BAB II
LANDASAN TEORI

2.1 Definisi Graf

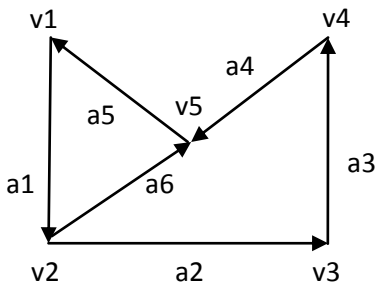
Graf adalah himpunan benda-benda yang disebut simpul (*vertex* atau node) yang terhubung oleh sisi (*edge*) atau busur (*arc*). Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan simpul) yang dihubungkan oleh garis-garis (melambangkan sisi) atau garis berpanah (melambangkan busur). Suatu graf G didefenisikan sebagai pasangan himpunan (V,E), di mana V = himpunan yang berisikan simpul pada graf tersebut $\{v_1, v_2, .., v_n\}$ dan E adalah himpunan sisi yang menghubungkan simpul-simpul $\{e_1, e_2, .., e_n\}$ atau dapat ditulis dengan notasi G = (V,E) . [2] [6]

Berdasarkan orientasi arah pada sisi, graf dapat dibedakan atas dua jenis yaitu :

1. Graf Berarah
2. Graf Tidak Berarah

2.1.1 Graf Berarah (*directed* graf)

Graf G disebut graf berarah (*directed* graf) jika setiap sisinya berarah. titik awal dari suatu sisi disebut simpul awal (*initial vertex*) sedangkan titik akhir dari suatu sisi disebut simpul akhir (*terminal vertex*). *Loop* pada graf adalah sisi yang simpul awal dan simpul akhirnya sama.

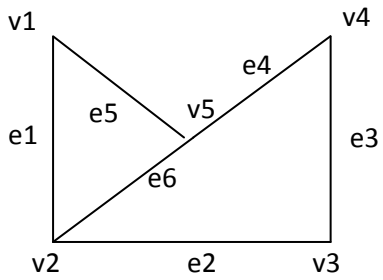


Gambar 2.1 Graf Berarah atau *Digraf*

Digraf pada Gambar 2.1 menunjukkan graf berarah dengan himpunan simpul, V(G) = {$v1, v2, v3, v4, v5$} dan himpunan busur A(G) = {a1,a2, a3, a4, a5, a6} yaitu pasangan terurut dari {(v1,v2), (v2,v3), (v3,v4), (v4,v5), (v5,v1), (v2,v5)}.

2.1.2 Graf Tidak Berarah

Graf Tak Berarah *(Undirected Graph)* Graf G disebut graf tak berarah jika setiap sisinya tidak berarah.



Gambar 2.2 Graf Tidak Berarah

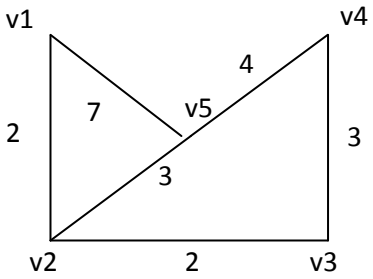
Graf pada gambar 2.2 menunjukkan graf tidak berarah dengan himpunan simpul, V(G) = {$v1, v2, v3, v4, v5$} dan himpunan sisi E(G) {e1, e2, e3, e4, e5, e6} yaitu pasangan terurut dari {(v1,v2), (v2,v3), (v3,v4), (v4,v5), (v5,v1), (v5,v2)}.

2.1.3 Graf Berbobot

Di dalam model graf, ada informasi yang ditambahkan pada busur graf. Misalnya pada graf yang menggambarkan antara kota-kota, dapat ditambahkan sebuah bilangan pada setiap busur untuk menujukkan jalur antara kedua kota yang dibutuhkan oleh busur tersebut.

Graf berbobot adalah suatu graf tanpa busur parallel dimana setiap busurnya berhubungan dengan suatu bilangan riil tak negatif yang menyatakan bobot busur (w(a)) tersebut [2].

Graf pada gambar 2.3 adalah graf berbobot dan tidak berarah yang terdiri dari lima node yaitu node 1, 2, 3, 4, 5.



