Analisis Penerapan Algoritma Kruskal dalam Pembuatan Jaringan Distribusi Listrik

Maureen Linda Caroline (13508049)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18049@itb.ac.id

***Abstract*—Di jaman sekarang, kehidupan manusia tidak akan jauh dari listrik. Listrik sudah menjadi kebutuhan primer manusia. Lampu, televisi, kulkas, mesin cuci, *water heater*, dan lain-lain memerlukan listrik. Pembangunan jalur jaringan distribusi listrik pun harus dipikirkan secara serius sehingga tidak ada pasokan listrik yang terbuang percuma. Oleh karena itu dibutuhkan sebuah konsep untuk memecahkan masalah ini. Salah satu konsepnya adalah dengan membuat pohon merentang minimum yang dapat dihasilkan jaringan distribusi listrik minimum. Untuk membuat pohon merentang minimum dapat dilakukan dengan menggunakan berbagai algoritma salah satunya adalah algoritma kruskal. Algoritma kruskal termasuk kedalam algoritma *greedy* dimana algoritma ini memilih bobot terkecil terlebih dahulu.**

***Index Terms*—algoritma *greedy*, algoritma kruskal, jaringan distribusi listrik, pohon merentang minimum.**

I. PENDAHULUAN

Pohon merupakan graf yang khusus. Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Pohon yang didefinisikan seperti itu adalah pohon yang bebas.

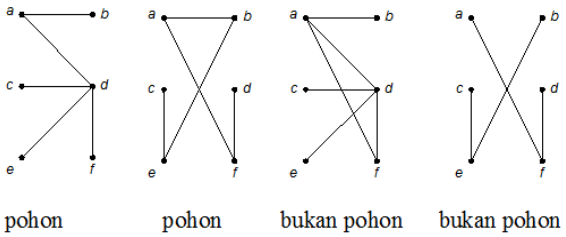
Pohon merentang adalah graf yang seluruh sirkuitnya diputus dan dibuang sehingga masing-masing simpul hanya memiliki satu jalur ke simpul lain. Aplokasi dari pohon merentang ni sangat banyak, salah satunya adalah dalam pembuatan jaringan distribusi listrik sehingga biaya yang dikeluarkan minimum.

Salah satu algoritma yang dapat digunakan untuk mendapatkan biaya yang dikeluarkan minimum adalah algoritma *greedy*. Dalam hal ini dapat digunakan algoritma kruskal yang termasuk dalam algoritma *greedy*.

II. METODE

1. Pohon dan Pohon Merentang

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit.



Gambar 2.1. Contoh pohon dan bukan pohon

Sifat-sifat pohon:

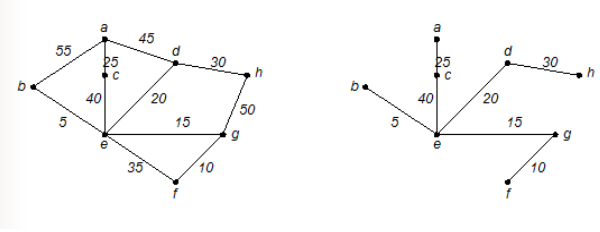
Misalkan G = (V,E) adalah graf tak-berarah sederhana dan jumlah simpulnya n buah, maka semua pernyataan dibawah ini adalah ekivalen

a. G adalah pohon
b. Setiap pasang simpul dalam G terhubung dengan lintasan tunggal
c. G terhubung dan memiliki m-n-1 buah sisi
d. G tidak mengandung sirkuit dan memiliki m-n-1 buah sisi.
e. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit
f. G terhubung dan smua sisinya adalah jembatan

Teorema ini juga sering dikatakan sebagai definisi lain dari pohon.

Pohon merentang dari graf terhubung adalah upagraf merentang yang berupa pohon. Pohon merentang diperoleh dengan memutus sirkuit didalam graf. Setiap graf terhubung mempunyai paling sedikit satu buah pohon merentang. Graf tak-berarah dengan k komponen mempunyai k buah hutan merentang yang disebut hutan merentang.

