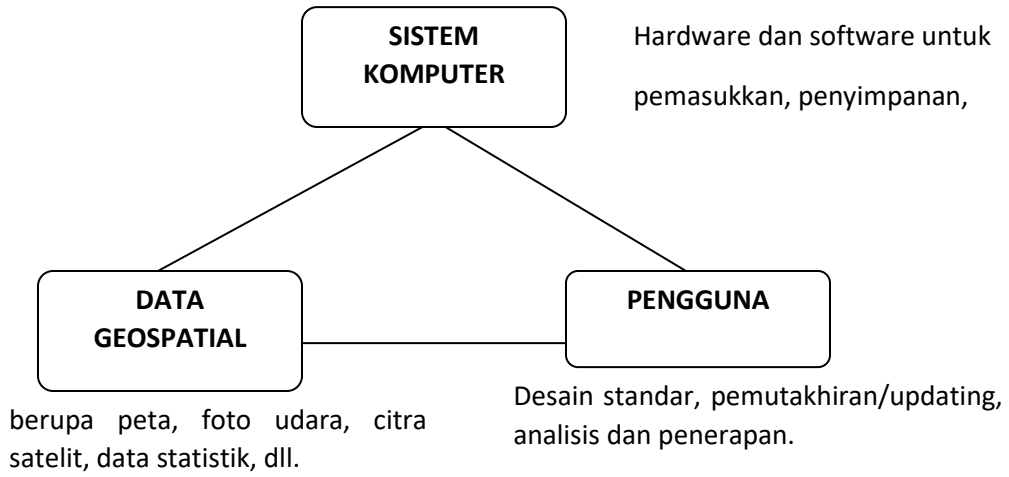
Supporting teory nya saya ambil dulu dari penggambungan 2 Algoritma:

1. Algoritma Dijkstra

Pengertian Sistem Informasi Geografis

GIS (Geografical Information System) atau dikenal pula dengan SIG (Sistem Informasi Geografis) merupakan sistem infomasi berbasis komputer yang menggabungkan antara unsur peta (geografis) dan informasinya tentang peta tersebut (data atribut) yang dirancang untuk mendapatkan, mengolah, memanipulasi, analisa, memperagakan dan menampilkan data spatial untuk menyelesaikan perencanaan, mengolah dan meneliti permasalahan. Dengan definisi ini, maka terlihat bahwa aplikasi SIG dilapangan cukup luas terutama bagi bidang yang memerlukan adanya suatu sistem informasi tidak hanya menyimpan, menampilkan, dan menganalisa data atribut saja tetapi juga unsur geografisnya seperti PT. Telkom, Pertamina, Departemen Kelautan, Kehutanan, Bakosurtanal, Marketing, Perbankan, Perpajakan, dan yang lainnya.

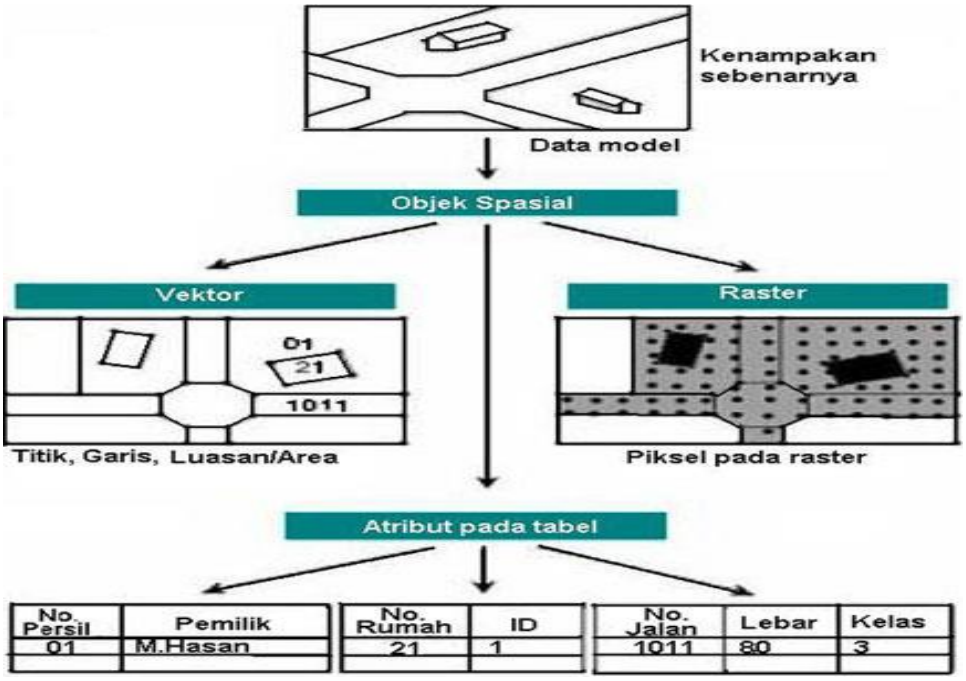
Komponen kunci dalam GIS adalah sistem komputer, data geospatial (data atribut) dan pengguna , yang dapat digambarkan sebagai berikut :



Gambar II.2. Komponen kunci SIG

Sistem komputer untuk SIG terdiri dari perangkat keras (*hardware*), perangkat lunak (*software*) dan prosedur untuk penyusunan pemasukkan data, pengolahan, analisis, pemodelan (*modelling*), dan penayangan data geospatial. Sumber-sumber data geospatial adalah peta digital, foto udara, citra satelit, tabel statistik dan dokumen lain yang berhubungan.

Data geospatial dibedakan menjadi data grafis (atau disebut juga data geometris) dan data atribut (data tematik), lihat Gambar II.3. Data grafis mempunyai tiga elemen: titik (*node*), garis (*arc*) dan luasan (*poligon*) dalam bentuk vector ataupun raster yang mewakili geometri topologi, ukuran, bentuk, posisi dan arah.

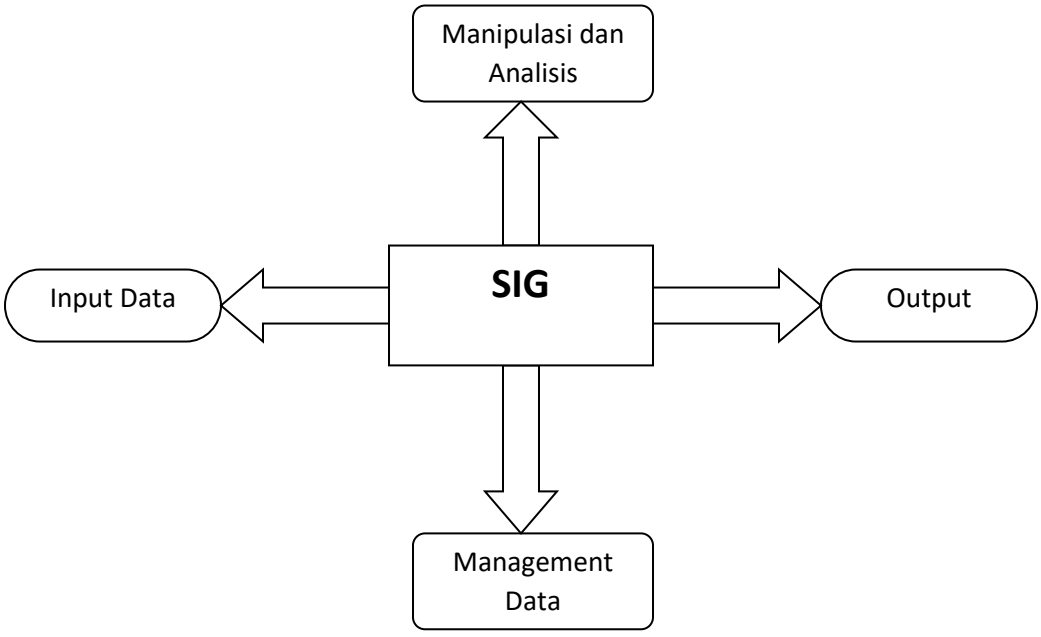


Gambar II.3. Konsep Data Geospasial

Fungsi pengguna adalah untuk memilih informasi yang diperlukan, membuat standar, membuat jadwal pemutakhiran (*updating*) yang efisien, menganalisis hasil yang dikeluarkan untuk kegunaan yang diinginkan dan merencanakan aplikasi.

II.2.1 Subsistem

Sistem informasi geografis merupakan sistem yang dapat mendukung pengambilan keputusan spasial dan mampu mengintegrasikan deskripsi-deskripsi lokasi dengan karakteristik-karakteristik fenomena yang ditemukan di lokasi tersebut. SIG dapat diuraikan menjadi beberapa subsistem berikut :



Gambar II.4. Subsistem-subsistem SIG

1. Subsistem masukan (input)

Subsistem ini bertugas mengumpulkan data dan mempersiapkan data spasial dan atribut dari berbagai sumber. Subsistem ini juga bertanggungjawab mengkonversi atau mentransformasi format-format data asli ke dalam format yang dapat digunakan oleh SIG.

2. Subsistem manajemen

Subsistem ini mengorganisasikan data spasial maupun atribut ke dalam sebuah sistem basisdata sedemikian rupa sehingga data spsial tersebut mudah dicari, di-*update* dan diedit.

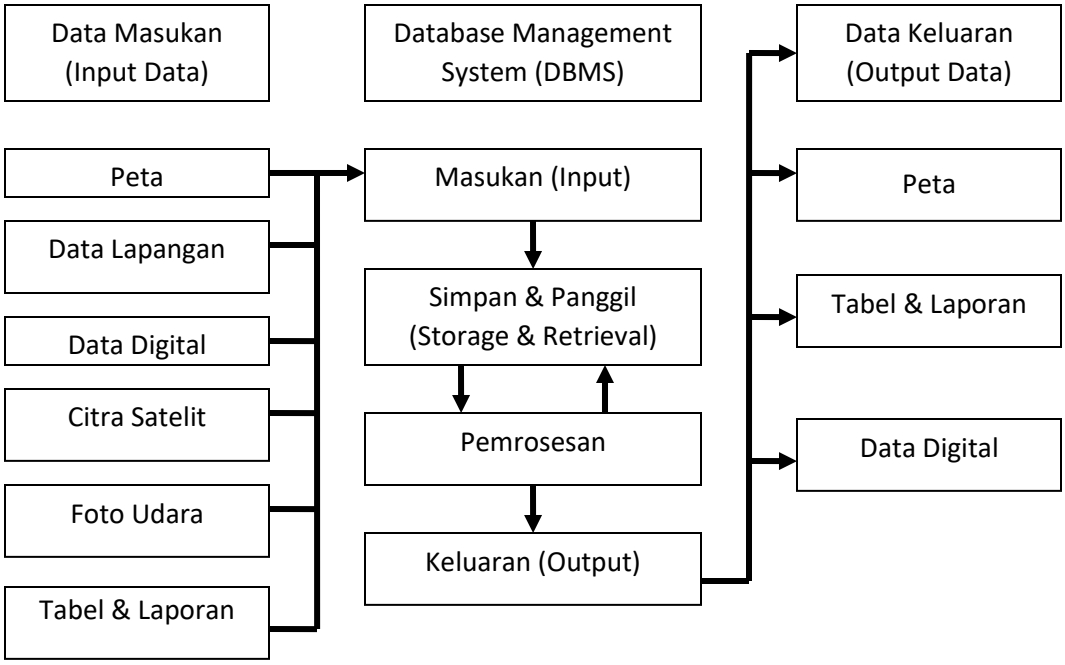
3. Subsistem manipulasi dan analisis

Subsistem ini menentukan informasi-informasi yang dapat dihasilkan oleh SIG. Selain itu, subsistem ini juga melakukan manipulasi dan pemodelan data untuk menghasilkan informasi yang diharapkan.