Gambar 2.3 Graf berbobot dan tidak berarah

2.1.4 Representasi Graf dalam Matriks

Matriks dapat digunakan untuk menyatakan suatu graf, graf yang direpresentasikan pada matriks, dapat dibedakan sebagai berikut :

1. Matriks *Adjacency*

   Misalkan G adalah graf yang tidak berarah yang terdiri dari n node tanpa *edge* paralel. Matriks *Adjacency* pada graf G adalah matriks bujursangkar n x n, A= (a_{ij}) dengan

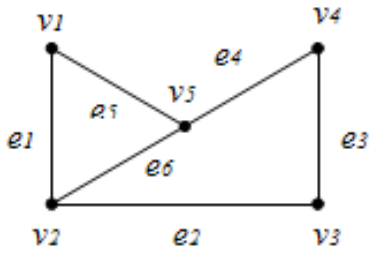
$$a = \begin{cases} \text{Tersambung, 1 jika ada } edge \text{ dari titik } v_i \text{ ke titik } v_j \\ \text{Tidak tersambung, 0 jika tidak ada } edge \text{ dari titik } v_i \text{ ke titik } v_j \end{cases}$$

Matriks *adjacency* dari graf pada gambar 2.2 adalah:

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

Jika graf yang diberikan adalah graf berbobot, maka elemen matriks yang terhubung antara verteks adalah bobot graf. [6]

2. Matriks *Incidency* atau matriks bersisian adalah matriks yang merepresentasikan hubungan antara node dan *edge*. Misalkan *B* adalah matriks dengan *m* baris untuk setiap node dan *n* kolom untuk setiap *edge*. Jika node terhubung dengan *edge*, maka elemen matriks bernilai 1. Sebaliknya, jika node tidak terhubung dengan *edge,* maka elemen matriks bernilai 0.



Gambar 2.4 Graf dengan 5 node dan 6 *edge*

Matriks bersisian dari graf pada gambar 2.4 adalah:

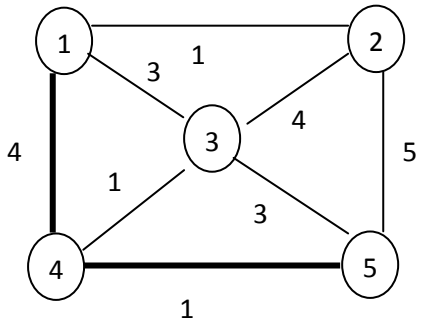
$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccc} e1 & e2 & e3 & e4 & e5 & e6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

2.2 Definisi Lintasan (*Path*)

Lintasan adalah hubungan antar titik atau node dalam sebuah graf. Suatu lintasan yang berawal dan berakhir pada node yang sama, maka disebut lintasan tertutup (*close path*), jika node awal dan node akhir dari lintasan tersebut berbeda, disebut lintasan terbuka (*open path*). [3] [7]

2.2.1 Lintasan Terpendek (*Shortest Path*)

Lintasan terpendek adalah lintasan yang memiliki total bobot minimum untuk mencapai suatu tempat dari tempat tertentu. Lintasan terpendek dapat dicari dengan menggunakan graf. Graf yang digunakan adalah graf yang berbobot, yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan, waktu, biaya dan sebagainya. [3]



Gambar 2.5 *Shortest Path* (garis tebal)

Gambar 2.5 menunjukkan lintasan terpendek (garis tebal) dari node 1 ke node 5 dengan jalur 1-4-5 dengan jumlah bobot lima (5).

2.3 Algoritma Dijkstra

2.3.1 Sejarah Algoritma Dijkstra

Algoritma Dijkstra ditemukan oleh seorang ilmuwan asal Belanda, Edger Wybe Dijkstra. Algoritma ini digunakan untuk mencari lintasan terpendek pada *graf* berarah, tetapi algoritma ini juga bisa diterapkan pada *graf* tak berarah.

Algoritma Dijkstra adalah sebuah algoritma untuk memecahkan permasalahan jalur terpendek (*shortest path problem*) pada sebuah graf berarah (*directed graf*) atau graf tidak berarah (*undirected graf*) dengan bobot-bobot sisi (*edge weights*) yang bernilai tidak negatif. Bila node dari sebuah graf melambangkan kota-kota dan bobot sisi (*edge weights*) melambangkan jalur antara kota-kota tersebut, maka algoritma Dijkstra dapat digunakan untuk menemukan jalur terpendek antara dua kota.