Pohon merentang minimum adalah pohon merentang yang berbobot minimum.



(a) **(b)**

Gambar 2.2. (a) Graf, (b) Pohon merentang minimum

2. Algoritma Greedy dan Algoritma Kruskal

2.1 Algoritma *Greedy*

Algoritma *greedy* merupakan metode yang paling popular untuk memecahkan persoalan optimasi. Persoalan optimasi merupakan persoalan mencari solusi optimum. Hanya ada dua macam persoalan optimasi yaitu maksimasi dan minimasi.

Algoritma *greedy* membentuk solusi langkah per langkah. Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.

Pada setiap langkah, kita membuat pilihan optimum lokal dengan harapan bahwa sisanya mengarah ke solusi optimum global.

Algoritma *greedy* adalah algoritma yang memecahkan masalah langkah per langkah, pada setiap langkah:

a. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now*!")
b. Berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir denan optimum global.

Elemen-elemen algoritma *greedy*:

a. Himpunan kandidat, C
b. Himpunan solusi, S
c. Fungsi seleksi
d. Fungsi kelayakan
e. Fungsi obyektif

Dengan kata lain, algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C, yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan yaitu menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

Pseudo-code algoritma *greedy* adalah sebagai berikut:

```
function greedy(input C: himpunan_kandidat) → himpunan_kandidat
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy
Masukan: himpunan kandidat C
Keluaran: himpunan solusi yang bertipe himpunan_kandidat
}
Deklarasi
  x : kandidat
  S : himpunan_kandidat
Algoritma:
  S ← {} { inisialisasi S dengan kosong }
  while (not SOLUSI(S)) and (C ≠ {} ) do
    x ← SELEKSI(C) { pilih sebuah kandidat dari C}
    C ← C - {x} { elemen himpunan kandidat berkurang satu }
    if LAYAK(S ∪ {x}) then
      S ← S ∪ {x}
    endif
  endwhile
  {SOLUSI(S) or C = {} }
  if SOLUSI(S) then
    return S
  else
    write('tidak ada solusi')
  endif
```

Gambar 2.3. *pseudo-code* algoritma *greedy*

Ada kalanya optimum global belum tentu merupakan solusi optimum, tetapi dapat merupakan solusi *sub-optimum* atau *pseudo-optimum.*

2.2 Algoritma Kruskal

Pada algoritma Kruskal, sisi-sisi graf diurutkan terlebih dahulu berdasarkan bobotnya dari kecil ke besar. Sisi yang dimasukkan kedalam himpunan T adalah sisi graf G sedemikian sehingga T adalah pohon. Pada keadaan awal, sisi-sisi sudah diurut berdasarkan bobot membentuk pohon, masing-masing pohon di hutan hanya berupa satu buah simpul. Hutan tersebut dinamakan hutan merentang. Sisi dari graf G ditambahkan ke T jika ia tidak membentuk siklus di T.

Langkah-langkah pada algoritma Kruskal adalah sebagai berikut:

(langkah 0: sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya dari bobot kesil ke bobot besar)

1. Langkah 1 : T masih kosong
2. Langkah 2: pilih sisi (u,v) dengan bobot minimum yang tidak membentuk sirkuit di T. Tambahkan (u,v) ke dalam T
3. Langkah 3: ulangi langkah 2 sebanyak n-1 kali

Dalam notasi *pseudo-code* , algoritma Kruskal dituliskan sebagai berikut:

```
function Kruskal(input E : himpunan_sisi) → himpunan_sisi
{ menghasilkan pohon merentang minimum.
Asumsi: sisi-sisi di dalam graf sudah diurut menaik berdasarkan bobotnya, dari bobot kecil ke bobot besar }

Deklarasi
   T : himpunan_sisi
   E : sisi

Algoritma
   S → {}
   while (T(V, S) not pohon merentang) and (E ≠ {}) do
      e ← sisi yang mempunyai bobot terkecil di dalam E
      E ← E - {e}
      if T ∪ e tidak membentuk sirkuit then
        S ← S ∪ {e}
      endif
   endwhile
   ( T(V, S) pohon merentang) or (E = {})
   return S
```

