Penyelesaian *Minimum Spanning Tree* Menggunakan Algoritma Prim dan Kruskal

[1]Arif Yulianto
1301168560
arify@outlook.co.id

[2]Amira Nur Khalipah
1301168559
amiranurkhalipah@gmail.com

[3]Ardhi Akmaludin Jadhira
1301168522
ardhiakhmal@gmail.com

Tugas Besar Mata Kuliah Desain Dan Analisis Algoritma IF-40-EXT02
Program Studi S1 Teknik Informatika, Fakultas Informatika
Universitas Telkom

Abstrak --- Pada permasalahan menentukan pohon merentang minimum dapat diselesaikan dengan beberapa metode. Namun, ada dua cara yang paling efektif untuk menyelesaikan masalah menentukan pohon merentang minimum, yaitu dengan algoritma Prim dan algoritma Kruskal. Algoritma Prim adalah suatu langkah-langkah untuk membuat pohon (tree) dengan cara memilih atau mengambil sisi dari graf yang memiliki bobot minimum dan bersisian dengan simpul di dalam pohon tetapi sisi tersebut tidak membentuk sirkuit di dalam pohon. Algoritma Kruskal adalah suatu langkah untuk membuat pohon (tree) dengan cara mengurutkan sisi dari graf berdasarkan urutan bobot dari yang terkecil hingga terbesar tetapi tidak membentuk sirkuit.

I. PENDAHULUAN

Teori graf dan pohon merupakan cabang dari matematika diskrit yang didapatkan dengan banyak pemodelan masalah dalam kehidupan sehari-hari. Salah satu pembahasannya yaitu pohon merentang minimum (minimum spanning tree). Salah satu contoh penyelesaian masalah dengan pemodelan pohon merentang minimum adalah input data dari file.txt yang isinya beruapa matriks (angka).

Dalam permasalahan pohon merentang minimum dapat diselesaikan dengan beberapa cara. Namun cara yang paling digunakan adalah algoritma Prim dan algoritma Kruskal. Kedua algoritma ini terbukti dapat menyelesaikan pohon merentang minimum. Namun dalam penggunannya, pengguna sering merasa sulit untuk memilih algoritma mana yang lebih tepat. Oleh Karena itu, kami akan membahas tentang kedua algoritma tersebut.

Beberapa terminologi dasar yang harus diketahui:

a. Graf Berarah (*Directed Graph*/*Digraph*)
Graf berarah adalah graf yang setiap sisinya diberi orientasi arah. Dalam hal ini sisi yang ditulis (v_1,v_2) berbeda dengan sisi (v_2,v_1)

b. Graf Berbobot (*Weight Graf*)
Graf berbobot adalah graf yang setiap sisinya diberi sebuah nilai bobot.

c. Graf Lengkap (*Complete Graph*)
Graf lengkap adalah graf sederhana, tidak mengandung gelang (sisi yang kedua simpulnya sama) maupun sisi ganda (dua sisi yang memiliki simpul asal dan simpul tujuan yang sama), serta setiap sisinya mempunyai sisi ke simpul lain.

d. Bertetangga (*Adjacent*)
Dua buah simpul pada graf tak berarah dikatakan bertetangga bila keduanya terhubung dengan sebuah sisi. Dapat dikatakan, jika ada v_1 dan v_2 yang bertetangga, maka harus ada sisi (v_1,v_2)

e. Bersisian (*Incident*)
Untuk sembarang sisi e = (v_1,v_2), sisi e dikatakan bersisian dengan simpul v_1 dan simpul v_2.

f. Simpul Terpencil (*Isolated Vertex*)
Simpul terpencil adalah simpul yang tidak mempunyai sisi yang bersisian dengannya. Dengan kata lain, simpul ini ialah simpul yang tidak satupun bertetangga dengan simpul-simpul lain.

g. Graf Kosong (*Empty Graf*)
Graf kosong yaitu graf yang himpunan sisinya merupakan himpunan kosong.