Dalam menentukan jalur terpendek dari suatu graf oleh algoritma dijkstra akan didapatkan jalur yang terbaik, karena pada waktu penentuan jalur yang akan dipilih, akan dianalisis bobot dari node yang belum terpilih, lalu dipilih node dengan bobot yang terkecil. Jika ternyata ada bobot yang lebih kecil melalui node tertentu maka bobot akan dapat berubah. Algoritma dijkstra akan berhenti ketika semua node sudah terpilih. Sehingga akan ditemukan jalur terpendek dari seluruh node, tidak hanya untuk node dari asal dan node tujuan tertentu saja. [5]

2.3.2 Cara Kerja Algoritma Dijkstra

Algoritma ini mencari panjang lintasan terpendek dari node asal ke node tujuan dalam sebuah graf.

Langkah-langkah dalam menentukan lintasan terpendek pada algoritma Dijkstra yaitu :

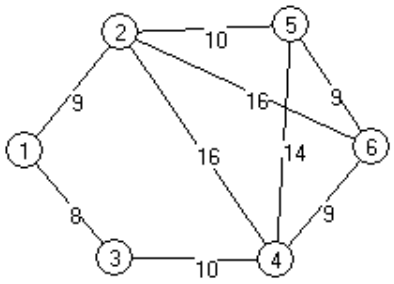
1. Pada awalnya inisialisasikan node asal (V1) dan node tujuan (V2).
2. Buat 2 buah list, *open list* dan *closed list.* Keduanya tidak ada data atau kosong, dan formatnya {node, bobot, node induk}.
3. Masukkan V1 ke *open list.*
4. Pilih 1 node dengan bobot terkecil pada *open list*, tambahkan kedalam *closed list.*
5. Cari node yang bertetangga langsung dari node sebelumnya, yang masuk terakhir dalam *closed list*. Tambahkan bobot dengan node yang terkait, apabila sudah ada dalam *closed list* abaikan.
6. Apabila dalam *open list* terdapat node yang sudah ada bandingkan, lalu cari yang terkecil, dan perbaharui. Bila ternyata jumlah bobotnya sama dalam node yang sama, maka abaikan.
7. Apakah data dalam *open list* kosong ? jika belum ulangi langkah 4

8. Dalam *closed list* cari V2, telusuri jalur berdasarkan node induk sampai mengacu ke node asal (V1), dan balikkan urutan node.

9. Lintasan terpendek ditemukan bersama bobotnya.

Algoritma dijkstra digunakan untuk pencarian jalur terpendek dari suatu graf, sehingga akan didapatkan jalur yang akan ditempuh. [5]

2.3.3 Penerapan Cara Kerja Algoritma Dijkstra



Gambar 2.6 Sebelum pencarian jalur terpendek

Gambar 2.6 menunjukkan pencarian jalur terpendek dari graf yang memiliki enam buah node dan sembilan *edge* yang saling terhubung, pencarian dimulai dari node satu (1) ke node enam (6) sebagai node yang dituju.

Langkah awalnya inisialisasikan node 1 sebagai node awal (V1) dan node 6 sebagai node tujuan (V2), buat 2 buah *list*, *open list* dan *closed list*.

Tabel 2.1 Hasil Iterasi Ke-1

Open List	Closed List
{1,0,null}	-

Dari tabel 2.1 masukkan node 1 ke dalam *open list*, node 1 memiliki bobot 0, dan node induknya *null* karena sebagai awal proses pencarian.

Tabel 2.2 Hasil Iterasi Ke-2

Open List	Closed List

{2,9,1}	{1,0,null}
{3,8,1}	

Dari tabel 2.2 Proses dari bobot terkecil lebih dahulu, dikarenakan ada satu yaitu, node 1 maka masukkan kedalam *closed list*. Node yang bertetangga dengan node 1 adalah, node 2 dan node 3, lalu masukkan ke dalam *open list*, node 2 memiliki bobot 9, dan node induknya node 1, node 3 memiliki bobot 8, dan node induknya node 1.