Gambar 2.4. Algoritma Kruskal

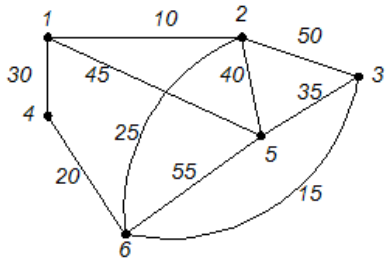
Mengingat pohon merentang dengan n simpul mempunyai n-1 buah sisi, maka kalang while di dalam algoritma diatas dilakukan sebanyak n-1 kali. Dengan demikian, algoritmanya menjadi lebih elegan sebagai berikut:

```
function Kruskal(input E : himpunan_sisi)
→ himpunan_sisi
{ menghasilkan pohon merentang minimum.
Asumsi: sisi-sisi di dalam graf sudah
diurut menaik berdasarkan bobotnya, dari
bobot kecil ke bobot besar }

Deklarasi
    T : himpunan_sisi
    E : sisi

Algoritma
    S → {}
    while (jumlah sisi di dalam S < n-1)
and (E ≠ {}) do
       e ← sisi yang mempunyai bobot
terkecil di dalam E
       E ← E - {e}
       if T ∪ e tidak membentuk sirkuit
then
        S ← S ∪ {e}
       endif
    endwhile
    ( T(V, S) pohon merentang) or (E = {})
    return S
```

Gambar 2.5. Algorima kruskal setelah mengalami perubahan

Adapun penerapan dari algoritma ini adalah sebagai berikut.

Diberikan sebuah graf G.



Gambar 2.6. Graf G

Penyelesaian :

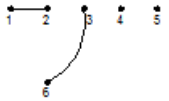
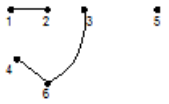
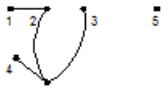
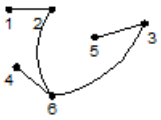
Sisi-sisi graf diurut menaik berdasarkan bobotnya:

Sisi	(1,2)	(3,6)	(4,6)	(2,6)	(1,4)	(3,5)	(2,5)	(1,5)	(2,3)	(5,6)
Bobot	10	15	20	25	30	35	40	45	50	55

Gambar 2.7. Tabel bobot

Langkah-langkah pembentukan pohon merentang minimum diperlihatkan pada Tabel 2.1. bobot pohon merentang minimum ini adalah 10 + 25 + 15 + 20 + 35 = 105.

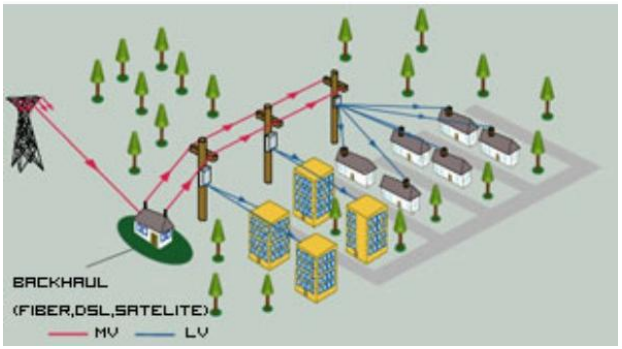
Tabel 2.1 Tabel pembentukan pohon merentang minimum

Langkah	Sisi	Bobot	Hutan merentang
0			
1	(1, 2)	10	
2	(3, 6)	15	
3	(4, 6)	20	
4	(2, 6)	25	
5	(1, 4)	30	ditolak
6	(3, 5)	35	

