kelas Simpul

```
class Simpul {
    String nama;
    Simpul berikut;
    Tetangga kepalaTetangga;
    boolean dikunjungi;
    int jarak;
    Simpul jalur;

    public Simpul(String nama) {
        this.nama = nama;
        this.berikut = null;
        this.kepalaTetangga = null;
        this.dikunjungi = false;
        this.jarak = Integer.MAX_VALUE;
        this.jalur = null;
    }
}
```

b. Kelas Tetangga

```
class Tetangga {
    Simpul simpul;
    Tetangga berikut;
    int bobot;

    public Tetangga(Simpul simpul, int bobot) {
        this.simpul = simpul;
        this.bobot = bobot;
```

```
        this.berikut = null;
    }
}
```

c. Kelas Graph

```
class Graph {
    Simpul head;

    public Graph() {
        this.head = null;
    }

    void tambahSimpul(String nama) {
        Simpul simpulBaru = new Simpul(nama);
        if (head == null) {
            head = simpulBaru;
        } else {
            Simpul temp = head;
            while (temp.berikut != null) {
                temp = temp.berikut;
            }
            temp.berikut = simpulBaru;
        }
    }

    Simpul find(String nama) {
        Simpul temp = head;
        while (temp != null) {
            if (temp.nama.equals(nama)) {
                return temp;
            }
            temp = temp.berikut;
        }
        return null;
    }

    void tambahSisi(String host, String tujuan, int bobot) {
        Simpul simpulHost = find(host);
        Simpul simpulTujuan = find(tujuan);

        if (simpulHost != null && simpulTujuan != null) {
            Tetangga tetanggaBaru = new Tetangga(simpulTujuan,
bobot);
            if (simpulHost.kepalaTetangga == null) {
                simpulHost.kepalaTetangga = tetanggaBaru;
            } else {
                Tetangga temp = simpulHost.kepalaTetangga;
                while (temp.berikut != null) {
                    temp = temp.berikut;
                }
                temp.berikut = tetanggaBaru;
            }
        }
    }

    void Dijkstra(String host, String tujuan, boolean useFenrir,
boolean useEMP) {
        Simpul start = find(host);
        Simpul end = find(tujuan);
        Simpul fenrir = find("Fenrir");
        Simpul emp = find("EMP");
```

```
        if (start == null || end == null)
            return;

        resetVertices();
        start.jarak = 0;

        Simpul current = start;
        while (current != null) {
            Tetangga tetangga = current.kepalaTetangga;

            while (tetangga != null) {
                int totalBobot = current.jarak + tetangga.bobot;
                if (totalBobot < tetangga.simpul.jarak) {
                    tetangga.simpul.jarak = totalBobot;
                    tetangga.simpul.jalur = current;
                }
                tetangga = tetangga.berikut;
            }
            current.dikunjungi = true;
            current = findSimpulJarakkecil();
        }

        if (useFenrir && fenrir != null) {
            if (fenrir.jarak < end.jarak) {
                int jarakViaFenrir = fenrir.jarak +
getDistance(fenrir, end);
                if (jarakViaFenrir < end.jarak) {
                    end.jarak = jarakViaFenrir;
                    end.jalur = fenrir;
                }
            }
        }

        if (useEMP && emp != null) {
            if (emp.jarak < end.jarak) {
                int jarakViaEMP = emp.jarak + getDistance(emp,
end);
                if (jarakViaEMP < end.jarak) {
                    end.jarak = jarakViaEMP;
                    end.jalur = emp;
                }
            }
        }

        if (end.jarak == Integer.MAX_VALUE) {
            System.out.println("Tidak ada jalur dari " + host +
" ke " + tujuan);
        } else {
            printPath(end);
            System.out.println();
        }
    }

    int getDistance(Simpul from, Simpul to) {
        Tetangga tetangga = from.kepalaTetangga;
        while (tetangga != null) {
            if (tetangga.simpul == to) {
                return tetangga.bobot;
            }
            tetangga = tetangga.berikut;
```

```
        }
        return Integer.MAX_VALUE;
    }

    Simpul findSimpulJarakkecil() {
        Simpul temp = head;
        Simpul vertexEfisien = null;
        int jarakTerkecil = Integer.MAX_VALUE;

        while (temp != null) {
            if (!temp.dikunjungi && temp.jarak < jarakTerkecil)
{
                jarakTerkecil = temp.jarak;
                vertexEfisien = temp;
            }
            temp = temp.berikut;
        }
        return vertexEfisien;
    }

    void resetVertices() {
        Simpul temp = head;
        while (temp != null) {
            temp.jarak = Integer.MAX_VALUE;
            temp.dikunjungi = false;
            temp.jalur = null;
            temp = temp.berikut;
        }
    }

    void printPath(Simpul simpul) {
        if (simpul.jalur != null) {
            printPath(simpul.jalur);
            System.out.print(" => ");
        }
        System.out.print(simpul.nama);
    }
}
```

