

RESPONSI
IMPLEMENTASI ALGORITMA KRUSKAL DALAM MENENTUKAN
TOPOLOGI JARINGAN ANTAR GEDUNG DI FAKULTAS TEKNIK
UNIVERSITAS PALANGKA RAYA



NAMA : JORDI IRAWAN
NIM : 223010503002
KELAS : E

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS PALANGKARAYA
2023

BAB I

PENDAHULUAN

1.1. Latar Belakang

Seiring dengan perkembangan teknologi dan kebutuhan akan akses internet yang semakin meningkat, jaringan komputer yang baik dan handal sangat penting untuk mendukung kegiatan akademik dan administratif di kampus. Dalam hal ini, topologi jaringan antar gedung di Fakultas Teknik Universitas Palangka Raya perlu dipikirkan dengan matang agar jaringan dapat beroperasi secara efisien dan aman.

Algoritma Kruskal dipilih sebagai algoritma yang digunakan dalam menentukan topologi jaringan antar gedung karena algoritma ini sangat efektif dalam mencari MST (Minimum Spanning Tree) pada grafik yang terhubung. MST ini akan menjadi topologi jaringan antar gedung yang optimal, sehingga jaringan dapat beroperasi secara optimal dan efisien. Oleh karena itu, implementasi algoritma Kruskal sangat penting dalam menentukan topologi jaringan antar gedung di Fakultas Teknik Universitas Palangka Raya.

1.2. Teori

Algoritma Kruskal adalah salah satu algoritma greedy yang digunakan untuk mencari MST (*Minimum Spanning Tree*) pada grafik yang terhubung. MST sendiri merupakan subgrafik dari grafik yang terhubung dan tidak memiliki siklus, serta menghubungkan semua simpul dengan biaya minimum. Dalam hal ini, simpul dapat diartikan sebagai gedung atau titik konektivitas di dalam gedung yang akan dihubungkan.

Berikut ini adalah langkah-langkah yang dilakukan dalam algoritma Kruskal untuk menyelesaikan masalah menentukan topologi jaringan antar gedung di Fakultas Teknik Universitas Palangka Raya:

1. Mengurutkan semua edge pada grafik berdasarkan bobotnya, dari yang terkecil hingga yang terbesar.

2. Pilih edge dengan bobot terkecil dan periksa apakah pengambilan edge tersebut akan membentuk siklus. Jika tidak membentuk siklus, edge tersebut ditambahkan ke MST.
3. Lakukan langkah kedua hingga semua simpul terhubung dengan MST.
4. MST yang dihasilkan akan menjadi topologi jaringan antar gedung yang optimal.

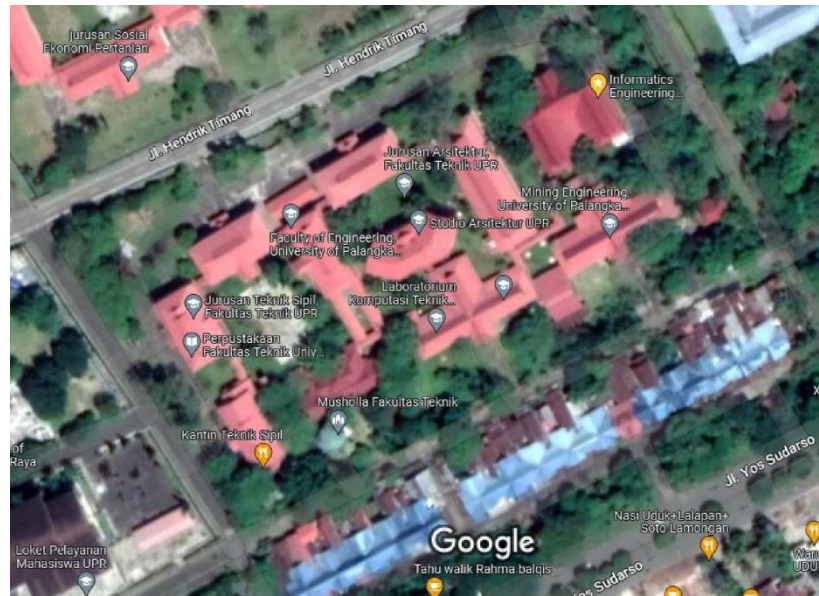
Dalam konteks menentukan topologi jaringan antar gedung di Fakultas Teknik Universitas Palangka Raya, algoritma Kruskal digunakan untuk menghubungkan gedung-gedung yang ada dengan biaya minimum. Dengan menggunakan algoritma Kruskal, akan didapatkan topologi jaringan antar gedung yang optimal dan efisien, sehingga jaringan dapat beroperasi dengan baik dan dapat diakses dari semua gedung yang terhubung.

