BAB II

MATERI PENUNJANG

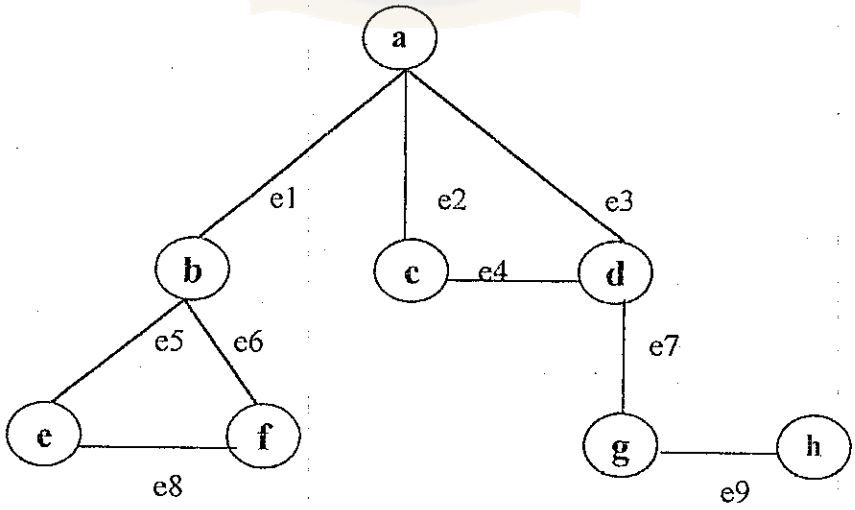
2.1 TREE

Definisi 2.1.1 :

Suatu graph (atau graph tak berarah) G terdiri dari suatu himpunan V yang beranggotakan titik (vertices atau nodes) dan himpunan V tidak boleh kosong ($V \neq \{\}$) serta suatu himpunan E yang beranggotakan garis (edge) sedemikian sehingga garis $e \in E$ berasosiasi dengan pasangan titik yang tak terurut. Notasi yang digunakan adalah G= (V,E)

Jika sebuah garis e berasosiasi dengan sepasang titik v dan w yang tunggal (unique) maka dinotasikan dengan e= {v,w} atau e= {w,v}. Dalam konteks ini {v,w} menyatakan sebuah garis dalam graph tak berarah dan bukan merupakan pasangan terurut. Untuk selanjutnya titik dalam V disebut vertex.

Contoh :



Gambar 2.1.a. Graph G

Keterangan gambar 2.1.a :

G = (V,E)

V(G) = {a,b,c,d,e,f,g,h}

E(G) = {e1,e2,e3,e4,e5,e6,e7,e8,e9}

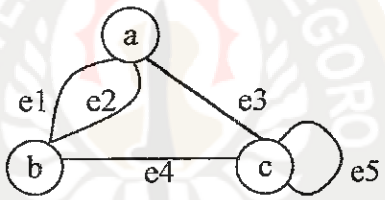
atau E(G) = { {a,b},{a,c},{a,d},{c,d},{b,e},{b,f},{d,g},{e,f},{g,h} }

Definisi 2.1.2 :

Loop adalah suatu garis dengan bentuk {v,v}.

Definisi 2.1.3 :

Multiple edges adalah dua garis atau lebih yang menghubungkan pasangan vertex yang sama.



Gambar 2.1.b. Graph G dengan Loop dan Multiple edges

Loop : { {c,c} } atau { e5 }

Multiple edges : { {a,b},{a,b} } atau { e1,e2 }

Definisi 2.1.4 :

Jika v dan w adalah vertex-vertex dalam G, maka setiap sepasang x={v,w} adalah suatu garis dari G, dan x disebut menghubungkan v dan w ditulis dengan notasi x=vw dan dikatakan bahwa v dan w adalah vertex-vertex yang adjacent. Dan vertex v dan garis x adalah incident satu sama lain, begitu juga antara vertex w dan garis x adalah incident satu sama lain.

Pada gambar 2.1.a dapat diketahui bahwa, misalnya untuk garis e1={a,b}, maka vertex a dan vertex b adalah vertex-vertex yang adjacent, dan vertex a dengan garis e1 adalah incident satu sama lain serta vertex b dengan garis e1 adalah incident satu sama lain.