d. Kelas Main

```
public class Main {
    public static void main(String[] args) {
        Graph graphEva = new Graph();

        graphEva.tambahSimpul("Eva");
        graphEva.tambahSimpul("Ash");
        graphEva.tambahSimpul("Rhongo");
        graphEva.tambahSimpul("PSY");
        graphEva.tambahSimpul("Curius");
        graphEva.tambahSimpul("Fenrir");
        graphEva.tambahSimpul("WH");
        graphEva.tambahSimpul("Zen");
        graphEva.tambahSimpul("Redox");
        graphEva.tambahSimpul("Manadium");
        graphEva.tambahSimpul("Tau");
        graphEva.tambahSimpul("EMP");
        graphEva.tambahSimpul("Primeval");
        graphEva.tambahSimpul("Sol");
        graphEva.tambahSimpul("Eldwurm");
        graphEva.tambahSimpul("Unicorn");
        graphEva.tambahSimpul("Axel");
```

```
        graphEva.tambahSisi("Eva", "Ash", 9);
        graphEva.tambahSisi("Eva", "Rhongo", 11);
        graphEva.tambahSisi("Ash", "PSY", 9);
        graphEva.tambahSisi("Rhongo", "Curius", 8);
        graphEva.tambahSisi("Rhongo", "Fenrir",11);
        graphEva.tambahSisi("PSY", "WH", 5);
        graphEva.tambahSisi("WH", "Zen", 2);
        graphEva.tambahSisi("PSY", "Redox", 8);
        graphEva.tambahSisi("Zen", "Manadium", 2);
        graphEva.tambahSisi("WH", "Tau", 7);
        graphEva.tambahSisi("Eva", "EMP", 30);
        graphEva.tambahSisi("WH", "Primeval", 8);
        graphEva.tambahSisi("WH", "Sol", 9);
        graphEva.tambahSisi("Zen", "Eldwurm", 6);
        graphEva.tambahSisi("Primeval", "Unicorn", 10);
        graphEva.tambahSisi("Eldwurm", "Axel", 9);

        graphEva.Dijkstra("Eva", "Eva", false, false);
        graphEva.Dijkstra("Eva", "Ash", false, false);
        graphEva.Dijkstra("Eva", "Rhongo", false, false);
        graphEva.Dijkstra("Eva", "PSY", false, false);
        graphEva.Dijkstra("Eva", "Curius", false, false);
        graphEva.Dijkstra("Eva", "Fenrir", false, false);
        graphEva.Dijkstra("Eva", "WH", false, false);
        graphEva.Dijkstra("Eva", "Zen", false, false);
        graphEva.Dijkstra("Eva", "Redox", false, false);
        graphEva.Dijkstra("Eva", "Manadium", false, false);
        graphEva.Dijkstra("Eva", "Tau", false, false);
        graphEva.Dijkstra("Eva", "EMP", false, false);
        graphEva.Dijkstra("Eva", "Primeval", false, false);
        graphEva.Dijkstra("Eva", "Sol", false, false);
        graphEva.Dijkstra("Eva", "Eldwurm", false, false);
        graphEva.Dijkstra("Eva", "Unicorn", false, false);
        graphEva.Dijkstra("Eva", "Axel", false, false);


        System.out.println();

        Graph graphManadium = new Graph();

        graphManadium.tambahSimpul("Eva");
        graphManadium.tambahSimpul("Ash");
        graphManadium.tambahSimpul("Rhongo");
        graphManadium.tambahSimpul("PSY");
        graphManadium.tambahSimpul("Curius");
        graphManadium.tambahSimpul("Fenrir");
        graphManadium.tambahSimpul("WH");
        graphManadium.tambahSimpul("Zen");
        graphManadium.tambahSimpul("Redox");
        graphManadium.tambahSimpul("Manadium");
        graphManadium.tambahSimpul("Tau");
        graphManadium.tambahSimpul("EMP");
        graphManadium.tambahSimpul("Primeval");
        graphManadium.tambahSimpul("Sol");
        graphManadium.tambahSimpul("Eldwurm");
        graphManadium.tambahSimpul("Unicorn");
        graphManadium.tambahSimpul("Axel");

        graphManadium.tambahSisi("Manadium", "Zen", 2);
        graphManadium.tambahSisi("Zen", "WH", 2);
```

```
        graphManadium.tambahSisi("WH", "PSY", 5);
        graphManadium.tambahSisi("Zen", "Eldwurm", 6);
        graphManadium.tambahSisi("WH", "Tau", 7);
        graphManadium.tambahSisi("WH", "Primeval", 8);
        graphManadium.tambahSisi("WH", "Sol", 9);
        graphManadium.tambahSisi("WH", "Redox", 11);
        graphManadium.tambahSisi("Tau", "EMP", 4);
        graphManadium.tambahSisi("Eldwurm", "Axel", 9);
        graphManadium.tambahSisi("PSY", "Ash", 9);
        graphManadium.tambahSisi("PSY", "Curius", 9);
        graphManadium.tambahSisi("Primeval", "Unicorn", 10);
        graphManadium.tambahSisi("Curius", "Rhongo", 8);
        graphManadium.tambahSisi("Ash", "Eva", 9);
        graphManadium.tambahSisi("Rhongo", "Fenrir", 11);

        graphManadium.Dijkstra("Manadium", "Manadium", false,
false);
        graphManadium.Dijkstra("Manadium", "Zen", false, false);
        graphManadium.Dijkstra("Manadium", "WH", false, false);
        graphManadium.Dijkstra("Manadium", "PSY", false, false);
        graphManadium.Dijkstra("Manadium", "Eldwurm", false,
false);
        graphManadium.Dijkstra("Manadium", "Tau", false, false);
        graphManadium.Dijkstra("Manadium", "Primeval", false,
false);
        graphManadium.Dijkstra("Manadium", "Sol", false, false);
        graphManadium.Dijkstra("Manadium", "Redox", false,
false);
        graphManadium.Dijkstra("Manadium", "EMP", false, false);
        graphManadium.Dijkstra("Manadium", "Axel", false,
false);
        graphManadium.Dijkstra("Manadium", "Ash", false, false);
        graphManadium.Dijkstra("Manadium", "Curius", false,
false);
        graphManadium.Dijkstra("Manadium", "Unicorn", false,
false);
        graphManadium.Dijkstra("Manadium", "Rhongo", false,
false);
        graphManadium.Dijkstra("Manadium", "Eva", false, false);
        graphManadium.Dijkstra("Manadium", "Fenrir", false,
false);

        System.out.println("\nJalur setelah menggunakan
teleportation device");
        System.out.println("= = = = = = = = = = = = = = = = = = =
= = = = = =\n");

        graphEva.Dijkstra("Eva", "Eva", true, false);
        graphEva.Dijkstra("Eva", "Ash", true, false);
        graphEva.Dijkstra("Eva", "Rhongo", true, false);
        graphEva.Dijkstra("Eva", "PSY", true, false);
        graphEva.Dijkstra("Eva", "Curius", true, false);
        graphEva.Dijkstra("Eva", "Fenrir", true, false);
        graphEva.Dijkstra("Eva", "Unicorn", true, false);
        graphEva.Dijkstra("Eva", "Primeval", true, false);
        graphEva.Dijkstra("Eva", "EMP", true, false);
        graphEva.Dijkstra("Eva", "WH", true, false);
        graphEva.Dijkstra("Eva", "Tau", true, false);
        graphEva.Dijkstra("Eva", "Sol", true, false);
        graphEva.Dijkstra("Eva", "Manadium", true, false);
        graphEva.Dijkstra("Eva", "Eldwurm", true, false);
```

```
        graphEva.Dijkstra("Eva", "Zen", true, false);
        graphEva.Dijkstra("Eva", "Redox", true, false);
        graphEva.Dijkstra("Eva", "Axel", true, false);

        System.out.println();

        graphManadium.Dijkstra("Manadium", "Manadium", false,
true);
        graphManadium.Dijkstra("Manadium", "Zen", false, true);
        graphManadium.Dijkstra("Manadium", "WH", false, true);
        graphManadium.Dijkstra("Manadium", "PSY", false, true);
        graphManadium.Dijkstra("Manadium", "Eldwurm", false,
true);
        graphManadium.Dijkstra("Manadium", "Tau", false, true);
        graphManadium.Dijkstra("Manadium", "Primeval", false,
true);
        graphManadium.Dijkstra("Manadium", "Sol", false, true);
        graphManadium.Dijkstra("Manadium", "Redox", false,
true);
        graphManadium.Dijkstra("Manadium", "EMP", false, true);
        graphManadium.Dijkstra("Manadium", "Fenrir", false,
true);
        graphManadium.Dijkstra("Manadium", "Unicorn", false,
true);
        graphManadium.Dijkstra("Manadium", "Rhongo", false,
true);
        graphManadium.Dijkstra("Manadium", "Eva", false, true);
        graphManadium.Dijkstra("Manadium", "Curius", false,
true);
        graphManadium.Dijkstra("Manadium", "Ash", false, true);
        graphManadium.Dijkstra("Manadium", "Axel", false, true);
    }
}
```

