Tabel 4.2 Penghitungan *Minimal Spanning Tree* Menggunakan Algoritma Prim

Iterasi	W	V - W	Garis-garis yang dilalui (e)	Bobot (keputusan)	Verteks berikutnya
1	$\{v_1\}$	$\{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$	$(v_1, v_2) = 0{,}0965$, $(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$	0,0965 (terima)	v_2 karena (v_1, v_2) minimum
2	$\{v_1, v_2\}$	$\{v_3, v_4, v_5, v_6, v_7, v_8\}$	$(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$, $(v_2, v_3) = 0{,}1012$, $(v_2, v_4) = 0{,}0971$, $(v_2, v_5) = 0{,}0949$, $(v_2, v_7) = 0{,}0939$, $(v_2, v_8) = 0{,}0950$	0,0939 (terima)	v_7 karena (v_2, v_7) minimum
3	$\{v_1, v_2, v_7\}$	$\{v_3, v_4, v_5, v_6, v_8\}$	$(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$, $(v_2, v_3) = 0{,}1012$, $(v_2, v_4) = 0{,}0971$, $(v_2, v_5) = 0{,}0949$, $(v_2, v_8) = 0{,}0950$	0,0949 (terima)	v_5 karena (v_2, v_5) minimum

Tabel 4.2 lanjutan

4	$\{v_1, v_2, v_7, v_5\}$	$\{v_3, v_4, v_6, v_8\}$	$(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$, $(v_2, v_3) = 0{,}1012$, $(v_2, v_4) = 0{,}0971$, $(v_2, v_8) = 0{,}0950$, $(v_3, v_5) = 0{,}1164$	0,0950 (terima)	v_8 karena (v_2, v_8) minimum.
5	$\{v_1, v_2, v_7, v_5, v_8\}$	$\{v_3, v_4, v_6\}$	$(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$, $(v_2, v_3) = 0{,}1012$, $(v_2, v_4) = 0{,}0971$, $(v_3, v_5) = 0{,}1164$	0,0971 (terima)	v_4 karena (v_2, v_4) minimum
6	$\{v_1, v_2, v_7, v_5, v_8, v_4\}$	$\{v_3, v_6\}$	$(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$, $(v_2, v_3) = 0{,}1012$, $(v_3, v_5) = 0{,}1164$, $(v_4, v_3) = 0{,}0863$	0,0863 (terima)	v_3 karena (v_4, v_3) minimum

Tabel 4.2 lanjutan

7	$\{v_1, v_2, v_7, v_5, v_8, v_4, v_3\}$	$\{v_6\}$	$(v_1, v_3) = 0{,}1005$, $(v_1, v_4) = 0{,}1231$, $(v_1, v_5) = 0{,}1203$, $(v_1, v_6) = 0{,}1023$, $(v_1, v_7) = 0{,}1068$, $(v_1, v_8) = 0{,}1054$, $(v_2, v_3) = 0{,}1012$, $(v_3, v_5) = 0{,}1164$,	0,1005 dan 0,1012 (tolak). 0,1023 (terima)	v_6 karena (v_1, v_6) minimum. (v_1, v_3) dan (v_2, v_3) membentuk *cycle*
8	$\{v_1, v_2, v_7, v_5, v_8, v_4, v_3, v_6\}$ Selesai karena semua titik telah dievaluasi semua	{}	-	Total bobot = 0,666	-

Keterangan: Terima = jika tidak membentuk *cycle* (*acyclic*),
Tolak = jika membentuk *cycle*.

Lampiran 5. Matriks Berbobot dari Probabilitas Tersambung Antar Sentral Telepon

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	0,8007	0,7934	0,7532	0,7581	0,7901	0,7819	0,7845
v_2	0,8007	0	0,7922	0,7997	0,8036	∞	0,8055	0,8035
v_3	0,7934	0,7922	0	0,8198	0,7648	∞	∞	∞
v_4	0.7532	0,7997	0,8198	0	∞	∞	∞	∞
v_5	0,7581	0,8036	0,7648	∞	0	∞	∞	∞
v_6	0,7901	∞	∞	∞	∞	0	∞	∞
v_7	0,7819	0,8055	∞	∞	∞	∞	0	∞
v_8	0,7845	0,8035	∞	∞	∞	∞	∞	0

Keterangan:

- Sentral Malang Kota (MLK) = v_1
- Sentral Malang Belimbing = v_2
- Sentral Klojen = v_3
- Sentral Gadang = v_4
- Sentral Sawojajar = v_5
- Sentral Sumber Pucung = v_6
- Sentral Tumpang = v_7
- Sentral Singosari = v_8
- 0 jika i=j
- ∞ = tidak terdapat hubungan antara v_i dan v_j.

Lampiran 6. Matriks Berbobot dari Probabilitas Hilangnya Panggilan Akibat Kanal Yang Penuh Antar Sentral Telepon

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	0,0965	0,1005	0,1231	0,1203	0,1023	0,1068	0,1054
v_2	0,0965	0	0,1012	0,0971	0,0949	∞	0,0939	0,0950
v_3	0,1005	0,1012	0	0,0863	0,1164	∞	∞	∞
v_4	0,1231	0,0971	0,0863	0	∞	∞	∞	∞
v_5	0,1203	0,0949	0,1164	∞	0	∞	∞	∞
v_6	0,1023	∞	∞	∞	∞	0	∞	∞
v_7	0,1068	0,0939	∞	∞	∞	∞	0	∞
v_8	0,1054	0,0950	∞	∞	∞	∞	∞	0

Keterangan:

- Sentral Malang Kota (MLK) = v_1
- Sentral Malang Belimbing = v_2
- Sentral Klojen = v_3
- Sentral Gadang = v_4
- Sentral Sawojajar = v_5
- Sentral Sumber Pucung = v_6
- Sentral Tumpang = v_7
- Sentral Singosari = v_8
- 0 jika i=j
- ∞ = tidak terdapat hubungan antara v_i dan v_j.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dewasa ini, perkembangan ilmu pengetahuan dan teknologi sangat pesat, hal ini tidak lepas dari peran ilmu matematika. Ilmu ini mempunyai sifat fleksibel, dalam arti dapat dimanfaatkan dan diaplikasikan dalam berbagai cabang disiplin ilmu lainnya. Salah satu teori yang dapat diaplikasikan adalah teori graf.

Dalam kehidupan sehari-hari terdapat permasalahan mengenai optimasi yang dapat diselesaikan menggunakan *minimal spanning tree* misalnya, masalah jarak terpendek dalam pembangunan jalan, jaringan telepon selular, maupun jaringan listrik. Penyelesaian masalah-masalah ini pada dasarnya menentukan terjadinya semua spanning tree yang mungkin dan memperhitungkan *minimal spanning tree* dalam menentukan *bobot* terkecilnya.

Kebutuhan masyarakat akan sarana telepon semakin meningkat. Peningkatan kebutuhan akan sarana telepon ini ditandai dengan semakin padatnya lalu lintas (trafik) pada kanal antar sentral yang melebihi kapasitas kanal. Hal ini mengakibatkan terjadinya kegagalan pembentukan hubungan antara sentral.

Untuk itu diperlukan suatu algoritma yang tepat untuk menentukan probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral sehingga didapatkan probabilitas tersambung terbesar antar sentral. Berdasarkan uraian di atas dan mengingat pentingnya aplikasi graf dalam menentukan bobot yang minimal, maka dalam skripsi ini, memotivasi penulis untuk membahas secara khusus bagaimana cara memperoleh *minimal spanning tree* dalam suatu graf sederhana terhubung, terboboti, dan tidak berarah.

1.2 Rumusan Masalah

Beberapa pokok permasalahan yang akan dipecahkan dalam penulisan skripsi ini adalah:

1. Bagaimana menentukan *minimal spanning tree* dengan menggunakan Algoritma Kruskal pada jaringan konfigurasi *Multi Exchange Area* (MEA) Malang?
2. Bagaimana menentukan *minimal spanning tree* dengan menggunakan Algoritma Prim pada jaringan konfigurasi *Multi Exchange Area* (MEA) Malang?
3. Bagaimana perbandingan antara Algoritma Kruskal dan Algoritma Prim dalam penentuan *minimal spanning tree* pada jaringan konfigurasi *Multi Exchange Area* (MEA) Malang?

1.3 Batasan Masalah

Agar pembahasan yang dilakukan lebih terfokus, maka dibuat batasan-batasan masalah yaitu

1. Graf yang dibicarakan adalah graf sederhana, tidak berarah (*undirect graph*), memiliki bobot (*weighted graph*), dan terhubung (*connected graph*),
2. Semua sentral yang dilalui dalam keadaan sibuk.

1.4 Tujuan

Tujuan dari penulisan skripsi ini adalah:

1. Menentukan *minimal spanning tree* dengan menggunakan Algoritma Kruskal pada jaringan konfigurasi *Multi Exchange Area* (MEA) Malang,
2. Menentukan *minimal spanning tree* dengan menggunakan Algoritma Prim pada jaringan konfigurasi *Multi Exchange Area* (MEA) Malang,

3. Membandingkan antara Algoritma Kruskal dan Algoritma Prim dalam penentuan *minimal spanning tree* pada jaringan konfigurasi *Multi Exchange Area* (MEA) Malang.

1.5 Manfaat

Manfaat dari penulisan skripsi ini adalah:

1. Bagi penulis, tulisan ini merupakan kesempatan bagi penulis untuk menambah pengetahuan tentang masalah sebenarnya yaitu perbandingan dan penggunaan teori-teori yang diterima di bangku kuliah terutama tentang Algoritma Kruskal dan Algoritma Prim,
2. Bagi pihak lain, tulisan ini semoga dapat memberikan wacana, tambahan pengetahuan, maupun sebagai bahan pembanding bagi pihak yang ingin mengetahui lebih banyak tentang *minimal spanning tree*.

BAB II
TINJAUAN PUSTAKA

2.1 Graf Secara Umum

Teori graf adalah salah satu teori dalam matematika yang cukup luas yang memiliki aplikasi di berbagai bidang ilmu yang lain seperti bidang ilmu komunikasi, transportasi, dll.

Definisi 2.1.1

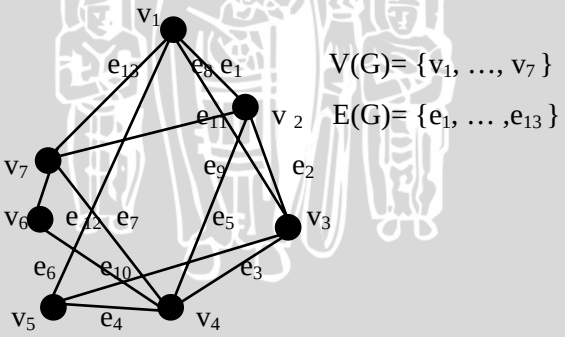
Graf G terdiri dari himpunan tidak kosong dari elemen-elemen yang disebut titik (*vertex*) dan himpunan dari elemen- elemen yang disebut garis (*edge*). Graf G adalah pasangan himpunan (V(G), E(G)) yang dinotasikan dalam bentuk G={V(G),E(G)} dengan:

V(G) adalah himpunan titik yang tidak kosong yang jumlahnya berhingga, dan

E(G) adalah himpunan garis yang dapat merupakan himpunan kosong

(Chartrand dan Ortrud, 1993).

Contoh 2.1:



Gambar 2.1 Graf dengan tujuh titik dan tiga belas garis

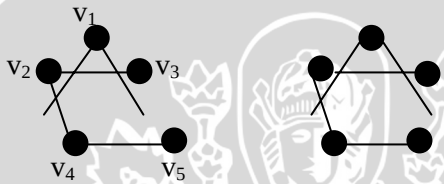
Definisi 2.1.2

Graf sederhana (*simple graph*) adalah graf yang tidak mempunyai garis paralel atau *loop* (Chartrand dan Ortrud, 1993).

Definisi 2.1.3

Suatu graf G dengan n titik disebut graf berlabel jika setiap titik pada graf diberi label dengan bilangan bulat dari 1 sampai n (Wilson dan Watkins, 1990).

Contoh 2.2:



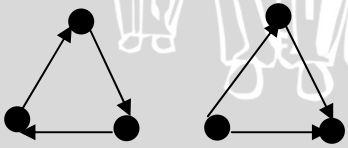
(a) (b)

Gambar 2.2 Graf berlabel (a) dan graf tidak berlabel (b)

Definisi 2.1.4

Graf berarah (*digraph*) adalah graf yang terdiri dari himpunan titik- titik V dan himpunan garis- garis E yang mempunyai orientasi arah (Doerr dan Levvaseur, 1991).

Contoh 2.3:



(a) (b)

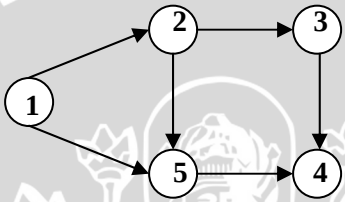
Gambar 2.3 Dua digraf dengan tiga titik dan tiga garis

Definisi 2.1.5

Suatu jaringan merupakan himpunan titik-titik atau node-node yang dihubungkan oleh busur. Secara umum node-node ini ditandai dengan angka ataupun huruf tunggal, sedangkan busur ditandai dengan pasangan angka atau huruf.

G=(N,A) merupakan notasi standar untuk menggambarkan sebuah jaringan dimana N menyatakan himpunan node sedangkan A menyatakan himpunan busur (Dimitri dan Robert, 1992).

Contoh 2.4:



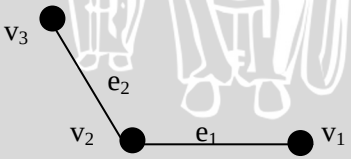
Gambar 2.4 Jaringan dengan lima node dan enam busur.

2.2 Terminologi dalam Graf

Definisi 2.2.1

Jika dua titik v_1 dan v_2 dihubungkan oleh suatu garis e, maka dikatakan v_1 dan v_2 terhubung langsung (*adjacent*) (Wilson & Watkins, 1990).

Contoh 2.5:



Gambar 2.5 Graf yang *adjacent*

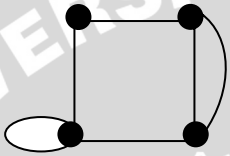
Definisi 2.2.2

Misal v dan w adalah titik- titik dari graf G dan e adalah garis pada graf G yang menghubungkan titik v dan w, maka kita katakan bahwa e *incident* dengan titik v dan w (Roman S, 1989).

Definisi 2.2.3

Loop adalah garis yang berawal dan berakhir pada suatu titik yang sama (Wilson & Watkins, 1990).

Contoh 2.6:



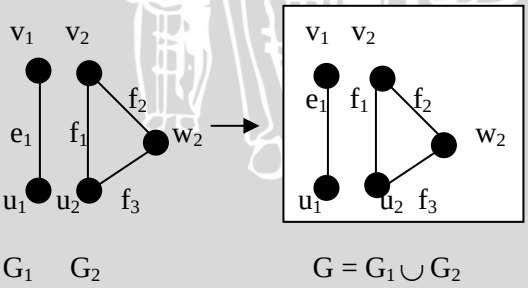
Gambar 2.6 Graf yang memuat *loop* dan *multiple edge*

2.3 Operasi Pada Graf

Definisi 2.3.1

Gabungan dari dua graf sederhana $G_1 = (V_1, E_1)$ dan $G_2 = (V_2, E_2)$ dengan $V_1 \cap V_2 = \phi$ dinotasikan $G = G_1 \cup G_2 = (V, E)$ dimana $V = V_1 \cup V_2$ dan $E = E_1 \cup E_2$ (Marsudi, 2006).