4. Subsistem keluaran (output) dan Penyajian (display)

Subsistem ini menampilkan atau menghasilkan keluaran seluruh atau sebagian basisdata, baik dalam bentuk softcopy maupun hardcopy, dalam format table, grafik, peta atau format lainnya.

Subsistem ini dapat diperjelas berdasarkan uraian jenis masukan, proses dan jenis keluarannya dengan bagan berikut :



Gambar II.5. Uraian subsistem-subsistem SIG

II.2.2 Kemampuan SIG

SIG dapat merepresentasikan dunia nyata (real world) pada layar komputer seperti lembaran peta kertas. SIG mempunyai kekuatan dan fleksibilitas lebih dari lembaran peta. Beberapa keunggulan SIG dibanding pekerjaan manual dapat dilihat pada tabel II.1

Tabel II.1. kelebihan SIG dan kekurangan pekerjaan manual tanpa SIG.

	SIG	Pekerjaan Manual
Penyimpanan	Basisdata Digital baku dan terpadu	Skala dan standar berbeda
Pemanggilan kembali	Pencarian dengan komputer	Cek manual
Pemutakhiran	Sistematis	Mahal dan memakan waktu

Analisis overlay	Sangat cepat	Memakan waktu dan tenaga
Analisis spasial	Mudah	Rumit
Penayangan (display)	Murah dan cepat	Mahal dan memakan waktu

Berikut adalah alasan dibutuhkannya SIG:

- penanganan data geospatial sangat buruk
- peta dan statistik sangat cepat kadaluarsa
- data dan informasi sering tidak akurat
- tidak ada pelayanan penyediaan data
- tidak ada pertukaran data

Dan begitu SIG diterapkan, didapat keuntungan berikut:

- penanganan data geospatial menjadi lebih baik dalam format baku
- revisi dan pemutakhiran data menjadi lebih mudah
- data geospatial dan informasi lebih mudah dicari, dianalisis dan direpresentasikan
- menjadi produk bernilai tambah
- data geospatial dapat dipertukarkan
- produktivitas staf meningkat dan lebih efisien
- penghematan waktu dan biaya
- keputusan yang akan diambil menjadi lebih baik

II.3 Pengetahuan Peta

Peta dalam SIG dapat digunakan baik sebagai input maupun output. Pemetaan merupakan suatu proses yang terdiri dari beberapa tahapan kerja (pengumpulan data, pengolahan data, penyajian data), serta melibatkan beberapa disiplin ilmu (surveying, fotogrametri, penginderaan jauh, kartografi) yang satu sama lain berkaitan.

Peta merupakan penyajian grafis dari sebagian atau seluruh permukaan bumi pada suatu bidang datar dengan menggunakan suatu skala dan sistem proyeksi tertentu. Penyajian unsusr-unsur permukaan bumi pada suatu peta dilakukan dengan caramemilih, mengenerelisasi data permukaan bumi, sesuai dengan maksud dan tujuan pembuatan peta tersebut. Peta menyajikan sejumlah informasi mengenai permukaan bumi yang diharapakan dapat digunakan secara baik oleh pengguna.

Peta mempunyai beberapa fungsi yaitu :

1. memperlihatkan posisi atau lokasi relatif dari suatu tempat.
2. memperlihatkan bentuk atau ukuran unsur yang terdapat dipermukaan bumi
3. memperlihatkan ukuran dalam pengertian jarak dan arah
4. menghimpun serta menyeleksi data permukaan bumi.

Persyaratan-persyaratan geometrik yang harus dipenuhi oleh peta ayang ideal adalah :

1. jarak antara titik-titik yang terletak di atas peta harus sesuai dengan jarak aslinya di permukaan bumi (dengan memperhatikan faktor skala peta)
2. luas suatu unsur yang direpresentasikan di atas peta harus sesuai dengan luas sebenarnya (dengan memperhatikan faktor skala peta)
3. sudut atau arah suatu garis yang direpresentasikan dia atas peta harus sesuai dengan arah sebenarnya seperti di permukaan bumi

4. bentuk suatu unsur yang direpresentasikan di atas peta harus sesuai dengan bentuk yang sebenarnya (dengan memperhatikan faktor skala peta).

Adalah tidak mungkin membuat suatu peta yang ideal sebagaimanayang disebutkan di atas karena permukaan bumi merupakan bidang lengkung yang tidak teratur. Akan tetapi, dapat dibuat peta yang memenuhi salah satu syarat di atas, yang disesuaikan dengan tujuan pembuatan peta tersebut.

II.4. Pengertian MapInfo

Perangkat lunak Sistem Informasi Geografis saat ini telah banyak dijumpai dipasaran. Masing-masing perangkat lunak ini mempunyai kelebihan dan kekurangan dalam menunjang analisis informasi geografis. Salah satu yang sering digunakan saat ini adalah mapinfo. MapInfo merupakan perangkat lunak aplikasi untuk keperluan sistem informasi geografis yang dikembangkan oleh MapInfo Coorporatin. MapInfo mempunyai karakteristikyang menarik, mudah digunakan , user friendly dan memiliki tampilan yang interaktif.

Arsitektur Mapinfo

Elemen-elemen dasar MapInfo adalah :

- Icon Menu Control yang digunakan untuk mengontrol jendela MapInfo yang sedang aktif.
- Title Bar yang berisi nama program aplikasi dan nama file yang sedang aktif. Title Bar (baris judul) ini dapat digunakan untuk memindahkan jendela ke posisi lain yang diingikan.
- Menu Bar yang berisi barisan perintah berupa menu-menu, misalnya menu File, Edit, Tools, Object, Query, Table, Options, Window, dan Help.
- Toolbar yang berisi tombol-tombol yang digunakan untuk menjalankan suatu perintah secara cepat dan mudah, terutama perintah-perintah yang sering digunakan.

- Status Bar yang berisi tool-tool untuk mengetahui keadan suatu proses dalam MapInfo.

II.5. Kota Medan

Kota Medan (dahulu daerah tingkat II berstatus kotamadya) adalah ibu kota provinsi Sumatera Utara. Medan adalah pintu gerbang wilayah Indonesia bagian barat dan juga sebagai pintu gerbang bagi para wisatawan untuk menuju objek wisata Brastagi di daerah dataran tinggi Karo, objek wisata Orangutan di Bukit Lawang, Danau Toba, yang terkenal sebagai tempat wisata, serta Pantai Cermin, yang tekenal dengan pemandangan lautnya dilengkapi dengan waterboom Theme Park. Kota Medan dipimpin oleh seorang walikota, yang saat ini dijabat oleh Drs. H. Afifuddin Lubis, MSi (penjabat walikota Medan). Wilayah Kota Medan kemudian dibagi lagi menjadi 21 kecamatan dan 151 kelurahan.

- Medan Tuntungan
- Medan Johor
- Medan Amplas
- Medan Denai
- Medan Area
- Medan Kota
- Medan Maimun
- Medan Polonia
- Medan Baru
- Medan Selayang
- Medan Sunggal
- Medan Helvetia
- Medan Petisah
- Medan Barat
- Medan Timur
- Medan Perjuangan
- Medan Tembung
- Medan Deli
- Medan Labuhan
- Medan Marelan
- Medan Belawan

Geografi

Kota Medan memiliki luas 26.510 Hektar (265,10 Km^2) atau 3,6% dari keseluruhan wilayah Sumatera Utara. Dengan demikian, dibandingkan dengan kota/kabupaten lainya, Kota Medan memiliki luas wilayah yang relatif kecil, tetapi dengan jumlah penduduk yang relatif

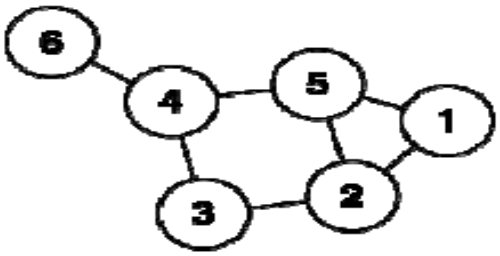
besar. Secara geografis kota Medan terletak pada 3° 30' – 3° 43' Lintang Utara dan 98° 35' - 98° 44' Bujur Timur.

Untuk itu topografi kota Medan cenderung miring ke utara dan berada pada ketinggian 2,5 - 37,5 meter diatas permukaan laut. Secara administratif , wilayah kota medan hampir secara keseluruhan berbatasan dengan Daerah Kabupaten Deli Serdang, yaitu sebelah Barat, Selatan dan Timur. Sepanjang wilayah Utara nya berbatasan langsung dengan Selat Malaka, yang diketahui merupakan salah satu jalur lalu lintas terpadat di dunia. Kabupaten Deli Serdang merupakan salah satu daerah yang kaya dengan Sumber Daya alam (SDA), khususnya di bidang perkebunan dan kehutanan. Karenanya secara geografis kota Medan didukung oleh daerah-daerah yang kaya Sumber daya alam seperti Deli Serdang, Labuhan Batu, Simalungun, Tapanuli Utara, Tapanuli Selatan, Mandailing Natal, Karo, Binjai dan lain-lain. Kondisi ini menjadikan kota Medan secara ekonomi mampu mengembangkan berbagai kerjasama dan kemitraan yang sejajar, saling menguntungkan, saling memperkuat dengan daerah-daerah sekitarnya. Di samping itu sebagai daerah yang pada pinggiran jalur pelayaran Selat Malaka. Maka Kota Medan memiliki posisi strategis sebagai gerbang (pintu masuk) kegiatan perdagangan barang dan jasa, baik perdagangan domestik maupun kuar negeri (ekspor-impor). Posisi geografis Kota Medan ini telah mendorong perkembangan kota dalam 2 kutub pertumbuhan secara fisik , yaitu daerah terbangun Belawan dan pusat Kota Medan saat ini.

