BAB II

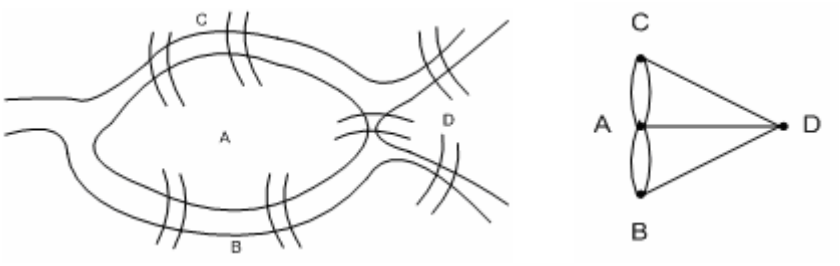
LANDASAN TEORI

2.1. Pendahuluan

Teori graf dikenal sejak abad ke-18 Masehi. Saat ini teori graf telah berkembang sangat pesat dan digunakan untuk menyelesaikan persoalan-persoalan pada berbagai bidang ilmu pengetahuan seperti ekonomi, sosial, kimia, biologi, fisika, geografi, dan lain-lain.

2.2. Sejarah Graf

Pada mulanya penggunaan jaringan yang memuat titik dan sisi digunakan oleh matematikawan Swiss, Leonhard Euler (1707-1783), untuk memecahkan masalah tujuh jembatan *Konigsberg*. Di kota Prussia, Jerman, sungai Pregel mengalir melewati kota, dan menutupi Pulau Kneiphof. Pulau tersebut dihubungkan oleh dua jembatan ke masing-masing tepi daratan C dan B, dan tambahan tiga jembatan yang menghubungkan mereka ke sebuah wilayah. Masalah yang ingin diselesaikan adalah "Dapatkah seseorang melewati semua jembatan dengan masing-masing jembatan terlewati tepat satu kali, dan kembali ke tempat semula?"



(a) (b)

Gambar 2.1. (a) Jembatan Konigsberg,
(b) Graf yang merepresentasikan Jembatan Konigsberg.

2.2.1. Pengertian Graf

Graf *G* didefinisikan sebagai pasangan himpunan (*V*,*E*), yang dalam hal ini:

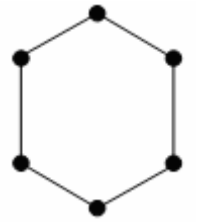
V = Himpunan tak kosong dari titik-titik (*vertices*)

= $\{v_1,v_2,v_3,\ldots,v_n\}$

E = Himpunan sisi (*edges*) yang menghubungkan sepanjang titik-titik

= $\{e_1,e_2,e_3,\ldots,e_n\}$

Atau dapat di tulis dengan notasi $G=(V,E)$.

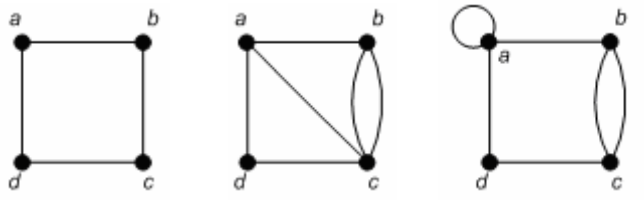


Gambar 2.2. Graf dengan $|V(G)| = 6$ dan $|E(G)| = 6$.

2.2.2. Terminologi Dasar Graf

Jika terdapat sisi *e* yang menghubungkan titik *u* dan *v*, maka *u* dan *v* disebut bertetangga. Titik-titik *u* dan *v* dari sisi *e* disebut sebagai titik-titik ujung (*end vertices*) dari sisi *e*. Sisi yang kedua titik ujungnya sama disebut *loop*. Bila ada

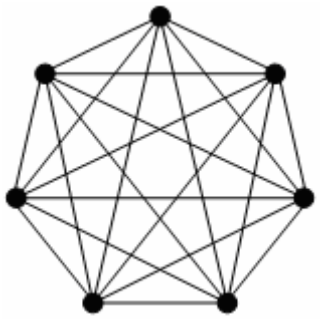
lebih dari satu sisi yang menghubungkan dua buah titik maka sisi-sisi itu disebut sisi ganda (*multiple edges)*. Graf yang tidak memiliki *loop* dan *sisi ganda* disebut graf sederhana (*simple graph*).