BAB II

PEMBAHASAN

2.1. Menentukan Lokasi atau Peta

Berikut ini peta Fakultas Teknik, Universitas Palangka Raya yang dihasilkan oleh *Google Maps*



Gambar 2.1. Peta

Dari peta tersebut nantinya bisa dibuat sebagai informasi atau untuk membuat referensi graf

2.2. Membuat Data

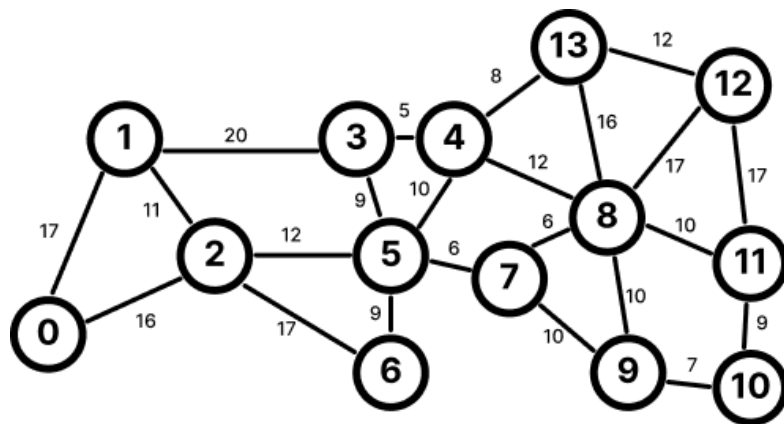
Berikut ini data tabel informasi gedung yang didapat dari **Gambar 2.1.**

0	Teknik Informatika
1	Teknik Pertambangan
2	Ruang Kuliah & Audit
3	Lab. Mekanika Tanah Teknik Sipil
4	Lab. Komputasi Teknik Sipil
5	Studio Arsitektur
6	Teknik Arsitektur
7	HIMA

8	Kantor Fakultas
9	Lapangan Basket
10	Teknik Sipil
11	Perpustakaan Fakultas Teknik
12	Kantin Teknik Sipil
13	Mushola

2.3. Membuat Graf

Graf yang akan dibuat adalah graf yang tidak berarah yang memiliki 14 vertex dan 23 edge. Berikut ini ilustrasi dari graf tersebut.