Tabel 2.3 Hasil Iterasi Ke-3

Open List	Closed List
{2,9,1}	{1,0,null}
{4,18,3}	{3,8,1}

Dari tabel 2.3 Proses dari bobot terkecil lebih dahulu yaitu, node 3 dan masukkan kedalam *closed list*, node yang bertetangga dengan node 3 adalah node 1 dan node 4, lalu masukkan ke dalam open list, karena node 1 sudah masuk ke dalam *closed list* diabaikan dan node 4 memiliki bobot 18, dan node induknya node 3.

Tabel 2.4 Hasil Iterasi Ke-4

Open List	Closed List
{4,18,3}	{1,0,null}
{5,19,2}	{3,8,1}
{6,25,2}	{2,9,1}

Dari tabel 2.4 Proses dari bobot terkecil lebih dahulu yaitu, node 2 dan masukkan kedalam *closed list*, node yang bertetangga dengan node 2 adalah node 1, node 4, node 5 dan node 6, lalu masukkan ke dalam *open list*, karena node 1 sudah masuk ke dalam *closed list* diabaikan, karena node 4 sudah ada dalam *open list* bandingkan bobotnya, ambil yang terkecil. Sehingga node 4 memiliki bobot

18, dan node induknya node 3. Node 5 memiliki bobot 19, dan node induknya node 2, dan node 6 memiliki bobot 25, dan node induknya node 2.

Tabel 2.5 Hasil Iterasi Ke-5

Open List	Closed List
{5,19,2}	{1,0,null}
{6,25,2}	{3,8,1}
	{2,9,1}
	{4,18,3}

Dari tabel 2.5 Proses dari bobot terkecil lebih dahulu yaitu, node 4 dan masukkan kedalam *closed list*, node yang bertetangga dengan node 4 adalah node 2, node 3, node 5 dan node 6, lalu masukkan ke dalam *open list*, karena node 2, dan node 3 sudah masuk ke dalam *closed list* diabaikan, karena node 5 dan node 6 sudah ada dalam *open list* bandingkan bobotnya, ambil yang terkecil. Sehingga node 5 memiliki bobot 19, dan node induknya node 2. Node 6 memiliki bobot 25, dan node induknya node 2.

Tabel 2.6 Hasil Iterasi Ke-6

Open List	Closed List
{6,25,2}	{1,0,null}
	{3,8,1}
	{2,9,1}
	{4,18,3}
	{5,19,2}

Dari tabel 2.6 Proses dari bobot terkecil lebih dahulu yaitu, node 5 dan masukkan kedalam *closed list*, node yang bertetangga dengan node 5 adalah node 2, node 4 dan node 6, lalu masukkan ke dalam *open list*, karena node 2, dan node 4 sudah masuk ke dalam *closed list* diabaikan, karena node 5 sudah ada dalam

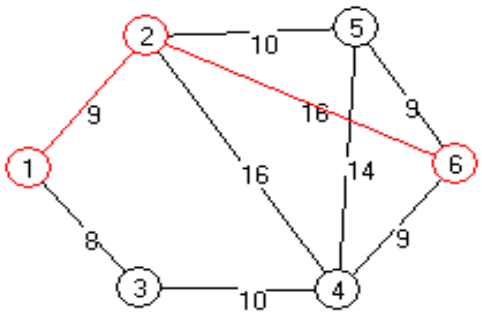
open list bandingkan bobotnya, ambil yang terkecil. Sehingga node 6 memiliki bobot 25, dan node induknya node 2.

Tabel 2.7 Hasil Iterasi Ke-7

Open List	Closed List
-	{1,0,null}
	{3,8,1}
	{2,9,1}
	{4,18,3}
	{5,19,2}
	{6,25,2}

Dari tabel 2.7 Proses dari bobot terkecil lebih dahulu, karena tinggal 1 maka node 6 masukkan kedalam *closed list*, node yang bertetangga dengan node 6 adalah node 2, node 4 dan node 5, lalu masukkan ke dalam *open list*, karena node 2, node 4 dan node 5 sudah masuk ke dalam *closed list* diabaikan.

Pencarian akan berhenti dikarenakan *open list* sudah kosong, didalam *closed list* telusuri V2 atau node tujuan dengan mengacu pada node induk, dan balikkan urutan nodenya, sehingga pencarian jalur terpendek dari graf yang dimulai dari node satu (1) ke node enam (6) diperoleh jalur **1-2-6** dengan jumlah bobot **25**. Dapat dilihat pada gambar 2.6.