(a) (b) (c)

Gambar 2.3. (a) Graf sederhana, (b) Graf *multiple edges*,
(c) Graf dengan *loop.*

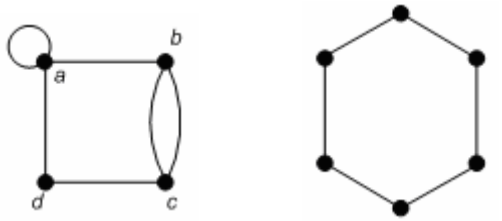
Banyaknya sisi yang terjadi pada suatu titik *v*, dimana *loop* dihitung sebagai dua buah sisi, dinyatakan sebagai derajat (*degree*) dari *v*, dan dinotasikan dengan simbol $d(v)$. Titik berderajat nol disebut titik terisolasi (*isolated vertices*). Graf dengan titik-titik berderajat sama disebut graf regular (*regular graph*). Graf *G* dengan *n* titik yang setiap dua titiknya bertetangga, disebut graf lengkap (*complete graph*), dan dinotasikan oleh K_n.



Gambar 2.4. Graf lengkap K_7.

Derajat suatu titik *v* di *G* ditulis $G(v)$ didefinisikan sebagai banyaknya sisi yang bersisian (*insident*) dengan *v*, dimana *loop* dihitung sebagai dua buah sisi

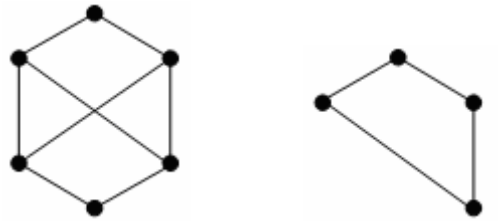
dan dinotasikan dengan simbol *d*(v). Graf teratur (*regular graph*) adalah *g*raf yang derajat setiap verteksnya sama. Graf *G* dengan *n* titik yang setiap titik terhubung ke setiap titik yang lainnya. Derajat setiap titik di K_n adalah (*n-1*).



(a) (b)

Gambar 2.5. (a) Derajat $d(a) = 4$, $d(b) = 3$, $d(c) = 3$, dan $d(d) = 2$,
(b) Graf teratur.

Suatu graf $G' = (V', E)$ adalah subgraf $G = (V, E)$ jika hanya jika $V' \leq V$ dan $E' = E$.



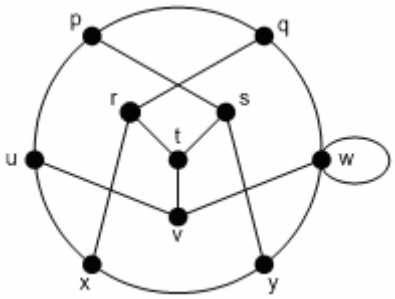
(a) (b)

Gambar 2.6. (a) Graf *G*, (b) Subgraf dari *G*.

Jalan (*walk*) adalah barisan dari titik dan sisi, $v_0e_1v_1e_2...e_kv_k$ dimana $e_i = (v_{i-1}, v_i)$ untuk $i = 1, 2, ..., k$. Titik-titik yang mengawali dan mengakhiri suatu jalan disebut titik terminal. Bila kedua titik terminalnya sama, jalan tersebut adalah jalan tertutup, kebalikannya disebut sebagai jalan terbuka. Jalan terbuka dimana setiap titik hanya muncul satu kali disebut lintasan (*path*). Dengan demikian, *loop* tidak mungkin muncul dalam lintasan. Pada lintasan, kedua titik terminal

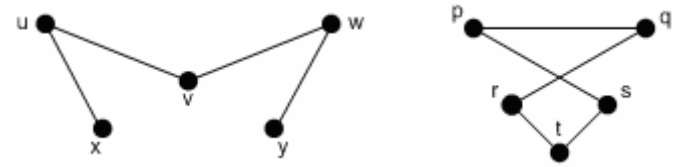
mempunyai derajat satu dan titik lainnya berderajat dua. Banyaknya sisi yang terdapat pada lintasan disebut panjang lintasan. Lintasan dari *u* ke *z* dinotasikan dengan lintasan-(*u*,*z*).