h. Lintasan (*Path*)
Lintasan yang panjangnya n dari simpul awal v_0 ke simpul akhir v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v_0, e_1, v_1, e_2, …, v_{n-1}, e_n, v_n sedemikian sehingga $e_1=(v_0,v_1)$, $e_2=(v_1,v_2)$, …, $e_n=(v_{n-1},v_n)$ adalah sisi-sisi dari graf G.

i. Siklus (*Cycle*) atau Sirkuit (*Circuit*)
Lintasan yang berawal dan berakhir pada simpul yang sama disebut siklus atau sirkuit.

j. Terhubung (*Connected*)
Graf disebut graf terhubung jika untuk setiap pasang simpul v_1 dan v_2 di dalam himpunan V terdapat lintasan dari v_1 ke v_2, yang juga berarrti ada lintasan dari v_2 ke v_1 (untuk graf berarah).

k. Upagraf (*Subgraf*) dan Komplemen Upagraf
Misalkan G = {V,E} sebuah graf, G_1 = $\{V_1,E_1\}$ dikatakan upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.

Komplemen dari upagraf G_1 terhadap G adalah graf G_2 = $\{V_2,E_2\}$ sedemikian sehingga $E_2 = E - E_1$ dan V_2 adalah himpunan simpul yang anggota-anggota E_2 bersisian dengannya.

l. Upagraf Merentang (*Spanning Subgraf*)
Upagraf G_1 = $\{V_1,E_1\}$ dari G = {V,E} dikatakan upagraf merentang jika V_1=V, G_1 mengandung semua simpul G.F

II. STRATEGI

Strategi algoritma prim yang digunakan:

1. Pada setiap langkah, pilih sisi *e* dari graf *G*(*V*, *E*) yang mempunyai bobot terkecil dan bersisian dengan simpul-simpul di *T* tetapi *e* tidak membentuk sirkuit di *T*.
2. Komplesiats algoritma: $O(n^2)$.
3. Pada algoritma prim, dimulai pada vertex yang mempunyai sisi (edge) dengan bobot terkecil.
4. Sisi yang dimasukkan ke dalam himpunan T adalah sisi graph G yang bersisian dengan sebuah simpul di T, sedemikian sehingga T adalah Tree (pohon). Sisi dari Graph G ditambahkan ke T jika ia tidak membentuk cycle.

(NOTE: dua atau lebih edge kemungkinan mempunyai bobot yang sama, sehingga terdapat pilihan vertice, dalam hal ini dapat diambil salah satunya.)

Contoh Algoritma Prim:

1. Ambil sisi (edge) dari graph yang berbobot minimum, masukkan ke dalam T.
2. Pilih sisi (edge) (i,j) yg berbobot minimum dan bersisisan dengan simpul di T, tetapi (i,j) tidak membentuk cycle di T. tambahkan (i,j) ke dalam T.
3. Ulangi prosedur no 2 sebanyak (n-2) kali.

Strategi algorima kruskal yang digunakan:

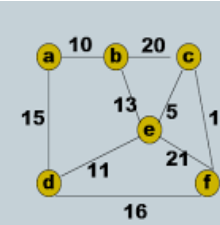
Pada setiap langkah, pilih sisi *e* dari graf *G* yang mempunyai bobot minimum tetapi *e* tidak membentuk sirkuit di *T*.

Kompleksitas algoritma: ***O*(|*E*| log |*E*|)**

- Pada algoritma kruskal, sisi (edge) dari Graph diurut terlebih dahulu berdasarkan bobotnya dari kecil ke besar.
- Sisi yang dimasukkan ke dalam himpunan T adalah sisi graph G yang sedemikian sehingga T adalah Tree (pohon). Sisi dari Graph G ditambahkan ke T jika ia tidak membentuk cycle.
    1. T masih kosong
    2. Pilih sisi (i,j) dengan bobot minimum
    3. Pilih sisi (i,j) dengan bobot minimum berikutnya yang tidak membentuk cycle di T, tambahkan (i,j) ke T
    4. Ulangi langkah 3 sebanyak (n-2) kali.
    5. Total langkah (n-1) kali