Contoh 2.7:



G_1 G_2 $G = G_1 \cup G_2$

Gambar 2.7 Graf yang memuat $G_1 \cup G_2$

Pada Gambar 2.7 terdapat dua graf sederhana yaitu G_1 dengan $V_1 = \{v_1, u_1\}$ dan $E_1 = \{e_1\}$ dan G_2 dengan $V_2 = \{v_2, u_2, w_2\}$

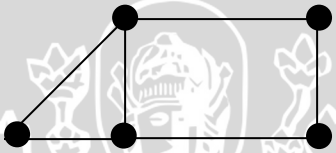
dan E_2= {f_1, f_2, f_3} sehingga didapatkan gabungan dari dua graf tersebut adalah $G = G_1 \cup G_2 = (V, E)$ dengan V= {v_1, u_1, v_2, u_2, w_2} dan E= {e_1, f_1, f_2, f_3}.

2.4 Graf Terhubung, Graf Berbobot, dan Subgraf

Definisi 2.4.1

Graf dikatakan terhubung (*connected*) jika diantara dua titik yang berbeda terdapat sedikitnya satu *path* (Foldes, 1993).

Contoh 2.8:

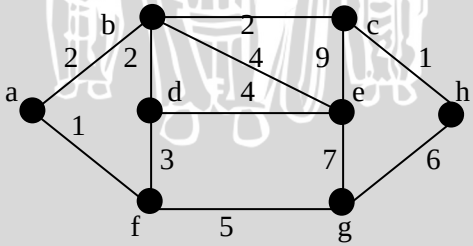


Gambar 2.8 Graf yang terhubung

Definisi 2.4.2

Graf berbobot (*weighted graph*) adalah graf yang mempunyai nilai atau bobot pada setiap garisnya (*edge*). Dan bobot *edge* dalam graf dinotasikan dengan W(e) (Fletcher, Hoyle, dan Patty, 1991).

Contoh 2.9:

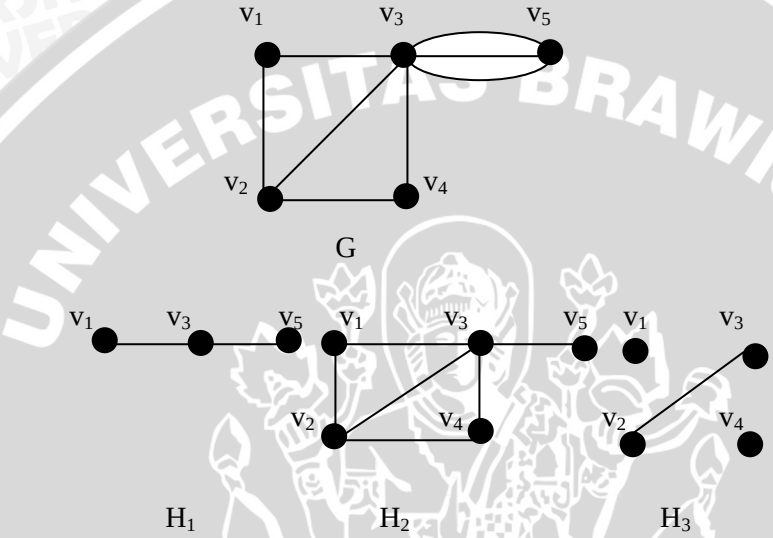


Gambar 2.9 Graf berbobot

Definisi 2.4.3

Graf H disebut subgraf dari graf G jika setiap titik dari graf H juga merupakan titik dari graf G dan setiap garis pada graf H juga merupakan garis pada graf G. Yang dinotasikan, H subgraf dari G jika $V(H) \subset V(G)$ dan $E(H) \subset E(G)$ (Roman, 1989).

Contoh 2.10:

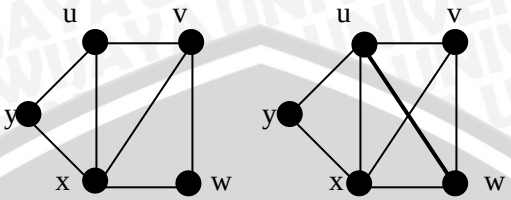


Gambar 2.10 Graf G dan tiga subgraf H

Definisi 2.4.4

Jika titik-titik u dan w tidak *adjacent* dalam graf G, penambahan garis uw menghasilkan supergraf terkecil dari G yang memuat garis uw dan dilambangkan dengan $G + uw$ (http://www.ecp6.jussieu.fr).

Contoh 2.11:



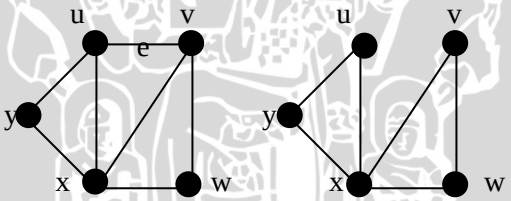
(a) (b)

Gambar 2.11 Graf G (a) dan supergraf terkecil G+uw (b)

Definisi 2.4.5

Jika e merupakan garis dari suatu graf G, maka G-e menyatakan graf yang diperoleh dari graf G dengan menghapus garis e. Penghapusan suatu garis e dari graf G menghasilkan *spanning* subgraf G-e yang memuat garis dari G kecuali garis e. Jadi G-e adalah subgraf maksimal dari G yang tidak memuat e (Marsudi, 2006).

Contoh 2.12:



(a) (b)

Gambar 2.12 Graf G (a) dan subgraf maksimal G-e (b)

2.5 Representasi Graf Dalam Matriks

Definisi 2.5.1

Matriks *adjacency* dari graf berlabel G dengan $V(G)=\{v_1, \dots, v_n\}$ adalah $A = [a_{ij}]_{n\times n}$ dengan:

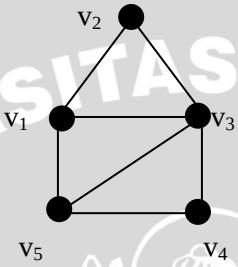
$$\begin{cases} 1, & v_i \textit{ adjacent} \text{ dengan } v_j \end{cases}$$

a_{ij} =

0, v_i tidak *adjacent* dengan v_j

(Marsudi, 2006).

Contoh 2.13:



G

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Gambar 2.13 Graf G dan matriks *adcacency* A

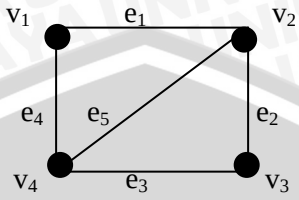
Definisi 2.5.2

Matriks *incident* dari graf berlabel dengan V(G)={v_1, … , v_p} dan E(G)={e_1, … ,e_q } adalah $B = [b_{ij}]_{p \times q}$ dengan:

$$b_{ij} = \begin{cases} 1, & v_i \textit{ incident} \text{ dengan } e_j \\ 0, & v_i \text{ tidak } \textit{incident} \text{ dengan } e_j \end{cases}$$

(Marsudi, 2006).

Contoh 2.14:



G

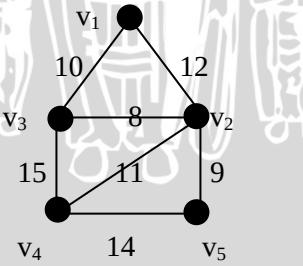
$$B=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Gambar 2.14 Graf G dan matriks *incident* B

Definisi 2.5.3

Matriks yang bersesuaian dengan graf berbobot G adalah matriks jarak $A=(a_{ij})$ dengan a_{ij}= bobot garis yang menghubungkan titik v_i dengan titik v_j. Jika titik v_i tidak *adjacent* dengan titik v_j maka $a_{ij}=\infty$ dan $a_{ij}=0$ jika i=j (Siang, 2002).

Contoh 2.15:



G

v_1 v_2 v_3 v_4 v_5

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 \\ 0 & 12 & 10 & \infty & \infty \\ 12 & 0 & 8 & 11 & 9 \\ 10 & 8 & 0 & 15 & \infty \\ \infty & 11 & 15 & 0 & 14 \\ \infty & 9 & \infty & 14 & 0 \end{bmatrix}$$

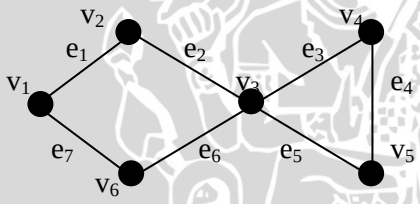
Gambar 2.15 Graf berbobot G dan matriks berbobot A

2.6 *Walk, Trail, Circuit, Path,* dan *Cycle*

Definisi 2.6.1

Walk dari titik u menuju titik v adalah urutan alternatif dari titik-titik dan garis-garis dari G yang dimulai dari titik u dan berakhir pada titik v, sehingga setiap garis *incident* dengan titik yang terdahulu dan titik yang kemudian (Roman, 1989).

Contoh 2.16:



Gambar 2.16 Graf yang mempunyai *walk*

Dari Gambar 2.16 didapatkan *walk* dari v_1 sampai v_5, urutannya adalah (v_1, e_7, v_6, e_6, v_3, e_3, v_4, e_4, v_5).

Definisi 2.6.2

Trail dari titik u menuju titik v adalah walk dari u menuju v dimana tidak ada garis yang dilalui lebih dari satu kali (Roman, 1989).

Dari Gambar 2.16 dihasilkan *trail* dengan urutannya adalah (v_2, e_2 v_3, e_3, v_4, e_4, v_5, e_5, v_3).

Definisi 2.6.3

Circuit adalah *trail* yang dimulai dan berakhir pada titik yang sama (Roman, 1989).
Dari Gambar 2.16 dihasilkan *circuit* dengan urutannya adalah (v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_5, e_5, v_3, e_6, v_6, e_7,v_1).

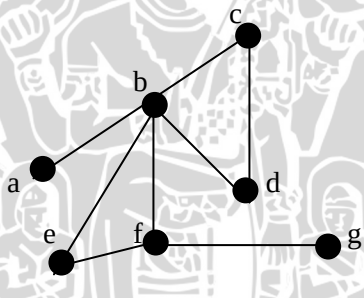
Definisi 2.6.4

Path dari titik u menuju ke titik v adalah *walk* dari titik u menuju titik v dimana tidak ada titik yang dilalui lebih dari satu kali (Roman, 1989).
Dari Gambar 2.16 dihasilkan *path* dengan urutannya adalah (v_1, e_1, v_2, e_2, v_3, e_5, v_5).

Definisi 2.6.5

Cycle adalah *path* yang dimulai dan diakhiri pada titik yang sama (Johnsonbaugh, 1984).

Contoh 2.17:



Gambar 2.17 Graf yang mempunyai *cycle*

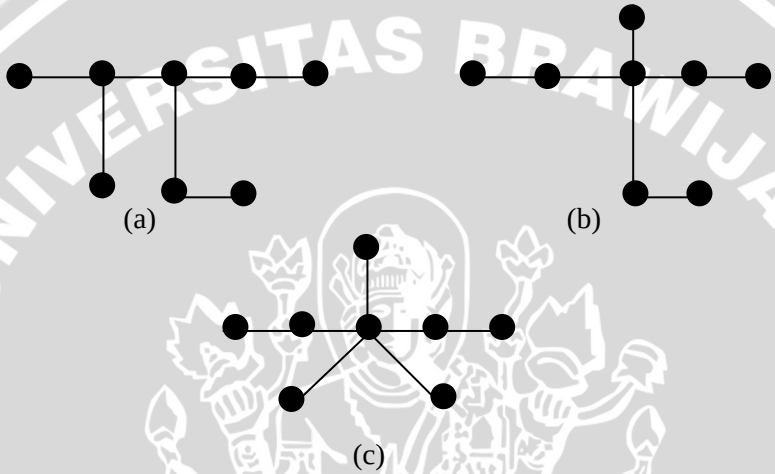
Dari Gambar 2.16 didapatkan *cycle* dengan urutan (e, f, b, e).

2.7 *Tree* (Pohon), *Forest* dan *Cut Edge*

Definisi 2.7.1

Sebuah graf dikatakan *acyclic* jika graf tersebut tidak mempunyai *cycle. Tree* (pohon) adalah graf terhubung tidak berarah yang *acyclic* (Harary, 1994).

Contoh 2.18:



Gambar 2.18 (a), (b), dan (c) adalah *tree* dengan delapan titik

Teorema 2.7.1

G graf dengan p titik dan q garis maka kelima pernyataan berikut adalah ekuivalen:

1. G adalah *tree*;
2. Setiap dua titik dari G dihubungkan dengan suatu *path* yang tunggal;
3. G terhubung dan p=q+1;
4. G *acyclic* dan p=q+1;
5. G *acyclic* dan jika dua titik (sebarang) tidak *adjacent* di G dihubungkan dengan suatu garis e, maka G+e mempunyai tepat satu *cycle*.

(Marsudi, 2006).

Bukti:

Cukup dibuktikan: $(1) \Rightarrow (2)$, $(2) \Rightarrow (3)$, $(3) \Rightarrow (4)$, $(4) \Rightarrow (5)$, $(5) \Rightarrow (1)$

$(1 \Rightarrow 2)$ Karena G terhubung, maka setiap dua titik di G dihubungkan dengan suatu *path*. Akan ditunjukkan bahwa *path* ini adalah tunggal. Andaikan P dan P' adalah dua *path* berlainan yang menghubungkan titik u dan v di G. Misalkan w titik pertama di P dan P' dengan sifat bahwa sesudah w *path* P berbeda dengan *path* P', sedangkan w' adalah titik pertama sesudah itu, yang berada di P dan P'.

Maka segmen dari P ke P' diantara w dan w' merupakan *cycle* dari G. Kontradiksi dengan G *acyclic*. Jadi, pengandaian salah dan terbukti ada *path* yang tunggal yang menghubungkan setiap dua titik di G.

$(2 \Rightarrow 3)$ Jelas G terhubung. Karena setiap titik di G dihubungkan dengan suatu *path* (tunggal). Akan ditunjukkan $p=q+1$ (dengan induksi).

- Untuk $p=1$, berarti ada satu titik dan tidak ada garis. Untuk $p=2$ berarti ada dua titik dan satu garis. Jadi, pernyataan $p=q+1$ benar untuk graf dengan p kecil.
- Misalkan pernyataan benar untuk graf dengan jumlah titik kurang dari p titik. Apabila G mempunyai p titik, maka penghapusan suatu garis x sebarang dari G menghasilkan dua komponen dengan jumlah dari titik masing-masing kurang dari p. Untuk kedua komponen ini pernyatan berlaku (hipotesis induksi). Sehingga banyak titik di masing- masing komponen satu lebih dari pada banyaknya garis. Dalam gabungan (*union*) dari komponen-komponen itu, banyaknya titik adalah dua lebih dari pada banyaknya garis. Sehingga setelah garis x ditambahkan lagi, diperoleh bahwa banyak titik satu lebih dari pada banyaknya garis. Jadi terbukti $p=q+1$.

$(3 \Rightarrow 4)$ Cukup ditunjukkan G *acyclic*.

Andaikan G mempunyai *cycle* dengan panjang n. Maka terdapat n titik dan n garis pada *cycle* itu dan ada (p-n) titik yang tidak terletak pada *cycle*. Pada setiap titik di luar *cycle*, dapat dikaitkan dengan minimal satu titik di luar *cycle*, yaitu pada *geodesic* (path terpendek) dari titik itu ke suatu titik

pada *cycle*. Garis-garis itu berbeda. Dengan demikian banyak garis pasti lebih besar atau sama dengan banyak titik ($q \geq p$). Kontradiksi dengan p=q+1 atau p>q. Jadi, pengandaian salah dan terbukti G *acyclic*.

(4 $\Rightarrow$ 5) Karena G *acyclc*, maka setiap komponen dari G merupakan *tree*. Misalkan terdapat k komponen. Dalam setiap komponen, banyaknya titik satu lebih dari pada banyaknya garis. Maka untuk *forest* seluruhnya, p=q+k. karena diketahui p=q+1, maka k=1(hanya ada satu komponen) dan G terhubung. Dengan demikian G adalah *tree*.