Sebuah jalan tertutup (*closed walk*) di *G* adalah rangkaian sisi-sisi dari *G* dalam bentuk *uv,vw,wx,...,yz,zu.* Jika titik-titiknya berbeda, disebut sebagai lingkaran (*cycle*). Misalkan sebuah graf *G* dengan 10 titik sebagai berikut:



Gambar 2.7. Graf *G* dengan 10 titik.

Sebuah contoh lintasan dari graf *G* di atas adalah lintasan-(*x*,*y*) = *xuvwy* dengan panjang 4. Sebuah jalan *pqrtsp* adalah contoh lingkaran dalam *G*.

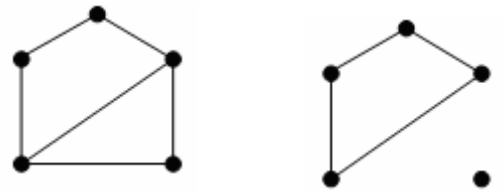


(a) (b)

Gambar 2.8. (a) Lintasan-*(x,y)=xuvwy,* (b) Lingkaran *pqrtsp.*

Dua titik *u* dan *v* di *G* dikatakan terhubung jika terdapat lintasan-(*u, v*) di *G*. Graf *G* adalah graf terhubung (*connected graph*) bila semua titik pada *G* dapat ditelusuri dari setiap titik lainnya. Dengan kata lain, suatu graf dikatakan terhubung bila untuk setiap pasangan titik *u* dan *v* pada *G* terdapat paling sedikit

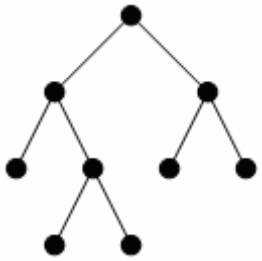
satu lintasan-(*u, v*) yang menghubungkan keduanya. Kebalikannya adalah graf tak terhubung (*disconnected graph*). Jarak antara dua titik *u* dan *v* menyatakan panjang lintasan terpendek antara *u* dan *v*, dan dinyatakan dengan *d*(*u, v*).



(a) (b)

Gambar 2.9. (a) Graf terhubung, (b) Graf tak terhubung.

Suatu graf tak terhubung terdiri dari dua atau lebih subgraf terhubung maksimal. Setiap subgraf terhubung maksimal itu disebut komponen. Pohon (*tree*) adalah graf terhubung minimal, dimana penghapusan sembarang sisi akan mengakibatkan graf yang tersisa tak terhubung. Pohon juga merupakan graf sederhana, tidak mempunyai lingkaran dan hanya ada satu lintasan diantara setiap pasangan titiknya.



Gambar 2.10. Sebuah pohon dengan 9 titik.

2.3. Konektivitas

Misalkan *G* adalah graf terhubung. *Cut-edge* adalah sebuah sisi yang bila dihapus akan membuat graf *G* tak terhubung. Keterhubungan sisi (*edge connectivity*) dari graf terhubung *G*, dinotasikan dengan $\lambda(\boldsymbol{G})$, adalah minimum banyaknya sisi yang jika dihilangkan akan membuat graf *G* tak terhubung. Ketika $\lambda(\boldsymbol{G}) \geq \boldsymbol{k}$, graf *G* dikatakan *k*-sisi terhubung (*k-edge connected*).

Sebuah *edge-cut* dari graf terhubung *G* adalah himpunan *S* dari sisi-sisi dengan sifat:

a. Menghapus semua sisi di *S* akan membuat *G* tak terhubung.
b. Menghapus beberapa (tidak semua) sisi di *S* tidak akan mengganggu keterhubungan *G*.

Secara serupa, untuk *G* graf terhubung, *cut-vertices* adalah sebuah titik yang bila dihapus akan membuat graf *G* tak terhubung. Keterhubungan titik dari graf terhubung *G* (bukan graf lengkap), dinotasikan dengan $\kappa(\boldsymbol{G})$, adalah minimum banyaknya titik yang jika dihapus akan membuat *G* tak terhubung. Ketika $\kappa(\boldsymbol{G}) \geq \boldsymbol{k}$, graf G disebut *k*-titik terhubung (*k-vertices connected*).

Sebuah *vertices-cut* dari graf terhubung *G* adalah himpunan *S* dari titik di *G* dengan sifat:

a. Menghapus semua titik di *S* akan membuat *G* tak terhubung.
b. Menghapus beberapa (tidak semua) titik di *S* tidak akan mengganggu keterhubungan *G*.