3. Hasil Program

```
Eva
Eva => Ash
Eva => Rhongo
Eva => Ash => PSY
Eva => Rhongo => Curius
Eva => Rhongo => Fenrir
Eva => Ash => PSY => WH
Eva => Ash => PSY => WH => Zen
Eva => Ash => PSY => Redox
Eva => Ash => PSY => WH => Zen => Manadium
Eva => Ash => PSY => WH => Tau
Eva => EMP
Eva => Ash => PSY => WH => Primeval
Eva => Ash => PSY => WH => SOL
Eva => Ash => PSY => WH => Zen => Eldwurm
Eva => Ash => PSY => WH => Primeval => Unicorn
Eva => Ash => PSY => WH => Zen => Eldwurm => Axel
```

Gambar 5.5 Hasil *Run* Planet Eva ke Semua Planet

Berdasarkan **Gambar 5.5**, Dapat diketahui alur perjalanan dari planet Eva menuju planet lainnya tanpa device teleport berhasil dengan banyak planet yang menjadi tempat kunjungan, seperti Eva menuju Ash, Eva ke Rhongo dan lain-lainnya.

```
Manadium
Manadium => Zen
Manadium => Zen => WH
Manadium => Zen => WH => PSY
Manadium => Zen => Eldwurm
Manadium => Zen => WH => Tau
Manadium => Zen => WH => Primeval
Manadium => Zen => WH => SOL
Manadium => Zen => WH => Redox
Manadium => Zen => WH => Tau => EMP
Manadium => Zen => WH => PSY => Curius => Rhongo => Fenrir
Manadium => Zen => WH => Primeval => Unicorn
Manadium => Zen => WH => PSY => Curius => Rhongo
Manadium => Zen => WH => PSY => Ash => Eva
Manadium => Zen => WH => PSY => Curius
Manadium => Zen => WH => PSY => Ash
Manadium => Zen => Eldwurm => Axel
```

Gambar 5.6 Hasil *Run* Planet Manadium ke Semua Planet dengan Teleport ke EMP

Berdasarkan **Gambar 5.6**, dapat diketahui alur perjalanan dari planet Manadium mennuju planet lainnya tanpa device teleport berhasil dengan banyak planet yang menjadi tempat kunjungan, seperti Manadium menuju Zen, Manadium ke PSY melalui Zen dan WH serta lain-lainnya.

```
Eva
Eva => Ash
Eva => Rhongo
Eva => Ash => PSY
Eva => Rhongo => Curius
Eva => Rhongo => Fenrir
Eva => Rhongo => Fenrir => Unicorn
Eva => Rhongo => Fenrir => Primeval
Eva => Rhongo => Fenrir => EMP
Eva => Rhongo => Fenrir => WH
Eva => Rhongo => Fenrir => Tau
Eva => Rhongo => Fenrir => SOL
Eva => Rhongo => Fenrir => Manadium
Eva => Rhongo => Fenrir => Eldwurm
Eva => Rhongo => Fenrir => Zen
Eva => Rhongo => Fenrir => Redox
Eva => Rhongo => Fenrir => Axel
```