Gambar 2.2. Graf

2.4. Membuat Program

Dari informasi yang sudah dibuat, maka saatnya untuk membuat kode program untuk menentukan *Minimum Spanning Tree* dari graf.

```
import java.util.*;

public class Kruskal {

    public static void main(String[] args) {
        // Inisialisasi graf
        int graph[][] = new int[][] {
            { 0, 17, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 17, 0, 11, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 16, 11, 0, 0, 0, 12, 17, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 20, 0, 0, 5, 9, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 5, 0, 10, 0, 0, 12, 0, 0, 0, 0, 8 },
            { 0, 0, 12, 9, 10, 0, 9, 6, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 17, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 6, 0, 0, 6, 10, 0, 0, 0, 0 },
        };
    }
}
```

```

        { 0, 0, 0, 0, 12, 0, 0, 6, 0, 10, 0, 10, 17, 16 },
        { 0, 0, 0, 0, 0, 0, 0, 10, 10, 0, 7, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 9, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 9, 0, 17, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 9, 17, 0, 12 },
        { 0, 0, 0, 0, 8, 0, 0, 0, 16, 0, 0, 12, 0 },
    };
    // Panggil fungsi kruskalMST dan berikan graf sebagai
    argumen
    kruskalMST(graph);
}

public static void kruskalMST(int[][] graph) {
    int vertices = graph.length;
    List<Edge> edges = new ArrayList<>(); // List untuk
    menyimpan semua sisi
    List<Edge> mst = new ArrayList<>(); // List untuk
    menyimpan MST

    // Loop untuk menambahkan setiap sisi ke dalam List
    edges
    for (int i = 0; i < vertices; i++) {
        for (int j = i + 1; j < vertices; j++) {
            if (graph[i][j] != 0) {
                edges.add(new Edge(i, j, graph[i][j]));
            }
        }
    }

    // Urutkan List edges berdasarkan bobotnya dari kecil ke
    besar
    Collections.sort(edges);

    int[] parent = new int[vertices]; // Array untuk
    menyimpan simpul-simpul dalam MST

    // Inisialisasi setiap simpul dengan dirinya sendiri
    sebagai parent
    for (int i = 0; i < vertices; i++) {
        parent[i] = i;
    }

    // Loop untuk memproses setiap sisi dalam List edges
    for (Edge edge : edges) {
        int u = edge.source;
        int v = edge.destination;
        int uParent = findParent(parent, u);
        int vParent = findParent(parent, v);

        // Jika sisi tidak membentuk siklus, tambahkan sisi
        tersebut ke dalam MST dan
        // gabungkan dua simpul yang berbeda
        if (uParent != vParent) {

```

```

        mst.add(edge);
        parent[uParent] = vParent;
    }
}

// Cetak MST pada output
System.out.println("Minimum Spanning Tree:");
for (Edge edge : mst) {
    System.out.println(edge.source + " - " +
edge.destination + " : " + edge.weight);
}
}

public static int findParent(int[] parent, int node) {
    // Loop untuk mencari parent dari simpul
    while (parent[node] != node) {
        node = parent[node];
    }
    return node;
}

static class Edge implements Comparable<Edge> {
    int source, destination, weight;

    public Edge(int source, int destination, int weight) {
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }

    @Override
    public int compareTo(Edge otherEdge) {
        return Integer.compare(this.weight,
otherEdge.weight);
    }
}
}

```

Gambar 2.3. Kode Program

Program di atas adalah implementasi algoritma Kruskal dalam bahasa pemrograman Java. Algoritma Kruskal digunakan untuk menyelesaikan masalah MST (*Minimum Spanning Tree*) dalam sebuah graf. Program ini memiliki tiga bagian utama, yaitu fungsi main, fungsi kruskalMST, dan kelas Edge.

Fungsi main berisi inisialisasi graf dan pemanggilan fungsi kruskalMST dengan graf sebagai argumen. Dalam array graf tersebut, setiap elemen

mewakili bobot (weight) edge antara dua titik (vertex) dalam sebuah graf berbentuk matriks adjacency.

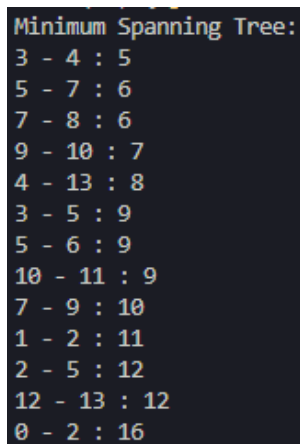
Jadi, dalam array graf tersebut, setiap baris dan kolom merepresentasikan sebuah vertex, dan setiap nilai dalam array merepresentasikan bobot edge antara dua vertex yang dihubungkan oleh edge tersebut. Misalnya, `graph[0][1]` adalah bobot edge antara vertex 0 dan vertex 1. Jika bobotnya 0, artinya tidak ada edge yang menghubungkan kedua vertex tersebut.

Fungsi `kruskalMST` adalah implementasi algoritma Kruskal. Fungsi ini mengambil graf sebagai argumen dan mengembalikan MST. Fungsi ini pertama-tama menambahkan semua sisi dalam graf ke dalam List edges. Kemudian, List edges diurutkan berdasarkan bobotnya dari kecil ke besar menggunakan metode `Collections.sort`. Fungsi ini juga membuat array parent untuk menyimpan simpul-simpul dalam MST dan menginisialisasi setiap simpul dengan dirinya sendiri sebagai parent. Fungsi ini kemudian memproses setiap sisi dalam List edges dan menambahkan sisi tersebut ke dalam MST jika sisi tersebut tidak membentuk siklus.