Misal *G* graf terhubung, dua atau lebih lintasan-(*s, t*) disebut saling lepas sisi (*edges disjoint*) jika mereka tidak memiliki sisi yang sama, dan disebut saling lepas titik (*vertices disjoint*) jika mereka tidak memiliki titik yang sama.

Misal *G* graf terhubung, dan misal *s* dan *t* titik di *G*. Kita katakan beberapa sisi memisahkan (*separate*) *s* dari *t* jika penghapusan sisi-sisi ini menghilangkan semua lintasan antara *s* dan *t*. Secara serupa kita katakan beberapa titik memisahkan *s* dari *t* jika penghapusan titik-titik ini menghilangkan semua lintasan antara *s* dan *t*.

Misal *G* graf terhubung, dan misal *s* dan *t* titik di *G*. Maka maksimum banyaknya lintasan-(*s, t*) yang saling lepas sisi adalah sama dengan minimum banyaknya sisi-sisi minimum yang memisahkan *s* dari *t*.

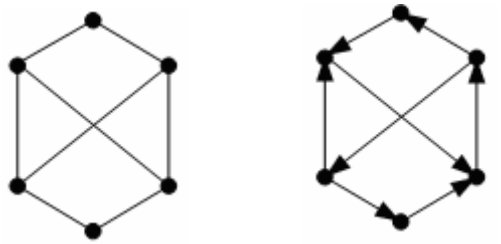
Sebuah graf terhubung *G* adalah *k*-sisi terhubung (*k-edge connected*) jika dan hanya jika sembarang dua titik dari *G* terhubung oleh sedikitnya *k* lintasan saling lepas sisi.

Sebuah graf adalah 2-sisi terhubung jika dan hanya jika sembarang dua titik terhubung oleh minimum dua lintasan yang saling lepas.

2.4. Graf Berarah (*directed graph* atau *digraph*)

Graf berarah adalah graf yang setiap sisinya diberikan orientasi arah. Graf dengan (v_j,v_k) dan (v_k,v_j) merupakan dua buah busur yang bebeda, dengan kata lain $(v_j,v_k) \neq (v_k,v_j)$. Untuk busur (v_j,v_k), simpul v_j disebut titik asal (*initial vertices*) dan v_k disebut titik terminal (*terminal vertices*).

Untuk sebuah *digraph D*, dapat dibuat graf *G* dengan menghilangkan arah pada setiap busur di *D*, dan graf *G* tersebut dinamakan *underlying graph* dari *D*. Sebaliknya, dari sebuah graf *G* dapat dibuat graf berarah *D* dengan memberikan arah tepat satu pada setiap sisi di *G*, dan *digraph D* tersebut dinamakan orientasi dari *G*.

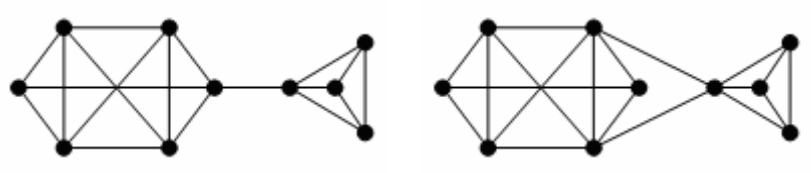


G *D*

Gambar 2.11. Graf *G* adalah *underlying graph* dari *D*,
dan *D* salah satu orientasi dari *G*.

2.5. Orientasi Dua Arah

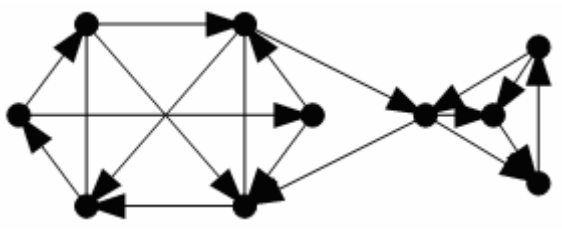
Sebuah *digraph D* dikatakan berorientasi dua arah jika untuk setiap dua titik *u*, *v* di *D* terdapat lintasan-(*u*, *v*) dan lintasan-(*v*, *u*) yang saling lepas.



G_1 G_2

Gambar 2.12. G_1 bukan graf berorientasi arah,
G_2 graf berorientasi arah.