Gambar 5.7 Hasil *Run* Planet Manadium ke Semua Planet dengan Teleport ke EMP

Berdasarkan **Gambar 5.7**, dapat diketahui alur perjalanan dari planet Manadium mennuju planet lainnya dengan device teleport Fenrir berhasil dengan banyak planet yang menjadi tempat kunjungan, seperti Manadium menuju Zen, Manadium ke PSY melalui hanya Ash. Hal ini dapat dilakukan dengan memberikan edge ke salah satu planet ke planet tujuan dengan bobot seringan mungkin, sehingga jalur akan diprioritaskan.

```
Manadium
Manadium => Zen
Manadium => Zen => WH
Manadium => Zen => WH => PSY
Manadium => Zen => Eldwurm
Manadium => Zen => WH => Tau
Manadium => Zen => WH => Primeval
Manadium => Zen => WH => SOL
Manadium => Zen => WH => Redox
Manadium => Zen => WH => Tau => EMP
Manadium => Zen => WH => Tau => EMP => Fenrir
Manadium => Zen => WH => Tau => EMP => Unicorn
Manadium => Zen => WH => Tau => EMP => Rhongo
Manadium => Zen => WH => Tau => EMP => Eva
Manadium => Zen => WH => Tau => EMP => Curius
Manadium => Zen => WH => Tau => EMP => Ash
Manadium => Zen => WH => Tau => EMP => Axel
```

Gambar 5.8 Hasil *Run* Planet Manadium ke Semua Planet dengan Teleport ke EMP

Berdasarkan **Gambar 5.8**, dapat diketahui alur perjalanan dari planet Manadium mennuju planet lainnya dengan device teleport pindah ke EMP berhasil dengan banyak planet yang menjadi tempat kunjungan, seperti Manadium menuju Zen, Manadium ke PSY melalui hanya Zen dan WH.

5.3 ANALISIS DATA

5.3.1 Verda Kalanta

```
class Tetangga {
    Simpul simpul;
    Tetangga berikut;
    int bobot;

    public Tetangga(Simpul simpul, int bobot) {
        this.simpul = simpul;
        this.bobot = bobot;
        this.berikut = null;
    }
}
```

Script di atas merupakan deklarasi variabel dan objek yang disebut sebagai atribut, seperti simpul, bobot, dan berikut. Atribut-atribut ini dapat diakses dalam kelas yang sama atau melalui metode yang tersedia, jika diperlukan di luar kelas. Atribut simpul merepresentasikan objek dari kelas Simpul, sedangkan bobot adalah nilai *integer* yang menunjukkan hubungan berbobot antara simpul-simpul. Sementara itu, atribut berikut adalah referensi ke objek lain dari kelas Tetangga, yang memungkinkan pembentukan struktur data berbasis *node*, seperti l*inked list*. Dengan atribut ini, kelas Tetangga dapat digunakan untuk memodelkan jaringan simpul dengan relasi berbobot secara terstruktur.

```
class Simpul {
    String nama;
    Simpul berikut;
    Tetangga kepalaTetangga;
    boolean dikunjungi;
    int jarak;
    Simpul jalur;

    public Simpul(String nama) {
        this.nama = nama;
        this.berikut = null;
        this.kepalaTetangga = null;
        this.dikunjungi = false;
        this.jarak = Integer.MAX_VALUE;
        this.jalur = null;
    }
}
```

Script di atas kumpulan deklarasi variabel dan objek yang disebut sebagai atribut, seperti nama, tetangga, dikunjungi, jarak, berikut, dan jalur. Atribut-atribut ini dapat diakses dalam kelas yang sama atau melalui metode yang tersedia untuk penggunaan di luar kelas.

Atribut nama menyimpan nama simpul sebagai representasi unik, sementara tetangga adalah referensi ke objek dari kelas Tetangga yang menunjukkan daftar tetangga dari simpul ini. Atribut dikunjungi adalah penanda boolean untuk melacak apakah simpul telah dikunjungi dalam suatu proses, seperti pencarian graf. Atribut jarak digunakan untuk menyimpan jarak terpendek dari simpul awal ke simpul ini dalam algoritma pencarian jarak terpendek, dengan nilai awal diatur ke *Integer*.MAX_VALUE. Atribut next mereferensikan simpul berikutnya, mendukung struktur data berbasis *node* seperti *linked list*, sedangkan jalur menyimpan referensi ke simpul sebelumnya dalam jalur terpendek. Dengan atribut-atribut ini, kelas Simpul dapat digunakan untuk memodelkan simpul dalam struktur *graf*.

```
class Graph {
    Simpul head;

    public Graph() {
        this.head = null;
    }

    void tambahSimpul(String nama) {
        Simpul simpulBaru = new Simpul(nama);
        if (head == null) {
            head = simpulBaru;
        } else {
            Simpul temp = head;
            while (temp.berikut != null) {
                temp = temp.berikut;
            }
            temp.berikut = simpulBaru;
        }
    }
    Simpul find(String nama) {
        Simpul temp = head;
        while (temp != null) {
            if (temp.nama.equals(nama)) {
                return temp;
            }
            temp = temp.berikut;
        }
        return null;
    }
```