3. Distribusi Listrik

Sistem jaringan distribusi listrik dapat diilustrasikan sebagai graf. Sisi-sisi menggambarkan hubungan antara gardu listrik dengan rumah-rumah pelanggan atau gardu listrik dengan gardu listrik utama atau gardu utama dengan sumber pembangkit listrik. Simpul-simpulnya menggambarkan rumah pelanggan, gardu listrik, gardu listrik utama atau sumber pembangkit listrik. Graf yang diilustrasikan ini berupa graf berbobot yang tak berarah.

Pada pendistribusian listrik, listrik harus didistribusikan dari sumber pembangkit listrik ke pelanggan. Prosesnya harus melalui gardu-gardu listrik. Dimulai dari pembangkit listrik yang mengalirkan ke gardu-gardu utama, kemudian dari gardu-gardu utama dialirkan ke gardu-gardu listrik hingga akhirnya sampai ke rumah-rumah warga atau pabrik.

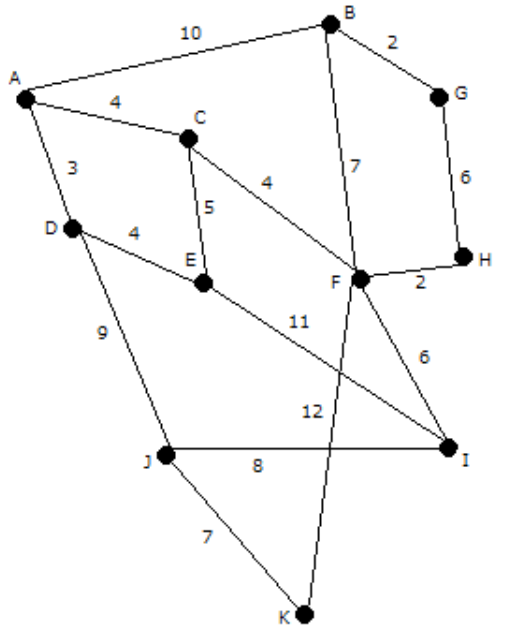


Gambar 2.8. Distribusi listrik

Karena jalur pendistribusian listrik harus seperti itu, maka harus dipikirkan jalur terpendek penyampaian listrik ke pengguna sehingga biaya pendistribusian listrik menjadi minimum. Penyelesaian permasalahan ini sangat cocok dengan pohon merentang minimum.

III. STUDI KASUS

Ada sebuah kota yang baru akan membuat jaringan distribusi listrik baru. Diperlukan cara untuk membuat biaya yang dikeluarkan dari pembuatan jaringan distribusi listrik tersebut menjadi minimum. Berikut adalah gambar dari jaringan yang akan dibuat:



Gambar 3.1. Studi kasus jaringan distribusi listrik

Menurut langkah algoritma Kruskal, langkah awal adalah mengurutkan bobot sisi-sisi graf.

Tabel 3.1 tabel pengurutan bobot

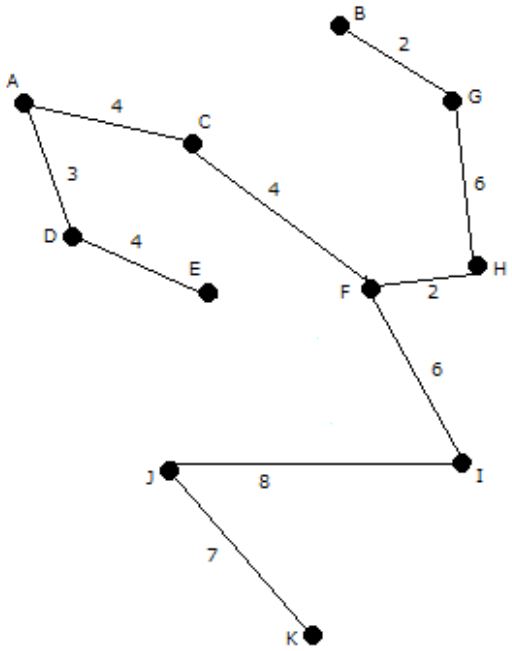
Sisi	Bobot
(B,G)	2
(F,H)	2
(A,D)	3
(A,C)	4
(D,E)	4
(C,F)	4
(C,E)	5
(G,H)	6
(F,I)	6
(B,F)	7
(J,K)	7
(J,I)	8
(D,J)	9
(A,B)	10
(E,I)	11
(F,K)	12