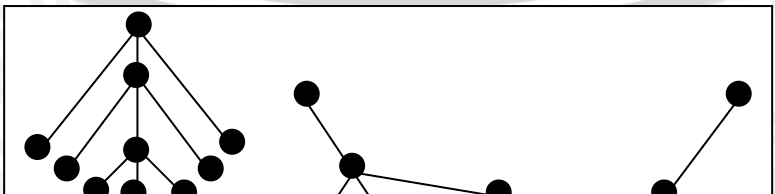
Misalkan titik u dan v tidak adjacent di G. Menurut bukti (1 $\Rightarrow$ 2), ada tepat satu *path* yang menghubungkan u dan v di G. Apabila garis x= uv ditambahkan ke G, maka garis x beserta *path* tunggal dalam G yang menghubungkan u dan v membentuk suatu *cycle*. *Cycle* itu tunggal sebab *path* dari u ke v adalah tunggal. Jadi, G + x mempunyai tepat satu *cycle*.

(5 $\Rightarrow$ 1) Cukup ditunjukkan bahwa G terhubung. Andaikan G tidak terhubung. Jika suatu garis x ditambahkan pada G yang menghubungkan dua titik dalam komponen yang berlainan, G + x tidak mempunyai *cycle*. Kontradiksi dengan ketentuan pernyataan (5). Jadi, pengandaian salah dan terbukti G terhubung. Dengan demikian, G *acyclic* dan terhubung. Dengan kata lain G adalah sebuah *tree*.

Definisi 2.7.2

Forest adalah graf *acyclic* dimana setiap komponen terhubungnya merupakan *tree* (Rosen, 2003).

Contoh 2.19:



Gambar 2.19 *Forest*

Definisi 2.7.3

Garis e dari graf terhubung G(V,E) adalah garis yang dihapus (*cut edge*) dari G jika graf (V,E-$\{e\}$) tidak terhubung (Fletcher, Hoyle, dan Patty, 1991).

Contoh 2.20:

G

Gambar 2.20 Graf G yang memuat *cut edge*

Keterangan: Garis yang bergaris tebal adalah *cut edge*

Teorema 2.7.2

Garis e dari graf G adalah garis yang dihapus (*cut edge*) jika dan hanya jika e tidak termuat di dalam *cycle* pada G (http://www.ecp6.jussieu.fr)

Bukti:

($\Rightarrow$) Pertama dibuktikan dengan kontradiksi. Misalkan terdapat sebuah *cut edge* e dari G yang termuat dalam *cycle* C. Karena G-$\{e\}$ tidak terhubung, maka terdapat titik u dan v pada G yang terhubung dalam G tetapi tidak dalam G-$\{e\}$. Dengan demikian terdapat sebuah *path* P (u, v) dalam G dan e berada di dalamnya.

Dikatakan bahwa e= $\{a,b\}$dan a mendahului b dalam P. Maka suatu bagian dari P membentuk sebuah *path* (a,b) dalam P. Karena e adalah anggota dari *cycle* C, C- e adalah *path* (a,b). Oleh karena itu u dan v terhubung dalam G-$\{e\}$.Terjadi kontradiksi. Telah dibuktikan bahwa jika e adalah sebuah *cut edge* dari G, maka e tidak termuat di dalam *cycle* pada G.

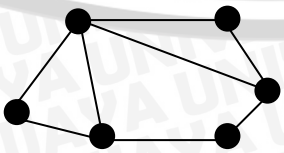
($\Leftarrow$) Misalkan bahwa e adalah bukan sebuah *cut edge* dari G. Jika e adalah *loop,* maka *loop* tersebut adalah sebuah *cycle*. Misalkan e adalah bukan *loop* dan misal f(e)= $\{u,v\}$ dimana $u \neq v$. Maka e adalah bukan cut edge dari G. G-$\{e\}$ terhubung. Oleh karena itu, terdapat sebuah *path* P dalam G-$\{e\}$ dari u menuju v. Maka P $\cup \{e\}$ adalah sebuah *cycle* dalam G dan e termuat dalam cycle tersebut. Sehingga telah dibuktikan bahwa jika e tidak termuat di dalam *cycle* pada G, maka e adalah sebuah *cut edge* dari G.

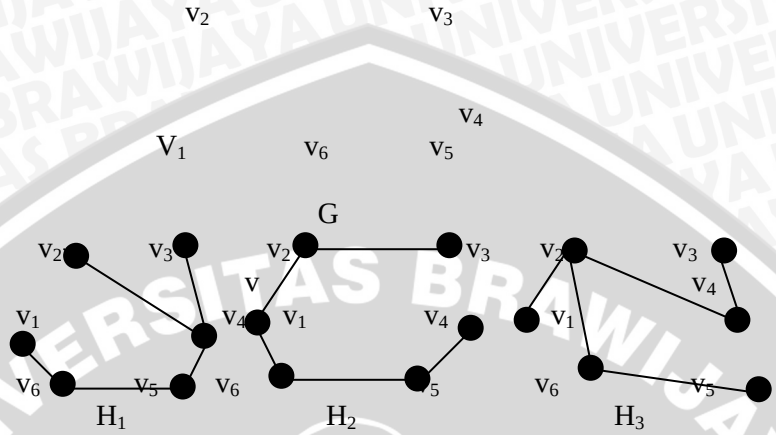
2.8 *Spanning Tree* dan *Minimal Spanning Tree*

Definisi 2.8.1

Subgraf H dari graf G yang terhubung disebut *spanning tree* jika dan hanya jika H merupakan *tree* dan H mempunyai himpunan titik yang sama dengan himpunan titik- titik pada graf G (Munro, 1993).

Contoh 2.21 :





Gambar 2.21 Graf G dan tiga *spanning tree* H dari graf G

Teorema 2.8.1

Setiap graf terhubung memuat *spanning tree*. (http://www.ecp6.jussieu.fr).

Bukti:

Misal $G = (V,E)$ adalah graf terhubung. Maka terdapat $G' = (V',E')$ yang merupakan subgraf terhubung dari G dengan syarat $V = V'$ dan jika terdapat garis dari G' yang dihapus maka G' tidak terhubung. (Oleh karena itu G' adalah graf sederhana karena sudah jelas bahwa *loop* dan garis ganda dapat dihapus dari graf terhubung tanpa memutuskan graf).

Menurut Teorema 2.7.2 bahwa e garis yang dihapus (*cut edge*) jika dan hanya jika e tidak termuat dalam *cycle* pada G'. Oleh karena itu G' tidak memuat *cycle*, maka berdasarkan Definisi 2.7.1 dikatakan bahwa G' merupakan *tree*, sehingga jelas sekali jika G' adalah *spanning tree*.

Teorema 2.8.2

Misalkan T *spanning tree* dari graf terhubung G dan misalkan e garis di G dan tidak pada T. Maka T+e memuat *cycle* yang tunggal. (http://www.ecp6.jussieu.fr)

Bukti:

T *spanning tree* berarti T terhubung dan *acyclic*. Karena T *acyclic*, maka setiap *cycle* dari T+e memuat e. C merupakan *cycle* dari T+e jika dan hanya jika C-e adalah *path* dalam T yang menghubungkan garis akhir e. Jadi T merupakan sebuah *tree* karena terhubung dan *acyclic*. Misalkan u dan v tidak adjacent dalam T dan garis e=uv ditambahkan ke T. Melalui Teorema 2.7.1 (2) maka didapatkan bahwa setiap dua titik dari T dihubungkan dengan suatu *path* yang tunggal. Garis e beserta *path* tunggal dalam T yang menghubungkan u dan v membentuk suatu *cycle*. *Cycle* itu tunggal sebab *path* dari u ke v adalah tunggal Dengan demikian *cycle* tersebut adalah tunggal. Jadi, T+e memuat *cycle* yang tunggal.

Definisi 2.8.2

Minimal spanning tree dari graf G adalah subgraf-subgraf dari G dengan bobot terkecil diantara subgraf-subgraf dari G yang terhubung. Jika N adalah suatu jaringan terhubung yang tidak berarah, maka *minimum spanning tree* dari N adalah bobot terkecil dari seluruh *spanning tree* yang ada pada N (Roman, 1989).

Definisi 2.8.3

Dalam Algoritma Kruskal, graf G=(V,E) dengan n titik ($n \in N$) dan garis-garis $e_1, e_2, \ldots, e_n$, diurutkan sedemikian sehingga $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_n)$.

Langkah-langkah Algoritma Kruskal adalah sebagai berikut:

Langkah (1) Misal $T = \emptyset$ dan $U(T) = \emptyset$,

Langkah (2) Misal k adalah bilangan terkecil sehingga e_k tidak berada di dalam T dan graf ($U(T \cup \{e_k\})$, $T \cup \{e_k\}$) tidak memuat *cycle*,

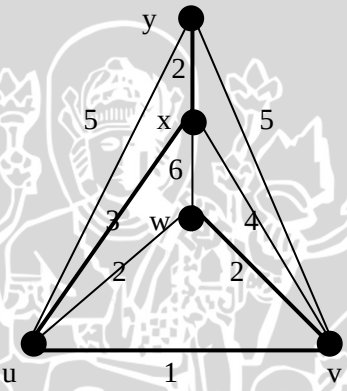
Langkah (3) Gantikan T dengan $T \cup \{e_k\}$ dan U(T) dengan $U(T \cup \{e_k\})$,

Langkah (4) Jika T memuat n–1 garis maka langkah pencarian dihentikan, (U(T),T) merupakan *minimal spanning tree*,

Langkah (5) Kembali menuju langkah (2).

Keterangan: T adalah himpunan dari garis- garis,
U(T) adalah himpunan titik dari G yang merupakan titik-titik dari garis-garis pada T.
(Fletcher, Hoyle, dan Patty, 1991).

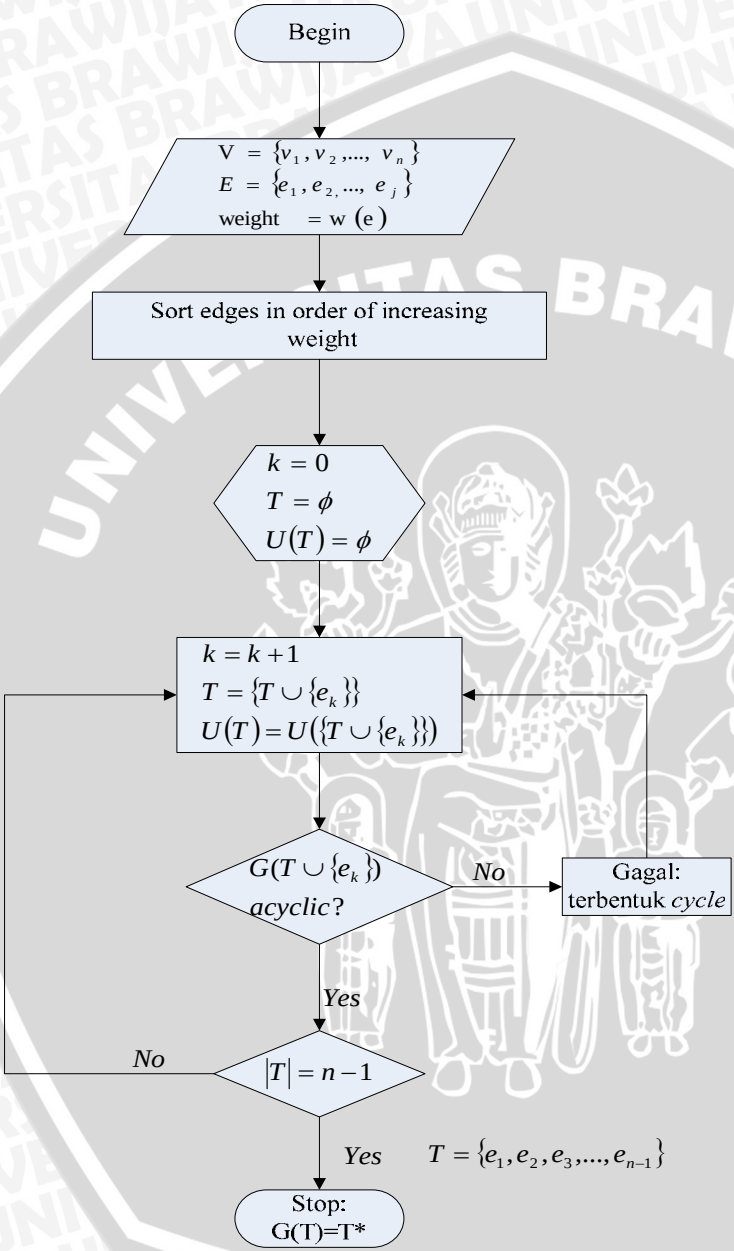
Contoh 2.22:



Gambar 2.22 Graf yang memuat *minimal spanning tree*

Keterangan: Garis yang bergaris tebal adalah garis dari *minimal spanning tree.*

Dari uraian langkah-langkah di atas, maka Algoritma Kruskal dapat disajikan dalam bentuk diagram di bawah ini:



Gambar 2.23 Diagram Algoritma Kruskal

Teorema 2.8.3

Algoritma Kruskal menghasilkan *minimal spanning tree* dari graf sederhana terhubung berbobot (Susanna, 1992).

Bukti :

Misalkan G adalah graf terhubung berbobot dengan n titik dan T adalah subgraf dari G yang dihasilkan oleh Algoritma Kruskal. Pasti T *acyclic* (karena tidak terdapat garis yang ditambahkan pada T untuk membentuk sebuah *cycle*). Dengan demikian T terhubung. Karena sepanjang T mempunyai lebih dari satu komponen terhubung, himpunan garis dari G dapat ditambahkan pada T tanpa membentuk *cycle* yang tidak kosong. (Karena G terhubung, diberikan sebarang titik v_1 dalam satu komponen terhubung C_1 dari T dan sebarang titik v_2 di dalam komponen terhubung lainnya yaitu dalam C_2, terdapat sebuah *path* dalam G dari v_1 menuju v_2. Karena C_1 dan C_2 berbeda, terdapat sebuah garis e dari *path* tersebut yang tidak berada pada T. Penambahan terhadap T tidak membentuk *cycle* dalam T karena penghapusan garis dari *cycle* tidak akan memutuskan suatu graf). Hal tersebut menyatakan bahwa T *acyclic* dan terhubung. Karena T memuat setiap titik dari G, maka T adalah *spanning tree* dari G.