Definisi 2.1.5 :

Walk dari suatu graph G adalah barisan vertex dan garis secara bergantian $v_0, x_1, v_1, x_2, \ldots, v_{n-1}, x_n, v_n$ yang dimulai dan diakhiri dengan vertex, dan setiap garis dalam walk yang menghubungkan 2 vertex yaitu vertex-vertex yang berada tepat sebelum dan sesudahnya, adalah incident dengan kedua vertex tersebut.

Walk ini menghubungkan v_0 dan v_n, dan juga dapat dinotasikan dengan $v_0\ v_1\ v_2\ v_3\ \ldots\ v_n$; dan biasanya disebut v_0-v_n walk. Walk disebut tertutup jika $v_0 = v_n$ dan walk disebut terbuka jika $v_0 \neq v_n$. Panjang dari walk $v_0\ v_1\ v_2\ v_3\ \ldots\ v_n$ adalah n yaitu jumlah dari edge (garis) sepanjang walk tersebut.

Contoh dari walk pada gambar 2.1.a adalah sebagai berikut :

Walk : abebf

Walk tertutup : abefba

Walk terbuka : acdgd

Definisi 2.1.6 :

Trail dari suatu graph G adalah suatu walk dengan syarat bahwa semua edge (garis) saling berbeda. Contoh suatu trail dari graph G pada gambar 2.1.a adalah

Trail : abefb

Definisi 2.1.7 :

Sebuah lintasan (path) dari suatu graph G adalah suatu walk dengan syarat bahwa semua vertex dan semua edge saling berbeda. Jika v dan w adalah vertex-vertex dalam G, maka geodesic adalah lintasan (path) terpendek dari vertex v ke vertex w.

Contoh suatu lintasan (path) dari graph G pada gambar 2.1.a adalah sebagai berikut :

Lintasan : acdgh

Definisi 2.1.8 :

Sebuah cycle dari suatu graph G adalah suatu walk tertutup dengan n vertex yang berbeda dan $n \geq 3$. Contoh suatu cycle dari graph G pada gambar 2.1.a adalah sebagai berikut :

Cycle : acda

Definisi 2.1.9 :

Graph G dikatakan terhubung (connected) jika diberikan sembarang vertex-vertex v dan w yang berbeda, terdapat suatu lintasan dari v ke w.

Gambar 2.1.a merupakan contoh dari suatu graph G yang terhubung (connected). Vertex-vertex yang berada dalam graph G tersebut terhubung oleh suatu lintasan. Vertex a misalnya, terhubung oleh suatu lintasan ke vertex b, vertex c dan juga vertex d. Begitu pula vertex-vertex yang lainnya seperti yang terlihat pada gambar 2.1.a.

Definisi 2.1.10 :

Subgraph dari G adalah suatu graph yang himpunan vertexnya merupakan himpunan bagian dari V(G) dan himpunan garisnya merupakan himpunan bagian dari E(G) atau suatu graph G` disebut subgraph dari graph G bila hanya bila V`(G`) ⊂ V(G) dan E`(G`) ⊂ E(G). Notasi yang digunakan adalah G`=(V`,E`).

Contoh dari subgraph G' pada gambar 2.1.a adalah sebagai berikut :

G' = (V',E')

V' = { b,e,f }

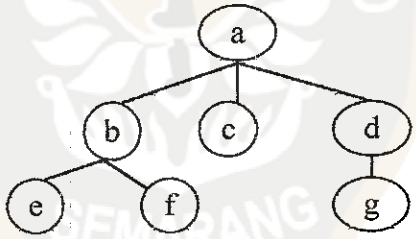
E' = { e5, e6, e8 }

Definisi 2.1.11 :

Tree adalah suatu graph yang :

- Acyclic (tidak mempunyai cycle)
- Connected / terhubung (antara dua vertex sekurang-kurangnya ada satu lintasan)

Contoh :



Gambar 2.1.d. Tree

Masing-masing anggota dalam suatu tree disebut vertex tree. Sebuah tree dapat juga mempunyai banyak sekali subtree. Masing-masing vertex dapat sebagai root dari tree dengan 0 atau lebih subtree.