Kelas Edge adalah kelas untuk merepresentasikan sisi dalam graf. Kelas ini memiliki tiga atribut yaitu source (simpul awal), destination (simpul tujuan), dan weight (bobot sisi). Kelas ini juga mengimplementasikan metode `compareTo` untuk membandingkan bobot dua sisi.

2.5. Hasil atau Ouput

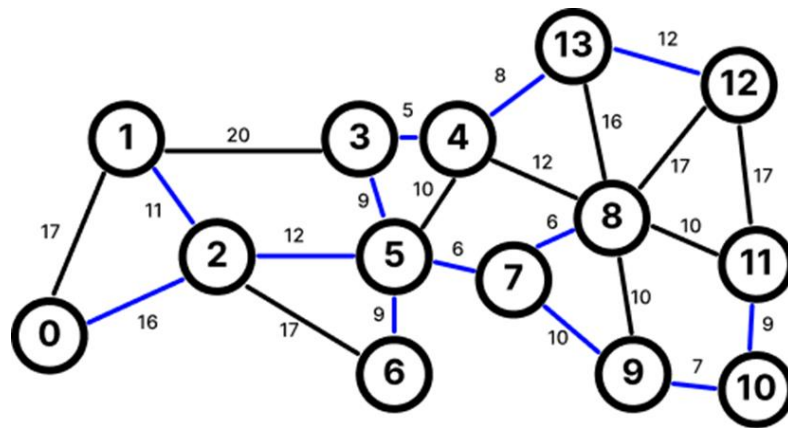
Berikut ini tampilan ketika kode program dijalankan di terminal.



```
Minimum Spanning Tree:
3 - 4 : 5
5 - 7 : 6
7 - 8 : 6
9 - 10 : 7
4 - 13 : 8
3 - 5 : 9
5 - 6 : 9
10 - 11 : 9
7 - 9 : 10
1 - 2 : 11
2 - 5 : 12
12 - 13 : 12
0 - 2 : 16
```

Gambar 2.4. Output Program

Berdasarkan output pada **Gambar 2.4**. Maka graf yang dihasilkan setelah mendapatkan *Minimum Spanning Tree* adalah sebagai berikut



Gambar 2.5. MST

BAB III

KESIMPULAN

Dapat disimpulkan bahwa implementasi algoritma Kruskal dalam menentukan topologi jaringan antar gedung di Fakultas Teknik Universitas Palangka Raya sangat penting. Algoritma Kruskal dapat membantu dalam menentukan MST (*Minimum Spanning Tree*) pada graf yang terhubung, sehingga topologi jaringan antar gedung yang dihasilkan akan optimal dan efisien. Yang awalnya memerlukan 23 edge untuk terhubung kesemua gedung, setelah dilakukan MST hanya membutuhkan 13 edge agar semua gedung terhubung.

Dalam hal ini, topologi jaringan antar gedung perlu dipikirkan dengan matang agar jaringan dapat beroperasi secara efisien dan aman. Dengan menggunakan algoritma Kruskal, akan didapatkan topologi jaringan antar gedung yang terintegrasi dan terkoneksi dengan baik, sehingga kegiatan akademik dan administratif di kampus dapat berjalan dengan lancar.

Dalam implementasinya, algoritma Kruskal dilakukan dengan langkah-langkah sederhana seperti mengurutkan edge berdasarkan bobotnya, memilih edge dengan bobot terkecil, dan menambahkan edge tersebut ke MST jika tidak membentuk siklus. Dalam hal ini, simpul dapat diartikan sebagai gedung atau titik konektivitas di dalam gedung yang akan dihubungkan.

Dengan demikian, implementasi algoritma Kruskal dapat membantu dalam menentukan topologi jaringan antar gedung yang optimal dan efisien di Fakultas Teknik Universitas Palangka Raya.