Script di atas deklarasi kelas *Graph* yang memiliki *atribut head*, yang merepresentasikan simpul pertama (*head node*) dalam struktur *graf* berbasis simpul. Atribut ini dapat diakses dalam kelas yang sama atau melalui metode terkait untuk pengelolaan *graf*. Atribut *head* diinisialisasi dengan nilai *null* dalam konstruktor *Graph*, menunjukkan bahwa *graf* awalnya kosong tanpa simpul apa pun. Dengan pendekatan ini, kelas *Graph* dapat digunakan untuk membangun dan mengelola struktur *graf* dinamis, di mana simpul-simpulnya dapat ditambahkan secara bertahap melalui referensi yang dimulai dari *head*. Hal ini mempermudah implementasi berbagai operasi pada *graf*, seperti pencarian simpul, penambahan hubungan antar simpul, atau penelusuran *graf*. *Script* "tambahSimpul" bertugas untuk menambahkan simpul baru ke dalam *graf*. Metode ini menerima parameter nama, yang akan digunakan untuk membuat objek simpul baru dari kelas *Vertex*. Jika atribut *head* bernilai *null*, artinya *graf* masih kosong, maka simpul baru langsung dijadikan sebagai *head*. Namun, jika *graf* sudah memiliki simpul, metode akan menggunakan sebuah variabel sementara "(temp)" untuk menelusuri simpul-simpul dalam *graf* hingga menemukan simpul terakhir (simpul dengan atribut next bernilai null). Setelah simpul terakhir ditemukan, simpul baru dihubungkan sebagai simpul berikutnya dengan cara mengatur "temp.next" menjadi simpul baru. Dengan mekanisme ini, metode "tambahSimpul" memastikan simpul baru selalu ditambahkan di akhir *graf*, membangun struktur berbasis *linked list* untuk menyimpan semua simpul dalam *graf*.

```
void tambahSisi(String host, String tujuan, int bobot) {
        Simpul simpulHost = find(host);
        Simpul simpulTujuan = find(tujuan);

        if (simpulHost != null && simpulTujuan != null) {
            Tetangga tetanggaBaru = new Tetangga(simpulTujuan, bobot);
            if (simpulHost.kepalaTetangga == null) {
                simpulHost.kepalaTetangga = tetanggaBaru;
            } else {
                Tetangga temp = simpulHost.kepalaTetangga;
                while (temp.berikut != null) {
                    temp = temp.berikut;
                }
                temp.berikut = tetanggaBaru;
            }
        }
    }
```

Script “tambahSisi” digunakan untuk menambahkan sebuah sisi “(edge)” yang menghubungkan dua simpul dalam graf dengan bobot tertentu. Metode ini menerima tiga parameter: “host” sebagai nama simpul asal, tujuan sebagai nama simpul tujuan, dan bobot sebagai nilai yang merepresentasikan berat sisi tersebut. Pertama, metode menggunakan fungsi “find” untuk mencari simpul asal “(simpulHost)” dan simpul tujuan (simpulTujuan) berdasarkan nama yang diberikan. Jika kedua simpul ditemukan, sebuah objek Tetangga baru dibuat untuk merepresentasikan hubungan antara simpul asal dan simpul tujuan dengan bobot tertentu. Jika simpul asal belum memiliki tetangga, tetangga baru langsung ditambahkan sebagai tetangga pertama. Namun, jika sudah ada daftar tetangga, metode akan menelusuri daftar tersebut hingga akhir, kemudian menambahkan tetangga baru di posisi terakhir. Dengan cara ini, metode “tambahSisi” memastikan relasi berbobot antara simpul-simpul dapat dikelola secara dinamis dalam graf berbasis *linked list*.

```
void Dijkstra(String host, String tujuan, boolean useFenrir, boolean
useEMP) {
        Simpul start = find(host);
        Simpul end = find(tujuan);
        Simpul fenrir = find("Fenrir");
        Simpul emp = find("EMP");

        if (start == null || end == null)
            return;

        resetVertices();
        start.jarak = 0;

        Simpul current = start;
        while (current != null) {
            Tetangga tetangga = current.kepalaTetangga;

            while (tetangga != null) {
                int totalBobot = current.jarak + tetangga.bobot;
                if (totalBobot < tetangga.simpul.jarak) {
                    tetangga.simpul.jarak = totalBobot;
                    tetangga.simpul.jalur = current;
                }
                tetangga = tetangga.berikut;
            }
            current.dikunjungi = true;
```

```
            current = findSimpulJarakkecil();
        }

        if (useFenrir && fenrir != null) {
            if (fenrir.jarak < end.jarak) {
                int jarakViaFenrir = fenrir.jarak + getDistance(fenrir,
end);
                if (jarakViaFenrir < end.jarak) {
                    end.jarak = jarakViaFenrir;
                    end.jalur = fenrir;
                }
            }
        }

        if (useEMP && emp != null) {
            if (emp.jarak < end.jarak) {
                int jarakViaEMP = emp.jarak + getDistance(emp, end);
                if (jarakViaEMP < end.jarak) {
                    end.jarak = jarakViaEMP;
                    end.jalur = emp;
                }
            }
        }

        if (end.jarak == Integer.MAX_VALUE) {
            System.out.println("Tidak ada jalur dari " + host + " ke " +
tujuan);
        } else {
            printPath(end);
            System.out.println();
        }
    }
```

Script “Dijkstra” merupakan implementasi algoritma Dijkstra yang digunakan untuk mencari jalur terpendek antara dua simpul dalam *graf*, dengan opsi tambahan yang melibatkan simpul khusus “Fenrir” dan EMP. Metode ini menerima parameter *host* (simpul awal), tujuan (simpul akhir), serta dua bendera boolean “useFenrir” dan “useEMP” untuk mengaktifkan fitur tambahan. Pertama, metode mencari simpul awal “(start)”, simpul akhir “(end)”, serta simpul “Fenrir” dan EMP menggunakan metode “find”. Jika salah satu dari