Pada gambar 2.1.d, a sebagai root dengan 3 subtree. Masing-masing subtree tersebut mempunyai :

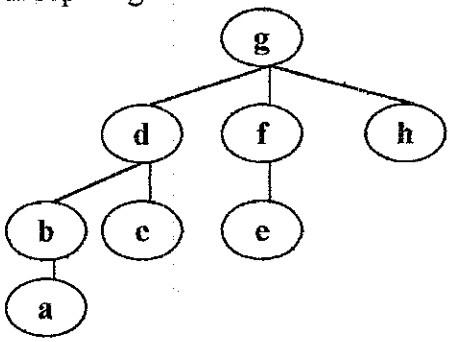
- Root b dengan 2 subtree berkara pada e dan f
- Root c dengan 0 subtree (tanpa subtree)

- Root d dengan 1 subtree berakar pada g

Contoh suatu tree dalam kehidupan sehari-hari yang dapat disajikan dalam bentuk tree adalah sebagai berikut : Dalam suatu pertandingan bola voli antar desa diikuti oleh 8 regu yaitu a,b,c,d,e,f,g dan h. Ketentuan dalam pertandingan adalah dengan menggunakan sistem gugur. Apabila disajikan dengan tree, regu-regu digambarkan dengan vertex, sedangkan edge menyajikan regu yang satu bertanding dengan regu yang lain. Setelah pertandingan berakhir, maka didapat hasil seperti berikut ini :

a. a melawan b dimenangkan oleh b
b. c melawan d dimenangkan oleh d
c. e melawan f dimenangkan oleh f
d. g melawan h dimenangkan oleh g
e. b melawan d dimenangkan oleh d
f. f melawan g dimenangkan oleh g
g. d melawan g dimenangkan oleh g

Sistem pertandingan bola voli antar desa di atas, apabila disajikan sebagai tree, maka akan menjadi seperti gambar berikut ini :



Gambar 2.1.e. Tree dari sistem pertandingan bola voli antar desa

Definisi 2.1.12 :

Degree (derajat) dari suatu vertex v dalam graph G, yang dinotasikan dengan deg v adalah jumlah dari garis (edge) yang incident dengan v.

Berikut adalah contoh degree vertex dari graph G pada gambar 2.1.a :

deg a = 3 deg c = 2 deg e = 2 deg g = 2

deg b = 3 deg d = 3 deg f = 2 deg h = 1

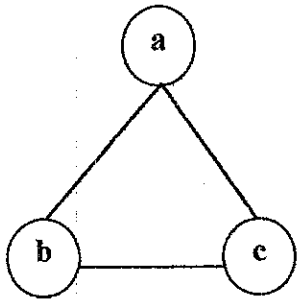
Definisi 2.1.13 :

Leaf suatu tree adalah sebuah vertex tanpa subtree atau sebuah vertex yang mempunyai degree satu . Bila dilihat pada gambar 2.1.a, maka dapat diketahui bahwa vertex h merupakan contoh dari suatu leaf.

Definisi 2.1.14 :

Complete graph atau graph komplet K_p adalah graph yang mempunyai p pasang vertex yang adjacent. Dan K_p mempunyai edges (garis) sejumlah $\binom{p}{2}$ dan setiap vertex mempunyai degree sejumlah p-1.

Berikut ini adalah contoh dari complete graph K_p dengan p=3 :



Gambar 2.1.f. Complete Graph

Teorema 2.1.15 :

Pernyataan berikut adalah ekuivalen untuk suatu graph G dengan n vertex dan q edges (garis) yaitu:

(1) G adalah Tree.

(2) Setiap dua vertex dari G terhubung oleh suatu lintasan yang tunggal

(3) G terhubung dan n = q+1

(4) G tidak mempunyai cycle dan n = q+1

(5) G tidak mempunyai cycle dan jika sembarang dua vertex yang tidak adjacent dari G terhubung oleh edge (garis) x maka G+x mempunyai tepat satu cycle.