II.6 Pengertian Graf

Graf G didefinisikan sebagai pasangan himpunan (V,E) yang dalam hal ini V adalah himpunan tak kosong dari simpul-simpul (vertices atau node), dan E adalah himpunan sisi-sisi

(edges atau arcs) yang menghubungkan sepasang simpul, atau dapat ditulis singkat dengan notasi G = (V,E).



Gambar II.6. Graf dengan 6 verteks

Sebuah struktur graf dapat dikembangkan dengan memberi bobot pada setiap *edge.* Graf berbobot dapat digunakan untuk melambangkan banyak konsep berbeda. Jika suatu graf melambangkan jaringan jalan, maka bobotnya dapat berarti panjang jalan maupun batas kecepatan pada batas tertentu. Graf berbobot inilah yang digunakan untuk mencari lintasan terpendek.

Pencarian lintasan terpendek dalam graf berarti meminimalisasi bobot suatu lintasan dalam graf. Aplikasinya banyak ditemukan dalam kehidupan sehari-hari seperti untuk mencari lintasan terpendek antara dua kota dan menentukan jalur komunikasi terpendek antara dua buah terminal komputer. Ada beberapa macam persoalan lintasan terpendek, antara lain : lintasan terpendek antara dua buah simpul, lintasan terpendek antara semua pasangan simpul, lintasan terpendek dari simpul tertentu ke semua simpul lain, dan lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu. Terdapat banyak cara untuk menentukan lintasan terpendek suatu graf. Hingga saat ini sudah banyak algoritma pencarian lintasan terpendek yang ditulis orang. Diantaranya algoritma Floyd-

Warshall, algoritma Johnson, algoritma Dijkstra, dan algoritma Bellman-Ford. Namun yang paling banyak digunakan ialah algoritma Dijkstra.

Algoritma Dijkstra awalnya diterapkan untuk mencari lintasan terpendek pada graf berarah, namun algoritma ini juga benar untuk digunakan pada graf tak berarah.

II.7 Jenis dan Sifat Graf

Teori graf memiliki banyak terapan hingga saat ini. Graf digunakan untuk merepresentasikan objek diskrit dan hubungan antara objek-objek tertentu. Representasi visual graf adalah dengan menyatakan objek sebagai noktah, bulatan, atau titik. Sedangkan hubungan antara objek dinyatakan dengan garis.

II.7.1 Jenis Graf

Pengelompokkan graf dapat didasarkan pada ada tidaknya sisi ganda atau sisi kalang, pada jumlah simpul, atau berdasarkan orientasi arah pada sisi.

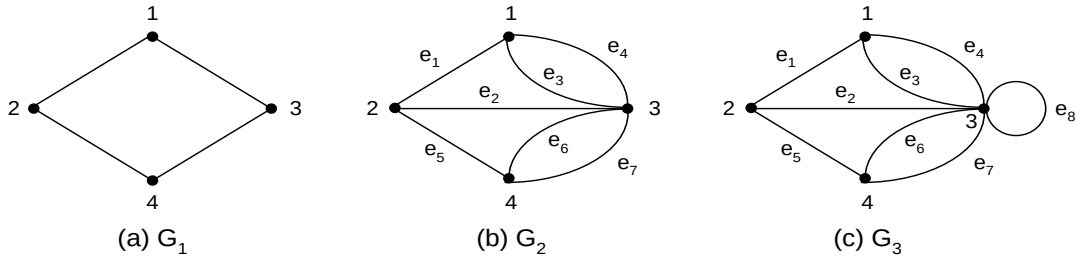
Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, secara umum graf dapat digolongkan menjadi dua jenis :

1. Graf sederhana

   Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana. Jaringan komputer merupakan contoh aplikasi graf sederhana.

2. Graf tak sederhana

Dibagi menjadi 2, yaitu graf ganda dan graf semu. Graf ganda ialah graf yangmengandung sisi ganda. Graf semu ialah graf yang mengandung gelang.



(a) G_1 (b) G_2 (c) G_3

Gambar II.7. Graf berdasarkan ada tidaknya sisi gelang atau sisi ganda,

(a). Graf sederhana, (b). Graf ganda, (c). Graf semu

Berdasarkan jumlah simpul pada suatu graf, secara umum graf dapat digolongkan menjadi dua jenis :

1. Graf berhingga

   Ialah graf yang jumlah simpulnya berhingga.

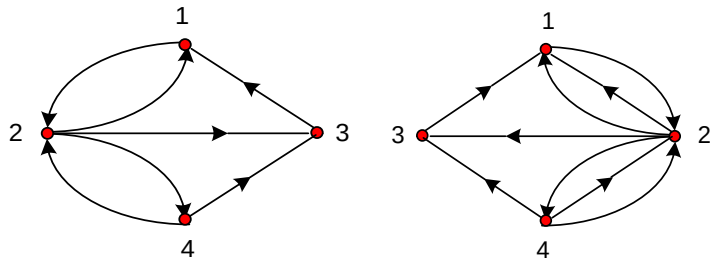
2. Graf tak berhingga

   Ialah graf yang jumlah simpulnya tak berhingga.

Berdasarkan orientasi arah pada sisi, secara umum graf dibedakan atas dua jenis :

1. Graf berarah

   Graf yang setiap sisinya diberikan orientasi arah disebut graf berarah.



(a) (b)

Gambar II.8. (a). Graf berarah, (b). Graf berarah ganda

2. Graf tak berarah

   Graf yang sisinya tidak mempunyai orientasi arah disebut *graf* tak berarah. Pada *graf* tak berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi, (u, v) = (v, u) adalah sisi yang sama. Tiga buah *graf* pada Gambar II.7 adalah *graf* tak berarah. Pada jaringan telepon, sisi pada *graf* menyatakan bahwa saluran telepon dapat beroperasi pada dua arah.

II.7.2 Terminologi Dasar

Banyak terminologi yang akan sering digunakan dalam pembahasan graf, diantaranya:

1. Bertetangga (Adjacent)

   Dua buah simpul pada graf tak-berarah G dikatakan bertetangga jika keduanya terhubung langsung dengan sebuah sisi.

2. Bersisian (Incident)

   Untuk sembarang sisi e = (v_j, v_k), sisi e dikatakan bersisian dengan simpul v_j dan v_k.

3. Simpul Terpencil (Isolated Vertex)

   Simpul yang tidak mempunyai sisi yang bersisian dengannya disebut simpul terpencil.

4. Graf Kosong (Null Graf atau Empty Graf)

   Graf yang himpunan sisinya merupakan himpunan kosong disebut graf kosong.

Gambar II.9. Graf kosong

5. Derajat (Degree)

   Derajat suatu simpul pada graf tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut. Pada graf berarah, derajat suatu simpul ialah jumlah busur yang masuk ke simpul ditambah dengan jumlah busur yang keluar dari simpul.

6. Lintasan (Path)

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

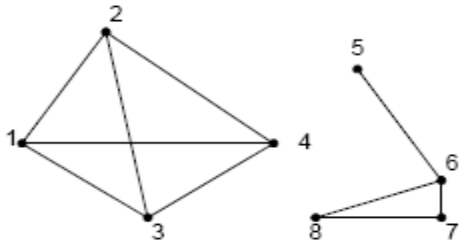
   Panjang lintasan adalah jumlah sisi dalam lintasan tersebut.

7. Siklus (Cycle) atau Sirkuit (Circuit)

   Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus. Panjang sirkuit adalah jumlah sisi di dalam sirkuit tersebut.

8. Terhubung (Connected)

   Graf tak berarah G disebut graf terhubung jika untuk setiap pasang simpul v_i dan v_j dalam himpunan V terdapat lintasan dari v_i ke v_j. Jika tidak, maka graf G tak terhubung.



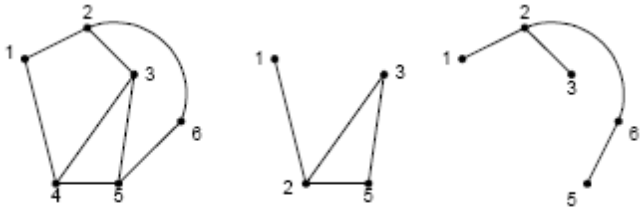
(a) (b)

Gambar II.10. (a). Graf terhubung, (b). Graf tak terhubung

9. Upagraf (Subgraf) dan Komplemen Upagraf

Misalkan G = (V,E) adalah sebuah graf. $G_1 = (V_1, E_1)$ adalah upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.

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



(a) (b) (c)

Gambar II.11. (a). Graf G_1, (b). Upagraf dari G_1,

(c). Komplemen dari upagraf yang bersesuaian.

10. Upagraf Merentang (Spanning Subgraf)

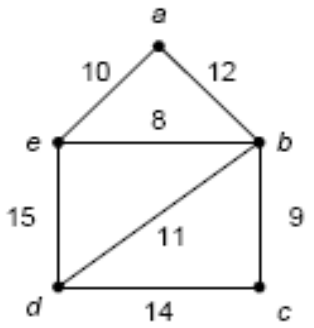
Upagraf $G_1 = (V_1, E_1)$ dari G = (V,E) dikatakan upagraf merentang jika $V_1 = V$ (yaitu G_1 mengandung semua simpul dari G)

11. Cut-Set

    Cut-set dari graf terhubung G adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung. Jadi, cut-set selalu menghasilkan dua buah komponen terhubung. Nama lain untuk cut-set ialah bridge (jembatan).

12. Graf Berbobot (Weighted Graf)

    Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga (bobot). Graf inilah yang digunakan untuk mencari lintasan terpendek.



Gambar II.12. Graf berbobot

II.8 Lintasan Terpendek

Persoalan mencari lintasan terpendek (shortest path problem) dalam graf merupakan persoalan optimasi. Pencarian lintasan terpendek termasuk masalah yang paling umum dalam suatu weighted-connected graf. Dalam masalah ini terdapat subkelas-subkelas masalah yang lebih spesifik. Misalnya pada jaringan jalan raya yang menghubungkan kota-kota disuatu wilayah, hendak dicari lintasan terpendek yag menghubungkan antara dua kota berlainan tertentu

(Single-source shortest path problems), semua lintasan terpendek masing-masing dari suatu kota ke setiap kota lainnya (Single-source shortest path problems), semua lintasan terpendek masing-masing antara tiap kemungkinan pasang kota yang berbeda (all-pairs shortest path problems). Untuk memecahkan masing-masing dari masalah-masalah tersebut terdapat sejumlah solusi. Yaitu algoritma Dijkstra untuk single-source shortest path, algoritma Floyd-Warshall untuk masalah all-pairs shortest path, dan algoritma Johnson untuk masalah all-pairs shortest path pada sparse graf. Dalam beberapa masalah graf lain, suatu graf dapat memiliki bobot negatif dan kasus ini dipecahkan oleh algoritma Bellman-Ford.