Setelah itu diambil sisi-sisi yang membuat pohon merentang minimum. Karena graf yang digambar bukan graf berarah, maka simpulnya dapat dibalik. Contohnya : (A,B) = (B,A).

Berikut adalah sisi-sisi yang terambil.

1. (F,H) dengan bobot 2
2. (B,G) dengan bobot 2
3. (A,D) dengan bobot 3
4. (C,F) dengan bobot 4
5. (D,E) dengan bobot 4
6. (A,C) dengan bobot 4
7. (F,I) dengan bobot 6
8. (G,H) dengan bobot 6
9. (J,K) dengan bobot 7
10. (I,J) dengan bobot 8

Maka jadilah pohon merentang minimum dengan bobot pohon merentang minimum adalah 2 + 2 + 3 + 4 + 4 + 4 + 6 + 6 + 7 + 8 = 46.



Gambar 3.2. Pohon merentang minimum

Jadi, jaringan distribusi yang dibuat adalah seperti ini dengan F adalah pembangkit listrik; C, H, I adalah gardu listrik utama; A, J, G adalah gardu listrik; dan E, K, B adalah rumah warga atau pabrik.

V. CONCLUSION

Jaringan distribusi listrik dapat diilustrasikan dengan menggunakan graf. Dan algoritma Kruskal cocok dipakai untuk pembuatan jaringan distribusi listrik. Dengan menggunakan algoritma Kruskal, jaringan distribusi listrik yang dibuat memiliki biaya yang minimum.

REFERENSI

[1] Munir, Rinaldi, Diktat Kuliah Matematika Diskrit, Program Studi Teknik Informatika Institut Teknologi Bandung, 2003.

[2] Munir, Rinaldi, Diktat Kuliah IF3051 Strategi Algoritma, Program Studi Teknik Informatika Institut Teknologi Bandung 2009.

[3] http:/daewo.files.wordpress.com/2007/05, tanggal akses : 28 November 2010.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 November 2010

ttd

Maureen Linda Caroline – 13508049

Lampiran :

Source code algoritma kruskal dalam bahasa C:

```
/* file : kruskal.h */
/* author : Maureen Linda Caroline - 13508049 */

#ifndef kruskal_h
#define kruskal_h

#include <stdio.h>

#define infinity 9999

int c;                          //angka pembanding
int MatriksBobot[100][100];     //matriks ini untuk menyimpan bobot.
int MatriksTitik[100][100];     //matriks ini untuk menyimpan titik-titik yang nantinya akan di
ambil satu persatu hingga dieprlukan matriks untuk mengetahui titik mana saja yang sudah diambil.
int n;                          //untuk ukuran matriks. matriks selalu persegi.
int ArrayRela[100];             //untuk membuat hubungan. agar tidak ada siklus. pada akhirnya
semua ArrayRela berisi dengan 1 yang menandakan sudah terubung secara minimal.
int GaNol;                      //untuk menunjukkan berapah banyak titik yang memiliki bobot
int Node1,Node2;

void MinBobot(int n);           //mencari titik di matriks bobot yang terkecil
void Init_Rela(int a);          //menginisialisasi ArrayRela
void MakeARela(int a, int b);   //Membuat relasi antar titik sehingga tidak ada siklus
void ReadMatriksFile();         //membaca matriks dari File. Matriks harus di buat pencerminan.
maksudnya matriks[baris][kolom] = matriks [kolom][baris]. matriks[baris][baris]=0
void PrintMatriks(int n);       //untuk mencetak matriks bobot dan matriks titik
#endif
```