Gambar 2.6 Sesudah pencarian jalur terpendek

2.4 DFD (*Data Flow Diagram*)

Data Flow Diagram (DFD) adalah alat pembuatan model yang memungkinkan professional sistem untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data, baik secara manual maupun komputerisasi. DFD ini sering disebut juga dengan nama *Bubble chart, Bubble* diagram, model proses, atau model fungsi. DFD ini adalah salah satu alat pembuatan model yang sering digunakan, khususnya bila fungsi-fungsi sistem merupakan bagian yang lebih penting dan kompleks dari pada data yang dimanipulasi oleh sistem.

Dengan kata lain, DFD adalah alat perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi dapat digunakan untuk penggambaran analisa maupun rancangan sistem yang mudah dikomunikasikan oleh professional sistem kepada pemakai maupun pembuat program.

2.4.1 Penggambaran DFD

Tidak ada aturan baku untuk menggambarkan DFD. Tapi dari berbagai referensi yang ada, secara garis besar langkah untuk membuat DFD adalah :

1. Identifikasai terlebih dahulu semua entitas luar yang terlibat di sistem.
2. Identifikasi semua input dan output yang terlibat dengan entitas luar.
3. Buat diagram konteks, disebut juga diagram level tertinggi dari DFD yang menggambarkan hubungan sistem dengan lingkungan luarnya. Caranya dengan tentukan nama sistemnya, tentukan batasan sistemnya, tentukan terminator apa saja yang ada dalam sistem, lalu tentukan apa yang diterima/diberikan terminator dari/ke sistem, dan gambarkan diagram konteks.
4. Buat diagram *Level Zero*, diagram ini merupakan dekomposisi dari diagram konteks. Caranya dengan tentukan proses utama yang ada pada sistem, tentukan apa yang diberikan/diterima masing-masing proses ke/dari sistem sambil memperhatikan konsep keseimbangan, apabila diperlukan muculkan data *store* (master) sebagai sumber maupun tujuan alur data dan gambarkan diagram *Level Zero*.

5. Buat diagram *Level* Satu, diagram ini merupakan dekomposisi dari level zero. Caranya tentukan proses yang lebih kecil dari proses utama yang ada di level zero, tentukan apa yang diberikan/diterima masing-masing proses ke/dari sistem sambil memperhatikan konsep keseimbangan, apabila diperlukan muculkan data *store* (transaksi) sebagai sumber maupun tujuan alur data dan gambarkan diagram Level Satu.
6. DFD *Level* Dua, Tiga, ..

   Diagram ini merupakan dekomposisi dari level sebelumnya. Proses dekomposisi dilakukan sampai dengan proses siap dituangkan ke dalam program. Aturan yang digunakan sama dengan level satu.

2.5 Pemrograman Borland Delphi

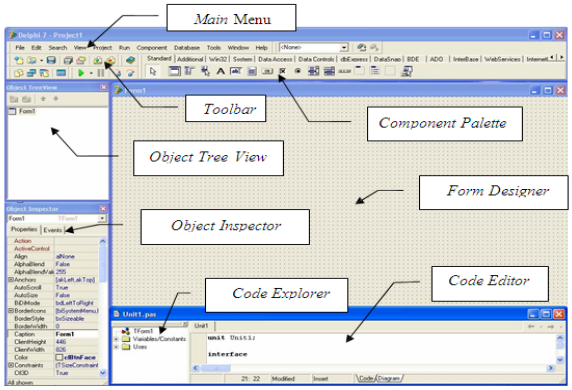
2.5.1 Pengenalan Borland Delphi

Delphi adalah sebuah *IDE COMPILER* untuk bahasa pemrograman Pascal dan lingkungan pengembangan perangkat lunak. Produk ini dikembangkan oleh CodeGear sebagai divisi pengembangan perangkat lunak milik Embarcadero, divisi tersebut sebelumnya adalah milik Borland. Bahasa Delphi, atau dikenal pula sebagai *object pascal* (pascal dengan ekstensi pemrograman berorientasi objek (PBO/OOP)) pada mulanya ditujukan hanya untuk Microsoft Windows, namun saat ini telah mampu digunakan untuk mengembangkan aplikasi untuk Linux dan Microsoft .NET framework.