Akan dibuktikan bahwa T mempunyai bobot yang minimal. Misal T_1 sebarang *minimal spanning tree* dari G. Jika $T=T_1$, maka T adalah sebuah *minimal spanning tree* dari G dan akan dibuktikan. Jika $T \neq T_1$, maka terdapat sebuah garis di dalam T yang bukan merupakan garis dari T_1. (Karena T dan T_1 keduanya mempunyai himpunan titik yang sama, jika keduanya berbeda,maka harus mempunyai himpunan titik yang berbeda). Dengan teorema 2.8.2 penambahan garis e pada T_1 menghasilkan sebuah graf dengan *cycle* yang tunggal. Misal e' adalah garis dari *cycle* tersebut tetapi tidak berada di dalam T (garis tersebut harus ada karena T adalah tree dan *acyclic*). Misal T_2 adalah graf yang diperoleh dari T_1 melalui penghapusan e' dan penjumlahan e. Dengan catatan bahwa T_2 mempunyai n-1 garis dan n titik dan T_2 terhubung(karena subgraf yang didapatkan dari penghapusan garis pada *cycle* dalam graf terhubung adalah terhubung). Hal tersebut berakibat T_2 merupakan sebuah *spanning tree* dari graf G. Sehingga,

$$w(T_2) = w(T_1) - w(e') + w(e) \quad (1)$$

Karena dalam Algoritma Kruskal, e dipilih dengan bobot terkecil, maka:

$$w(e) \leq w(e') \quad (2)$$

Ketika e ditambahkan dalam T, e' tersedia untuk ditambahkan (karena e' tidak berada dalam T, dan pada langkah tersebut penambahan tidak akan menghasilkan sebuah *cycle* karena e tidak berada pada T). Sehingga,

$$w(T_2) = w(T_1) \underbrace{- w(e') + w(e)}_{\geq 0} \quad (3)$$

$$\leq w(T_1).$$

Tetapi T_1 adalah sebuah *minimal spanning tree*. Karena T_2 adalah *spanning tree* dengan bobot lebih kecil atau sama dengan bobot dari T_1, T_2 juga merupakan *minimal spanning tree* dari G. Akhirnya dari pengkonstruksian tersebut, T_2 mempunyai satu lagi garis yang berhubungan dengan T dibandingkan dengan T_1. Jika T sama dengan T_2, maka T adalah *minimal spanning tree* dan akan dibuktikan. Jika tidak, maka dapat dilakukan proses pengulangan berdasarkan penguraian dia atas untuk mendapatkan *minimal spanning tree* T_3 yang mempunyai satu lagi garis yang berhubungan dengan T dibandingkan dengan T_2. Pelanjutan dari proses tersebut akan menghasilkan urutan dari *minimal spanning tree* T_1, T_2, T_3, …, setiap *minimal spanning tree* tersebut mempunyai satu lagi garis yang berhubungan dengan T dibandingkan dengan *tree* sebelumnya. Karena T hanya mempunyai sejumlah garis berhingga, urutan tersebut juga berhingga dan juga terdapat *minimal spanning tree* T_k, yang sama dengan T. Hal tersebut menyatakan bahwasanya T adalah sebuah *minimal spanning tree* dari G.

Definisi 2.8.4

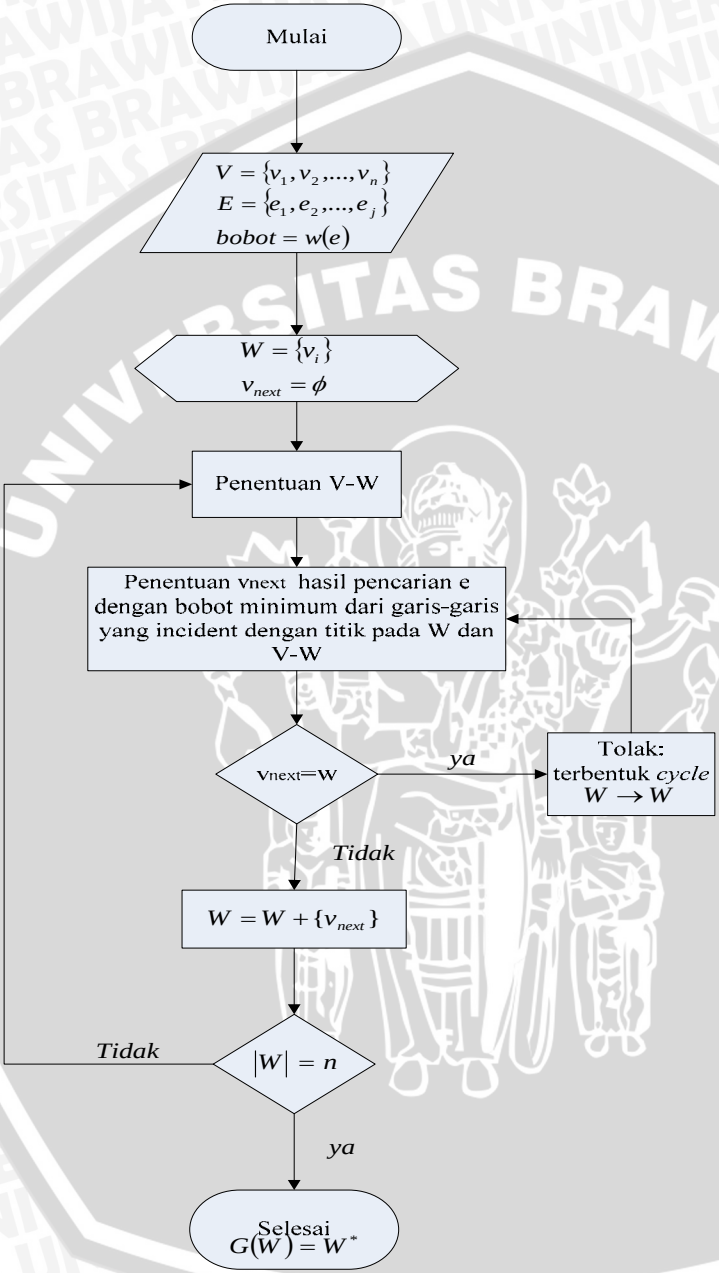
Algoritma Prim dimulai dari suatu titik awal tertentu bisa dipilih sebarang oleh algoritma. Misalnya titik awal tersebut adalah v.

Pada setiap iterasi terdapat kondisi dimana himpunan titik V terbagi dua dalam:

- W yaitu himpunan titik yang sudah dievaluasi sebagai node di dalam pohon, serta
- (V-W) yaitu himpunan titik yang belum dievaluasi.
- Di awal algoritma,W diinisialisasi berisi titik awal v.
- Selanjutnya, di dalam iterasinya:
    - Pada setiap *adjacency* dari tiap titik dalam W dengan titik dalam (V-W) dicari garis dengan panjang minimal.
    - setelah diperoleh, garis tersebut ditandai sebagai garis yang membentuk *tree* dan titik yang *adjacent* dengan garis tersebut dalam (V-W) dipindahkan ke W (menjadi anggota W).
    - Jika titik tersebut telah dievaluasi semua maka proses selesai.

(Http://id.Wikipedia.Org/wiki).

Dari uraian langkah-langkah di atas, maka Algoritma Prim dapat disajikan dalam bentuk diagram sebagai berikut:



Gambar 2.24 Diagram Algoritma Prim

Teorema 2.8.4

Algoritma Prim menghasilkan *minimal spanning tree* dari graf sederhana terhubung berbobot (Rosen, 2003).

Bukti:

Misal G=(V,E) adalah graf sederhana terhubung berbobot dengan n titik dengan urutan garis-garis yang terpilih oleh Algoritma Prim adalah e_1, e_2, …, e_{n-1}. Didefinisikan S adalah *spanning tree* yang dipilih oleh algoritma prim (garis-garis dari S adalah e_1, e_2, …, e_{n-1}). Karena *spanning tree* dari G ada, maka G mempunyai *minimal spanning tree*. Misalkan S_k adalah *tree* dengan garis-garis e_1, e_2, …, e_k, dan S'_k adalah *tree* dengan garisnya adalah garis-garis dari S yang bukan merupakan garis-garis dari S_k, untuk setiap k= 1, 2, …, n–1.

k adalah bilangan bulat terbesar sedemikian sehingga terdapat *minimal spanning tree* dari G yang memuat k garis yang dibentuk oleh algoritma prim. Misal T dinotasikan sebagai sebuah *minimal spanning tree*. Hal tersebut cukup untuk membuktikan bahwa S=T. Dimisalkan $S \neq T$ sedemikian sehingga $k \leq n-1$. Yang mengakibatkan T memuat e_1, e_2, …, e_k, tetapi T tidak memuat e_{k+1}. Andaikan T' adalah graf yang diperoleh dari penjumlahan garis e_{k+1}. Maka T' mempunyai n garis, dan oleh karena itu tidak dapat menjadi sebuah *tree*.

Karena T' terhubung, pasti T' memuat sebuah *cycle* C. Karena tidak terdapat *cycle* pada T, C harus memuat e_{k+1}. Karena S'_k adalah *tree*, terdapat garis pada C yang bukan merupakan garis pada S'_k. Dimulai dari titik e_{k+1} yang juga merupakan salah satu titik dari e_1, e_2, …, e_k dan mengikuti C sampai didapatkan jangkauan garis e yang tidak berada pada S'_k yang juga merupakan salah satu titik dari garis-garis e_1, e_2, …, e_k. Hapus e dari T' dan dapatkan sebuah graf T'' dengan n – 1 garis.

Karena T'' tidak memuat *cycle*, maka T'' merupakan *tree*. Sehingga memuat e_1, e_2, …, e_k, e_{k+1}. Karena e ada ketika e_{k+1} telah

dipilih oleh algoritma prim, bobot dari e_{k+1} adalah terkecil atau sama dengan bobot dari e. Oleh karena itu, T'' merupakan *minimal spanning tree* dari G, karena jumlah bobot dari garis-garisnya tidak melebihi dari jumlah bobot garis-garis pada T. Terjadi kontradiksi dengan pilihan dari k sebagai bilangan bulat terbesar sedemikian sehingga minimal spanning tree memuat e_1, e_2, …, e_k . Oleh karena itu k=n-1 dan S=T. Terbukti jika Algoritma Prim menghasilkan *minimal spanning tree.*

Definisi 2.8.5

Pada kasus pancarian *path* terpendek pada graf berbobot terhubung dengan bobot yang berupa probabilitas digunakan konversi di bawah ini.

Misalkan $P_{1k} = P_1 \times P_2 \times ... \times P_k$ probabilitas pada *path* spesifik (1,k) maka:

$$\log P_{1k} = \log P_1 + \log P_2 + ... + \log P_k \qquad (2.1)$$

Maksimum dari P_{1k} dalam persamaan aljabar ekuivalen dengan memaksimumkan log P_{1k} dan berakibat memaksimumkan jumlah dari logaritma-logaritma dari setiap probabilitas pada *path* yang dipilih. Karena $\log P_j \leq 0$, j= 1, 2, … , k, maka maksimum jumlah dari $\log P_j$ ekuivalen dengan meminimumkan jumlah dari

$$\left(-\log P_j\right) \qquad (2.2)$$

(Taha H. A, 1992).

BAB III
METODOLOGI PENELITIAN

3.1 Sumber Data

Dalam skripsi ini, penulis menggunakan data sekunder sebagai bahan untuk dikaji dan diteliti. Data tersebut diperoleh dari Tugas Akhir mahasiswa Fakultas Teknik yang bernama Linda Norwatiningsih. Dalam Tugas Akhirnya menjelaskan bagaimana cara untuk mendapatkan sebuah graf dengan menggunakan data jumlah panggilan telepon yang telah diperoleh dari PT. Telkom kemudian mencari peluang tersambung antar sentral-sentral yang ada dan menentukan rute-rute yang dilalui berdasarkan cara mendapatkan peluang tersebut. Data yang diambil dari Tugas Akhir tersebut antara lain ditunjukkan pada lampiran yaitu gambar konfigurasi sentral telepon MEA Malang, tabel hubungan antar sentral telepon lokal di MEA Malang, dan tabel data jumlah panggilan antar sentral. Keseluruhan data tersebut disajikan dalam bentuk sebuah matriks *adjacency* dan matriks berbobot yang dibentuk berdasarkan data sekunder yang diperoleh. Data ini akan dibentuk menjadi sebuah graf sederhana terhubung, berbobot dan tidak berarah. Kemudian dari graf tersebut akan dicari dan dihitung *minimal spanning tree*nya yaitu berupa probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral dengan tujuan mendapatkan probabilitas terbesar semua sentral tersambung dengan menggunakan Algoritma Kruskal dan Algoritma Prim.

3.2 Analisis Data

Dalam skripsi ini akan membahas tentang Algoritma Kruskal dan Algoritma Prim beserta langkah-langkahnya dalam menetukan *minimal spanning tree* dalam suatu graf sederhana terhubung, berbobot dan tidak berarah

Beberapa langkah yang harus dilakukan untuk menyelesaikan masalah konfigurasi sentral telepon *Multi Exchange Area* (MEA) Malang adalah:

1) Penentuan titik-titik atau *vertex* dalam pembentukan graf
    - Untuk setiap titik pada pembentukan graf dari konfigurasi sentral telepon MEA Malang ini diwakili oleh setiap sentral,
2) Penentuan bobot dari setiap garis, yang dapat dilakukan melalui perhitungan-perhitungan di bawah ini:
    - Penghitungan besarnya probabilitas tersambung antara sentral (P) menggunakan persamaan di bawah ini:

$$P = \frac{Nc}{No} \qquad (3.\ 1)$$

Dengan: Nc = panggilan yang berhasil (*carried call*)

No = panggilan yang ditawarkan (*offered call*),

    - Dalam analisa selanjutnya, dilakukan penghitungan jarak antara sentral telepon (bobot) dari setiap garis yang diwakili oleh probabilitas hilangnya panggilan akibat kanal yang penuh antar sentral dengan menggunakan Persamaan (2.2) yaitu: $\left(-\log P\right)$,

3) Penghitungan *minimal spanning tree* dari hasil pembentukan graf menggunakan Algoritma Kruskal dengan menggunakan perhitungan manual dan menggunakan program,
4) Penghitungan *minimal spanning tree* dari hasil pembentukan graf menggunakan Algoritma Prim dengan menggunakan perhitungan manual dan menggunakan program,
5) Perbandingan hasil yang diperoleh dari penghitungan menggunakan Algoritma Kruskal dan Algoritma Prim.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Masalah Jaringan *Multi Exchange Area* (MEA)

Jaringan telepon merupakan media yang digunakan untuk perpindahan informasi dari suatu tempat ke tempat yang lain. Trafik adalah perpindahan informasi dari satu tempat ke tempat lain dengan menggunakan suatu media jaringan telekomunikasi pada waktu tertentu dan selang waktu tertentu. Trafik pada telepon dibangkitkan oleh sejumlah pelanggan dalam suatu proses pemanggilan sampai penyambungan di level sentral sehingga setiap peralatan dapat diidentifikasi besar trafiknya.

Jaringan lokal merupakan saluran penghubung langsung antara sentral dengan pelanggan. Sentral telepon adalah tempat dilakukannya penyambungan permintaan pembicaraan. Sentral telepon secara garis besar mempunyai dua fungsi utama yaitu:

1. Menghubungkan antara pelanggan telepon yang satu dengan pelanggan telepon yang lain,
2. Menghubungkan antara sentral telepon yang satu dengan sentral telepon yang lain.

Multi Exchange Area (MEA) adalah kumpulan beberapa sentral lokal yang terletak di dalam suatu wilayah atau kota dan antara sentral satu dengan sentral lainnya dapat saling berhubungan. Dalam suatu wilayah luas dengan jumlah pelanggan telepon yang besar, diperlukan banyak sentral telepon lokal. Sentral-sentral telepon lokal tersebut dihubungkan dangan saluran penghubung antar sentral dan membentuk suatu *Multi Exchange Area*. Seperti halnya dengan wilayah Malang memiliki delapan sentral yang berfungsi sebagai sentral lokal.

Kebutuhan masyarakat akan sarana telepon semakin meningkat. Peningkatan kebutuhan akan sarana telepon ini ditandai dengan semakin padatnya lalu lintas (trafik) pada kanal antar sentral

yang melebihi kapasitas kanal. Hal ini mengakibatkan terjadinya kegagalan pembentukan hubungan antara sentral.

Untuk menentukan *minimal spanning tree* (probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral) digunakan algoritma kruskal dan algoritma prim sehingga didapatkan probabilitas tersambung terbesar antar sentral.

4.2 Pembentukan Graf

Daerah komunikasi di Malang memiliki lebih dari satu sentral lokal (*Multi Exchange Area*). Sentral telepon tersebut adalah:

1. Sentral Malang Kota, Jl. Jenderal Basuki Rachmad,
2. Sentral Malang Blimbing, Jl. A. Yani,
3. Sentral Klojen, Jl. Terusan Surabaya,
4. Sentral Gadang, Jl. Raya Gadang,
5. Sentral Sawojajar, Jl. Danau Sentani,
6. Sentral Sumber Pucung, Jl. Panglima Sudirman,
7. Sentral Tumpang, Jl. Raya Tumpang,
8. Sentral Singosari, Jl. Raya Singosari.