DAFTAR PUSTAKA

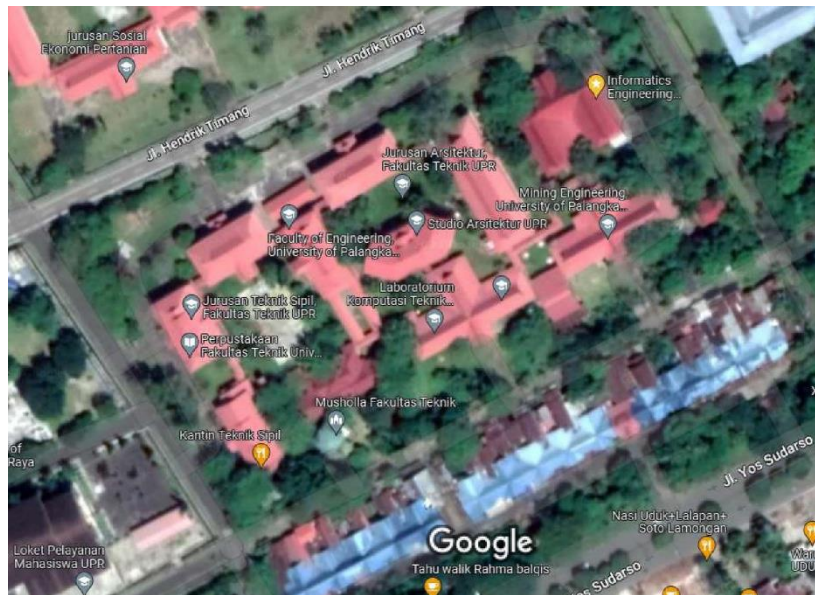
Dosen Teknik Informatika. 2022. Modul Algoritma dan Pemrograman II. Palangka Raya. Jurusan Teknik Informatika Fakultas Teknik Universitas Palangka Raya.

GusJiGang. 2023. *Perlu di Ketahui Dalam Membangun Jaringan Antar Gedung*.

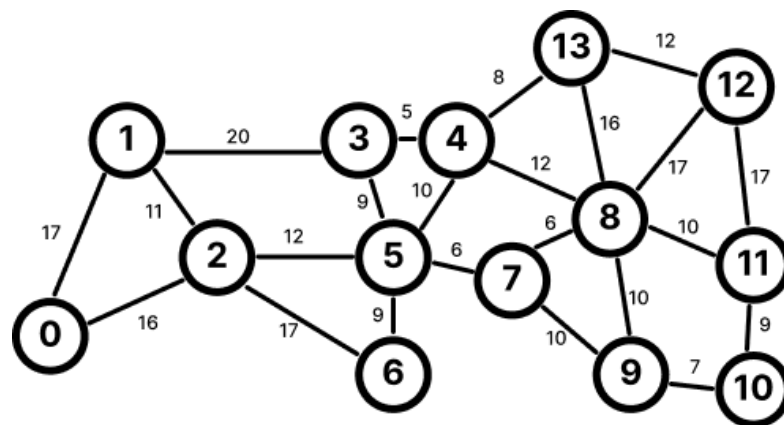
<https://networking.indo-solution.com/perlu-di-ketahui-dalam-membangun-jaringan-antar-gedung/>

(Diakses pada tanggal 10 April 2023)