Borland Delphi merupakan multi fungsi yang bekerja dalam sistem operasi Windows dan juga sering disebut alat bantu (*tool*). Dalam sejarah pemrograman sudah banyak digunakan oleh programmer untuk menyusun sebuah aplikasi. Microsoft Delphi 7.0 adalah bahasa pemrograman yang bekerja dalam lingkup MS-Windows dan merupakan bahasa pemrograman tercepat saat ini untuk membuat aplikasi pada sistem operasi Windows. Hal ini disebabkan dengan menggunakan Microsoft Delphi tidak perlu menuliskan instruksi pemrograman dalam kode-kode baris tetapi secara mudah cukup dengan melakukan *Drag and Drop* obyek-obyek yang akan digunakan. [1] [8]

2.5.2 Bagian Umum Lingkungan Borland Delphi 7

Lingkungan pengembangan terpadu atau *Integrated Development Environment* (IDE) dalam program Delphi terbagi menjadi delapan bagian umum, yaitu *Main Menu, ToolBar, Component Palette, Form Designer, Code Editor,Object Inspector, Exploring,* dan *Object Tree View*. Untuk lebih jelasnya ditunjukkan pada gambar 2.5.



Gambar 2.7 Lembar Kerja Borland Delphi

IDE merupakan sebuah lingkungan di mana semua tombol perintah yang diperlukan untuk mendesain aplikasi, menjalankan dan menguji sebuah aplikasi disajikan dengan baik untuk memudahkan pengembangan program. [1] [8]

Lingkungan pengembangan terpadu dalam program delphi terdiri dari:

(1) *Main* Menu

Menu utama pada Delphi memiliki kegunaan yang sama seperti program aplikasi windows lainnya. Dengan mengunakan fasilitas menu, program dapat dipanggil atau disimpan. Pada dasarnya semua perintah yang diberikan dapat ditemukan pada bagian menu utama ini.

(2) *Tool bar*

Delphi memiliki beberapa *toolbar* yang masing-masing memiliki perbedaan fungsi dan setiap tombol pada bagian *toolbar* berfungsi sebagai pengganti suatu menu perintah yang sering digunakan. Tombol-tombol yang terletak pada bagian *toolbar* dapat ditambah atau dikurangi sesuai kebutuhan.

(3) *Component Palette*

Component palette berisi kumpulan ikon yang melambangkan komponen-komponen yang terdapat pada VCL (*Visual Component Library*). Pada komponen *palette* dapat ditemukan beberapa *page control*, seperti *standard, additional, win 32, system, data access* dan lain-lain.

(4) *Form Designer*

Form designer merupakan suatu obyek yang dapat dipakai sebagai tempat untuk merancang program aplikasi. Dalam sebuah *form* terdapat titik-titik yang disebut *grid* yang berguna untuk membantu pengaturan tata letak obyek yang dimasukkan dalam *form*.

(5) *Object Inspector*

Object inspector digunakan untuk mengubah properti atau karakteristik dari sebuah komponen. *Object inspector* terdiri dari dua tab yaitu *properties* dan *event*.

(6) *Code Editor*

Code editor merupakan tempat kode program dituliskan. Keuntungan bagi pemakai Delphi adalah tidak perlu menuliskan kode- kode sumber, karena Delphi telah menyediakan kerangka penulisan sebuah program. *Code editor* dilengkapi dengan fasilitas *highlight* yang memudahkan pemakai menemukan kesalahan.

(7) *Code Explorer*

Jendela pada *code explorer* adalah lembar kerja baru yang terdapat di dalam Delphi 7 yang tidak ditemukan pada versi-versi sebelumnya. *Code explorer* digunakan untuk memudahkan pemakai berpindah antar *file* unit yang terdapat di dalam jendela *code editor*.

(8) *Object Tree View*

Object Tree View menampilkan diagram pohon dari komponen-komponen yang bersifat *visual* maupun *non visual* yang telah terdapat dalam *form*, data *module*, atau *frame*. *Object tree view* juga menampilkan hubungan logika antar komponen.