simpul awal atau akhir tidak ditemukan, proses dihentikan. Metode kemudian menginisialisasi ulang graf menggunakan "ResetVertex" dan mengatur jarak simpul awal menjadi 0. Dalam *loop* utama, metode memproses simpul saat ini "(current)" dan mengevaluasi semua tetangganya. Untuk setiap tetangga, metode menghitung total bobot jalur dan memperbarui jarak serta jalur terpendek jika total bobot lebih kecil dari nilai jarak yang ada. Setelah semua tetangga diproses, simpul saat ini ditandai sebagai telah dikunjungi, dan metode memilih simpul berikutnya dengan jarak terkecil menggunakan "findSimpulJarakkecil". Proses ini terus berlanjut hingga semua simpul selesai diproses, memungkinkan pencarian jalur terpendek yang efisien. *Script* di atas juga bertugas untuk menampilkan hasil jalur terpendek yang ditemukan oleh algoritma "Dijkstra". Pertama, kode memeriksa apakah atribut jarak dari simpul tujuan "(end)" masih memiliki nilai "Integer.MAX_VALUE", yang berarti tidak ada jalur yang menghubungkan simpul awal (*host*) ke simpul tujuan (tujuan). Jika kondisi ini terpenuhi, pesan "Tidak ada jalur dari [host] ke [tujuan]" akan dicetak ke konsol. Namun, jika jalur ditemukan, metode "printPath" akan dipanggil untuk mencetak jalur terpendek mulai dari simpul awal hingga simpul tujuan. Selain itu, terdapat opsi untuk mencetak jarak total jalur dengan menampilkan nilai atribut jarak dari simpul tujuan, meskipun bagian ini dikomentari. Dengan cara ini, kode memastikan bahwa hasil pencarian jalur terpendek dapat disampaikan dengan jelas kepada pengguna, baik saat jalur ditemukan maupun tidak.

```
int getDistance(Simpul from, Simpul to) {
        Tetangga tetangga = from.kepalaTetangga;
        while (tetangga != null) {
            if (tetangga.simpul == to) {
                return tetangga.bobot;
            }
            tetangga = tetangga.berikut;
        }
        return Integer.MAX_VALUE;
    }
```

Script "GetDistance" digunakan untuk mendapatkan bobot (jarak) dari sisi yang menghubungkan dua simpul tertentu dalam *graf*. Metode ini menerima dua parameter: "from", yaitu simpul asal, dan to, yaitu simpul tujuan. Proses dimulai dengan mengambil daftar tetangga dari simpul asal melalui atribut "tetangga". Kemudian, menggunakan "loop", metode memeriksa setiap tetangga dari simpul asal. Jika ditemukan tetangga yang menunjuk ke simpul tujuan (to), metode langsung mengembalikan nilai atribut bobot dari sisi tersebut.

Jika tidak ada tetangga yang cocok hingga akhir daftar, metode mengembalikan nilai "Integer.MAX_VALUE", yang menunjukkan bahwa tidak ada sisi langsung antara simpul asal dan tujuan. Dengan pendekatan ini, metode "GetDistance" memastikan efisiensi dalam menentukan hubungan berbobot antara dua simpul tertentu dalam *graf*.

```
Simpul findSimpulJarakkecil() {
        Simpul temp = head;
        Simpul vertexEfisien = null;
        int jarakTerkecil = Integer.MAX_VALUE;

        while (temp != null) {
            if (!temp.dikunjungi && temp.jarak < jarakTerkecil) {
                jarakTerkecil = temp.jarak;
                vertexEfisien = temp;
            }
            temp = temp.berikut;
        }
        return vertexEfisien;
    }
```

Script "findSimpulJarakkecil" digunakan untuk menemukan simpul dalam *graf* yang memiliki jarak terkecil dari simpul awal dan belum dikunjungi. Metode ini memulai pencarian dengan menginisialisasi variabel temp untuk menunjuk ke simpul pertama ("head") dan dua variabel tambahan: "vertexEfisien", yang akan menyimpan simpul dengan jarak terkecil, serta "jarakTerkecil", yang diinisialisasi dengan nilai "Integer.MAX_VALUE" sebagai pembanding awal. Dalam *loop*, metode memeriksa setiap simpul dalam *graf*. Jika simpul belum dikunjungi "(!temp.dikunjungi)" dan jaraknya lebih kecil daripada nilai jarakTerkecil saat ini, nilai "jarakTerkecil" diperbarui dengan jarak simpul tersebut, dan "vertexEfisien" diset ke simpul tersebut. Setelah semua simpul diperiksa, metode mengembalikan "vertexEfisien", yang merupakan simpul dengan jarak terkecil dan belum diproses. Pendekatan ini mendukung efisiensi algoritma "Dijkstra" dalam menentukan simpul berikutnya untuk diproses.

```
void resetVertices() {
        Simpul temp = head;
        while (temp != null) {
            temp.jarak = Integer.MAX_VALUE;
            temp.dikunjungi = false;
            temp.jalur = null;
```

```
            temp = temp.berikut;
        }
    }
```

Script "ResetVertices" berfungsi untuk mengatur ulang atribut-atribut pada semua simpul dalam *graf* ke kondisi awal, terutama sebelum memulai algoritma seperti "Dijkstra". Metode ini memanfaatkan variabel sementara "temp" yang mulai menunjuk ke simpul pertama "(head)". Dalam loop, setiap simpul pada graf diakses satu per satu. Atribut jarak diatur ke "Integer.MAX_VALUE" untuk merepresentasikan bahwa jarak dari simpul awal belum dihitung. Atribut dikunjungi diatur ke "false", menandakan bahwa simpul belum diproses. Atribut jalur juga diatur ke "null", menghapus referensi ke simpul sebelumnya dalam jalur terpendek. Setelah semua simpul diproses, variabel temp akan menunjuk ke "null", menandakan akhir "graf". Dengan cara ini, metode "ResetVertices" memastikan semua simpul berada dalam keadaan siap untuk proses perhitungan atau traversal baru.

```
void printPath(Simpul simpul) {
        if (simpul.jalur != null) {
            printPath(simpul.jalur);
            System.out.print(" => ");
        }
        System.out.print(simpul.nama);
    }
}
```