LAMPIRAN



Gambar 2.1. Peta



Gambar 2.2. Graf

```
import java.util.*;

public class Kruskal {

    public static void main(String[] args) {
        // Inisialisasi graf
        int graph[][] = new int[][] {
            { 0, 17, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 17, 0, 11, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 16, 11, 0, 0, 0, 12, 17, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 20, 0, 0, 5, 9, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 5, 0, 10, 0, 0, 12, 0, 0, 0, 0, 8 },
            { 0, 0, 12, 9, 10, 0, 9, 6, 0, 0, 0, 0, 0, 0 },
        };
    }
}
```

```

        { 0, 0, 17, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 6, 0, 0, 6, 10, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 12, 0, 0, 6, 0, 10, 0, 10, 17, 16 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 10, 10, 0, 7, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 9, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 9, 0, 17, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 9, 17, 0, 12 },
        { 0, 0, 0, 0, 8, 0, 0, 0, 16, 0, 0, 0, 12, 0 },
    };
    // Panggil fungsi kruskalMST dan berikan graf sebagai
argumen
    kruskalMST(graph);
}

public static void kruskalMST(int[][] graph) {
    int vertices = graph.length;
    List<Edge> edges = new ArrayList<>(); // List untuk
menyimpan semua sisi
    List<Edge> mst = new ArrayList<>(); // List untuk
menyimpan MST

    // Loop untuk menambahkan setiap sisi ke dalam List
edges
    for (int i = 0; i < vertices; i++) {
        for (int j = i + 1; j < vertices; j++) {
            if (graph[i][j] != 0) {
                edges.add(new Edge(i, j, graph[i][j]));
            }
        }
    }

    // Urutkan List edges berdasarkan bobotnya dari kecil ke
besar
    Collections.sort(edges);

    int[] parent = new int[vertices]; // Array untuk
menyimpan simpul-simpul dalam MST

    // Inisialisasi setiap simpul dengan dirinya sendiri
sebagai parent
    for (int i = 0; i < vertices; i++) {
        parent[i] = i;
    }

    // Loop untuk memproses setiap sisi dalam List edges
    for (Edge edge : edges) {
        int u = edge.source;
        int v = edge.destination;
        int uParent = findParent(parent, u);
        int vParent = findParent(parent, v);

        // Jika sisi tidak membentuk siklus, tambahkan sisi
tersebut ke dalam MST dan

```

```

        // gabungkan dua simpul yang berbeda
        if (uParent != vParent) {
            mst.add(edge);
            parent[uParent] = vParent;
        }
    }

    // Cetak MST pada output
    System.out.println("Minimum Spanning Tree:");
    for (Edge edge : mst) {
        System.out.println(edge.source + " - " +
edge.destination + " : " + edge.weight);
    }
}

public static int findParent(int[] parent, int node) {
    // Loop untuk mencari parent dari simpul
    while (parent[node] != node) {
        node = parent[node];
    }
    return node;
}

static class Edge implements Comparable<Edge> {
    int source, destination, weight;

    public Edge(int source, int destination, int weight) {
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }

    @Override
    public int compareTo(Edge otherEdge) {
        return Integer.compare(this.weight,
otherEdge.weight);
    }
}
}

```

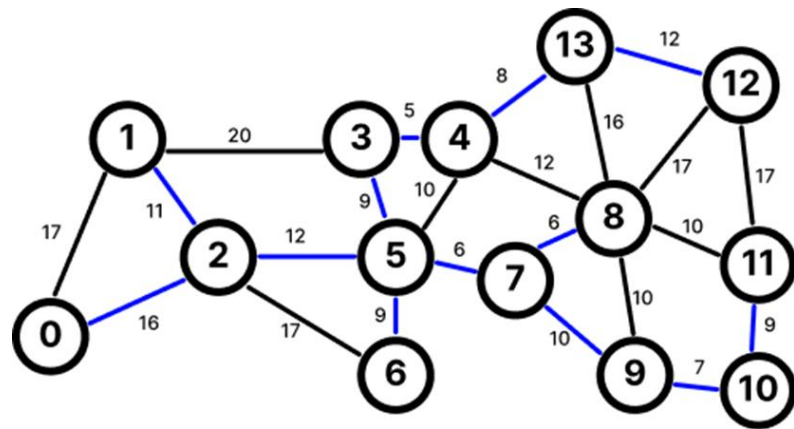
Gambar 2.3. Kode Program

```

Minimum Spanning Tree:
3 - 4 : 5
5 - 7 : 6
7 - 8 : 6
9 - 10 : 7
4 - 13 : 8
3 - 5 : 9
5 - 6 : 9
10 - 11 : 9
7 - 9 : 10
1 - 2 : 11
2 - 5 : 12
12 - 13 : 12
0 - 2 : 16

```

Gambar 2.4. Output Program



Gambar 2.5. MST