Bagaimanapun graf G_1 diorientasikan, hasilnya tidak akan dapat berorientasi dua arah, sebab G_1 memiliki *cut edge*, sehingga tidak setiap titik dapat mencapai semua titik lainnya. Lain halnya dengan G_2, kita dapat membuat orientasi dua arah sehingga setiap titik dapat dicapai oleh titik lainnya melalui sebuah lintasan.



D_2

Gambar 2.13. Graf D_2 yang merupakan orientasi dua arah dari graf G_2.

Kondisi yang diperlukan agar sebuah graf G memiliki orientasi dua arah adalah bahwa G 2-sisi terhubung. Graf G memiliki orientasi dua arah jika dan hanya jika G adalah graf 2-sisi terhubung.

2.6. Algoritma Dijkstra

Dalam mencari lintasan terpendek antara dua titik terdapat banyak algoritma yang dapat digunakan. Salah satu algoritma yang terkenal adalah algoritma Dijkstra (1959). Algoritma ini tidak hanya menemukan lintasan terpendek antara dua titik (u_0-v_0) tetapi juga lintasan terpendek dari u_0 ke semua titik lain di G.

Misal diberikan graf terhubung G dengan himpunan titik terbatas V_i, i=1,2,3,..,|V| dengan |V| menyatakan banyaknya titik dalam himpunan titik V dan titik 1 sebagai titik asal (*source*). s adalah himpunan bagian dari V dan merupakan titik asal, v adalah titik dalam graf G, $l(v)$ menyatakan bobot lintasan yang menghubungkan titik asal dengan titik lainnya di G, pada tahap pertama nilai $l(v)$ bernilai takhingga untuk semua $v \varepsilon V(G) \backslash \{s\}$, R adalah himpunan titik yang jarak terpendeknya telah diketahui, pada langkah pertama R berisi himpunan kosong.

$c(v,w)$ adalah *array* dua dimensi yang menyatakan bobot atau jarak dari titik v ke titik w bila tidak ada sisi yang menghubungkan titik v ke titik w, maka kita asumsikan $c(v,w)$ bernilai takhingga.

Pada tiap langkah, $l(v)$ akan bernilai dengan jarak terpendek dari titik asal ke titik v, dari semua v akan dipilih $l(v)$ yang paling kecil dan v tersebut ditambahkan kedalam himpunan R sehingga R akan berisi semua titik dengan jarak terpendeknya yang telah diketahui.

Setelah v dimasukan kedalam himpunan R, maka muncul kemungkinan terdapat lintasan lain melalui himpunan R yang lebih pendek ke w malalui v. dengan demikian nilai $l(w)$ semula perlu dibandingkan dengan nilai $l(v) + c(v,w)$ untuk tiap titik w dalam himpunan $V(G)\backslash R$. Dari perbandingan tersebut nilai $l(w)$ akan muncul jika lintasan baru lebih pendek. Untuk membangun struktur lintasan dari titik awal ke titik lainnya dibangun *array* p, dengan $p(w)$ adalah lintasan terpendek.

Algoritma Dijkstra digunakan untuk mencari lintasan terpendek dari sebuah titik s di graf G ke semua titik di G, beserta panjang lintasannya. Sepanjang algoritma, masing-masing titik v membawa label $l(v)$ yaitu jarak terpendek lintasan-(s,v).

Input : Graf G, bobot $c : E(G) \to R_+$ dan titik $s \in V$

Output : Lintasan terpendek dari s ke semua $v \in V(G)$ dan panjang lintasannya.

Langkah :

1. Inisialisasikan label $l(s) = 0$ dan $l(v) = \infty$ untuk semua $v \in V(G) \setminus \{s\}$, Himpunan $R = \varnothing$

2. Cari sebuah titik $v \in V(G) \setminus R$ sedemikian hingga $l(v) = \min_{w \in V(G) \setminus R} l(w)$
3. $R = R \cup (v)$
4. Untuk semua $w \in V(G) \setminus R$ demikian hingga $(v, w) \in E(G)$ lakukan:
   Jika $l(w) > l(v) + c((v, w))$ maka
   $l(w) = l(v) + c((v, w))$ dan $p(w) = v$
5. Jika $R \neq V(G)$ kembali ke no 2, dan jika $R = V(G)$ algoritma selesai

2.7. Algoritma Pemeriksa 2-Sisi Terhubung

Penyelidikan sebuah graf 2-sisi terhubung dilakukan dengan memeriksa apakah setiap dua titik terhubung oleh dua lintasan yang saling lepas sisi. Untuk sembarang dua titik, pencarian dua lintasan yang saling lepas sisi dilakukan dengan menggunakan bantuan algoritma Dijkstra.