III. STRUKTUR DATA

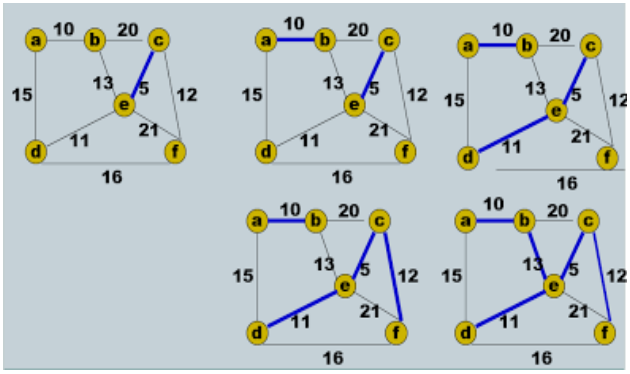
Dalam penyelesaian masalah *minimum spanning tree* menggunakan algoritma prim dan kruskal, Langkah pertama yang harus kita lakukan tentunya adalah memilih struktur data yang tepat untuk digunakan dalam merepresentasikan permasalahan MST. Jika dilihat kembali, sebuah peta seperti pada gambar di atas pada dasarnya hanya menunjukkan titik-titik yang saling berhubungan, dengan jarak tertentu pada masing-masing titik tersebut. Misalnya, peta di atas dapat direpresentasikan dengan titik-titik penghubung seperti berikut:



Langkah	Sisi	bobot
0		
1	e-c	5
2	a-b	10
3	d-e	11
4	c-f	12
5	b-e	13

Gambar 1.1 Contoh graph MST

Maka dari itu, untuk menyelesaikan permasalahan jarak terpendek ini kita akan menggunakan **struktur data graph** untuk merepresentasikan peta. Berikut adalah contoh untuk merepresentasikan graph yang akan digunakan:

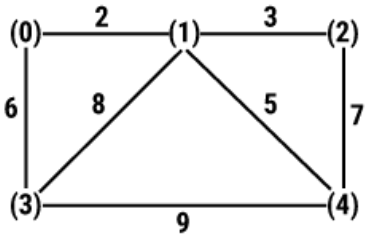


Gambar 1.2 Contoh graph solusi MST

Struktur data graph yang telah dirancang dan diimplementasikan mampu untuk menyimpan data matriks berukuran besar sesuai kapasitas memori komputer. Program yang telah dibuat dengan memanfaatkan struktur data graph mampu memecahkan masalah jaringan (*minimum spanning tree*) dengan tepat dan memberikan pemecahan langkah demi langkah.

IV. PENYELESAIAN

Berikut merupakan contoh penyelesaian permasalahan menggunakan strategi algoritma kruskal dan prim :



	0	1	2	3	4
0	0	2	0	6	0
1	2	0	3	8	5
2	0	3	0	0	7
3	6	8	0	0	9
4	0	5	7	9	0

- **Algoritma Kruskal**

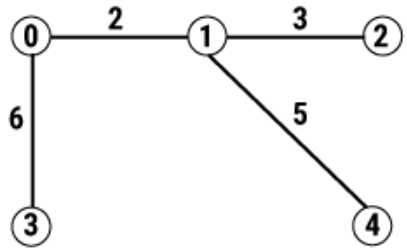
  Langkah-langkah penyelesaikan menggunakan algoritma kruskal :

  (i) Dari contoh graph tersebut, kita ubah terlebih dahulu menjadi matrik diagonal 4x4 seperti pada gambar diatas. Matrik tersebut digunakan sebagai input dari program.