Script "printPath" digunakan untuk mencetak jalur dari simpul awal hingga simpul tertentu dalam *graf*, yang telah dihitung sebelumnya menggunakan algoritma seperti "Dijkstra". Metode ini bekerja secara rekursif dengan memeriksa atribut jalur dari simpul yang diberikan sebagai parameter simpul. Jika atribut jalur tidak bernilai "null", metode akan memanggil dirinya sendiri dengan simpul sebelumnya (simpul.jalur), sehingga menavigasi jalur secara berurutan hingga simpul awal. Setelah mencapai simpul awal, metode mulai mencetak nama simpul-simpul di jalur tersebut, memisahkan setiap simpul dengan tanda " => " untuk menunjukkan hubungan antar simpul. Dengan pendekatan ini, metode "printPath" memastikan jalur terpendek dicetak dalam urutan yang benar, dari simpul awal ke simpul tujuan.

```
public class Main {
    public static void main(String[] args) {
        Graph graphEva = new Graph();
```

```
        graphEva.tambahSimpul("Eva");
        graphEva.tambahSimpul("Ash");
        graphEva.tambahSimpul("Rhongo");
        graphEva.tambahSimpul("PSY");
        graphEva.tambahSimpul("Curius");
        graphEva.tambahSimpul("Fenrir");
        graphEva.tambahSimpul("WH");
        graphEva.tambahSimpul("Zen");
        graphEva.tambahSimpul("Redox");
        graphEva.tambahSimpul("Manadium");
        graphEva.tambahSimpul("Tau");
        graphEva.tambahSimpul("EMP");
        graphEva.tambahSimpul("Primeval");
        graphEva.tambahSimpul("Sol");
        graphEva.tambahSimpul("Eldwurm");
        graphEva.tambahSimpul("Unicorn");
        graphEva.tambahSimpul("Axel");

        graphEva.tambahSisi("Eva", "Ash", 9);
        graphEva.tambahSisi("Eva", "Rhongo", 11);
        graphEva.tambahSisi("Ash", "PSY", 9);
        graphEva.tambahSisi("Rhongo", "Curius", 8);
        graphEva.tambahSisi("Rhongo", "Fenrir",11);
        graphEva.tambahSisi("PSY", "WH", 5);
        graphEva.tambahSisi("WH", "Zen", 2);
        graphEva.tambahSisi("PSY", "Redox", 8);
        graphEva.tambahSisi("Zen", "Manadium", 2);
        graphEva.tambahSisi("WH", "Tau", 7);
        graphEva.tambahSisi("Eva", "EMP", 30);
        graphEva.tambahSisi("WH", "Primeval", 8);
        graphEva.tambahSisi("WH", "Sol", 9);
        graphEva.tambahSisi("Zen", "Eldwurm", 6);
        graphEva.tambahSisi("Primeval", "Unicorn", 10);
        graphEva.tambahSisi("Eldwurm", "Axel", 9);

        graphEva.Dijkstra("Eva", "Eva", false, false);
        graphEva.Dijkstra("Eva", "Ash", false, false);
        graphEva.Dijkstra("Eva", "Rhongo", false, false);
        graphEva.Dijkstra("Eva", "PSY", false, false);
        graphEva.Dijkstra("Eva", "Curius", false, false);
        graphEva.Dijkstra("Eva", "Fenrir", false, false);
```

```
        graphEva.Dijkstra("Eva", "WH", false, false);
        graphEva.Dijkstra("Eva", "Zen", false, false);
        graphEva.Dijkstra("Eva", "Redox", false, false);
        graphEva.Dijkstra("Eva", "Manadium", false, false);
        graphEva.Dijkstra("Eva", "Tau", false, false);
        graphEva.Dijkstra("Eva", "EMP", false, false);
        graphEva.Dijkstra("Eva", "Primeval", false, false);
        graphEva.Dijkstra("Eva", "Sol", false, false);
        graphEva.Dijkstra("Eva", "Eldwurm", false, false);
        graphEva.Dijkstra("Eva", "Unicorn", false, false);
        graphEva.Dijkstra("Eva", "Axel", false, false);


        System.out.println();

        Graph graphManadium = new Graph();

        graphManadium.tambahSimpul("Eva");
        graphManadium.tambahSimpul("Ash");
        graphManadium.tambahSimpul("Rhongo");
        graphManadium.tambahSimpul("PSY");
        graphManadium.tambahSimpul("Curius");
        graphManadium.tambahSimpul("Fenrir");
        graphManadium.tambahSimpul("WH");
        graphManadium.tambahSimpul("Zen");
        graphManadium.tambahSimpul("Redox");
        graphManadium.tambahSimpul("Manadium");
        graphManadium.tambahSimpul("Tau");
        graphManadium.tambahSimpul("EMP");
        graphManadium.tambahSimpul("Primeval");
        graphManadium.tambahSimpul("Sol");
        graphManadium.tambahSimpul("Eldwurm");
        graphManadium.tambahSimpul("Unicorn");
        graphManadium.tambahSimpul("Axel");

        graphManadium.tambahSisi("Manadium", "Zen", 2);
        graphManadium.tambahSisi("Zen", "WH", 2);
        graphManadium.tambahSisi("WH", "PSY", 5);
        graphManadium.tambahSisi("Zen", "Eldwurm", 6);
        graphManadium.tambahSisi("WH", "Tau", 7);
        graphManadium.tambahSisi("WH", "Primeval", 8);
```

```
        graphManadium.tambahSisi("WH", "Sol", 9);
        graphManadium.tambahSisi("WH", "Redox", 11);
        graphManadium.tambahSisi("Tau", "EMP", 4);
        graphManadium.tambahSisi("Eldwurm", "Axel", 9);
        graphManadium.tambahSisi("PSY", "Ash", 9);