Yang akan dibahas di sini adalah algoritma Dijkstra yaitu mencari lintasan terpendek dari suatu verteks asal tertentu ke setiap verteks lainnya (algoritma ini juga berfungsi sangat optimal pada masalah single-destination).

II.9 Algoritma Dijkstra

Algoritma ini diberi nama sesuai nama penemunya, Edsger Wybe Dijkstra. Algoritma Dijkstra mencari lintasan terpendek dalam sejumlah langkah. Algoritma Dijkstra adalah sebuah prosedur iteratif untuk mencari lintasan terpendek antara dua simpul, *a* dan *z*, di dalam graf dengan pembobot. Prosedur ini dilaksanakan dengan cara mencari panjang lintasan terpendek dari sebuah simpul pendahulu dan menambahkan simpul-simpul tersebut ke himpunan simpul *S*. Algoritma berhenti setelah simpul *z* tercapai. Algoritma ini menggunakan prinsip greedy yang menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi. Algoritma Dijkstra dimulai dari sebuah simpul asal dan dalam setiap iterasinya menambahkan sebuah verteks lain ke lintasan terpendek pohon merentang. Verteks ini merupakan titik terdekat ke akar namun masih di luar bagian pohon.

Algoritma dikjstra akan mencari panjang lintasan terpendek antara dua simpul dalam graf yang terhubung, sederhana, tidak berarah, dengan pembobot.

II.10 Analisa Algoritma Dijkstra

Algoritma ini mirip dengan algoritma Prim untuk mencari MST, yaitu pada tiap iterasi memeriksa sisi-sisi yang menghubungkan subset verteks W dan subset verteks (V-W) dan memindahkan verteks w dari (VW) ke W yang memenuhi kriteria tertentu. Perbedaannya terletak pada kriteria itu sendiri. Dalam algoritma Dijkstra, yang dicari adalah sisi yang menghubungkan ke suatu verteks di (V-W) sehingga jarak dari verteks asal Vs ke verteks tersebut adalah minimal. Dalam implementasinya penghitungan jarak dari verteks asal vs disederhanakan dengan menambahkan field minpath pada setiap verteks. Field-field minpath ini diinisialisasi sesuai dengan adjacency-nya dengan v_s, kemudian dalam setiap iterasi di-update bersamaan masuknya w dalam W. Field minpath ini menunjukkan jarak dari v_s ke verteks yang bersangkutan terpendek yang diketahui hingga saat itu. Jadi pada verteks dalam W, minpath sudah menunjukkan jarak terpendek dari Vs untuk mencapai verteks yang bersangkutan, sementara pada (V-W) masih perlu di-update pada setiap iterasi dalam mendapatkan verteks w seperti diterangkan di atas. Yaitu, setiap mendapatkan w maka update minpath setiap adjacent verteks x dari w di (V-W) sisa dengan: minimum (minpath dari x , total minpath w + panjang sisi yang bersangkutan). Agar dapat berlaku umum maka di awal algoritma seluruh minpath diinisialisasi dengan $+\infty$. Demikian halnya pencarian w itu sendiri disederhanakan menjadi pencarian node di (V-W) dengan minpath terkecil.

Selengkapnya algoritma Dijkstra adalah sebagai berikut :

1. Inisialisasi

   W berisi mula-mula hanya vs, field minpath tiap verteks v dengan Weight[vs, v], jika ada sisi tersebut, atau, $+\infty$ jika tidak ada.

2. Lalu, dalam iterasi lakukan hingga (V-W) tak tersisa (atau dalam versi lain: jika ve ditemukan) dari field minpath tiap verteks cari verteks w dalam (V-W) yang memiliki minpath terkecil yang bukan tak hingga. Jika ada w maka masukkan w dalam W. Update minpath pada tiap verteks t adjacent dari w dan berada dalam (VW) dengan: minimum (minpath[t], minpath[w] + weight [w, t])

Verteks-verteks dalam W dapat dibedakan dari verteks dalam (V-W) dengan suatu field yang berfungsi sebagai flag atau dengan struktur liked-list. Informasi lintasan dapat diketahui dengan pencatatan predesesor dari setiap verteks yang dilakukan pada saat update harga minpath tersebut (fungsi minimum). Jika minpath[t] di-update dengan (minpath[w] + Weight [w, t]) maka predesesor dari t adalah w. Pada tahap inisialisasi predesesor setiap verteks diisi oleh v_s.

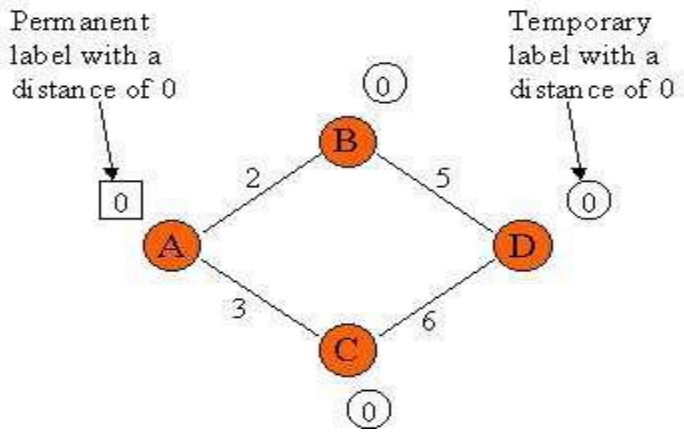
Algoritma Dijkstra membuat label yang menunjukkan simpul-simpul. Label-label ini melambangkan jarak dari simpul asal ke suatu simpul lain. Dalam graf, terdapat dua macam label, sementara dan permanen. Label sementara diberikan untuk simpul-simpul yang belum dicapai. Nilai yang diberikan untuk label sementara ini dapat beragam. Label permanen diberikan untuk simpul-simpul yang sudah dicapai dan jarak ke simpul asal diketahui. Nilai yang diberikan untuk label ini ialah jarak dari simpul ke simpul asal. Suatu simpul pasti memiliki label permanen atau label sementara. Tetapi tidak keduanya.

(a) (b)

Gambar II.13. (a). Simpul A berlabel sementara dengan jarak 0

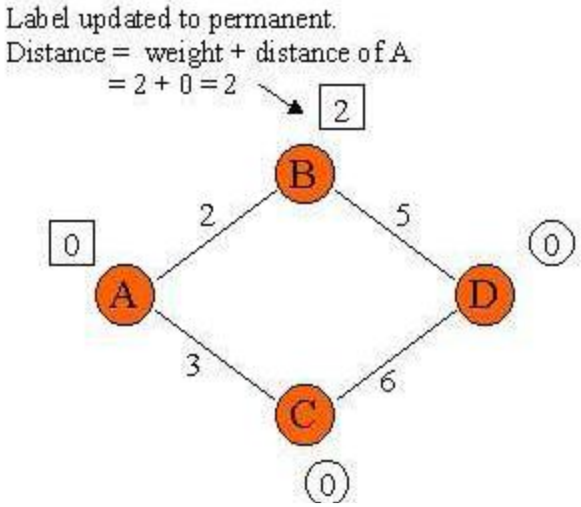
(b). Simpul B berlabel permanen dengan jarak 5

Algoritma dimulai dengan menginisialisasi simpul manapun di dalam graf (misalkan simpul A) dengan label permanen bernilai 0 dan simpul-simpul sisanya dengan label sementara bernilai 0.



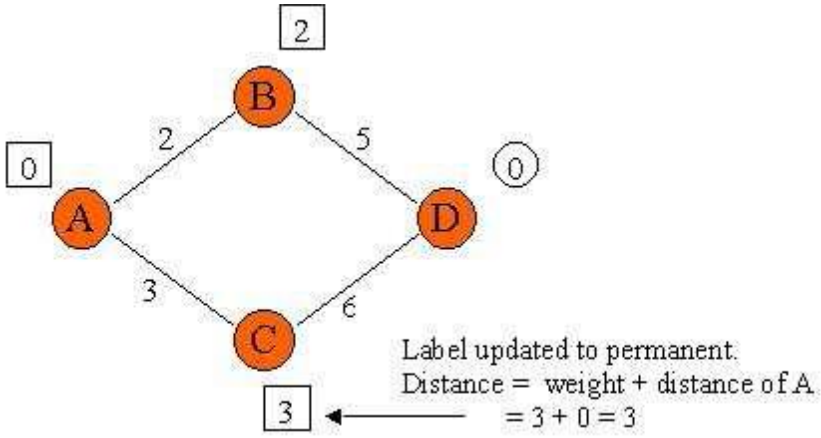
Gambar II.14. Inisialisasi awal

Algoritma ini kemudian memilih nilai sisi terendah yang menghubungkan simpul dengan label permanen (dalam hal ini simpul A) ke sebuah simpul lain yang berlabel sementara (misalkan simpul B). Kemudian label simpul B di-update dari label sementara menjadi label permanen. Nilai simpul B merupakan penjumlahan nilai sisi dan nilai simpul A.



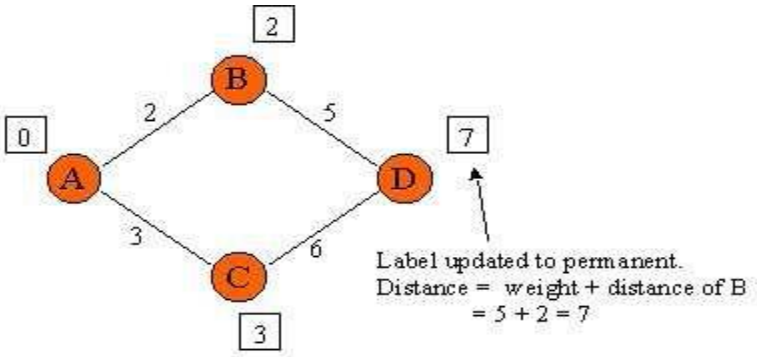
Gambar II.15. Nilai simpul B menjadi permanen

Langkah selanjutnya ialah menemukan nilai sisi terendah dari simpul dengan label sementara, baik simpul A maupun simpul B (misalkan simpul C). Ubah label simpul C menjadi permanen, dan ukur jarak ke simpul A.



Gambar II.16. Nilai simpul C berubah

Proses ini berulang hingga semua label simpul menjadi permanen.



Gambar II.17. Semua nilai simpul menjadi permanen

Lamanya waktu untuk menjalankan Algoritma Dijkstra's pada suatu graf dengan E (himpunan sisi) dan V (himpunan simpul) dapat dinyatakan sebagai fungsi E dan V menggunakan notasi Big-O. Waktu yang dibutuhkan algoritma Dijkstra untuk bekerja ialah sebesar O(V*log V + E). Implementasi paling

sederhana dari algoritma Dijkstra ialah penyimpanan simpul dari suatu himpunan ke dalam suatu array atau list berkait.