  (ii) Mencatat semua edge beserta bobot atau costnya, dari graph diatas terdapat total 7 edge yaitu :
  - (0,1) = 2
  - (0,3) = 6
  - (1,2) = 3
  - (1,3) = 8
  - (1,4) = 5
  - (2,4) = 7
  - (3,4) = 9

  (iii) Melakukan pengurutan (sorting) berdasarkan bobot atau costnya.
  - (0,1) = 2
  - (1,2) = 3
  - (1,4) = 5
  - (0,3) = 6
  - (2,4) = 7
  - (1,3) = 8
  - (3,4) = 9

  (iv) Membuat tree berdasarkan dari data yang telah diurutkan.



(0,1) = 2 -> Sudah dikunjungi
(1,2) = 3 -> Sudah dikunjungi
(1,4) = 5 -> Sudah dikunjungi
(0,3) = 6 -> Sudah dikunjungi
(2,4) = 7 -> Tidak dihubungkan, karena jika dihubungkan akan terjadi siklik
(1,3) = 8 -> Tidak dihubungkan, karena jika dihubungkan akan terjadi siklik
(3,4) = 9 -> Tidak dihubungkan, karena jika dihubungkan akan terjadi siklik

(v) Langkah terakhir adalah dengan menghitung total bobot atau cost dari vertex yang telah dikunjungi. Dari hasil analisa langkah sebelumnya, didapat total minimun spanning tree menggunakan algoritma kruskal adalah (2+3+5+6) = 16

- **Algoritma Prim**

  Langkah-langkah penyelesaikan menggunakan algoritma prim :

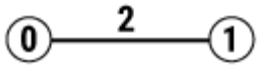
  (i) Dari contoh graph tersebut, kita ubah terlebih dahulu menjadi matrik diagonal 4x4 seperti pada gambar diatas. Matrik tersebut digunakan sebagai input dari program

  (ii) Mencatat semua edge beserta bobot atau costnya, dari graph diatas terdapat total 7 edge yaitu :
  - (0,1) = 2
  - (0,3) = 6
  - (1,2) = 3
  - (1,3) = 8
  - (1,4) = 5
  - (2,4) = 7
  - (3,4) = 9

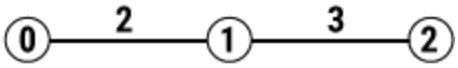
  (iii) Memilih salah satu vertex untuk menjadi permulaan. Misal kita pilih (1), lalu ada berapa edge yang terhubung dengan (1). Langkah selanjutnya adalah dengan melakukan analisa terhadap edge yang terhubung dengan (1). Untuk setiap langkahnya akan dijelaskan lebih lanjut.

No	Edge	Langkah IV	L-V	L-VI	L-VII	L-VIII
~~1~~	~~(0,1) = 2~~	←				
~~2~~	~~(0,3) = 6~~		←	←	←	
~~3~~	~~(1,2) = 3~~	←	←			
4	(1,3) = 8	←	←	←	←	(Siklik/Loop)
~~5~~	~~(1,4) = 5~~	←	←	←		
6	(2,4) = 7			←	←	(Siklik/Loop)
7	(3,4) = 9				←	(Siklik/Loop)

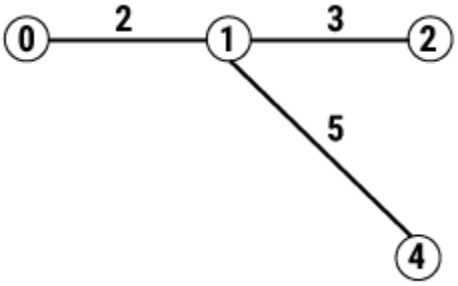
(iv) Memilih salah satu vertex yang terhubung dengan (1) dan yang memiliki bobot atau cost yang paling kecil. Dipilih (0,1) = 2, lalu tandai (0) dan (1) karena telah dikunjungi.