(6) G terhubung dan bukan merupakan K_p untuk $p \geq 3$ dan jika sembarang 2 vertex yang tidak adjacent dari G terhubung oleh edge (garis) x maka G+x mempunyai tepat satu cycle.

(7) G bukan merupakan $K_3 \cup K_1$ atau $K_3 \cup K_2$, n=q+1 dan jika sembarang dua vertex yang tidak adjacent dari G terhubung oleh edge (garis) x maka G+x mempunyai tepat satu cycle.

Bukti :

(1)⇒(2) karena G terhubung, maka setiap dua vertex dari G terhubung oleh suatu lintasan. Misalkan u dan v adalah dua vertex dalam G dan jika P_1 dan P_2 adalah dua buah lintasan yang menghubungkan vertex u dan v, serta jika w adalah suatu vertex pertama pada P_1 (jika melintasai P_1 dari u ke v) sedemikan

schingga w berada di P_1 dan P_2, tetapi hanya sebagai pengganti pada P_1 dan bukan pada P_2. Jika dilanjutkan dengan w` yaitu vertex berikutnya pada $P_1$ yang juga berada pada $P_2$, maka bagian S dari $P_1$ dan $P_2$ adalah antara w dan w`yang bersama-sama membentuk suatu cycle dalam G. Maka jika G adalah acyclic, maka ada lintasan tunggal yang menghubungkan setiap dua vertex.

(2)⇒(3) Karena G merupakan tree maka jelas bahwa G terhubung. Akan dibuktikan bahwa n=q+1 dengan induksi matematika.

- Untuk tree dengan jumlah vertex n=1, maka jumlah garis adalah q =n–1=1–1= 0
- Untuk tree dengan jumlah vertex n = k, maka jumlah garis adalah

  $q = n - 1 = k-1$
- Untuk tree dengan jumlah vertex n = k+1, maka

  $n - 1 = (k+1) - 1 = k \Leftrightarrow k > k - 1 = q \Leftrightarrow k > q$

jadi terbukti bahwa untuk suatu tree G, maka n = q+1.

Jika G mempunyai n vetex, maka penghapusan dari suatu edge (garis) dari G menyebabkan G tidak terhubung karena ketunggalan dari lintasan dan akan terbentuk tree baru yang mempunyai dua bagian. Dengan hipotesa induksi di atas, setiap bagian mempunyai vertex yang jumlahnya lebih satu dari jumlah edge (garis). Maka jumlah keseluruhan dari edge (garis) dalam G pasti akan n-1.

(3)⇒(4) Dianggap bahwa G mempunyai cycle dengan panjang p, maka ada p edge (garis) pada cycle dan untuk setiap n-p vertex tidak berada dalam cycle, ada suatu garis yang terjadi pada geodesic ke vertex dari cycle. Setiap garis tersebut berbeda, maka $q \geq n$ yang merupakan suatu kontradiksi.

(4) $\Rightarrow$(5) karena G tidak mempunyai cycle, maka setiap bagian dari G adalah tree. Jika ada k bagian, maka masing-masing bagian mempunyai jumlah vertex lebih satu daripada edge (garis), n=q+1, jadi k=1 dan G terhubung. Maka G adalah tree dan mempunyai satu lintasan tunggal yang menghubungkan antara dua vertex. Jika ditambahkan garis uv ke G (dimana u dan v merupakan vertex dalam G), maka garis tersebut bersama dengan lintasan tunggal dalam G yang menghubungkan u dan v membentuk suatu cycle. Cycle tersebut tunggal karena lintasan tersebut juga tunggal.

(5) $\Rightarrow$(6) karena setiap K_p untuk $p \geq 3$ mengandung suatu cycle, G tidak dapat merupakan salah satu bagian dari K_p. Graph G harus terhubung,untuk sebaliknya garis x dapat ditambahkan sehingga menghubungkan dua vertex dalam bagian yang berbeda dari G, dan G+x akan menjadi acyclic.