2. SPANNING CYCLE

II.1. Pengertian *Graph*

Graph adalah kumpulan dari simpul dan busur yang secara umum dinyatakan sebagai G = (V, E), dimana :

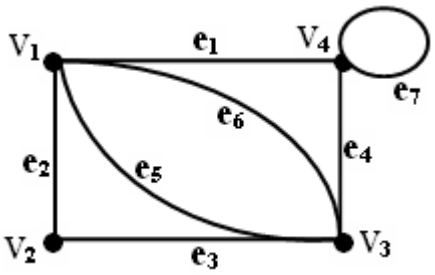
G = *Graph*

V = Simpul, atau Titik, atau *Vertex*, atau *Node*.

E = Busur, atau Sisi, atau *Edge*, atau *arc*

Atau *Graph* adalah pasangan V[G] dan E[G], dimana V[G] adalah himpunan tak kosong yang anggotanya berupa titik yang disebut simpul atau *vertex*, dan E[G] adalah himpunan yang anggotanya berupa garis yang disebut Sisi atau *Edge*. ***(****Jong Jek Siang, 2006****).***

Simpul atau *vertex* adalah suatu elemen dari *graph* G yang dapat disajikan berupa titik atau lingkaran kecil dan dinotasikan dengan V_i, dimana i = 1,2,3,…,n. Sisi adalah suatu elemen *graph* G yang dapat disajikan berupa garis lurus atau garis lengkung yang menghubungkan dua simpul (V_i, V_j) yang dinotasikan dengan E_k, dimana k = 1,2,3,…,m. V_i dan V_j disebut simpul-simpul ujung dari sisi E_k. Untuk memperjelas hal tersebut di atas berikut adalah gambar *graph* dengan empat *vertex* dengan tujuh sisi, atau secara matematis ditulis dengan G(4,7).



Gambar II.1. *Graph* 4 Simpul dan 7 Sisi.

II.2. Istilah-istilah dalam Teori *Graph*

Dalam teori *graph,* ditemukan beberapa istilah-istilah *graph,* antara lain sebagai berikut.

II.2.1. Gelang *(Loop)*

Edge yang menghubungkan pasangan *vertex* yang sama yaitu (V_i, V_j) disebut gelang atau kalang. Pada Gambar II.1 dapat dilihat bahwa e_7 merupakan gelang. *(Rinaldi Munir, 2007).*

II.2.2. Pararel *(Multiple)*

Dua atau lebih sisi yang mempunyai simpul-simpul ujung yang sama disebut sisi-sisi pararel atau sisi *multiple edge*. Pada Gambar II.1 dapat dilihat bahwa e_5 dan e_6 merupakan dua buah sisi pararel. *(Rinaldi Munir, 2007).*

II.2.3. Bersisian *(Incident)*

Dalam suatu *graph* G simpul sisi e_k disebut *incident* terhadap V_i jika dan hanya jika sisi tersebut merupakan penghubung dari simpul V_i. Dalam Gambar II.1 masing-masing sisi e_2 *incident* dengan simpul V_1 dan V_2, sisi e_1 *incident* dengan simpul V_1 dan V_4. *(Rinaldi Munir, 2007).*

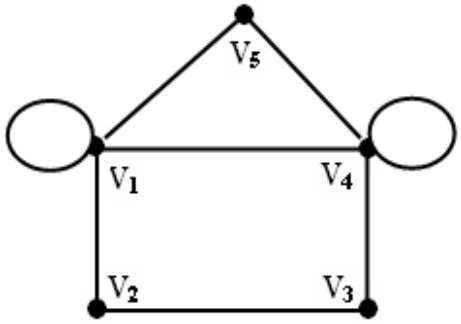
II.2.4. Bertetangga *(Adjacent)*

Dua buah simpul V_i dan V_j dikatakan saling bertetangga *(adjacent)* jika kedua *vertex* tersebut dihubungkan oleh satu *edge.* Dalam Gambar II.1 *vertex* V_1 dan V_4 adalah dua simpul yang *adjacent.* Sedangkan e_1 dan e_4 merupakan dua buah *edge* yang saling berdekatan disebut *edge* bertetangga. *(Rinaldi Munir, 2007).*

II.2.5. Derajat Suatu *Graph (Degree)*

Jumlah *edge* yang bersamaan pada sebuah *vertex* V_i di dalam suatu *graph* G dengan ketentuan bahwa suatu gelang *(loop)* dihitung dua kali, disebut dengan derajat *(valensi)*, *vertex* V_i biasanya dinotasikan dengan $d(V_i)$. ***(****Jong Jek Siang, 2006****).***

Contoh, tentukanlah banyaknya derajat *(valensi)* dari Gambar II.2 berikut.



Gambar II.2. Derajat Suatu *Graph*

Maka banyaknya *valensi* pada Gambar II.2 adalah :

$d(V_1)$ = 5 karena garis yang berhubungan dengan V_1 ada sebanyak 5 dan karena gelang *(loop)* dihitung sebanyak dua.

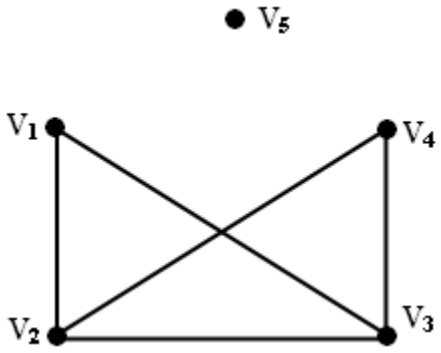
$d(V_2) = d(V_3) = d(V_5)$ = 2 karena garis yang berhubungan dengan masing-masing vertex V_2, V_3 dan V_5 adalah sebanyak dua.

$d(V_4)$ = 5 karena garis yang berhubungan dengan V_4 ada sebanyak 5 dan karena gelang *(loop)* dihitung sebanyak dua.

Derajat total = $\sum d(V_i)$ = 5 + 2 + 2 + 5 + 2 = 16

II.2.6. Simpul Terpencil *(Isolated Vertex)*

Simpul terpencil *(isolated vertex)* adalah simpul yang tidak mempunyai sisi yang bersisian dengannya. Atau dapat juga dinyatakan bahwa simpul terpencil adalah simpul yang tidak satupun bertetangga dengan simpul-simpul lainnya. Dari Gambar II.3 menunjukkan bahwa V_5 adalah simpul terpencil *(isolated vertex)*. *(Rinaldi Munir, 2007).*



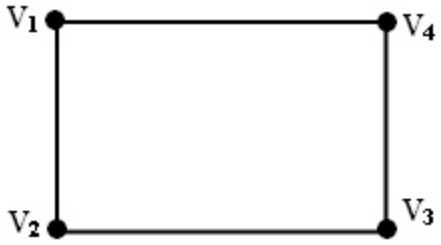
Gambar II.3. Simpul Terpencil *(Isolated Vertex)*

II.3. Jenis-jenis *Graph*

Jenis-jenis *graph* dapat dibagi dalam beberapa bagian, yang bisa diterapkan dalam bagian sub-sub yang dituliskan dalam pembentukan *graph.*

II.3.1. *Graph* Sederhana *(Simple Graph)*

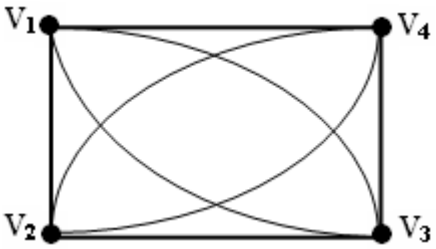
Suatu *graph* yang tidak mempunyai gelang *(loop)* dan *edge* pararel disebut *graph* sederhana. Salah satu contoh *graph* sederhana dapat dilihat pada Gambar II.4. *(Rinaldi Munir, 2007).*



Gambar II.4. *Graph* Sederhana *(Simple Graph)*

II.3.2. *Graph* Ganda *(Multigraph)*

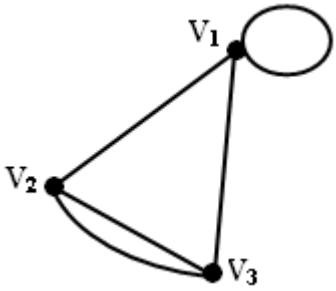
Suatu *graph* yang mengandung sisi ganda (*multipleedge)* tanpa gelang *(loop)* disebut *multigraph*. Contoh *multigraph* dapat dilihat pada Gambar II.5. *(Rinaldi Munir, 2007).*



Gambar II.5. Multi *Graph*

II.3.3. *Graph* Semu *(Pseudo Graph)*

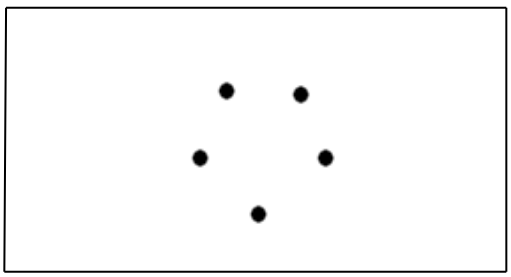
Suatu *graph* yang mengandung gelang *(loop)* dan *edge* pararel disebut *graph* semu *(pseudo graph)*, salah satu contoh *graph* semu dapat dilihat pada Gambar II.6. *(Rinaldi Munir, 2007).*



Gambar II.6. *Graph* Semu

II.3.4. *Graph* Kosong *(Null Graph* atau *Empty Graph)*

Dari pendefinisian sebuah *graph* G(V, E) memungkinkan adanya suatu *graph* dengan E suatu himpunan kosong, sehingga *graph* tersebut tidak memiliki *edge*. *Graph* yang demikian disebut dengan *null graph* yang berarti bahwa simpul atau *vertex* di dalam adalah simpul terasing. Salah satu contoh *null graph* dapat dilihat pada Gambar II.7. *(Rinaldi Munir, 2007).*

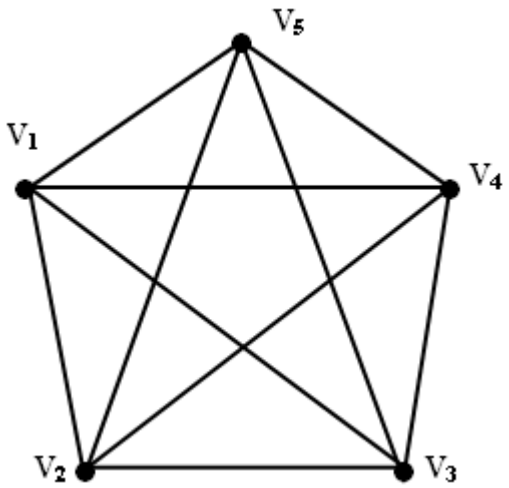


Gambar II.7. *Graph* Kosong *(Null Graph)*

II.3.5*. Graph* Lengkap *(Complete Graph)*

Suatu *graph* sederhana yang mana setiap dua simpul berbeda dihubungkan oleh satu garis (sisi) dinamakan *graph* lengkap. *Graph* lengkap pada n simpul biasanya dinotasikan dengan K_n.