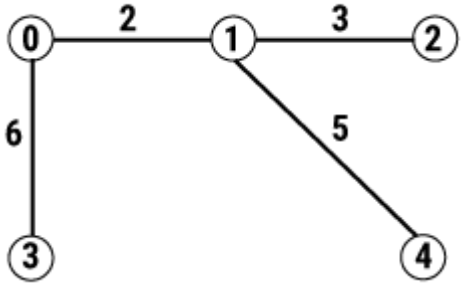
(v) Memilih salah satu vertex yang terhubung dengan (0) dan (1) dan yang memiliki bobot atau cost yang paling kecil. Dipilih (1,2) = 3, lalu tandai (2) karena telah dikunjungi.



(vi) Memilih salah satu vertex yang terhubung dengan (0), (1), (2) dan yang memiliki bobot atau cost yang paling kecil. Dipilih (1,4) = 5, lalu tandai (4) karena telah dikunjungi.



(vii) Memilih salah satu vertex yang terhubung dengan (0), (1), (2), (4) dan memiliki bobot atau cost yang paling kecil. Dipilih (0,3) = 6, lalu tandai (3) karena telah dikunjungi.



(viii) Pada edges (1,3) = 8, (2,4) = 7, (3,4) = 9 jika dihubungkan dengan vertexnya masing-masing akan terjadi siklik atau loop. Semua vertex yang telah dikunjungi seperti pada langkah sebelumnya, maka didapat dari hasil analisa total minimum spanning tree menggunakan algoritma prim adalah (2+3+5+6) = 16.

V. PENJELASAN

5.1 Algortima Prim

Algoritma Prim adalah suatu langkah-langkah untuk membuat pohon (tree) dengan cara memilih atau mengambil sisi dari graf yang memiliki bobot minimum dan bersisian dengan simpul di dalam pohon tetapi sisi tersebut tidak membentuk sirkuit di dalam pohon.

Langkah-langkah algoritma Prim:

1. Ambil sisi dari graf yang berbobot minimum, kemudian masukkan ke dalam pohon.
2. Pilihlah sisi yang mempunyai bobot minimum dan bersisian dengan pohon. Namun sisi tersebut tidak membentuk sirkuit di dalam pohon. Masukkan sisi tersebut ke dalam pohon.
3. Ulangi langkah 2 sampai pohon merentang minimum terbentuk, pohon merentang minimum terbentuk setelah mengalami penggulangan sebanyak n-2 kali. (n adalah jumlah simpul graf).

Penulisan algoritma Prim dalam bentuk**:**

Pseudocode Algoritma Prim

```
Procedure Prim(input G:graf, output T:pohon)
{
    1. Membentuk pohon merentang minimum T dari graf terhubung G.
    2. Masukan: graf-berbobot terhubung G = (V, E), yang mana |V| = n
    3. Keluaran: Minimum Spanning Tree T = (V, E')
}

Deklarasi
i, p, q, u, v : integer

Algoritma
Cari sisi (p,q) dari E yang berbobot terkecil
T ← {(p,q)}
for i ← 1 to n-1 do
   Pilih sisi (u,v) dari E yang bobotnya terkecil namun bersisian
   dengan suatu simpul didalam T
   T ← T U {(u,v)}
Endfor
```

Implementasi source code algoritma prim untuk menyelesaikan *problem minimum spanning tree* dalam bahasa pemrograman java.

mst.java

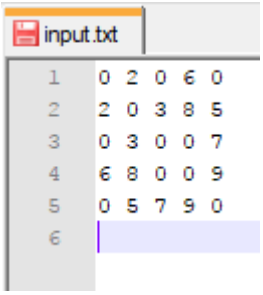
```
//Fungsi problem Minimal Spanning Tree dengan algoritma Prim
public void prim() {
try {
String fileDir = file.getParent();
//membaca inputan
   for (int i = 0; i < weights.size(); i++) {
            System.out.printf(weights.get(i) + "\t");
              if ((i + 1) % dimension == 0) {
                        System.out.println(""); }
                                  }
                    System.out.println("");

int[][] matrix = new int[dimension][dimension];
```