Lampiran 1. *Source code* kruskal.h

```
/* file : kruskal.c */
/* author : Maureen Linda Caroline - 13508049 */

#include "kruskal.h"

void MinBobot(int n)
{
    int baris,kolom;
    for (baris=0; baris<=n-1; ++baris)
    {
        for (kolom=0; kolom<= baris; ++kolom)
        {
            if (MatriksTitik[baris][kolom]==1)
            {
                if ((MatriksBobot[baris][kolom] <= c) && (MatriksBobot[baris][kolom] >= 1))
                {
                    c=MatriksBobot[baris][kolom];
                    Node1=baris;
                    Node2=kolom;
                }
            }
        }
    }
    MatriksTitik[Node1][Node2] = 0;
    c = infinity;
}

void Init_Rela(int a)
{
    int i=1;
    for (i; i<=n; ++i)
    {
        ArrayRela[i]=i;
    }
}

void MakeARela(int a, int b)
{
    int i,temp;
    temp = ArrayRela[b];
    ArrayRela[b]=ArrayRela[a];
    for (i=0;i<=n;++i)
    {
        if (ArrayRela[i]==temp)
        {
```

```
            ArrayRela[i] = ArrayRela[a];
        }
    }
}

void ReadMatriksFile()
{
    int baris, kolom;
    int nilai;
    GaNol=0;
    FILE *f = fopen("short.txt", "r");
    fscanf (f, "%d", &n);
    for (baris = 0; baris < n; ++baris)
    {
        for (kolom = 0; kolom < n; ++kolom)
        {
            fscanf (f, "%d", &MatriksBobot[baris][kolom]);
            if (MatriksBobot[baris][kolom]==0)
            {
                MatriksTitik[baris][kolom]=0;
            }
            else
            {
                MatriksTitik[baris][kolom]=1;
                GaNol+=1;
            }
        }
    }
    fclose(f);
    for (baris=0; baris<n; ++baris)
    {
        for (kolom=baris; kolom<n; ++kolom)
        {
            MatriksBobot[baris][kolom] = -1;
            MatriksTitik[baris][kolom] = 0;
        }
    }

}

void PrintMatriks(int n)
{
    int baris, kolom;
    printf("\nMatriks yang ditampilkan hanya berbentuk segitiga karena matriks tersebut merupakan
matriks simetri. \n");
    printf("Matriks bobot : \n");
    for (baris=0; baris<n; ++baris)
    {
        for(kolom=0; kolom<baris; ++kolom)
        {
            printf ("    %d", MatriksBobot[baris][kolom]);
        }
        printf("\n");
    }
    printf("\nMatriks titik : \n");
    for (baris=0; baris<n; ++baris)
    {
        for(kolom=0; kolom<baris; ++kolom)
        {
            printf ("    %d", MatriksTitik[baris][kolom]);
        }
        printf("\n");
    }
    printf("\n");
}
```