Salah satu contoh *complete graph* pada gambar II.8. jika G adalah *graph* lengkap dengan n *vertex*, maka setiap *vertex* memiliki derajat n-1. *(Rinaldi Munir, 2007).*



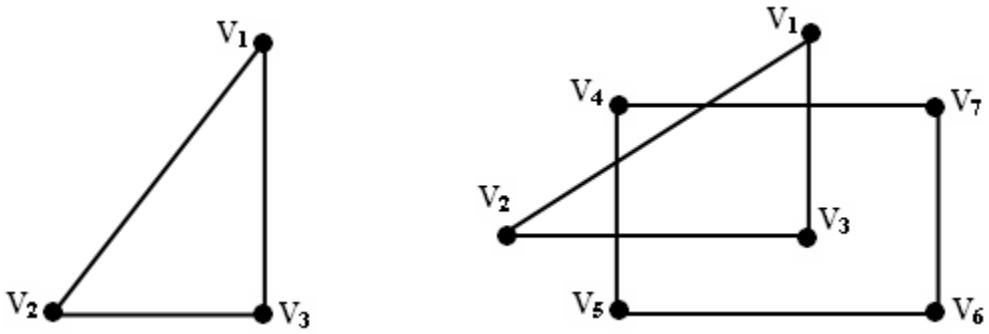
Gambar II.8. *Graph* Lengkap

II.3.6. *Graph* Terhubung *(Connected Graph)*

Semua *graph* disebut *graph* terhubung jika untuk setiap pasangan simpul di dalam G terdapat paling sedikit satu lintasan. Dengan kata lain penelusuran semua simpul dapat dilakukan dari sembarang melalui sisi yang ada. Jadi jika dalam suatu *graph* G terdapat pasangan simpul yang tidak mempunyai lintasan penghubung maka *graph* tersebut dinamakan *graph* tidak terhubung *(disconnected graph)*. Suatu *graph* tak terhubung *(disconnected graph)* memuat dua atau lebih *graph* terhubung dimana setiap

graph terhubung ini disebut dengan komponen. Atau dengan kata lain, sebuah *graph* disebut *graph* tak terhubung bila antara dua simpul tidak terhubung dan tidak terdapat satu lintasan yang melaluinya.

Gambar II.9 adalah contoh *graph* terhubung *(Connected Graph)* dan *graph* tak terhubung *(Disconnected Graph).* ***(*** *Jong Jek Siang, 2006****).***



(a) *Graph* Terhubung *(Connected Graph);* (b) *Graph* Tak Terhubung *(Disconnected Graph)*

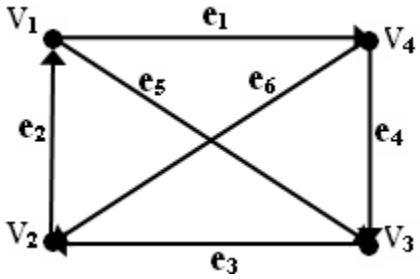
Gambar II.9. *Graph* Terhubung dan *Graph* Tak Terhubung

II.3.7. *Graph* Berarah *(Directed Graph)*

Graph berarah *(Digraph = Directed Graph)* G terdiri dari himpunan titik-titik V[G] : {v_1, v_2,… }, himpunan garis-garis E[G] : {e_1, e_2,… }, dan suatu fungsi yang mengawankan setiap garis dalam E[G] kesuatu pasangan berurut titik (v_i, v_j). Jika e_k = (v_i, v_j) adalah suatu garis dalam G, maka v_i disebut titik awal e_k dan v_j disebut titik akhir e_k. Arah garis adalah dari v_i ke v_j.

Graph tak berarah *(Undigraph = Undirected Graph)* adalah *graph* G terdiri dari pasangan himpunan (V, E) dimana V merupakan berhingga tak kosong yang anggota-anggotanya disebut simpul dan E merupakan himpunan berhingga yang anggota-anggotanya disebut *edge*

atau busur, yakni merupakan potongan garis berarah yang menghubungkan pasangan simpul atau *vertex* V. Untuk lebih jelasnya dapat dilihat pada Gambar II.10. ***(Jong Jek Siang, 2006*)*.**



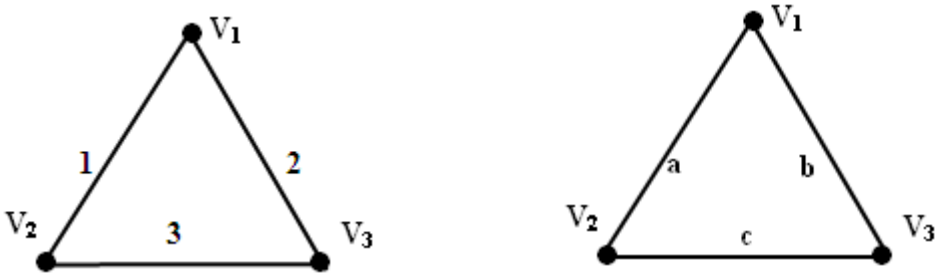
Gambar II.10. *Graph* Berarah

II.3.8. *Graph* Berbobot *(Weighted Graph)* Atau *Graph* Berlabel

Bobot pada setiap sisi dapat berbeda-beda bergantung pada masalah yang dimodelkan dengan *graph.* Bobot dapat menyatakan jarak antara dua buah kota, biaya perjalanan antara dua buah kota, waktu tempuh pesan *(message)* dari sebuah simpul komunikasi ke simpul komunikasi yang lain (hal ini bisa ditemukan dalam jaringan komputer maupun telepon seluler), ongkos produksi dan sebagainya. Gambar II.11 (a) menunjukkan *graph* berbobot, dengan bobot bilangan bulat positif.

Jika setiap *edge*nya dikaitkan dengan suatu simbol bukan bilangan *real* disebut dengan *graph* berlabel *(labeled graph).* Namun *graph* berlabel sesungguhnya lebih luas lagi definisinya. Label tidak hanya diberikan pada sisi, tetapi juga pada simpul. Sisi diberi label berupa bilangan tak-negatif, sedangkan simpul diberi label berupa data lain. Misalnya pada *graph* yang memodelkan kota. Simpul diberi dengan nama-nama kota sedangkan label pada sisi menyatakan jarak antara kota.

Suatu *graph* G disebut berlabel dimana setiap garisnya berhubungan dengan suatu bilangan *real* tak-negatif yang menyatakan bobot garis tersebut. Bobot garis e biasanya diberi simbol w(e). Untuk lebih jelasnya dapat dilihat pada Gambar II.11. *(Rinaldi Munir, 2007).*



(a) *Graph* Berbobot (b) *Graph* Berlabel

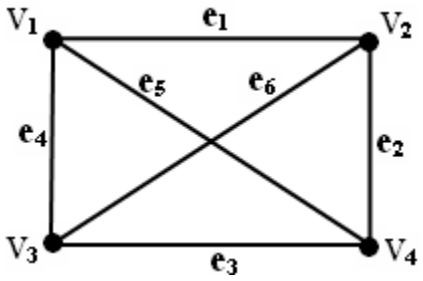
Gambar II.11. *Graph* Berbobot dan *Graph* Berlabel

I.4. Elemen-elemen *Graph*

II.4.1. Lintasan *(Path)*

Suatu lintasan dalam suatu *graph* G adalah suatu deretan dari suatu simpul dan sisi secara bergantian yang diawali dengan simpul dan diakhiri dengan simpul pula. Dalam hal ini sebuah simpul dan sebuah sisi boleh muncul dua kali.

Apabila lintasan dalam sebuah *graph* G, dimana simpul awal sama dengan simpul akhir, maka lintasan tersebut dinamakan dengan lintasan tertutup *(Close Walk*) dan sebaliknya, jika simpul awal dan simpul akhir berbeda dinamakan lintasan terbuka *(Open Walk). (Rinaldi Munir, 2007).*



Gambar II.12. *Graph* Lintasan *(walk)*

Dari Gambar II.12 didapati beberapa lintasan dalam *graph* yakni :

1) $V_1 V_2 V_4 V_3 V_1$ (Lintasan tertutup)

2) $V_2 V_3 V_4 V_2 V_1$ (Lintasan terbuka)

II.4.2. Jalan *(Trail)*

Jalan dari suatu *graph* G adalah suatu lintasan yang mana semua sisinya berlainan atau setiap sisi hanya boleh dilalui satu kali.

Dari Gambar II.12 didapati beberapa jalan *(trail)* dalam *graph* yakni :

1). $V_1 V_2 V_3 V_4 V_1$

2). $V_2 V_3 V_4 V_1$

II.4.3. Siklus *(Cycle)*

Siklus (*Cycle)* dari suatu *graph* G adalah alur dimana simpul awal sama dengan simpul akhir. Sebagai akibat dari definisi ini, susunan kalimat lain mengenai siklus didefinisikan sebagai alur tertutup *(Close path). (Rinaldi Munir, 2007).*

Contoh siklus dari Gambar II.12 adalah :

1). $V_1 V_2 V_4 V_1$ (Lintasan dengan panjang 3)

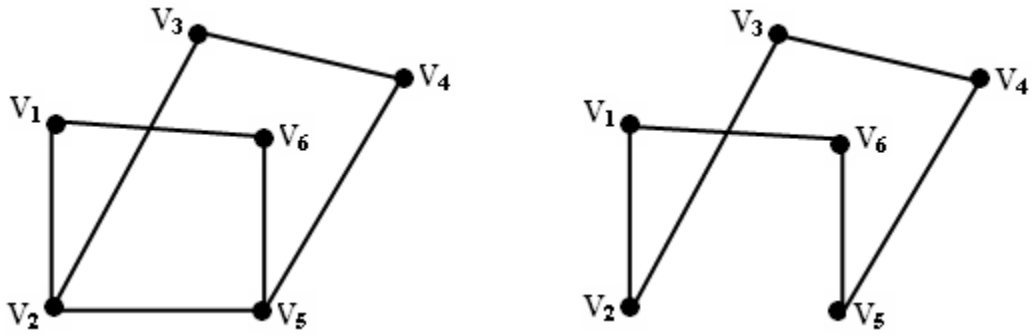
2). $V_3 V_1 V_4 V_2 V_3$ (Lintasan dengan panjang 4)

II.4.4. Sub *Graph*

Suatu *graph* dikatakan sub *graph* dari *graph* G jika semua simpul dan semua sisi dari *graph* g ada pada *graph* G, dan tiap sisi sub *graph* g mempunyai simpul sama dengan *graph* G.