Input : Graf G, $|V(G)| = n$

Output : Sifat graf apakah 2-sisi terhubung atau tidak

Langkah :

1. Untuk i =1 ke n-1
2. *{jalankan algoritma Dijkstra untuk mencari lintasan terpendek dari titik i ke semua titik lain pada G}*
3. Untuk $j = i$+1 ke n
4. *{diperoleh lintasan-1 dari i ke j}*
5. *{hapus semua sisi yang masuk lintasan-1 dari i ke j pada G menjadi G'}*
6. *{jalankan algoritma Dijkstra untuk mencari lintasan terpendek dari i pada G'}*

7. *{diperoleh lintasan-2 dari i ke j}*
8. *{jika tidak terdapat lintasan ke-2 maka algoritma berhenti dan hasilnya graf G tidak 2-edge-connected}*
9. Lanjut *j*
10. Lanjut *i*
11. *{jika setiap 2 titik terdapat 2 lintasan yang edge-disjoint maka graf G 2-edge connected}*

2.8. Algoritma Pembuat Orientasi Dua Arah

Berdasarkan penjelasan pada landasan teori bahwa *G* memiliki orientasi dua arah jika dan hanya jika *G* adalah graf 2-sisi terhubung**,** maka dari graf 2-sisi terhubung dapat dibuat orientasi dua arahnya.

Input : Graf *G* dengan sifat 2-sisi terhubung

Output : Orientasi dua arahnya

Langkah :

1. Berdasarkan hasil dari Algoritma Pemeriksa 2-sisi terhubung, buat sebuah lingkaran G_1.
2. Jika $|V(G_1)| < |V(G)|$, maka terdapat $\boldsymbol{v_i} \in \boldsymbol{G}$ tetapi $\boldsymbol{v_i} \notin \boldsymbol{G_i}$. Akibatnya terdapat dua lintasan yang saling lepas P_i dan Q_i dari v_i ke G_i. Definisikan: $G_{i+1} = G_i \cup P_i \cup Q_i$
3. Jika $|V(G_i)| \neq |V(G)|$, kembali ke langkah 2. Jika $|V(G_i)| = |V(G)|$, berikan orientasi sembarang untuk sisi di *G* yang tidak ada di G_i

2.9. Pengertian Program Aplikasi

2.9.1. Pengertian Program

Program adalah ekspresi, pernyataan atau kombinasi yang disusun dan dirangkai menjadi satu kesatuan prosedur yang berupa urutan langkah untuk menyelesaikan masalah yang diimplementasikan dengan menggunakan bahasa pemrograman, sehingga dapat dieksekusi oleh komputer.

2.9.2. Pengertian Aplikasi

Aplikasi merupakan penerapan, menyimpan sesuatu hal, data, permasalahan pekerjaan kedalam suatu sarana atau media yang digunakan untuk menerapkan atau mengimplementasikan hal atau permasalahan tersebut sehingga berubah menjadi suatu bentuk yang baru tanpa menghilangkan nilai-nilai dasar dari hal, data, permasalahan atau pekerjaan. Jadi dalam hal ini hanya bentuk dari tampilan data yang berubah, sedangkan isi yang memuat dalam data tersebut tidak mengalami perubahan.

Jadi Program Aplikasi adalah sederetan kode yang digunakan untuk mengatur komputer supaya dapat melakukan pekerjaan sesuai dengan keinginan dari permasalahan pengguna.

2.10. Alat Bantu Sistem

2.10.1. *Flowchart*

Bagan alir (*flowchart*) adalah bagan yang menunjukan alir di dalam program atau prosedur sistem secara logika. *Flowchart* digunakan terutama untuk alat bantu komunikasi dan untuk dokumentasi.

2.10.2. *Entity Relationship Diagram*

Entity Relationship Diagram (ERD) adalah model data yang menggunakan beberapa notasi untuk menggambarkan data dalam konteks entitas dan hubungan yang dideskripsikan oleh data tersebut.