Selain sentral-sentral lokal, MEA Malang juga memiliki sental Trunk (*Malang Trunk*). Sentral Malang Trunk digunakan untuk melayani permintaan hubungan komunikasi dari area Malang ke area lain.

Dalam pembentukan graf tersebut maka sentral-sentral telepon lokal yang ada dikonversikan sebagai node-node atau titik-titik yang ditandai dengan nomor-nomor berikut:

- Sentral Malang Kota (MLK) = titik 1 (v_1)
- Sentral Malang Blimbing = titik 2 (v_2)
- Sentral Klojen = titik 3 (v_3)
- Sentral Gadang = titik 4 (v_4)
- Sentral Sawojajar = titik 5 (v_5)

- Sentral Sumber Pucung = titik 6 (v_6)
- Sentral Tumpang = titik 7 (v_7)
- Sentral Singosari = titik 8 (v_8)

Probabilitas tersambung antara sentral telepon merupakan parameter yang digunakan untuk menentukan keberhasilan penyelenggaraan penyambungan antara sentral satu dengan sentral yang lainnya. Untuk menghitung besarnya probabilitas tersambung antara sentral (P) digunakan Persamaan (3.1):

$$P = \frac{Nc}{No}$$

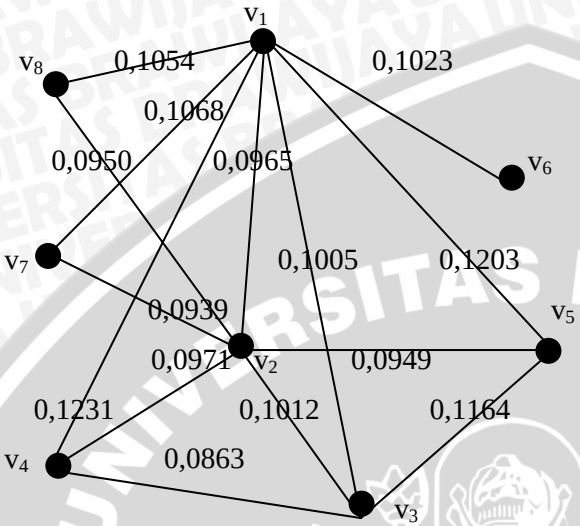
Dengan: Nc = panggilan yang berhasil (*carried call*)

No= panggilan yang ditawarkan (*offered call*)

Dari data-data pada Lampiran (3) yang berupa matriks berbobot dari jumlah panggilan yang ditawarkan dan Lampiran (4) yang berupa matriks berbobot dari jumlah panggilan yang berhasil diperoleh hasil perhitungan besarnya probabilitas tersambung antara sentral (P) yang ditunjukkan berupa matriks berbobot pada Lampiran (5).

Dalam analisa selanjutnya, jarak antara sentral telepon (bobot) dari setiap garis diwakili oleh probabilitas hilangnya panggilan akibat kanal yang penuh antar sentral. Probabilitas hilangnya panggilan akibat kanal yang penuh antar sentral ini diperoleh dengan cara mengkonversikan besarnya probabilitas tersambung antara sentral (P) yang didapatkan dari perhitungan sebelumnya (Lampiran 5) dengan menggunakan Persamaan (2.2). yaitu $\left(-\log P\right)$. Sehingga diperoleh hasil probabilitas hilangnya panggilan akibat kanal yang penuh antar sentral yang ditunjukkan dalam bentuk matriks berbobot pada Lampiran (6).

Sehingga data awal yang berupa gambar konfigurasi sentral telepon MEA Malang yang tertera pada Lampiran (1) dapat disederhanakan menjadi gambar dalam bentuk jaringan (graf sederhana terhubung, tidak bararah, dan berbobot) di bawah ini:



Gambar 4.1 Graf Konfigurasi sentral telepon MEA Malang

Gambar graf di atas memuat delapan titik (verteks) dan empat belas garis. Untuk mencari dan mendapatkan *minimal spanning tree* dari graf tersebut, digunakan Algoritma Kruskal dan Algoritma Prim. Dua algoritma tersebut mempunyai metodologi yang berbeda tetapi keduanya memang dikonstruksikan untuk mendapatkan *minimal spanning tree*. Kedua algoritma ini diproses melalui penambahan bobot- bobot terkecil dari garis-garis yang ada.

4.3 Penghitungan *Minimal Spanning Tree* Menggunakan Algoritma Kruskal

Berdasarkan langkah-langkah Algoritma Kruskal yang tertulis pada Definisi 2.8.3 maka dapat dilakukan perhitungan bobot *minimal spanning tree* dari graf konfigurasi sentral telepon MEA Malang dengan menggunakan Algoritma Kruskal sebagai berikut:

- Mengurutkan garis-garis dari graf di atas yaitu $e_1, e_2, ..., e_{14}$ sedemikian sehingga $w(e_1) \leq w(e_2) \leq ... \leq w(e_{14})$.
- Misal:

$e_1 = (v_3, v_4), e_2 = (v_2, v_7), e_3 = (v_2, v_5), e_4 = (v_2, v_8), e_5 = (v_1, v_2),$
$e_6 = (v_2, v_4), e_7 = (v_1, v_3), e_8 = (v_2, v_3), e_9 = (v_1, v_6), e_{10} = (v_1, v_8),$
$e_{11} = (v_1, v_7), e_{12} = (v_3, v_5), e_{13} = (v_v, v_5), e_{14} = (v_1, v_4).$

Tabel 4.1 Penghitungan *Minimal Spanning Tree* Menggunakan Algoritma Kruskal

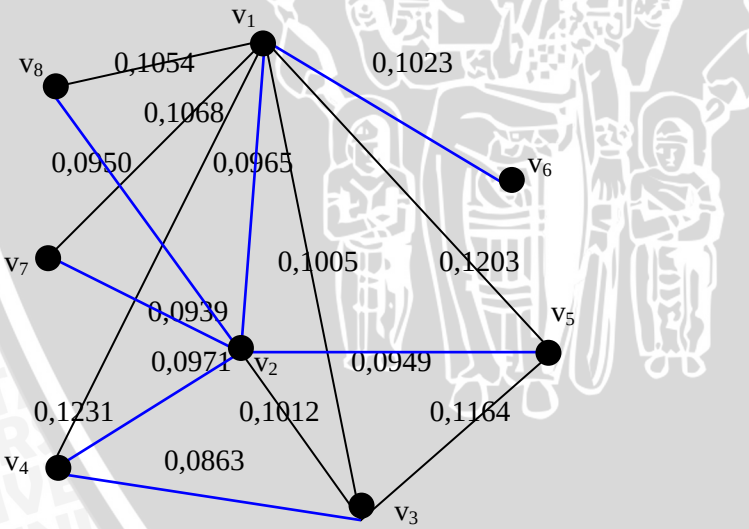
Iterasi	**T**	**U(T)**	**Bobot (keputusan)**
0	{}	{}	-
1	$\{e_1\}$	$\{v_3, v_4\}$	0,0863 (terima)
2	$\{e_1, e_2\}$	$\{v_3, v_4, v_2, v_7\}$	0,0939 (terima)
3	$\{e_1, e_2, e_3\}$	$\{v_3, v_4, v_2, v_7, v_5\}$	0,0949 (terima)
4	$\{e_1, e_2, e_3, e_4\}$	$\{v_3, v_4, v_2, v_7, v_5, v_8\}$	0,0950 (terima)
5	$\{e_1, e_2, e_3, e_4, e_5\}$	$\{v_3, v_4, v_2, v_7, v_5, v_8, v_1\}$	0,0965 (terima)
6	$\{e_1, e_2, e_3, e_4, e_5, e_6\}$	$\{v_3, v_4, v_2, v_7, v_5, v_8, v_1\}$	0,0971 (terima)

Tabel 4.1 Lanjutan

7	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$	$\{v_3, v_4, v_2, v_7, v_5, v_8, v_1\}$	0,1005 (tolak)
8	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_8\}$	$\{v_3, v_4, v_2, v_7, v_5, v_8, v_1\}$	0,1012 (tolak)
9	$\{e_1, e_2, e_3, e_4, e_5, e_6, e_9\}$	$\{v_3, v_4, v_2, v_7, v_5, v_8, v_1, v_6\}$	0,1023 (terima)
	Selesai karena sudah terbentuk n-1 garis *acyclic*	-	Total bobot = 0,666

Keterangan: Terima = jika tidak membentuk *cycle* (*acyclic*),
Tolak = jika membentuk *cycle*

Hasil *minimal spanning tree* yang diperoleh dari perhitungan menggunakan Algoritma Kruskal dapat dilihat dengan ditandai garis berwarna biru pada gambar di bawah ini:



Gambar 4.2 Graf yang memuat *minimal spanning tree*

Hasil *minimal spanning tree* tersebut diperoleh dari penggabungan garis yang didapatkan dari setiap langkah-langkah iterasi yang disajikan dalam gambar pada Lampiran (7).

Dari perhitungan Algoritma Kruskal di atas diperoleh *minimal spanning tree* dengan jumlah bobot sebagai berikut:

$$w(e_1)+w(e_2)+w(e_3)+w(e_4)+w(e_5)+w(e_6)+w(e_9)$$
$$= 0{,}0863+0{,}0939+0{,}0949+0{,}0950+0{,}0965+0{,}0971+0{,}1023$$
$$= 0{,}666$$

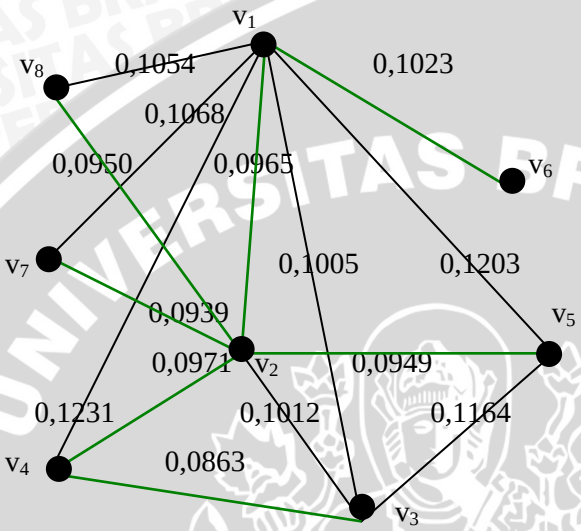
Jadi diperoleh *minimal spanning tree* $T=\{e_1,e_2,e_3,e_4,e_5,e_6,e_9\}$ dengan bobot 0,666 yang berarti besar probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral adalah 0,666.

Hasil bobot dari *minimal spanning tree* ini sama dengan penghitungan yang didapatkan dengan menggunakan program delphi yang dilampirkan pada Lampiran (9) yaitu didapatkan *minimal spanning tree* dengan bobot 0,666 yang dihasilkan setelah terjadi perhitungan dari sejumlah prosedur-prosedur yang ada sebanyak 592 kali.

4.4 Penghitungan *Minimal Spanning Tree* Menggunakan Algoritma Prim

Berdasarkan langkah-langkah Algoritma Prim yang tertulis pada Definisi 2.8.4 maka dapat dilakukan perhitungan bobot *minimal spanning tree* dari graf konfigurasi sentral telepon MEA Malang dengan menggunakan Algoritma Prim sebagai berikut:

Hasil *minimal spanning tree* yang diperoleh dari perhitungan menggunakan Algoritma Prim dapat dilihat dengan ditandai garis berwarna hijau pada gambar di bawah ini:



Gambar 4.3 Graf yang memuat *minimal spanning tree*

Hasil *minimal spanning tree* yang diperoleh dari perhitungan menggunakan Algoritma Prim sama dengan Gambar (3.2), hal tersebut dikarenakan hasil dari *minimal spanning tree* adalah tunggal karena bertujuan untuk mencari bobot terkecil dari suatu graf sederhana tidak berarah, terhubung, dan berbobot. Hasil *minimal spanning tree* yang diperoleh dari penggabungan garis yang didapatkan dari langkah-langkah pada setiap iterasi, disajikan dalam gambar pada Lampiran (8).

Dari perhitungan Algoritma Prim di atas diperoleh *minimal spanning tree* dengan jumlah bobot sebagai berikut:

$$w(v_1,v_2)+w(v_2,v_7)+w(v_2,v_5)+w(v_2,v_8)+w(v_2,v_4)+w(v_4,v_3)+w(v_1,v_6)$$
$$=0{,}0965+0{,}0939+0{,}0949+0{,}0950+0{,}0971+0{,}0863+0{,}1023$$
$$=0{,}666$$

Jadi diperoleh *minimal spanning tree* dengan $W = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ dan bobot 0,666 yang berarti besar probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral adalah 0,666.

Hasil bobot dari *minimal spanning tree* ini sama dengan penghitungan yang didapatkan dengan menggunakan program delphi yang dilampirkan pada Lampiran (10) yaitu didapatkan *minimal spanning tree* dengan bobot 0,666 yang dihasilkan setelah terjadi perhitungan dari sejumlah prosedur-prosedur yang ada sebanyak 97 kali.

Dari hasil perhitungan baik menggunakan Algoritma Kruskal maupun Algoritma Prim diperoleh bobot *minimal spanning tree* (probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral) adalah 0.666. Hal tersebut mengakibatkan besarnya probabilitas terbesar semua sentral tersambung menjadi:

Probabilitas terbesar semua sentral tersambung (P)
= ant log (-0,666)
= $10^{(-0,666)}$
= 0,2158

Jadi besarnya Probabilitas terbesar semua sentral tersambung (P) adalah 0,2158. Hal ini dikarenakan adanya beberapa faktor antara lain:

1. Semua sentral yang dilalui diasumsikan dalam keadaan sibuk sehingga harus melalui jalur-jalur dari semua sentral dengan probabilitas kegagalan terendah (berdasarkan pembentukan graf dengan bobot probabilitas hilangnya panggilan akibat kanal yang penuh),
2. Kemampuan daya tampung jumlah panggilan yang berbeda pada setiap kanal dalam sentral, dan
3. Adanya jalur hubungan yang tidak langsung dari setiap sentral untuk menuju ke sentral yang lain.

4.5 Perbandingan

Dari perhitungan di atas dapat dikatakan bahwasanya penggunaan Algoritma Kruskal dan Algoritma Prim mempunyai perbedaan yang mendasar yaitu pada algoritma kruskal pencarian *minimal spanning tree* didasarkan pada garis yang mempunyai bobot minimum dan garis tersebut tidak membentuk *cycle* dengan garis-garis yang telah terpilih sebagai *tree*. Sedangkan pada algoritma prim pencarian *minimal spanning tree* dilakukan dengan memperhatikan *adjacent* dari titik awal dan titik yang telah dievaluasi sebagai *tree*.

Berdasarkan perhitungan dari kedua algoritma di atas dan berdasarkan penggunaan program, maka dapat ditampilkan sebuah tabel perbandingan dari Algoritma Kruskal dan Algoritma Prim sebagai berikut:

Tabel 4.3 Perbandingan Hasil Algoritma Kruskal dan Algoritma Prim

Algoritma	Banyak Iterasi	Banyak Perhitungan	Bobot MST
Kruskal	9	592 kali	0,666
Prim	8	97 kali	0,666

Keterangan: MST = *Minimal Spanning Tree*

Dari Tabel 4.3 dapat dilihat perbandingan antara Algoritma Kruskal dan Algoritma Prim, yaitu:

1. Jumlah bobot dan gambar graf *Minimal Spanning Tree* yang dihasilkan sama. Hal ini terjadi karena garis-garis yang diambil dari kedua metode sama walaupun caranya berbeda,
2. Waktu proses algoritma prim lebih cepat. Dilihat dari segi program, proses perhitungan Prim lebih sederhana dibanding Kruskal. Pada Kruskal diperlukan lebih banyak prosedur yaitu pengurutan garis, pengecekan keterhubungan, dan pengecekan *cycle*. Dengan banyaknya prosedur maka variable yang diperlukan lebih banyak

sehingga lebih banyak memerlukan memori dan proses lebih lambat. Namun pada Algoritma Kruskal, prosedur-prosedur nya berguna untuk perhitungan pada graf yaitu prosedur pengecekan keterhubungan, dan pengecekan *cycle*. Di dalam prosedur pengecekan *Cycle* terdapat perhitungan *incidentcy* tiap garis. Algoritma Kruskal lebih lambat juga karena perhitungannya lebih banyak. Pada setiap langkah pemasangan garis, dilakukan pengecekan *cycle*. Pada Algoritma Prim, hal tersebut tidak perlu dilakukan karena adanya prosedur pengecekan titik-titik terdekat pada setiap pemasangan garis.