(6) $\Rightarrow$ (7) akan dibuktikan bahwa setiap dua vertex dari G terhubung oleh suatu lintasan yang tunggal, dan karena (2) $\Rightarrow$ (3), maka n=q+1. tentu saja setiap dua vertex dari G terhubung oleh beberapa lintasan. Jika setiap dua vertex dari G trehubung oleh dua lintasan, maka dengan bukti (1) $\Rightarrow$(2), G mempunyai cycle. Cycle tersebut tidak dapat mempunyai empat atau lebih vertex maka dapat dihasilkan lebih dari satu cycle dalam G+x dengan mengambil x yang menghubungkan dua vertex yang tidak adjacent pada cycle (jikatidak ada vertex yang tidak adjacent pada cycle, maka G sendiri mengandung lebih dari satu cycle). Jadi cycle adalah K_3 yang merupakan subgraph dari G, dimana dengan hipotesis G tidak complete dengan p≥3. Karena G terhubung, dapat dianggap ada vertex lain dalam G yang terhubung ke vertex dari K_3. Maka hal ini jelas bahwa

jika sembarang garis dapat ditambahkan, maka satu garis mungkin ditambahkan untuk membentuk sedikitnya dua cycle dalam G+x. Jika tidak ada garis lain yang mungkin ditambahkan sedemikian sehingga kondisi kedua dari G terpenuhi, maka G adalah K_p dengan $p \geq 3$ bertentangan dengan hipotesis.

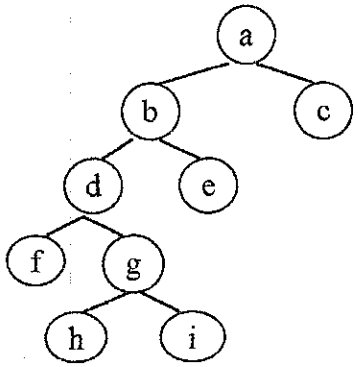
(7) $\Rightarrow$(1) Jika G mempunyai cycle, cycle tersebut pasti suatu segitigayang merupakan bagian dari G. Bagian ini mempunyai tiga vertex dan tiga garis. Seluruh bagian yang lain dari G pasti merupakan tree dan agar $n \pm q+1$, hanya dapat ada satu bagian yang lain dari G. Jika tree ini mengandung path dengan panjang 2, akan sangat mungkin untuk menambah garis x ke G dan mendapatkan dua cycle dalam G+x. Maka tree tersebut pasti salahsatu dari K_1 atau K_2. jadi G pasti merupakan $K_3 \cup K_1$ atau $K_3 \cup K_2$. Maka G adalah acyclic. Tetapi jika G acyclic dan $n=q+1$, maka G adalah tree dan teorema telah terbukti.

2.2 BINARY TREE

Definisi 2.2.1 :

Suatu binary tree adalah tree dengan banyak vertex lebih dari 2 dan mempunyai tepat 1 vertex dengan degree 2, sedangkan yang lain dengan degree 1 atau 3. vertex dengan degree 2 disebut akar/root dari binary tree tersebut.

Contoh binary tree :



Gambar 2.2.a. Binary Tree

Binary tree pada gambar 2.2.a mempunyai 9 vertex dan vertex a merupakan root. Subtree kiri berakar di b dan subtree kanan berakar di c. Hal ini ditunjukkan dari branch (cabang) yang berasal dari a ke b di sebelah kiri dan dari a ke c di sebelah kanan. Tidak adanya branch menunjukkan subtree yang kosong (tidak ada subtree). Subtree dari binary tree yang berakar di c adalah kosong.

Definisi 2.2.2 :

Jika v adalah root dari binary tree dan w adalah root kiri atau kanan dari subtree, maka v disebut father dari w.

Contoh yang dapat diambil dari gambar 2.2.a, yaitu bahwa vertex a merupakan father dari vertex b dan vertex c. Vertex b merupakan root kiri, karena merupakan root dari subtree kiri yang berakar di b dan vertex c merupakan root kanan, karena merupakan root dari subtree kanan yang berakar di c.

Definisi 2.2.3 :

Jika v merupakan father dari w maka w disebut son kiri atau son kanan dari v. Contoh yang dapat diambil dari gambar 2.2.a, yaitu bahwa vertex a merupakan father dari vertex b dan vertex c (sesuai dengan definisi 2.2.2), maka vertex b merupakan son kiri dari vertex a dan vertex c merupakan son kanan dari vertex a.