2.10.3. Diagram Konteks

Diagram konteks *(context diagram*) menggambarkan hubungan antara sistem dengan entitas luarnya. Diagram konteks berfungsi sebagai transformasi dari satu proses yang melakukan transformasi data *input* menjadi data *output*. Entitas yang dimaksud adalah entitas yang mempunyai hubungan langsung dengan sistem. Suatu diagram konteks selalu mengandung satu dan hanya satu proses saja. Proses ini mewakili proses dari seluruh sistem.

2.10.4. *Data Flow Diagram*

Data Flow Diagram (DFD) adalah model proses yang digunakan untuk menggambarkan aliran data melalui sebuah sistem dan tugas atau pengolahan yang dilakukan sistem.

DFD sering digunakan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau lingkungan fisik dimana data tersebut akan disimpan. DFD merupakan alat yang digunakan pada metodologi pengembangan sistem yang terstruktur (*structured analyst and design*). DFD merupakan alat yang cukup populer sekarang ini, karena dapat menggambarkan arus data di dalam sistem dengan terstruktur dan jelas, juga merupakan dokumentasi dari sistem yang baik.

2.10.5. Kamus Data

Kamus data (*data dictionary*) atau disebut juga dengan istilah *system data dictionary* adalah katalog fakta tentang data dan kebutuhan-kebutuhan informasi dari suatu sistem informasi. Dengan menggunakan kamus data, analisis sistem dapat mendefinisikan data yang mengalir di sistem dengan lengkap. Kamus data dibuat pada tahap analisis sistem dan digunakan baik pada tahap analisis maupun pada tahap perancangan sistem.

2.11. Perangkat Lunak Penunjung

2.11.1. Sekilas Visual Basic 6.0

Visual Basic 6.0 (VB6) merupakan salah satu aplikasi pemrograman visual yang dibuat oleh Microsoft. VB6 berjalan dalam sistem operasi Windows dan tergabung dalam *suite* aplikasi Microsoft Visual Studio 6.0 yang dikeluarkan pada akhir tahun 1998.

Aplikasi Visual Basic mulai diproduksi pertama kali pada tahun 1991. Setelah itu munculah versi-versi lanjutan dari Visual Basic, yaitu Visual Basic 3, 4, 5, dan 6. pada Visual Basic 4, dukungan terhadap aplikasi 32 bit mulai diberikan. Versi Visual Basic terbaru adalah Visual Basic.Net yang dirilis pada tahun 2002.

VB6 menyediakan berbagai perangkat yang dapat digunakan untuk membuat program aplikasi baik aplikasi kecil dan sederhana untuk keperluan sendiri, hingga aplikasi untuk sistem *enterprise* yang besar dan rumit, atau bahkan aplikasi yang dijalankan melalui internet.