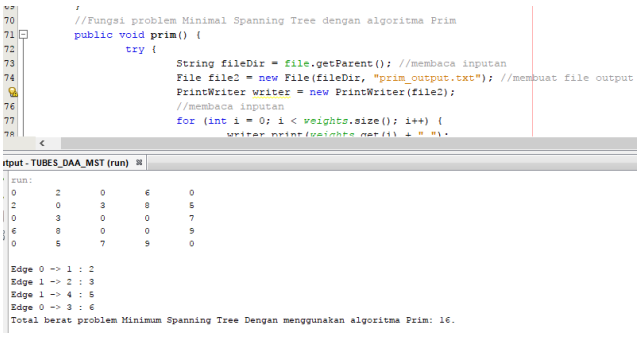
```
int[] visited = new int[dimension];
int min = 21; // inisiasi berat maksimal
int u = 0, v = 0; // iterasi untuk edge
int total = 0;
int count = 0; // iterasi array list dari berat (weight)
                    //Mencari nilai weight tekecil
                      for (int i = 0; i < dimension; i++) {
                                    visited[i] = 0;
                                for (int j = 0; j < dimension; j++) {
                                    matrix[i][j]                    =
weights.get(count);
                                                count++;
                                    if (matrix[i][j] == 0) {
                                        matrix[i][j] = 21;
                                        }
                                    }
                                }
                                visited[0] = 1;
                    //Mencari nilai weight terkecil
          for (int counter = 0; counter < dimension - 1; counter++)
{
              min = 21;
              for (int i = 0; i < dimension; i++) {
                  if (visited[i] == 1) {
                      for (int j = 0; j < dimension; j++) {
                          if (visited[j] == 0) {
                              if (min > matrix[i][j]) {
                                  min = matrix[i][j]; u = i; v = j;
                                  }
                              }
                          }
                      }
                  }
              visited[v] = 1; total += min; matrix[u][v] = 21;
System.out.println("Edge " + u + " -> " + v + " : " +
min);//menuliskan hasil ke line output
                                  }
System.out.println("Total berat problem Minimum Spanning
Tree Dengan menggunakan algoritma Prim: " + total + ".");
                    System.out.println();
                          } catch (FileNotFoundException e) {
                                    e.printStackTrace();
                              }

}
```

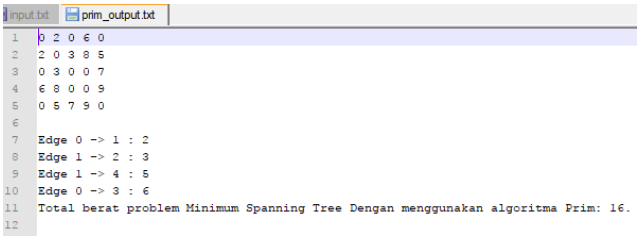
Input program algoritma prim:



Nilai inputan berbentuk matrix dalam file input.txt yang diletakkan di folder …/NetbeansProjects/TUBES_DAA_MST/



Gambar Output Program Algoritma Prim



Gambar Output Program Algoritma Prim

Gambar diatas merupakan hasil output program dalam bentuk prim_output.txt yang berlokasi di folder …/NetbeansProjects/TUBES_DAA_MST/

5.2 Algortima Kruskal

Algoritma Kruskal adalah suatu langkah untuk membuat pohon (tree) dengan cara mengurutkan sisi dari graf berdasarkan urutan bobot dari yang terkecil hingga terbesar tetapi tidak membentuk sirkuit.

Langkah-langkah algoritma Kruskal:

1. Lakukan pengurutan terhadap setiap sisi di graf mulai dari sisi dari bobot terkecil hingga bobot terbesar.
2. Pilih sisi yang mempunyai bobot minimum yang tidak membentuk sirkuit pada pohon, kemudian tambahkan sisi tersebut ke dalam pohon.
3. Ulangi langkah kedua sampai pohon minimum merentang terbentuk, pohon merentang minimum terbentuk setelah mengalami penggulangan sebanyak n-1 kali. (n adalah jumlah simpul graf).