3. Langkah pemasangan garis pada kedua algoritma dapat sama dan dapat berlainan. Langkah pemasangan akan sama bila pada tahap Algoritma Prim, garis yamg diambil mempunyai bobot yang berurutan secara menaik (*ascending).*

Dalam hal ini dapat disimpulkan bahwasanya Algoritma Prim lebih efisien dibandingkan algoritma Kruskal karena Algoritma Prim tidak membutuhkan waktu yang terlalu lama untuk mendapatkan *minimal spanning tree* dari suatu graf karena garis yang dimasukkan harus bersisian sehingga meminimalkan waktu dan dapat memperhitungkan semua bobot dari setiap garis berdasarkan *adjacent* titik yang dievaluasi. Sedangkan Algoritma Kruskal lebih sederhana jika dilihat dari konsepnya, namun lebih sulit dalam implementasinya karena iterasi yang dilakukan terlalu lama sehingga memerlukan ketelitian yang ekstra. Yang menjadi masalah dalam implementasinya adalah keperluan adanya pemeriksaan kondisi siklik.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil pembahasan dapat diambil kesimpulan bahwa:

1. Pada Algoritma Kruskal *minimal spanning tree* dapat dicari dengan cara penentuan garis yang mempunyai bobot minimum dan garis tersebut tidak membentuk *cycle* dengan garis-garis yang telah terpilih sebagai *tree*. Dari hasil perhitungan Algoritma Kruskal diperoleh besarnya *minimal spanning tree* (probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral) adalah 0,666.
2. Pada Algoritma Prim pencarian *minimal spanning tree* dilakukan dengan memperhatikan adjacent dari titik awal dan titik yang telah dievaluasi sebagai *tree*. Dari hasil perhitungan Algoritma Prim diperoleh besarnya *minimal spanning tree* (probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral) adalah 0,666.
3. Algoritma Kruskal dan Algoritma Prim mempunyai jumlah bobot dan gambar graf *Minimal Spanning Tree* yang sama. Algoritma Prim lebih efisien dibandingkan Algoritma Kruskal karena penentuan *minimal spanning tree* menggunakan Algoritma Kruskal dihasilkan setelah dilakukan iterasi sebanyak 9 dengan banyak perhitungan 592 kali. Sedangkan Pada Algoritma Prim terdapat 8 iterasi dengan banyak perhitungan 97 kali.

5.2 Saran

Pada penulisan tugas akhir ini hanya dibahas penentuan *minimal spanning tree* menggunakan Algoritma Kruskal dan Algoritma Prim. Untuk penulisan selanjutnya dapat dibahas dengan menggunakan algoritma yang lain seperti Algoritma Boruvka.

DAFTAR PUSTAKA

Dimitri Bertsekas, Robert Gallager. 1992. *Data Network*. Second edition. Prentice Hall International, Inc.

Chartrand, G. dan Ortrud, R. O. 1993. *Applied and Algorithmic Graph Theory*. McGraw- Hill, Inc. New York.

Doerr, Alan and Levasseur Kenneth. 1991. *Applied Discrete Strutures Computers Science.* Second Edition. Mac Millan Publishing Company. New York.

Fletcher, Hoyle dan Patty. 1991. *Foundation Of Discrete Mathematics.* PWS- KENS Publishing Company. Boston.

Foldes, S. 1993. *Fundamental Strutures Of Algebra and Discrete Mathematics.* John Whiley and Sons, Inc. America.

Harary, Frank. 1994. *Graph Theory*. Addison Wesley Publishing Company. America.

Http://id.Wikipedia.Org/wiki/Pohon_rentang_minimum. *Struktur Data Graf (Part 2)*. Tanggal akses: 01 Maret 2007.

Http://.www.ecp6.jussieu.fr/pageperso/bondy/books/gtwa/pdf/GTWA.pdf. *Trees*. Tanggal akses: 28 Februari 2007.

James, A. M. 1990. *Algorithmic Graph Theory*. Prentice- Hall, Inc. America.

Siang J. J. 2002. *Matematika Diskrit dan Aplikasinya Pada Ilmu Komputer Ed. 1*. ANDI Yogyakarta.

Johnsonbaugh, R. 1984. *Discrete Mathematics*. Mac Millan Publishing Company. New York.

Marsudi, 2006. *Pengantar Teori Graph*. Fakultas Matematika dan Ilmu Pengetahuan Alam. Universitas Brawijaya.

Munro, J. E. 1993. *Discrete Mathematics For Computing.* Chapman & Hall. Australia.

Norwatiningsih, L. 2002. *Analisis Penerapan Cyclic Algorithm Untuk Penentuan Rute Antar Sentral Telepon Pada Jaringan*

Telekomunikasi di Malang. Fakultas Teknik Jurusan Teknik Elektro. Universitas Brawijaya.

Roman, S. 1989. *An Introduction To Discrete Mathematics*. Second Edition. Harcourt Brace Jovanovich Publisher. Fullerton.

Rosen, K. H. 2003. *Discrete Mathematics and Its Application.* Fitht Edition. Mc Graw Hill Inc. New York.

Susanna, S. Epp. 1990. *Discrete Mathematics With Application.* Wadsworth Publishing Company. California.

Taha, H.A. 1992. *Operation Research an Introduction*. Fitht Edition. Mc. Milan Publishing Co. Inc. New York.

Wilson, R. J dan Watkins, J. J. 1990. *Graph An IntroductoryApproach a First Course In Discrete Mathemetics*. John Whiley and Sons, Inc. Canada.

PENENTUAN *MINIMAL SPANNING TREE* MENGGUNAKAN ALGORITMA KRUSKAL DAN ALGORITMA PRIM
(Studi kasus: Konfigurasi Sentral Telepon MEA Malang)

TUGAS AKHIR

Oleh:
NURUL ISNANIYAH
0310940045- 94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2007

PENENTUAN *MINIMAL SPANNING TREE* MENGGUNAKAN ALGORITMA KRUSKAL DAN ALGORITMA PRIM
(Studi kasus: Konfigurasi Sentral Telepon MEA Malang)

TUGAS AKHIR

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Sains dalam bidang Matematika

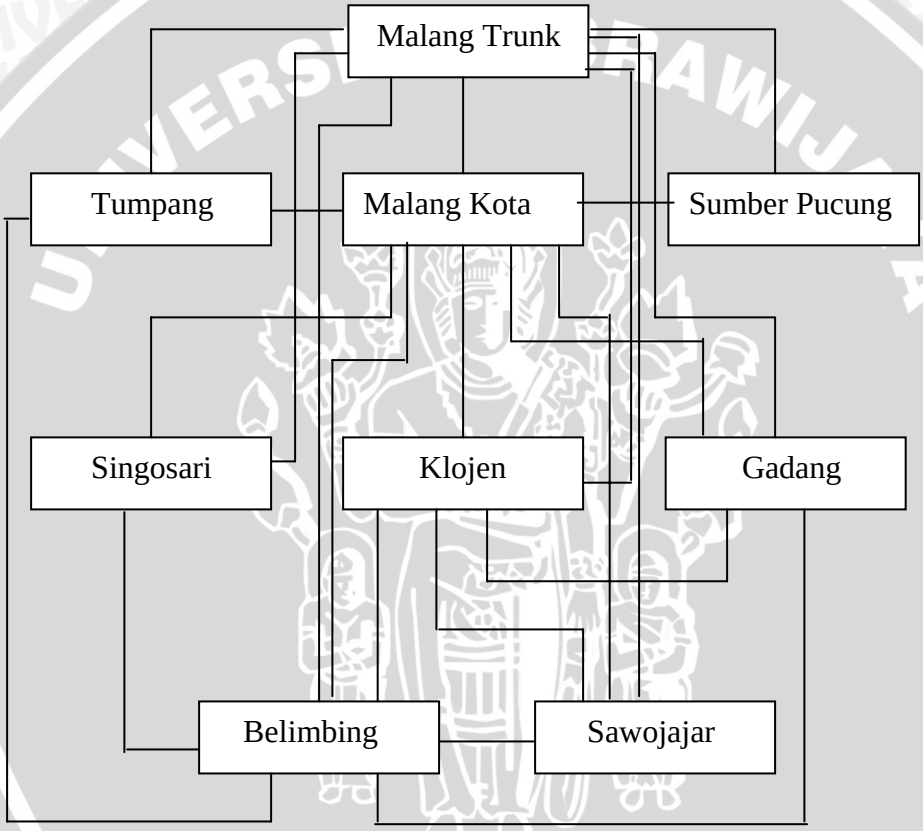
Oleh:
NURUL ISNANIYAH
0310940045- 94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2007

LAMPIRAN-LAMPIRAN

Lampiran 1. Gambar Konfigurasi Sentral Telepon MEA Malang



Lampiran 2. Matriks Hubungan Antar Sentral Telepon Lokal di MEA Malang

$$
\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Keterangan:

- Sentral Malang Kota (MLK) = v_1
- Sentral Malang Belimbing = v_2
- Sentral Klojen = v_3
- Sentral Gadang = v_4
- Sentral Sawojajar = v_5
- Sentral Sumber Pucung = v_6
- Sentral Tumpang = v_7
- Sentral Singosari = v_8
  - 1= terdapat hubungan antara v_i dan v_j,
  - 0= tidak terdapat hubungan antara v_i dan v_j.

Lampiran 3. Matriks Berbobot dari Jumlah Panggilan yang ditawarkan (*Offered Call*) Antar Sentral Telepon

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	6698	5416	2771	2191	624	1027	1007
v_2	6698	0	3496	589	891	∞	766	402
v_3	5416	3469	0	394	489	∞	∞	∞
v_4	2771	5891	394	0	∞	∞	∞	∞
v_5	2191	891	489	∞	0	∞	∞	∞
v_6	624	∞	∞	∞	∞	0	∞	∞
v_7	1027	766	∞	∞	∞	∞	0	∞
v_8	1007	402	∞	∞	∞	∞	∞	0

Keterangan:

- Sentral Malang Kota (MLK) = v_1
- Sentral Malang Belimbing = v_2
- Sentral Klojen = v_3
- Sentral Gadang = v_4
- Sentral Sawojajar = v_5
- Sentral Sumber Pucung = v_6
- Sentral Tumpang = v_7
- Sentral Singosari = v_8
  - ❖ 0 jika i=j
  - ❖ ∞ = tidak terdapat hubungan antara v_i dan v_j .

Lampiran 4. Matriks Berbobot dari Jumlah Panggilan yang berhasil (*Carried Call*) Antar Sentral Telepon

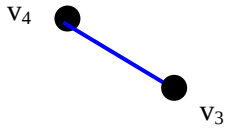
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	5363	4297	2087	1661	493	803	790
v_2	5363	0	2748	471	716	∞	617	323
v_3	4297	2748	0	323	374	∞	∞	∞
v_4	2087	471	323	0	∞	∞	∞	∞
v_5	1661	716	374	∞	0	∞	∞	∞
v_6	493	∞	∞	∞	∞	0	∞	∞
v_7	803	617	∞	∞	∞	∞	0	∞
v_8	790	323	∞	∞	∞	∞	∞	0

Keterangan:

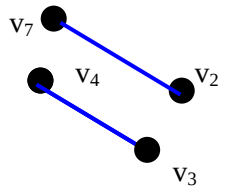
- Sentral Malang Kota (MLK) = v_1
- Sentral Malang Belimbing = v_2
- Sentral Klojen = v_3
- Sentral Gadang = v_4
- Sentral Sawojajar = v_5
- Sentral Sumber Pucung = v_6
- Sentral Tumpang = v_7
- Sentral Singosari = v_8
  - ❖ 0 jika i=j
  - ❖ ∞ = tidak terdapat hubungan antara v_i dan v_j.

Lampiran 7. Langkah-langkah *Pembentukan Minimal Spanning Tree* Menggunakan Algoritma Kruskal

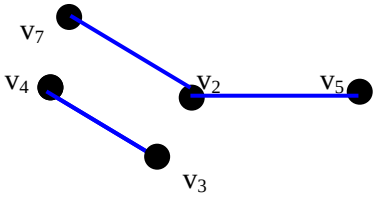
Langkah (1):



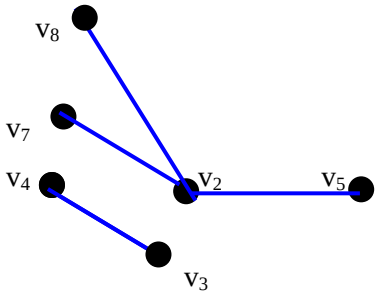
Langkah (2):



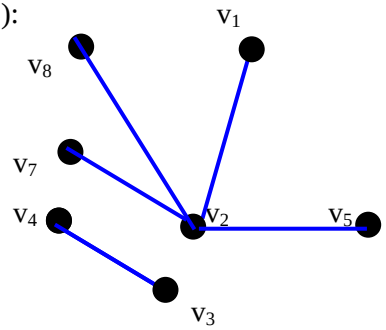
Langkah (3):



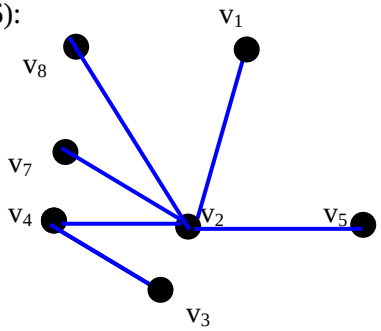
Langkah (4):



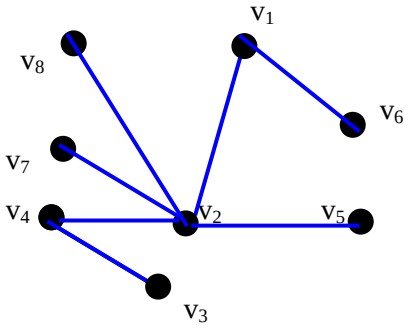
Langkah (5):



Langkah (6):



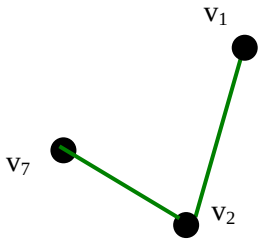
Langkah (7):



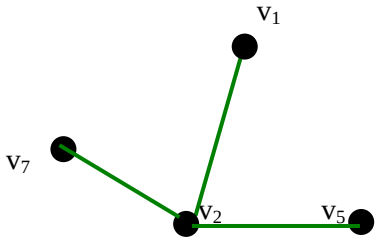
Lampiran 8. Langkah-langkah Pembentukan *Minimal Spanning Tree* Menggunakan AlgoritmaPrim

Langkah (1):

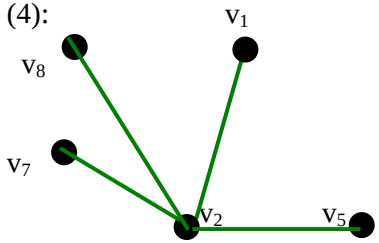
Langkah (2):



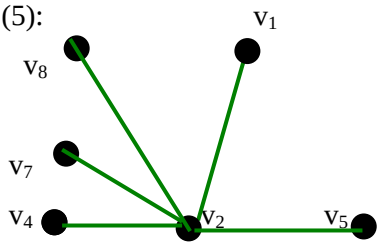
Langkah (3):