Misalkan *G = (V, E)* adalah sebuah *graph*. Kita sebut *(V', E')* sub *graph* dari G atau secara matematis dituliskan *G' = (V', E'). (Didiek Djunaedi, 2002).*

Untuk lebih jelasnya dapat dilihat pada Gambar II.13 dibawah ini :



(a) *Graph* G (6, 7) (b) Sub *Graph* G (6, 6)

Gambar II.13. Sub *Graph* G

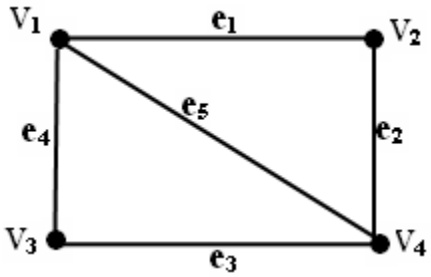
Dalam suatu sub *graph* terdapat beberapa konsekuensi yang disebut dengan konsekuensi *graph* antara lain sebagai berikut.

II.4.4.1. Penghapusan *(Deletetion)*

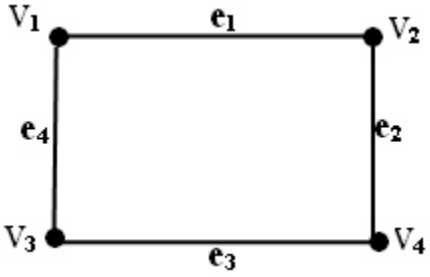
Jika V_1 sebuah simpul dari sebuah *graph* G, maka G-V_1 merupakan suatu sub *graph* dari G yang diperoleh dengan menghapus simpul V_1 beserta sisi-sisi yang bersamaan dengan simpul tersebut.

Jika e_j suatu sisi dalam suatu *graph* G, maka G-e_j merupakan suatu sub *graph* yang diperoleh dengan menghapus sisi e_j dari *graph* G.

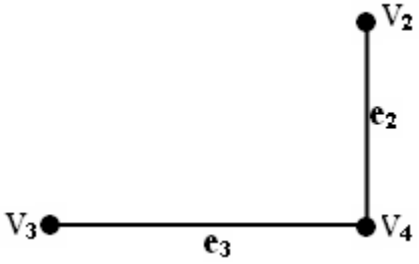
Pada Gambar II.14 di bawah ini akan diperlihatkan *Deletion* V_1 dan *Deletion* e_5 dari suatu *graph* G. *(Rinaldi Munir, 2007).*



(a) Graph G



(b) Deletion e_5



(c) Deletion V_1

Gambar II.14. Penghapusan *(Deletetion)*

II.4.4.2. Sirkuit dan Lintasan Euler

Lintasan Euler adalah lintasan yang melalui masing-masing *edge* di dalam *graph* tepat satu kali. Bila lintasan tersebut kembali ke *vertex* awal membentuk lintasan tertutup atau sirkuit, maka sirkuit itu disebut dengan sirkuit Euler. Jadi sirkuit Euler adalah sirkuit yang melewati masing-masing *edge* tepat satu kali dan kembali ke tempat asal.

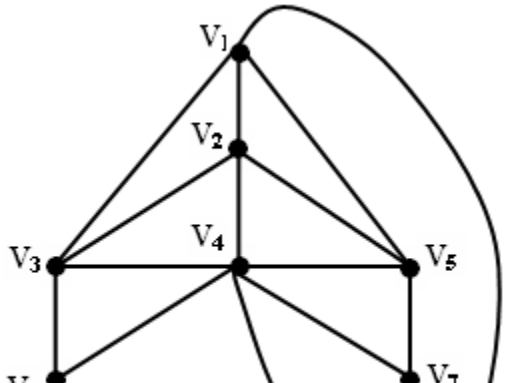
Graph yang mempunyai sirkuit Euler disebut *graph* Euler *(Eulerian Graph)*. *Graph* yang mempunyai lintasan Euler disebut dengan semi Euler *(Semi Eulerian Graph). (Rinaldi Munir, 2007).*

Jika sebuah *graph* G mempunyai siklus Euler, maka G tersambung dan setiap *vertex* mempunyai derajat genap, atau sebaliknya jika G adalah sebuah *graph* tersambung dan setiap *vertex* mempunyai derajat genap, maka G mempunyai sebuah siklus Euler. Gambar II.15 memperlihatkan contoh *graph* yang mengandung lintasan dan juga sirkuit Euler. *(Didiek Djunaedi, 2002).*

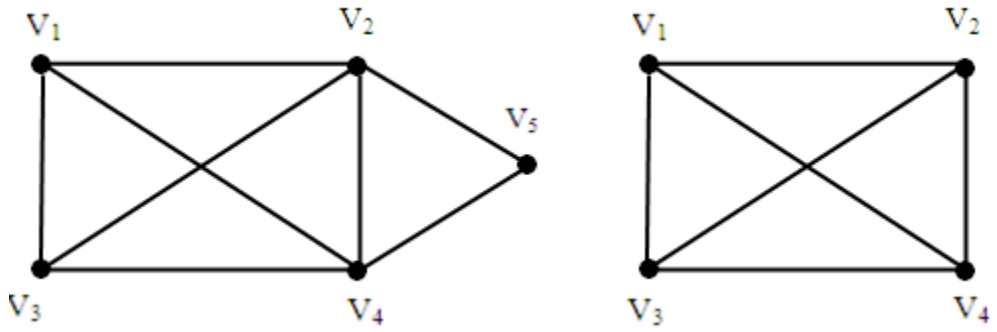
Misalkan G adalah *graph* dari gambar II.15 (a), kita menggunakan teorema diatas untuk membuktikan bahwa G mempunyai sebuah siklus atau sirkuit Euler. Setelah kita periksa bahwa *graph* G tersambung, dan derajat setiap *vertex* adalah, $d(V_1) = d(V_2) = d(V_3) = d(V_5) = 4$; $d(V_4) = 6$; $d(V_6) = d(V_7) = 2$.

Karena derajat setiap *vertex* adalah genap, maka *graph* G tersebut mempunyai sirkuit Euler. Dengan pemeriksaan atau menggunakan metode coba-coba kita mendapat sirkuit Euler sebagai berikut : ($V_6, V_4, V_7, V_5, V_1, V_3, V_4, V_1, V_2, V_5, V_4, V_2, V_3, V_6$).

Salah satu contoh lintasan Euler yang terbentuk dari Gambar II.15 (b) adalah : ($V_1, V_2, V_5, V_4, V_2, V_3, V_4, V_1, V_3$).



(a) *Graph* yang mempunyai *Sirkuit Euler*



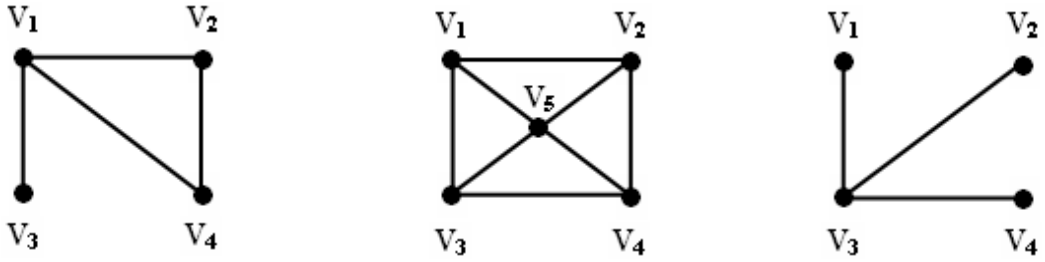
(b) Graph yang mempunyai *Lintasan Euler;* (c) *Graph* yang tidak mempunyai *Lintasan Euler* dan *Sirkuit Euler*

Gambar II.15. Sirkuit dan Lintasan Euler

II.4.4.3. Sirkuit dan Lintasan Hamilton

Lintasan Hamilton adalah lintasan yang melalui tiap *vertex* di dalam *graph* tepat satu kali. Bila lintasan itu kembali ke *vertex* asal membentuk lintasan tertutup, maka lintasan tertutup itu dinamakan

sirkuit Hamilton *(Hamilton Cycle).* Jadi sirkuit Hamilton adalah sirkuit yang melalui tiap *vertex* tepat satu kali kecuali *vertex* asal atau awal yang sama dengan *vertex* akhir.



Keterangan: (a). *Graph* yang mempunyai *Lintasan Hamilton*

(b). *Graph* yang mempunyai *Sirkuit Hamilton*

(c). *Graph* yang tidak mempunyai *Lintasan* dan *Sirkuit Hamilton*

Gambar II.16. Sirkuit dan Lintasan Hamilton

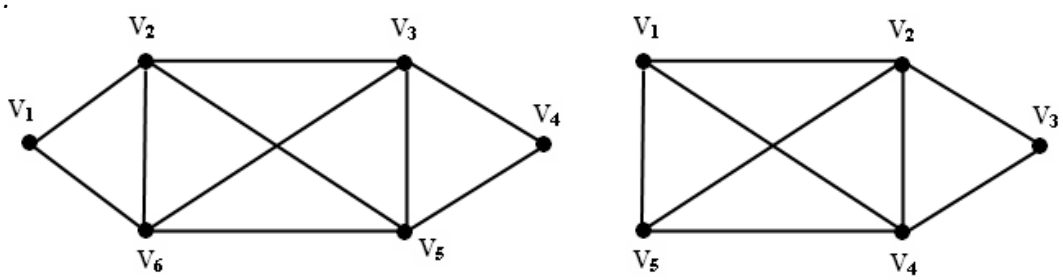
Perhatikan perbedaan sirkuit Euler dan sirkuit Hamilton. Dalam sirkuit Euler, semua garis *(edge)* harus dilalui tepat satu kali, sedangkan semua titiknya boleh dilalui lebih dari satu kali. Sebaliknya, dalam sirkuit Hamilton semua titik *(vertex)* harus dikunjungi tepat satu kali dan tidak harus melalui semua garisnya. Dalam sirkuit Euler, yang dipertimbangkan adalah garisnya. Sedangkan pada sirkuit hamilton yang dipertimbangkan adalah kunjungan pada titik atau *vertex*nya. ***(****Jong Jek Siang, 2006****).*** Gambar II.16 memperlihatkan contoh *graph* yang mengandung lintasan atau sirkuit hamilton.

Lintasan Hamilton pada gambar *graph* II.16 (a) adalah V_3–V_1–V_2–V_4.

Sirkuit Hamilton pada gambar *graph* II.16 (b) adalah V_1–V_2–V_4–V_3–V_5–V_1.