Penulisan algoritma Kruskal dalam bentuk pseudocode:

Pseudocode Algoritma Kruskal

```
Procedure Kruskal(input G:graf, output T:pohon)
{
    1. Membentuk Minimum Spanning Tree dari graph
       terhubung G
}

Deklarasi:
i, p, q, u, v : integer

Algoritma:
{Asumsi : sisi-sisi graph sudah diurut dari yang terkecil
berdasarkan bobotnya}
T ← { }
While jumlah sisi T < n-1 do
    Pilih sisi dari E yang bobotnya terkecil
    If (u,v) tidak membentuk siklus di T then
    T ← T u {(u,v)}
endfor
endwhile
```

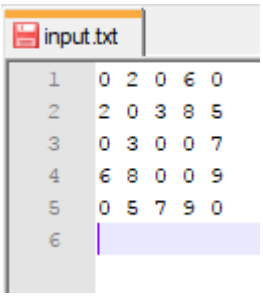
Implementasi source code algoritma Kruskal untuk menyelesaikan *problem minimum spanning tree* dalam bahasa pemrograman java.

mst.java

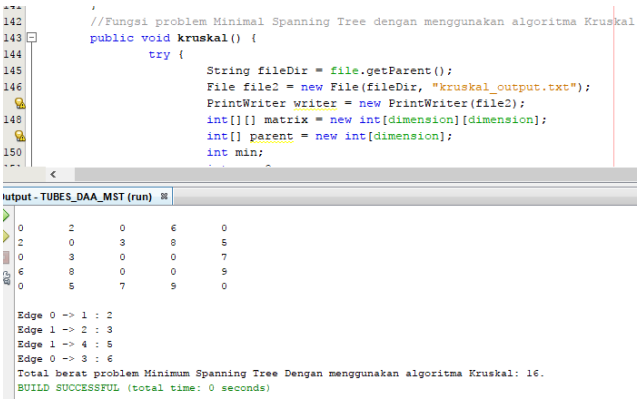
```
public void kruskal() {
try {
String fileDir = file.getParent();
File file2 = new File(fileDir, "kruskal_output.txt");
PrintWriter writer = new PrintWriter(file2);
int[][] matrix = new int[dimension][dimension];
int[] parent = new int[dimension];
int min;
int u = 0;
int v = 0;
int edges = 1;
int total = 0;
int count = 0;
for (int i = 0; i < weights.size(); i++) {
    writer.print(weights.get(i) + " ");
    System.out.printf(weights.get(i) + "\t");
    if ((i + 1) % dimension == 0) {
    writer.println("");
    System.out.println("");
     }
 }
writer.println("");
System.out.println("");
for (int i = 0; i < dimension; i++) {
    for (int j = 0; j < dimension; j++) {
         matrix[i][j] = weights.get(count);
         count++;
           }
     }
    for (int i = 0; i < dimension; i++) {
        parent[i] = 0;
        for (int j = 0; j < dimension; j++) {
           if (matrix[i][j] == 0) {
              matrix[i][j] = 21;
               }
           }
       }
    while (edges < dimension) {
           min = 21;
           for (int i = 0; i < dimension; i++) {
                for (int j = 0; j < dimension; j++) {
                     if (matrix[i][j] < min) {
                       min = matrix[i][j];
                       u = i;
                       v = j;
                       }
                  }
             }
           if (v != u) {
           edges++;

System.out.println("Edge " + u + " -> " + v + " : " + min);
                                total += min;
           parent[v] = u;
       }
           matrix[u][v] = 21;
           matrix[v][u] = 21;
       }
System.out.println("Total berat problem Minimum Spanning
Tree Dengan menggunakan algoritma Kruskal: "
                                                + total + ".");
           } catch (IOException e) {
                                    e.printStackTrace();
                    }
}
```