        graphManadium.tambahSisi("PSY", "Curius", 9);
        graphManadium.tambahSisi("Primeval", "Unicorn", 10);
        graphManadium.tambahSisi("Curius", "Rhongo", 8);
        graphManadium.tambahSisi("Ash", "Eva", 9);
        graphManadium.tambahSisi("Rhongo", "Fenrir", 11);

        graphManadium.Dijkstra("Manadium", "Manadium", false, false);
        graphManadium.Dijkstra("Manadium", "Zen", false, false);
        graphManadium.Dijkstra("Manadium", "WH", false, false);
        graphManadium.Dijkstra("Manadium", "PSY", false, false);
        graphManadium.Dijkstra("Manadium", "Eldwurm", false, false);
        graphManadium.Dijkstra("Manadium", "Tau", false, false);
        graphManadium.Dijkstra("Manadium", "Primeval", false, false);
        graphManadium.Dijkstra("Manadium", "Sol", false, false);
        graphManadium.Dijkstra("Manadium", "Redox", false, false);
        graphManadium.Dijkstra("Manadium", "EMP", false, false);
        graphManadium.Dijkstra("Manadium", "Axel", false, false);
        graphManadium.Dijkstra("Manadium", "Ash", false, false);
        graphManadium.Dijkstra("Manadium", "Curius", false, false);
        graphManadium.Dijkstra("Manadium", "Unicorn", false, false);
        graphManadium.Dijkstra("Manadium", "Rhongo", false, false);
        graphManadium.Dijkstra("Manadium", "Eva", false, false);
        graphManadium.Dijkstra("Manadium", "Fenrir", false, false);

         System.out.println("\nJalur setelah menggunakan teleportation
device");
        System.out.println("= = = = = = = = = = = = = = = = = = = = = = =
= =\n");

        graphEva.Dijkstra("Eva", "Eva", true, false);
        graphEva.Dijkstra("Eva", "Ash", true, false);
        graphEva.Dijkstra("Eva", "Rhongo", true, false);
        graphEva.Dijkstra("Eva", "PSY", true, false);
        graphEva.Dijkstra("Eva", "Curius", true, false);
        graphEva.Dijkstra("Eva", "Fenrir", true, false);
        graphEva.Dijkstra("Eva", "Unicorn", true, false);
```

```
        graphEva.Dijkstra("Eva", "Primeval", true, false);
        graphEva.Dijkstra("Eva", "EMP", true, false);
        graphEva.Dijkstra("Eva", "WH", true, false);
        graphEva.Dijkstra("Eva", "Tau", true, false);
        graphEva.Dijkstra("Eva", "Sol", true, false);
        graphEva.Dijkstra("Eva", "Manadium", true, false);
        graphEva.Dijkstra("Eva", "Eldwurm", true, false);
        graphEva.Dijkstra("Eva", "Zen", true, false);
        graphEva.Dijkstra("Eva", "Redox", true, false);
        graphEva.Dijkstra("Eva", "Axel", true, false);

        System.out.println();

        graphManadium.Dijkstra("Manadium", "Manadium", false, true);
        graphManadium.Dijkstra("Manadium", "Zen", false, true);
        graphManadium.Dijkstra("Manadium", "WH", false, true);
        graphManadium.Dijkstra("Manadium", "PSY", false, true);
        graphManadium.Dijkstra("Manadium", "Eldwurm", false, true);
        graphManadium.Dijkstra("Manadium", "Tau", false, true);
        graphManadium.Dijkstra("Manadium", "Primeval", false, true);
        graphManadium.Dijkstra("Manadium", "Sol", false, true);
        graphManadium.Dijkstra("Manadium", "Redox", false, true);
        graphManadium.Dijkstra("Manadium", "EMP", false, true);
        graphManadium.Dijkstra("Manadium", "Fenrir", false, true);
        graphManadium.Dijkstra("Manadium", "Unicorn", false, true);
        graphManadium.Dijkstra("Manadium", "Rhongo", false, true);
        graphManadium.Dijkstra("Manadium", "Eva", false, true);
        graphManadium.Dijkstra("Manadium", "Curius", false, true);
        graphManadium.Dijkstra("Manadium", "Ash", false, true);
        graphManadium.Dijkstra("Manadium", "Axel", false, true);
    }
}
```

Script implementasi simulasi jalur pada graf yang merepresentasikan konektivitas antara beberapa planet atau simpul. Pada metode main dari kelas Main, dua graf dibangun menggunakan kelas Graph, yaitu graphEva dan graphManadium. Graf ini memiliki simpul (vertex) dengan nama-nama planet seperti “Eva”, “Ash”, “Rhongo”, dan lain-lain, serta tepi (edge) dengan bobot tertentu yang menunjukkan jarak antar simpul.

Graf pertama (graphEva) memetakan jalur dari simpul awal “Eva” ke planet lainnya, sedangkan graf kedua (graphManadium) memetakan jalur dari simpul “Manadium”. Pada

bagian akhir, metode Dijkstra digunakan untuk menghitung dan mencetak jalur terpendek menggunakan algoritma Dijkstra dengan beberapa skenario. Skenario ini mencakup pencarian jalur tanpa menggunakan perangkat teleportasi, menggunakan perangkat teleportasi, atau dengan berpindah teleportasi ke simpul tertentu. Setiap panggilan metode Dijkstra mencetak jalur terpendek dari simpul awal ke simpul tujuan. Kode ini mengilustrasikan fleksibilitas algoritma Dijkstra dalam mencari jalur terpendek dengan variasi kondisi.