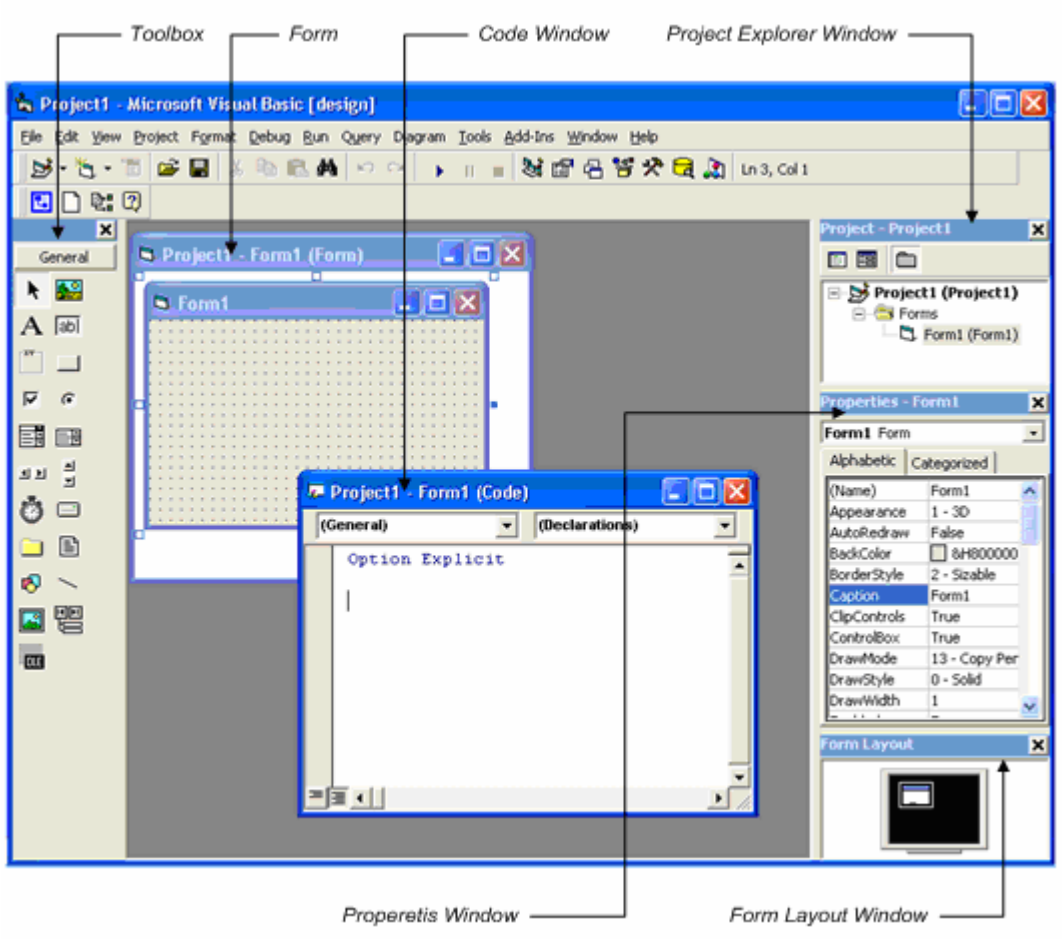
VB6 memanfaatkan pendekatan visual *Grafhical User Interface* (GUI) dalam proses penggunaannya. Dengan pendekatan GUI, proses pembuatan program aplikasi menjadi lebih mudah dan nyaman.

Basis bahasa pemrograman yang digunakan dalam VB6 adalah bahasa *Beginners All Purpose Symbolic Instruction Code* (BASIC). Bahasa BASIC merupakan salah satu bahasa pemrograman tingkat tinggi yang sederhana dan mudah dipelajari.

Oleh karena dibuat oleh Microsoft, VB6 memiliki keunggulan dalam hal pengaksesan terhadap beberapa pustaka (*library*) yang dimiliki oleh sistem operasi Windows. Para pemrogram (*programmer*) dapat memanfaatkan Windows *Application Programming Interfa*ce (API) untuk membuat program aplikasi yang lebih kompleks dan *powerfull*.

Kelebihan VB6 yang lain adalah sebagai berikut:

a. Kompiler yang sangat cepat.
b. *Control data object* untuk ActiveX yang baru.
c. Dapat mendukung *database* yang terintegrasi dengan variasi aplikasi yang sangat luas.
d. Perancangan data laporan yang lebih baru.
e. Adanya *Package* dan *Develoyment Wizard* yang bisa digunakan untuk membuat distribusi *disk* dari aplikasi yang kita buat.
f. Adanya tambahan dukungan terhadap internet.



Gambar 2.14. Antarmuka Visual Basic 6.0.

Komponen-komponen utama VB6 adalah sebagai berikut:

a. *Tool Box*

Tool box terdiri atas beberapa *class* objek yang dapat digunakan dalam proses pembuatan aplikasi.

b. *Form*

Merupakan tempat yang digunakan untuk merancang aplikasi yang sedang dibuat. Form dapat diibaratkan ‘kanvas’ untuk ‘melukis’ *user interface*.

c. *Code Window*

Merupakan jendela yang digunakan untuk menampilkan atau mengetikan kode program. Setiap form dalam VB6 memiliki satu buah *code window* sendiri.

d. *Propertis Window*

Merupakan jendela untuk menampilkan dan mengubah properti-properti yang dimiliki sebuah objek.

e. *Project Explorer Window*

Merupakan jendela untuk menampilkan *project-project*, *form-form* atau modul-modul yang terlibat dalam proses pembuatan aplikasi.

f. *Form Layout Window*

Merupakan jendela yang menunjukan posisi relatif *form* terhadap layar monitor.