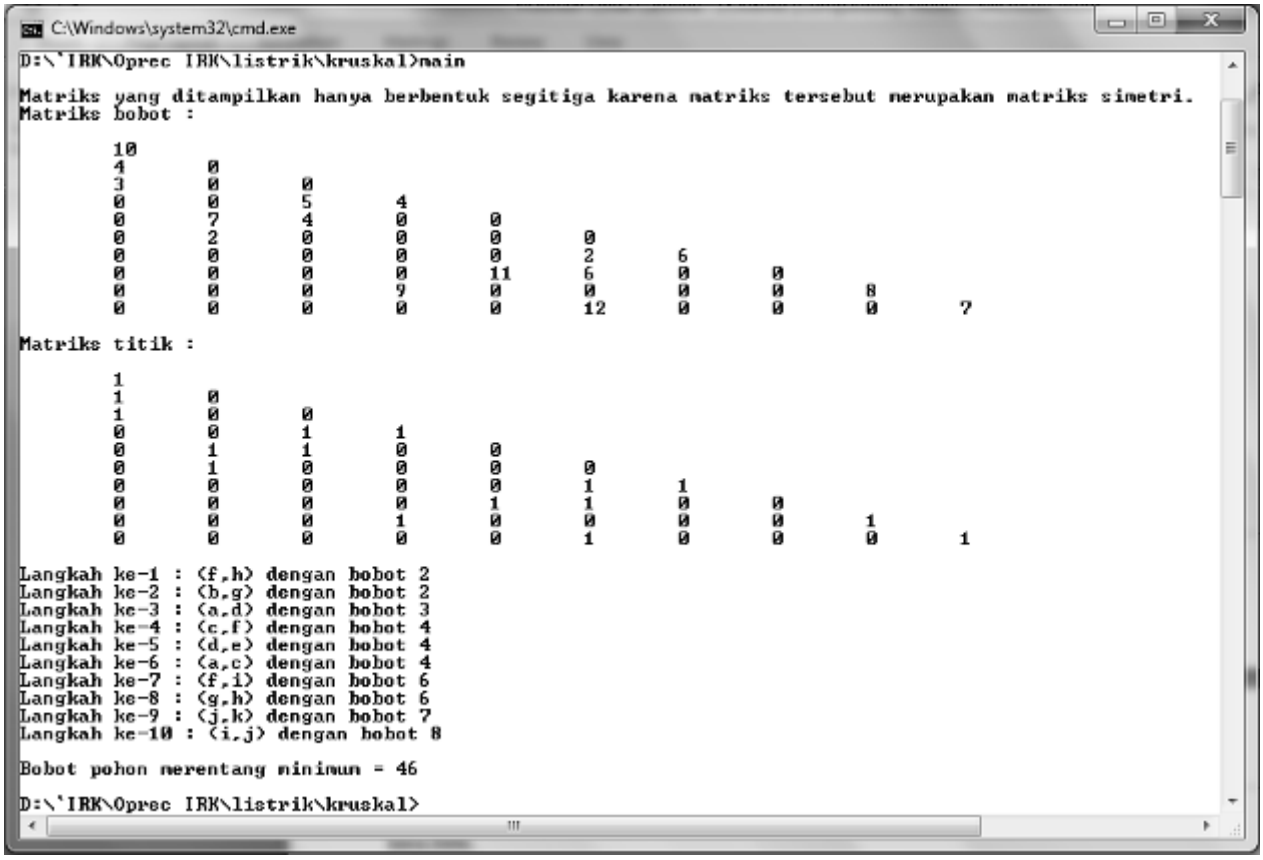
Lampiran 2. *Source code* kruskal.c

```
/* file : kruskalmain.c */
/* author : Maureen Linda Caroline - 13508049 */

#include "kruskal.h"

int main()
{
        int i=1;
        int k=0;
        int sum = 0;
        ReadMatriksFile();
        PrintMatriks(n);
        MinBobot(n);
```

```
        Init_Rela(n);
        for (; i<= (GaNol / 2); ++i)
        {
                MinBobot(n);
                if (ArrayRela[Node1]!=ArrayRela[Node2])
                {
                        k+=1;
                        sum+=MatriksBobot[Node1][Node2];
                        MakeARela(Node1,Node2);
                        printf("Langkah ke-%d : (%c,%c) dengan bobot %d\n",
k,(char)(Node2)+'a',(char)(Node1)+'a', MatriksBobot[Node1][Node2]);
                }
        }
        printf ("\nBobot pohon merentang minimum = %d\n", sum);
        return 0;
}
```

Lampiran 3. Source code kruskalmain.c

Hasil output soal studi kasus:



Lampiran 4. Output program