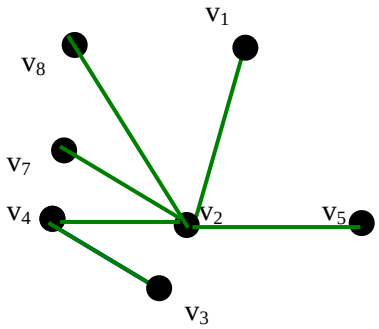
Langkah (4):



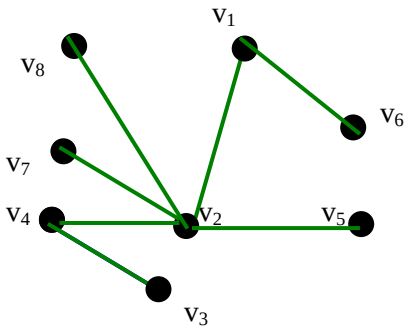
Langkah (5):



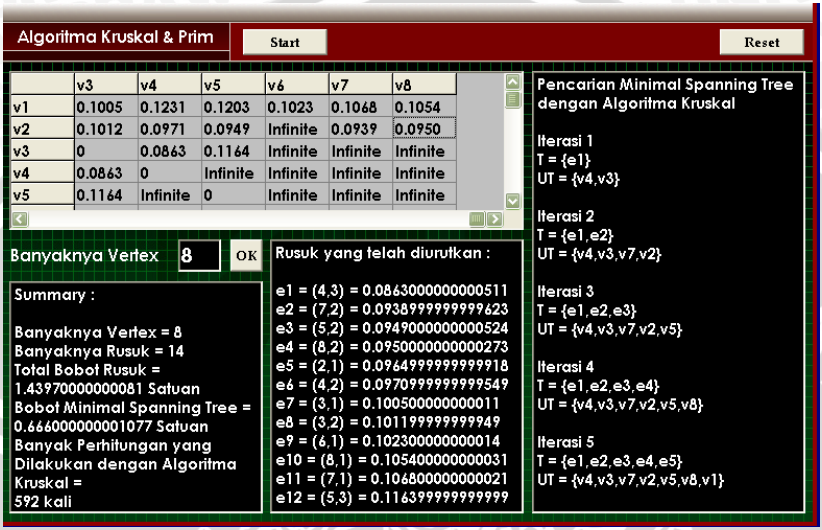
Langkah (6):



Langkah (7):



Lampiran 9. Tampilan Output dari Program Algoritma Kruskal



Memo 1:

Pencarian *Minimal Spanning Tree* dengan Algoritma Kruskal

Iterasi 1
T = {e1}
UT = {v4,v3}

Iterasi 2
T = {e1,e2}
UT = {v4,v3,v7,v2}

Iterasi 3
T = {e1,e2,e3}
UT = {v4,v3,v7,v2,v5}

Iterasi 4
T = {e1,e2,e3,e4}
UT = {v4,v3,v7,v2,v5,v8}

Iterasi 5
T = {e1,e2,e3,e4,e5}
UT = {v4,v3,v7,v2,v5,v8,v1}

Iterasi 6
T = {e1,e2,e3,e4,e5,e6}
UT = {v4,v3,v7,v2,v5,v8,v1}

Iterasi 7
T = {e1,e2,e3,e4,e5,e6}
UT = {v4,v3,v7,v2,v5,v8,v1}
Ditemukan *Cycle* yang Memuat
v1 v2 v3 v4

Iterasi 8
T = {e1,e2,e3,e4,e5,e6}
UT = {v4,v3,v7,v2,v5,v8,v1}
Ditemukan *Cycle* yang Memuat
v2 v3 v4

Iterasi 9
T = {e1,e2,e3,e4,e5,e6,e9}
UT = {v4,v3,v7,v2,v5,v8,v1,v6}

Iterasi Selesai karena Telah Terbentuk *Minimal Spanning Tree* yang Memuat Seluruh *Vertex*

Memo 2:

Rusuk yang telah diurutkan :

e1 = (4,3) = 0.0863000000000511
e2 = (7,2) = 0.0938999999999623
e3 = (5,2) = 0.0949000000000524
e4 = (8,2) = 0.0950000000000273
e5 = (2,1) = 0.0964999999999918
e6 = (4,2) = 0.0970999999999549
e7 = (3,1) = 0.100500000000011

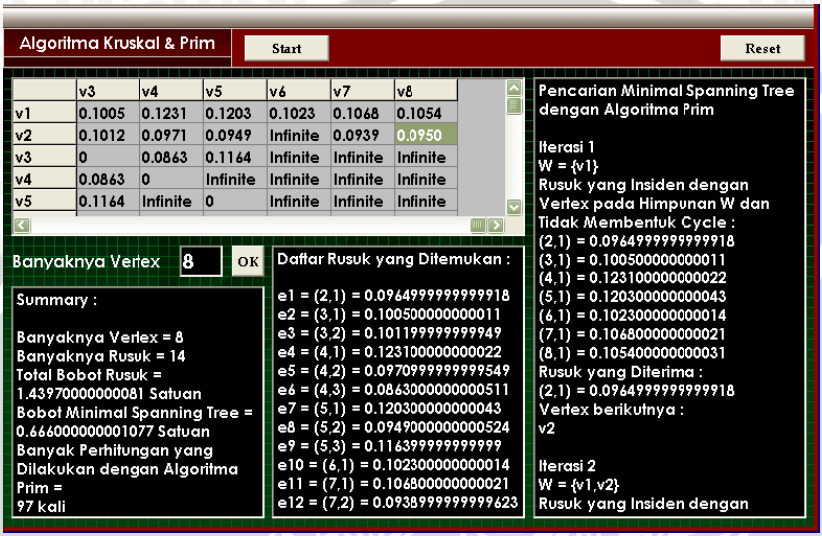
e8 = (3,2) = 0.101199999999949
e9 = (6,1) = 0.102300000000014
e10 = (8,1) = 0.105400000000031
e11 = (7,1) = 0.106800000000021
e12 = (5,3) = 0.116399999999999
e13 = (5,1) = 0.120300000000043
e14 = (4,1) = 0.123100000000022

Memo 3:

Summary :

Banyaknya *Vertex* = 8
Banyaknya Rusuk = 14
Total Bobot Rusuk =
1.43970000000081 Satuan
Bobot *Minimal Spanning Tree* =
0.666000000001077 Satuan
Banyak Perhitungan yang Dilakukan dengan Algoritma Kruskal = 592 kali

Lampiran 10. Tampilan Output dari Program Algoritma Prim



Memo 1:

Pencarian *Minimal Spanning Tree* dengan Algoritma Prim

Iterasi 1
W = {v1}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(2,1) = 0.0964999999999918
(3,1) = 0.100500000000011
(4,1) = 0.123100000000022
(5,1) = 0.120300000000043
(6,1) = 0.102300000000014
(7,1) = 0.106800000000021
(8,1) = 0.105400000000031
Rusuk yang Diterima :
(2,1) = 0.0964999999999918
Vertex berikutnya :
v2

Iterasi 2
W = {v1,v2}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(3,1) = 0.100500000000011
(4,1) = 0.123100000000022
(5,1) = 0.120300000000043
(6,1) = 0.102300000000014
(7,1) = 0.106800000000021
(8,1) = 0.105400000000031
(3,2) = 0.101199999999949
(4,2) = 0.0970999999999549
(5,2) = 0.0949000000000524
(7,2) = 0.0938999999999623
(8,2) = 0.0950000000000273
Rusuk yang Diterima :
(7,2) = 0.0938999999999623
Vertex berikutnya :
v7

Iterasi 3
W = {v1,v2,v7}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(3,1) = 0.100500000000011
(4,1) = 0.123100000000022
(5,1) = 0.120300000000043
(6,1) = 0.102300000000014
(8,1) = 0.105400000000031
(3,2) = 0.101199999999949
(4,2) = 0.0970999999999549
(5,2) = 0.0949000000000524
(8,2) = 0.0950000000000273
Rusuk yang Diterima :
(5,2) = 0.0949000000000524
Vertex berikutnya :
v5

Iterasi 4
W = {v1,v2,v7,v5}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(3,1) = 0.100500000000011
(4,1) = 0.123100000000022
(6,1) = 0.102300000000014
(8,1) = 0.105400000000031
(3,2) = 0.101199999999949
(4,2) = 0.0970999999999549
(8,2) = 0.0950000000000273
(5,3) = 0.116399999999999
Rusuk yang Diterima :
(8,2) = 0.0950000000000273
Vertex berikutnya :
v8

Iterasi 5
W = {v1,v2,v7,v5,v8}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(3,1) = 0.100500000000011
(4,1) = 0.123100000000022
(6,1) = 0.102300000000014
(3,2) = 0.101199999999949
(4,2) = 0.0970999999999549
(5,3) = 0.116399999999999
Rusuk yang Diterima :
(4,2) = 0.0970999999999549
Vertex berikutnya :
v4

Iterasi 6
W = {v1,v2,v7,v5,v8,v4}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(3,1) = 0.100500000000011
(6,1) = 0.102300000000014
(3,2) = 0.101199999999949

(5,3) = 0.116399999999999
(4,3) = 0.0863000000000511
Rusuk yang Diterima :
(4,3) = 0.0863000000000511
Vertex berikutnya :
v3

Iterasi 7
W = {v1,v2,v7,v5,v8,v4,v3}
Rusuk yang Insiden dengan *Vertex* pada Himpunan W dan Tidak Membentuk *Cycle* :
(6,1) = 0.102300000000014
Rusuk yang Diterima :
(6,1) = 0.102300000000014
Vertex berikutnya :
v6

Iterasi 8
W = {v1,v2,v7,v5,v8,v4,v3,v6}

Iterasi Selesai karena Telah Terbentuk *Minimal Spanning Tree* yang Memuat Seluruh *Vertex*

Memo 2:

Daftar Rusuk yang Ditemukan :

e1 = (2,1) = 0.0964999999999918
e2 = (3,1) = 0.100500000000011
e3 = (3,2) = 0.101199999999949
e4 = (4,1) = 0.123100000000022
e5 = (4,2) = 0.0970999999999549
e6 = (4,3) = 0.0863000000000511
e7 = (5,1) = 0.120300000000043
e8 = (5,2) = 0.0949000000000524
e9 = (5,3) = 0.116399999999999
e10 = (6,1) = 0.102300000000014
e11 = (7,1) = 0.106800000000021

e12 = (7,2) = 0.0938999999999623
e13 = (8,1) = 0.105400000000031
e14 = (8,2) = 0.0950000000000273

Memo 3:

Summary :

Banyaknya *Vertex* = 8
Banyaknya Rusuk = 14
Total Bobot Rusuk =
1.43970000000081 Satuan
Bobot *Minimal Spanning Tree* =
0.666000000001077 Satuan
Banyak Perhitungan yang Dilakukan dengan Algoritma Prim =
97 kali

LEMBAR PENGESAHAN TUGAS AKHIR

PENENTUAN *MINIMAL SPANNING TREE* MENGGUNAKAN ALGORITMA KRUSKAL DAN ALGORITMA PRIM
(Studi kasus: Konfigurasi Sentral Telepon MEA Malang)

Oleh :
NURUL ISNANIYAH
0310940045 – 94

Telah dipertahankan di depan Majelis Penguji
Pada tanggal 5 Juli 2007
Dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana Sains dalam bidang Matematika

Pembimbing I	Pembimbing II
Drs. Marsudi, MS	**Prof. Dr. Agus Widodo**
NIP. 131 759 585	**NIP. 131 281 894**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc
NIP. 132 126 049


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Nurul Isnaniyah
NIM : 0310940045 – 94
Jurusan : Matematika
Penulis Tugas Akhir Berjudul : Penentuan *Minimal Spanning Tree* Dengan Menggunakan Algoritma Kruskal dan Algoritma Prim (Studi kasus: Konfigurasi Sentral Telepon MEA Malang)

Dengan ini menyatakan bahwa :

1. **Isi dari Tugas Akhir yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Tugas Akhir ini.**
2. **Apabila dikemudian hari ternyata Tugas Akhir yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yan akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 5 Juli 2007
Yang menyatakan,

(Nurul Isnaniyah)
NIM. 0310940045 – 94


UNIVERSITAS BRAWIJAYA

PENENTUAN *MINIMAL SPANNING TREE* DENGAN MENGGUNAKAN ALGORITMA KRUSKAL DAN ALGORITMA PRIM
(Studi kasus: Konfigurasi Sentral Telepon MEA Malang)

ABSTRAK

Minimal spanning tree dari graf G adalah suatu subgraf dari G dengan jumlah bobot terkecil diantara subgraf-subgraf dari G yang terhubung. Penentuan *minimal spanning tree* dapat dilakukan menggunakan algoritma kruskal dan algoritma prim. Dalam tugas akhir ini dilakukan penerapan Algoritma Kruskal dan Algoritma Prim pada konfigurasi jaringan MEA (*Multi Exchange Area*) Malang. Dari hasil perhitungan diperoleh besarnya bobot *minimal spanning tree* (probabilitas terkecil hilangnya panggilan akibat kanal yang penuh antar sentral) adalah 0,666. Algoritma Prim lebih efisien dibandingkan Algoritma Kruskal karena tidak membutuhkan waktu yang terlalu lama untuk mendapatkan *minimal spanning tree* dari suatu graf dan dapat memperhitungkan semua bobot dari setiap garis berdasarkan *adjacent* titik yang dievaluasi. Sedangkan Algoritma Kruskal lebih sulit dalam implementasinya karena iterasi yang dilakukan terlalu panjang dan diperlukan adanya pemeriksaan kondisi siklik.

Kata kunci : Algoritma Kruskal, Algoritma Prim, *Minimal spanning tree*.

THE DETERMINATION OF MINIMAL SPANNING TREE BY USING KRUSKAL'S ALGORITHM AND PRIM'S ALGORITHM
(Case study: configuration of central telephone MEA Malang)

ABSTRACT

Minimal spanning tree from graph G is a subgraph from G with the smallest weight between subgraphs of G that are connected. The determinating of minimal spanning tree can be accomplished using kruskal's algorithm and prim's algorithm. In this final project proving about application of Kruskal's Algorithm and Prim's Algorithm at network configuration MEA (Multi Exchange Area) Malang. From the result of calculation founded the value weight of minimal spanning tree (the smallest probability of lossing call caused by looded between the central) is 0,666. Prim's Algorithm more efficient than Kruskal's Algorithm because Prim's Algorithm does not need a longer time to obtain minimal spanning tree from the graph and so can calculate all weight from every edge base on evaluated vertex's adjacent. While Kruskal's Algorithm is more difficult to be implemented because it is too long and need inspection cyclic condition.

Keywords : Kruskal's Algorithm, Prim's Algorithm, Minimal spanning tree.

KATA PENGANTAR

Segala puji syukur Alhamdulillah penulis panjatkan kehadirat Allah SWT, atas rahmat, taufik, dan hidayah-Nya, sehingga penulis dapat menyelesaikan skripsi ini dengan judul " **Penentuan *Minimal Spanning Tree* Dengan Menggunakan Algoritma Kruskal dan Algoritma Prim (Studi kasus: Konfigurasi Sentral Telepon MEA Malang)**

Keberhasilan penulis tidak lepas dari peran keluarga serta pihak-pihak yang terkait dalam penyusunan skripsi ini. Oleh karena itu, penulis menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. Drs. Marsudi, MS selaku Dosen Pembimbing I dan Prof. Dr. Agus Widodo selaku Dosen Pembimbing II atas bimbingan, pengarahan, dan nasehat yang telah diberikan kepada penulis dalam penyelesaian skripsi ini,
2. Dr. Agus Suryanto, MSc., selaku Ketua Jurusan Matematika.
3. Bapak dan Ibu Dosen Matematika yang telah memberikan ilmu pengetahuan serta seluruh staf tata usaha Jurusan Matematika atas bantuannya,
4. Kedua orang tuaku dan saudara-saudaraku yang selalu memberikan aku kasih sayang, doa, nasehat, serta semangat.
5. Semua sahabat-sahabat aku anak matematika 2003 khususnya dan angkatan 2002 atas persahabatan, bantuan, dan semangat yang telah diberikan kepada penulis,
6. Seluruh pihak yang telah membantu proses penulisan skripsi ini yang tidak dapat ditulis satu-persatu.

Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan. Oleh karena itu segala kritik dan saran yang bersifat membangun dari pembaca sangat diharapkan demi perbaikan. Semoga tulisan ini bermanfaat bagi penulis khususnya dan semua pihak pada umumnya.

Malang, Juli 2007

Penulis

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

Halaman

```
      var  Form1: TForm1;
           nVertices,nEdges,nT,nUT,k,nE,Exhaust:integer;
           TotalWeight:real;
           EdgeWeight,E,E1,E2,Vnext:array[1..100] of real;
           EdgeInc1,EdgeInc2,T,UT,W:array[1..100] of integer;


      procedure TForm1.OKClick(Sender: TObject);
      var i:integer;
B01   begin
      if nVertex.text<>'' then
B02     begin
        Start.visible:=true;
        WeightMatrix.visible:=false;
        WeightMatrix.visible:=true;
        nVertices:=StrtoInt(nVertex.text);
        WeightMatrix.ColCount:=nVertices+1;
        WeightMatrix.RowCount:=nVertices+1;
        for i:=1 to nVertices do
B03       begin
          WeightMatrix.Cells[0,i]:='v'+InttoStr(i);
          WeightMatrix.Cells[i,0]:='v'+InttoStr(i);
          WeightMatrix.Cells[i,i]:='0';
E03       end;
E02     end;
E01   end;


      procedure TForm1.StartClick(Sender: TObject);
      var i,j:integer;
B01   begin
      TotalWeight:=0;
      Kruskal.visible:=true;
      Prim.visible:=true;
      nEdges:=0;
      for i:=2 to nVertices do
          for j:=1 to i-1 do
B02         begin
            if WeightMatrix.Cells[i,j]='' then
               WeightMatrix.Cells[i,j]:='Infinite';
            if (WeightMatrix.Cells[i,j]<>'Infinite') and
               (StrtoFloat(WeightMatrix.Cells[i,j])>0) then
B03           begin
              nEdges:=nEdges+1;
              EdgeWeight[nEdges]:=StrtoFloat(WeightMatrix.Cells[i,j]);
              TotalWeight:=TotalWeight+EdgeWeight[nEdges];
              EdgeInc1[nEdges]:=i;
              EdgeInc2[nEdges]:=j;
E03           end;
            WeightMatrix.Cells[j,i]:=WeightMatrix.Cells[i,j];
E02         end;
      Memo2.Lines.Text:='Daftar Rusuk yang Ditemukan : '+chr(13);
      for i:=1 to nEdges do
          Memo2.Lines.Text:=Memo2.Lines.Text+chr(13)+'e'+InttoStr(i)+'
            ('+InttoStr(EdgeInc1[i])+','+InttoStr(EdgeInc2[i])+') =
            '+FloattoStr(EdgeWeight[i]);
      Memo3.Lines.Text:='Summary :'+chr(13)+chr(13)+'Banyaknya Vertex ='
                        +InttoStr(nVertices)+chr(13);
      Memo3.Lines.Text:=Memo3.Lines.Text+'Banyaknya Rusuk
                        ='+InttoStr(nEdges)+chr(13);
      Memo3.Lines.Text:=Memo3.Lines.Text+'Total Bobot Rusuk ='
                       +chr(13)+FloattoStr(TotalWeight)+' Satuan'+chr(13);
E01   end;
```

```
     procedure TForm1.KruskalClick(Sender: TObject);
     var i,j,l,eMin:integer;
         min,Total:real;
         Similar,Cycle:boolean;
         Sum:array[1..100]of integer;
         IncM:array[1..100,1..100]of integer;
B01  begin
     Kruskal.visible:=false;
     Prim.visible:=false;
     Exhaust:=0;
     Total:=0;
     nE:=0;
     Memo2.Lines.Text:='Rusuk yang telah diurutkan :'+chr(13);
R01  repeat
          min:=EdgeWeight[1];
          eMin:=1;
          for i:=2 to nEdges do
              if EdgeWeight[i]<min then
B02               begin
                  Exhaust:=Exhaust+1;
                  min:=EdgeWeight[i];
                  eMin:=i;
E02               end;
          nE:=nE+1;
          E[nE]:=EdgeWeight[eMin];
          E1[nE]:=EdgeInc1[eMin];
          E2[nE]:=EdgeInc2[eMin];
          Memo2.Lines.Text:=Memo2.Lines.Text+chr(13)+'e'+InttoStr(nE)+' = ('
                  +InttoStr(E1[nE])+','+InttoStr(E2[nE])+') = '
                  +FloattoStr(E[nE]);
          for i:=eMin+1 to nEdges do
B03           begin
              Exhaust:=Exhaust+1;
              EdgeWeight[i-1]:=EdgeWeight[i];
              EdgeInc1[i-1]:=EdgeInc1[i];
              EdgeInc2[i-1]:=EdgeInc2[i];
E03           end;
          nEdges:=nEdges-1;
U01  until nEdges=0;
     k:=0;
     nT:=0;
     nUT:=0;
     T[k]:=0;
     Memo1.Lines.Text:='Pencarian Minimal Spanning Tree dengan Algoritma
                       Kruskal'+chr(13)+chr(13);
R02  repeat
          nT:=nT+1;
          k:=k+1;
          T[nT]:=k;
          for i:=1 to nVertices do
B04           begin
              Sum[i]:=0;
              for j:=1 to nT do
B05               begin
                  Exhaust:=Exhaust+1;
                  if (E1[T[j]]=i) or (E2[T[j]]=i) then
                     IncM[T[j],i]:=1
                  else
                      IncM[T[j],i]:=0;
                  Sum[i]:=Sum[i]+IncM[T[j],i];
E05               end;
E04           end;
          Cycle:=false;
          if nT>2 then
```

```
R03          repeat
                 for i:=1 to nVertices do
B06                 begin
                    if (Sum[i]=1) then
                       for j:=1 to nT do
                          if (E1[T[j]]=i) or (E2[T[j]]=i) then
B07                          begin
                             Exhaust:=Exhaust+1;
                             IncM[T[j],E1[T[j]]]:=0;
                             incM[T[j],E2[T[j]]]:=0;
E07                          end;
E06                 end;
                 for i:=1 to nVertices do
B08                 begin
                    Sum[i]:=0;
                    Exhaust:=Exhaust+1;
                    for j:=1 to nT do
                        Sum[i]:=Sum[i]+IncM[T[j],i];
E08                 end;
                 Cycle:=true;
                 l:=0;
                 for i:=1 to nVertices do
B09                 begin
                    Exhaust:=Exhaust+1;
                    if (Sum[i]=1) then
                       Cycle:=false
                    else if (Sum[i]=0) then
                       l:=l+1;
E09                 end;
                 if l=nVertices then
                    Cycle:=false;
U03          until (Cycle) or (l=nVertices);
          if Cycle then
             nT:=nT-1
          else
B10          begin
             Similar:=false;
             for i:=1 to nUT do
                 if E1[k]=UT[i] then
B11                 begin
                    Exhaust:=Exhaust+1;
                    Similar:=true;
E11                 end;
             if not Similar then
B12             begin
                nUT:=nUT+1;
                UT[nUT]:=E1[k];
E12             end;
             Similar:=false;
             for i:=1 to nUT do
                 if E2[k]=UT[i] then
B13                 begin
                    Exhaust:=Exhaust+1;
                    Similar:=true;
E13                 end;
             if not Similar then
B14             begin
                nUT:=nUT+1;
                UT[nUT]:=E2[k];
E14             end;
E10          end;
          Memo1.Lines.Text:=Memo1.Lines.Text+'Iterasi '+InttoStr(k)+chr(13)+'T
                            = {';
          for i:=1 to nT do
```

```
B15             begin
                Memo1.Lines.Text:=Memo1.Lines.Text+'e'+InttoStr(T[i]);
                if i<nT then
                   Memo1.Lines.Text:=Memo1.Lines.Text+',';
E15             end;
           Memo1.Lines.Text:=Memo1.Lines.Text+'}'+chr(13)+'UT = {';
           for i:=1 to nUT do
B16             begin
                Memo1.Lines.Text:=Memo1.Lines.Text+'v'+InttoStr(UT[i]);
                if i<nUT then
                   Memo1.Lines.Text:=Memo1.Lines.Text+',';
E16             end;
           Memo1.Lines.Text:=Memo1.Lines.Text+'}'+chr(13);
           if Cycle then
B17           begin
              Memo1.Lines.Text:=Memo1.Lines.Text+'Ditemukan Cycle yang Memuat'
                                +chr(13);
              for i:=1 to nVertices do
                 if Sum[i]=2 then
                    Memo1.Lines.Text:=Memo1.Lines.Text+'v'+InttoStr(i)+' ';
              Memo1.Lines.Text:=Memo1.Lines.Text+chr(13);
E17           end;
           Memo1.Lines.Text:=Memo1.Lines.Text+chr(13);
U02     until nT=nVertices-1;
        for i:=1 to nT do
           Total:=Total+E[T[i]];
        Memo1.Lines.Text:=Memo1.Lines.Text+'Iterasi Selesai karena Telah Terbentuk
                          Minimal Spanning Tree yang Memuat Seluruh Vertex';
        Memo3.Lines.Text:=Memo3.Lines.Text+'Bobot Minimal Spanning Tree ='
                          +chr(13)+FloattoStr(Total)+' Satuan'+chr(13);
        Memo3.Lines.Text:=Memo3.Lines.Text+'Banyak Perhitungan yang Dilakukan
                          dengan Algoritma Kruskal ='+chr(13)+InttoStr(Exhaust)+'
                          kali';
E01     end;


        procedure TForm1.PrimClick(Sender: TObject);
        var i,j,l,eMin:integer;
            min,Total:real;
            Similar:boolean;
B01     begin
        Kruskal.visible:=false;
        Prim.visible:=false;
        Exhaust:=0;
        k:=1;
        W[1]:=1;
        Total:=0;
        Memo1.Lines.Text:='Pencarian Minimal Spanning Tree dengan Algoritma
        Prim'+chr(13)+chr(13);
R01     repeat
              nE:=0;
              Memo1.Lines.Text:=Memo1.Lines.Text+'Iterasi '+InttoStr(k)+chr(13);
              Memo1.Lines.Text:=Memo1.Lines.Text+'W = {';
              for i:=1 to k do
B02               begin
                  Memo1.Lines.Text:=Memo1.Lines.Text+'v'+InttoStr(W[i]);
                  if i<k then
                     Memo1.Lines.Text:=Memo1.Lines.Text+',';
E02               end;
              Memo1.Lines.Text:=Memo1.Lines.Text+'}';
              if k<nVertices then
```

```
B03         begin
            Memo1.Lines.Text:=Memo1.Lines.Text+chr(13)+'Rusuk yang Insiden
                        dengan Vertex pada Himpunan W dan Tidak
                        Membentuk Cycle :'+chr(13);
            for i:=1 to k do
                for j:=1 to nEdges do
                    if (EdgeInc1[j]=W[i]) or (EdgeInc2[j]=W[i]) then
B04                 begin
                    Exhaust:=Exhaust+1;
                    Similar:=false;
                    for l:=1 to nE do
                        if (EdgeInc1[j]=E1[l]) and (EdgeInc2[j]=E2[l])
then
                                Similar:=true;
                    nE:=nE+1;
                    E[nE]:=EdgeWeight[j];
                    E1[nE]:=EdgeInc1[j];
                    E2[nE]:=EdgeInc2[j];
                    if E1[nE]=W[i] then
                       Vnext[nE]:=E2[nE]
                    else
                        Vnext[nE]:=E1[nE];
                    for l:=1 to k do
                        if Vnext[nE]=W[l] then
B05                        begin
                           Similar:=true;
                           nE:=nE-1;
E05                        end;
                    if not Similar then
                        Memo1.Lines.Text:=Memo1.Lines.Text+'('+InttoStr(E1
                        [nE])+','+InttoStr(E2[nE])+' ) =
                        '+FloattoStr(E[nE])+chr(13);
E04                 end;
            min:=E[1];
            eMin:=1;
            for i:=2 to nE do
                if E[i]<min then
B06                begin
                   Exhaust:=Exhaust+1;
                   min:=E[i];
                   eMin:=i;
E06                end;
            for i:=1 to nEdges do
                if (EdgeInc1[i]=E1[eMin]) and (EdgeInc2[i]=E2[eMin]) then
B07                begin
                   Exhaust:=Exhaust+1;
                   Memo1.Lines.Text:=Memo1.Lines.Text+'Rusuk yang Diterima :'
                                                      +chr(13);
                    Memo1.Lines.Text:=Memo1.Lines.Text+'('+InttoStr(E1[eMin])
                                      +',' +InttoStr(E2[eMin])+') =
                                      '+FloattoStr(E[eMin]);
                   Memo1.Lines.Text:=Memo1.Lines.Text+chr(13)+'Vertex
                                      berikutnya:'+chr(13)+'v'+InttoStr(Vnext
                                      [eMin])+chr(13);
                   Total:=Total+E[eMin];
                   k:=k+1;
                   W[k]:=Vnext[eMin];
                   for j:=i+1 to nEdges do
B08                    begin
                       EdgeWeight[j-1]:=EdgeWeight[j];
                       EdgeInc1[j-1]:=EdgeInc1[j];
                       EdgeInc2[j-1]:=EdgeInc2[j];
E08                    end;
                   nEdges:=nEdges-1;
```

```
E07                 end;
                Memo1.Lines.Text:=Memo1.Lines.Text+chr(13);
E03             end
            else
B09             begin
                Memo1.Lines.Text:=Memo1.Lines.Text+chr(13)+chr(13)+
                                  'Iterasi Selesai karena Telah Terbentuk
                                  Minimal Spanning Tree yang Memuat Seluruh
                                  Vertex';
                Memo3.Lines.Text:=Memo3.Lines.Text+'Bobot Minimal Spanning Tree
                                  ='+chr(13)+FloattoStr(Total)+'
                                  Satuan'+chr(13);
                Memo3.Lines.Text:=Memo3.Lines.Text+'Banyak Perhitungan yang
                                  Dilakukan dengan Algoritma Prim
                                  ='+chr(13)+InttoStr(Exhaust)+ ' kali';
                k:=k+1;
E09             end;


U01   until k=nVertices+1;
E01   end;

      procedure TForm1.ResetClick(Sender: TObject);
      var i,j:integer;
B01   begin
      nVertex.text:='';
      nVertices:=0;
      nEdges:=0;
      for i:=1 to 100 do
B02       begin
          EdgeWeight[i]:=0;
          EdgeInc1[i]:=0;
          EdgeInc2[i]:=0;
          T[i]:=0;
          UT[i]:=0;
          W[i]:=0;
          E[i]:=0;;
          E1[i]:=0;
          E2[i]:=0;
          Vnext[i]:=0;
E02       end;
      WeightMatrix.ColCount:=7;
      WeightMatrix.RowCount:=6;
      for i:=0 to 100 do
          for j:=0 to 100 do
              WeightMatrix.Cells[i,j]:='';
      WeightMatrix.visible:=false;
      WeightMatrix.visible:=true;
      Start.visible:=false;
      Kruskal.visible:=false;
      Prim.visible:=false;
      Memo1.Lines.text:='';
      Memo2.Lines.text:='';
      Memo3.Lines.text:='';
E01   end;
```