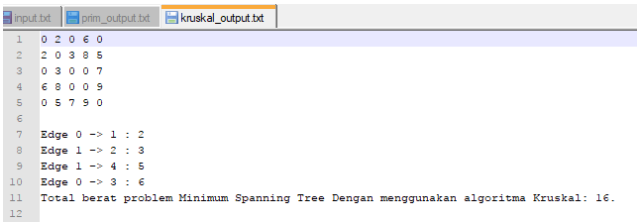
Input program algoritma kruskal:



Nilai inputan berbentuk matrix dalam file input.txt yang diletakkan di folder …/NetbeansProjects/TUBES_DAA_MST/



Gambar Output Program Algoritma Kruskal



Gambar Output Program Algoritma Kruskal

Gambar diatas merupakan hasil output program dalam bentuk prim_output.txt yang berlokasi di folder …/NetbeansProjects/TUBES_DAA_MST/

5.3 Analisis kompleksitas

1.1 Penentuan n

- **Algoritma Prim**
  n = 8.
- **Algoritma Kruskal**
  Pada algoritma kruskal ini kita menginputkan n = v-i

1.2 Basic operation & alasan pemilihan

- **Algoritma Prim**
  for (i = 0; i < Nodes; i++)
  for (j = 0; j < Nodes; j++)

Alasan : Karena for i dan for j sering dieksekusi dalam program algoritma prim.

- **Algoritma Kruskal**

  While (e < V-1)
  Alasan : eksekusi dilakukan berulang-ulang sesuai dengan nilai edge dan vertex yang diinputkan. Jika nilai yang diinputkan 5 maka eksekusi akan dilakukan 5 kali.

1.3 Proses memperoleh C(n)

- Penentuan kelas OOG
  - **Algoritma Prim**

$$C(n) = \sum_{i=1}^{n-m} . \sum_{j=1}^{n-m} 1$$
$$= \sum_{i=1}^{n-m} 1\ (n-m-1+1)$$
$$= \sum_{i=1}^{n-m} 1\ (n-m)$$
$$= (n-m-1+1)(n-m)$$
$$= (n-m)(n-m)$$
$$= n^2 - nm - nm - m^2$$
$$= n^2 - 2nm - m^2$$

$$T(n) = n(n-m) + 1$$

  - **Algoritma Kruskal**

Best case : Cn = $\sum_{i=1}^{1} 1$
e= edge dan v = vertex
Dengan kondisi e = v-1 dapat ditulis **e log v**
Waktu untuk 2e untuk menemukan operasinya adalah **= log v**
Untuk operasi v-1= n
Sehingga total running time
T(n) = e log v + 2e log v + n
T(n) = (e+v) log n

- Notasi asymptotic
  - **Algoritma Prim**
    $O(n(n-m)+1)$
  - **Algoritma Kruskal**
    Best case = O (1)
    T (n) = O ((n+m)log n)

VI. KESIMPULAN

Kesimpulan yang dapat diambil dari studi dan perbandingan dua algoritma pencarian pohon merentang minimum adalah:

1. Algoritma Prim dan Algoritma Kruskal dapat menyelesaikan masalah pencarian pohon merentang minimum dengan tepat dan akurat.
2. Algoritma Kruskal lebih efisien dibandingkan algoritma Prim saat graf yang diberikan memiliki banyak simpul dengan sisi yang sedikit.
3. Algoritma Prim lebih efisien dibandingkan algoritma Kruskal saat graf yang diberikan memiliki banyak sisi dengan simpul yang sedikit (graf lengkap).

REFERENSI

Munir, Rinaldi. (2005). Matematika Diskrit. Edisi revisi kelima. Bandung:Informatika Bandung.

Rosen, Kenneth. (2012). Discrete Mathematics and Its Applications. Edisi ketujuh. New York:The McGraw-Hill Companies.

Wikipedia. (2013, 29 November). Kruskal's algorithm. Diperoleh 28 Desember 2013, dari http://en.wikipedia.org/wiki/Kruskal%27s_algorithm.

Wikipedia. (2013, 5 April). Algoritma Prim. Diperoleh 28 Desember 2013, dari http://id.wikipedia.org/wiki/Algoritma_Prim.