Definisi 2.2.4 :

Jika $v_0, v_1, \ldots, v_n$ adalah barisan vertex dan untuk v_i ($i = 0,1,2,\ldots,n$) adalah father dari v_{i+1}, maka v_0 disebut ancestor dari v_n dan v_n adalah descendent dari v_0.

Gambar 2.2.a menunjukkan bahwa vertex a merupakan ancestor dari vertex c, dan vertex c merupakan descendent dari vertex a.

Definisi 2.2.5 :

Jika v_1 adalah ancestor (pendahulu) dari v_2, maka v_2 adalah descendent (turunan) dari v_1. Vertex v_2 adalah descendent kiri dari vertex v_1, jika v_2 son kiri dari v_1 atau descendent dari son kiri vertex v_1. Gambar 2.2.a menunjukkan bahwa vertex a merupakan ancestor dari vertex c, dan oleh karena vertex c merupakan son kanan dari vertex a, maka vertex c adalah descendent kanan dari vertex a.

Pengertian descendent kanan analog dengan pengertian descendent kiri.

Definisi 2.2.6 :

Depth dari suatu vertex adalah jumlah dari ancestor (pendahulu) dari vertex tersebut. Bila diambil contoh dari gambar 2.2.a, maka dapat diketahui bahwa vertex c mempunyai depth 1, karena hanya mempunyai ancestor yaitu 1 (vertex a).

Definisi 2.2.7 :

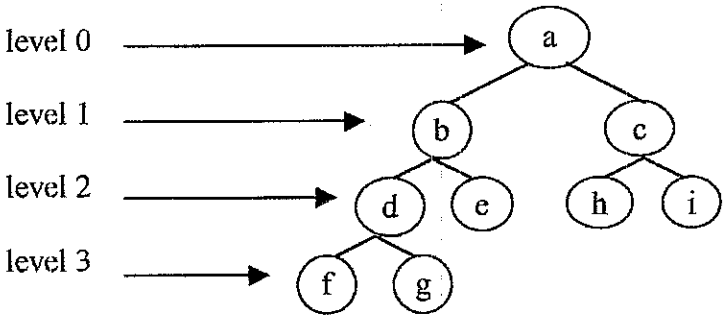
Himpunan semua vertex dari suatu tree dari depth d membentuk level d. Vertex b dan vertex c pada gambar 2.2.a mempunyai depth 1, maka himpunan vertex b dan vertex c membentuk level 1.

Definisi 2.2.8 :

Tinggi dari suatu tree adalah depth maksimal dari vertex-vertex yang ada dalam tree tersebut.

Definisi 2.2.9 :

Vertex interior adalah suatu vertex dari suatu tree yang mempunyai son. Contoh gambar 2.2.b yaitu binary tree yang mempunyai 9 vertex, 4 vertex interior (a, b, c, d) dan 5 leaf serta mempunyai tinggi 3 :

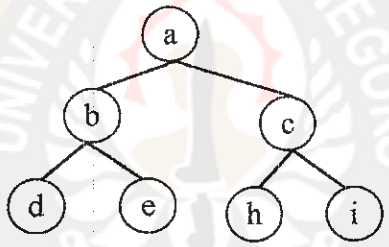


Gambar 2.2.b. Binary tree dengan level 3

Definisi 2.2.10 :

Suatu binary tree disebut full binary tree jika semua leaf berada pada depth yang sama dan setiap vertex interior mempunyai dua son yaitu son kiri dan son kanan.

Contoh dari suatu full binary tree dengan tinggi 2 seperti pada gambar di bawah ini :



Gambar 2.2.c. Full binary tree

Teorema 2.2.11 :

Full binary tree dengan tinggi h mempunyai $2^{h+1} - 1$ vertex yaitu $2^h - 1$ vertex interior dan 2^h leaf.