Beberapa *graph* dapat mengandung sirkuit Euler dan sirkuit Hamilton sekaligus, atau mengandung sirkuit Euler tetapi tidak memiliki sirkuit Hamilton, mengandung sirkuit Euler dan lintasan

Hamilton, mengandung lintasan Euler maupun lintasan Hamilton dan lain sebagainya. Untuk lebih jelasnya dapat dilihat pada Gambar II.17 (a) yang mengandung sirkuit Hamilton dan lintasan Euler. *(Rinaldi Munir, 2007)*



(a) *Graph Hamilton* sekaligus *Graph Euler* (b) *Graph Hamilton* sekaligus *Graph Semi Euler*

Gambar II.17. Persamaan *Graph Hamilton* dan *Graph Euler*

II.4.4.4. *Spanning Cycle*

Spanning cycle adalah merupakan alur yang melalui semua simpul hanya satu kali, dan simpul awal merupakan simpul akhir dan merupakan *graph* terhubung. *Spanning cycle* dari suatu *graph* adalah merupakan sub*graph*-sub*graph* dari *graph* Hamilton

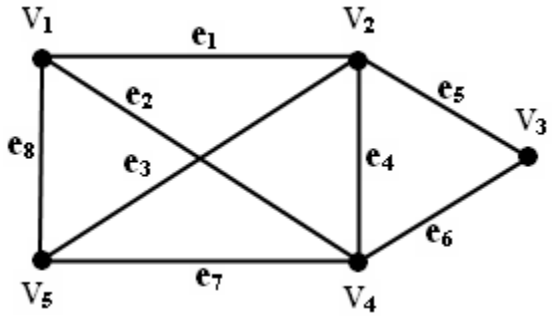
Jika suatu *graph* G mengandung atau memiliki *spanning cycle*, maka *graph* G mempunyai sub *graph* dengan sifat-sifat sebagai berikut :

1. H terhubung
2. H memuat semua titik G

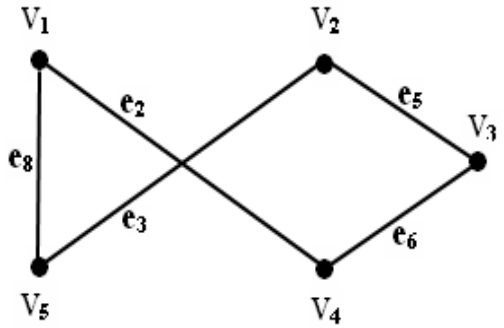
3. H mempunyai jumlah garis yang sama dengan jumlah titiknya.
4. Setiap titik dalam H mempunyai derajat 2.

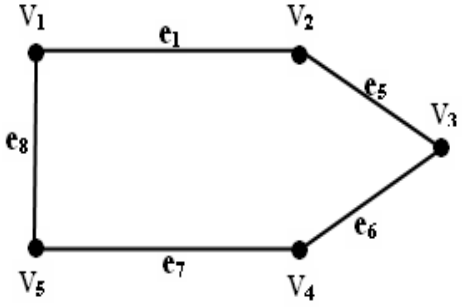
Syarat (1) dan (2) jelas menurut definisi *spanning cycle* mengharuskan mengunjungi semua titik dalam G. Syarat (4) ada sebagai akibat dari kunjungan semua titik yang hanya boleh dilakukan satu kali saja. Selama kunjungan, pasti ada satu garis masuk dan satu garis keluar sehingga derajat setiap titik (= 2). Karena dalam *spanning cycle*, setiap dua titik dihubungkan dengan satu garis, maka jumlah garis sama dengan jumlah titiknya. Hal ini dinyatakan dalam syarat (3).

Jika salah satu dari ke-4 syarat di atas tidak dipenuhi maka *graph*-nya bukanlah *spanning cycle*. ***(*** *Jong Jek Siang, 2006****).*** Contoh *spanning cycle* dapat kita lihat pada Gambar II.18.



(a) *Graph* G





(b) *Graph* H *(Spanning Cycle* dari *Graph* G); (c) *Graph* H (*Spanning Cycle* dari *Graph* G)

Gambar II.18. *Spanning Cycle*

Pada *graph* lengkap dengan n buah simpul (dengan n > 2), maka jumlah *spanning cycle* yang dapat terbentuk adalah (n-1)!. Rumus ini dihasilkan dari kenyataan bahwa dimulai dari sembarang simpul, maka suatu *graph* memiliki n-1 buah sisi untuk simpul pertama, n-2 buah sisi dari simpul kedua, n-3 buah sisi untuk simpul ketiga dan demikian seterusnya sampai pada simpul ke n, ini adalah pilihan yang bebas sehingga diperoleh (n-1)! pilihan. *(Rinaldi Munir, 2007).*

Pencarian *spanning cycle* adalah persoalan yang sulit *(hard problem)* dipandang dari sudut komputasinya. Artinya, secara teoritis, *spanning cycle* dapat dipecahkan dengan mengenumerasi (n-1)! buah sirkuit Hamilton, menghitung rute masing-masing sirkuit, dan kemudian memilih sirkuit yang memiliki panjang rute terpendek untuk mendapatkan *spanning cycle* minimum. Untuk jumlah n yang besar, jumlah *spanning cycle* yang akan dicari tentu saja sangat banyak.

Tabel II.1 menunjukkan jumlah *spanning cycle* yang dapat terbentuk pada *graph* lengkap *(complite graph)* sampai dengan 26 *vertex.*

Tabel II.1. Jumlah *Spanning Cycle* pada *graph* lengkap *(complite graph)*

Jumlah *Vertex*	Jumlah *Spanning Cycle*
3	2
4	6
5	24
6	120
7	720

8	**5,040**
9	**40,320**
10	**362,880**
11	**3,628,800**
12	**39,916,800**
13	**479,001,600**
14	**6,227,020,800**
15	**87,178,291,200**
16	**1,307,674,368,000**
17	**20,922,789,888,000**
18	**355,687,428,096,000**
19	**6,402,373,705,728,000**
20	**121,645,100,408,832,000**
21	**2,432,902,008,176,640,000**
22	**51,090,942,171,709,400,000**
23	**1,124,000,727,777,600,000,000**
24	**25,852,016,738,884,900,000,000**
25	**620,448,401,733,239,000,000,000**
26	**15,511,210,043,330,900,000,000,000**

II.5. Algoritma Pencarian Jalur

Pencarian adalah suatu proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (*state space*). Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin. Dalam ilmu komputer, sebuah algoritma pencarian dijelaskan secara luas adalah merupakan algoritma yang menerima masukan berupa sebuah masalah dan menghasilkan sebuah solusi untuk masalah tersebut, yang biasanya didapat dari evaluasi beberapa kemungkinan solusi.

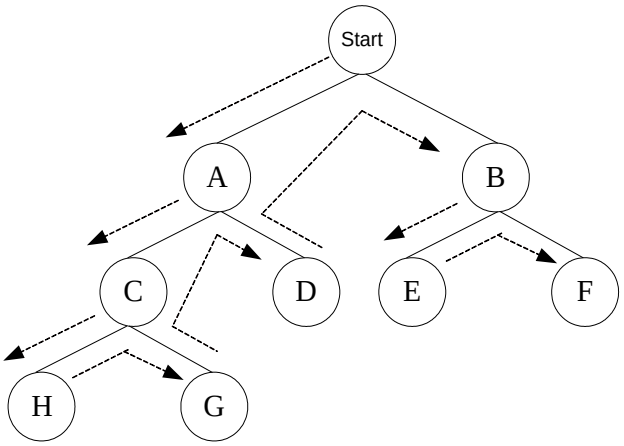
Himpunan semua kemungkinan solusi dari sebuah masalah disebut ruang pencarian. Algortima pencarian *brute-force* atau pencarian naif *(uninformed)* menggunakan metode yang sederhana dan sangat intuitif pada ruang pencarian, sedangkan algoritma pencarian *informed* menggunakan heuristik untuk menerapkan pengetahuan tentang struktur dari ruang pencarian untuk berusaha mengurangi banyaknya waktu yang dipakai dalam pencarian. *(Wikipedia, 2010)*

Untuk mengukur performansi metode pencarian, terdapat empat kriteria yang dapat digunakan :

1. *Completeness* : apakah metode tersebut menjamin penemuan solusi jika solusinya memang ada,
2. *Time complexity* : berapa lama waktu yang diperlukan,
3. *Space complexity* : berapa banyak memori yang diperlukan,
4. *Optimality* : apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi berbeda.

II.5.1. Algoritma Pencarian Mendalam Pertama (*Depth First Search*)

Salah satu algoritma yang dapat dipakai untuk mempermudah pencarian lintasan-lintasan pada suatu *graph* adalah dengan menggunakan algoritma *depth first search*. Pencarian dengan algoritma *depth first search* (DFS) dilakukan dari *node* awal secara mendalam hingga yang paling akhir (*dead-end*) atau sampai *goal* ditemukan. Dengan kata lain, simpul cabang atau anak yang terlebih dahulu dikunjungi. Sebagai ilustrasinya dapat dilihat pada Gambar II.19.



Gambar II.19. Pohon Pencarian Mendalam Pertama (*Depth First Search*)

Berdasarkan Gambar II.19, proses pencarian dilakukan dengan mengunjungi cabang terlebih dahulu hingga tiba di simpul terakhir. Jika tujuan yang diinginkan belum tercapai maka pencarian dilanjutkan ke cabang sebelumnya, turun ke bawah jika memang masih ada cabangnya. Begitu seterusnya hingga diperoleh tujuan (*goal*). Operasi semacam ini dikenal dengan sebutan *backtracking*. *(Kusumadewi, 2003)*

Adapun algoritma *depth first search* adalah sebagai berikut :

1. Jika keadaan awal merupakan tujuan, keluar (sukses).
2. Jika tidak demikian, kerjakan langkah-langkah berikut ini sampai tercapai keadaan sukses atau gagal :
    a. Bangkitkan *successor* dari keadaan awal. Jika tidak ada *successor*, maka akan terjadi kegagalan.
    b. Panggil *depth first search* dengan E sebagai keadaan awal
    c. Jika sukses berikan tanda sukses. Namun jika tidak, ulangi langkah 2.

Keuntungan penggunaan algoritma DFS adalah:

1. Membutuhkan memori yang relatif kecil, karena hanya *node-node* pada lintasan yang aktif saja yang disimpan.
2. DFS mungkin menemukan solusi tanpa menguji banyak jangkauan pencarian. DFS berhenti ketika salah satu solusi ditemukan.
3. Jika solusi yang dicari berada pada level yang dalam dan paling kiri, maka DFS akan menemukannya dengan cepat.

Kelemahan dari algoritma DFS adalah:

1. Hanya akan mendapatkan satu solusi pada setiap pencarian.
2. Jika pohon yang dibangkitkan mempunyai level yang sangat dalam (tak terhingga), maka tidak ada jaminan menemukan solusi. Artinya, DFS tidak complite.
3. Jika terdapat lebih dari satu solusi yang sama tetapi berada pada level yang berbeda, maka DFS tidak menjamin untuk menemukan solusi yang paling baik. Artinya tidak optimal**.** *(Suyanto, 2007).*