Bukti :

Full binary tree dengan tinggi h = 0 adalah suatu vertex leaf tunggal dan mempunyai $2^{h-1} = 2^0 - 1 = 0$ vertex interior dan mempunyai $2^h = 2^0 = 1$ leaf. Secara umum, dianggap bahwa teorema benar untuk semua full binary tree dengan tinggi h dimana h > 0. Maka dianggap suatui full binary tree dengan tinggi

h. kedua sub tree dari root mempunya tinggi h-1, sehingga dapat dirumuskan suatu bentuk seperti di bawah ini:

$n_L = 2^{h-1} - 1$, n_L adalah jumlah vertex interior pada subtree kiri.

$n_R = 2^{h-1} - 1$, n_R adalah jumlah vertex interior pada subtree kanan.

$l_L = 2^{h-1}$, l_L adalah jumlah leaf dari subtree kiri.

$l_R = 2^{h-1}$, l_R adalah jumlah leaf dari subtree kanan.

Maka $n = n_L + n_R + 1$

$n = (2^{h-1} - 1) + (2^{h-1} - 1) + 1 = 2.\ 2^{h-1} - 1 = 2^h - 1$

$l = l_R + l_L$

$l = 2^{h-1} + 2^{h-1}$

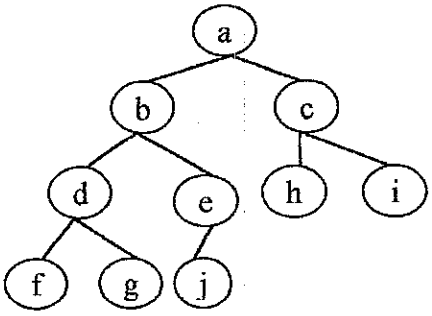
$l = 2.\ 2^{h-1}$

$l = 2^h$

jadi terbukti bahwa full binary tree dengan tinggi h mempunyai 2^h -1 vertex interior dan 2^h leaf.

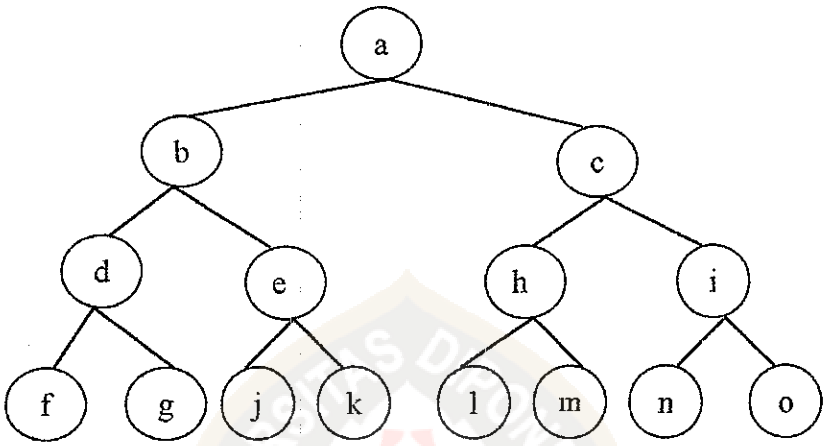
Definisi 2.2.12 :

Complete binary tree adalah suatu tree yang dapat dibentuk menjadi suatu full binary tree dengan cara penambahan leaf secara tepat pada level yang paling bawah. Contoh dari complete binary tree adalah seperti gambar di bawah ini :



Gambar 2.2.d. Complete binary tree

Dari gambar 2.2.d di atas, tree dapat dibentuk menjadi full binary tree dengan tinggi 3 dengan menambahkan 5 leaf secara tepat (masing-masing vertex k,l,m,n,o) pada level 3, sehingga menjadi seperti pada gambar di bawah ini.



Gambar 2.2.e. Full binary tree yang terbentuk setelah penambahan 5 leaf pada level 3

2.3 Bahasa Pemrograman C ++

❖ **Fungsi main ( )**

Pada program C++, main ( ) merupakan fungsi yang istimewa karena fungsi main ( ) harus ada pada program, sebab fungsi inilah yang menjadi titik awal dan titik akhir eksekusi program.

Tanda { diawal fungsi menyatakan awal tubuh fungsi dan sekaligus awal eksekusi program, sedangkan tanda } diakhir fungsi merupakan akhir tubuh fungsi dan sekaligus akhir dari eksekusi program.

Jika program terdiri lebih dari satu fungsi, fungsi main ( ) biasa ditempatkan pada posisi yang paling atas dalam pendefinisian fungsi.

❖ **Pengenalan Prosesor #include**

#include merupakan salah satu jenis pengarah praprosesor yang dipakai untuk membaca file yang dinamakan file-judul (header file) yaitu file yang diantaranya berisi deklarasi fungsi dan definisi konstanta.

File-file ini mempunyai ciri yaitu namanya diakhiri dengan akhiran **.h**. Bentuk umum dari #include adalah sebagai berikut :

#include <namafile>

atau **#include "namafile"**

Bentuk pertama mengisyaratkan bahwa pencarian file dilakukan pada direktori khusus (direktori file include) sedangkan bentuk kedua menyatakan bahwa pencarian file dilakukan pertama kali pada direktori aktif tempat program dimulai dan seandainya tidak ditemukan maka pencarian akan dilanjutkan pada direktori lainnya yang sesuai dengan perintah pada sistem operasi.

❖ **Fungsi printf( )**

Fungsi printf( ) merupakan fungsi umum yang dipakai untuk menampilkan suatu keluaran pada layar peraga. Misalkan untuk menampilkan tulisan :

Selamat datang

maka contoh pernyataan yang diperlukan berupa :

printf("Selamat datang");

pernyataan di atas adalah berupa pemanggilan fungsi printf( ) dengan argumen/ parameter berupa string "Selamat datang".

❖ **Konstanta**

Konstanta adalah nilai fixed atau nilai yang tidak dapat diubah pada waktu pengeksekusian program. Konstanta ini lain dengan konstanta variabel, sebab kalau konstanta variabel adalah variabel bukan fixed value. Konstanta dideklarasikan kalau diperlukan suatu variabel yang isinya sudah fixed dan tidak akan diganti-ganti atau tidak dapat berubah.

Syntax dari konstanta adalah sebagai berikut :

Const tipe-data nama_variabel = value;

Value adalah nilai yang fixed atau tidak dapat diganti, contoh :

Const int value = 100;

Const float pi = 3,141;

Konstanta juga dapat dipergunakan untuk kalkulasi dan hal-hal lainnya.

❖ **Operator**

- Operator arithmetic

Operator arithmetic adalah simbol yang menunjukkan suatu fungsi arithmetic. Listing dari operator arithmetic adalah sebagai berikut :

Operator	Arti
+	Tambah
-	Minus
*	Perkalian
/	Pembagian
%	Modus

Disamping operator arithmetic ada juga yang dinamakan operator unary yang merupakan perkembangan dari operator arithmetic. Listing dari operator unary adalah sebagai berikut :

Operator Unary	Arti
+=	Tambah
- =	Minus
*=	Perkalian
/=	Pembagian

- Operator Logika (logical operators)

Didalam C++ juga terdapat logical operators, seperti AND, OR, NOT. Tetapi logical operators ini bukan memakai kata melainkan mamakai simbol. Listing dari logical operators adalah sebagai berikut :

Logical operators	Arti
||	OR
&&	AND
!	NOT

- Operator Relational

Operator relational adalah operator untuk perbandingan dan berbagai jenis dari operator relational adalah sebagai berikut :

Operator Relational	Arti
>	Lebih besar
>=	Lebih besar atau sama dengan
<	Lebih kecil
<=	Lebih kecil atau sama dengan
==	Sama dengan
!=	Tidak sama dengan
=	Sama dengan

Perbedaan penggunaan '==' dengan '=' adalah kalau '=' adalah assignment untuk mengisi sebuah variabel dengan sebuah konstanta dan '==' adalah untuk persamaan.

- Operator Bit (Bits Operators)

Operator bit adalah operator-operator yang digunakan untuk memanipulasi bit-bit dari nilai data yang ada di memori. Bits adalah binary kode, dan binary value adalah 0 dan 1. Di dalam C++ juga terdapat fasilitas untuk bekerja dengan bits. Listing dari operator bit adalah sebagai berikut :

Bit Operator	Arti
&	AND adalah menghasilkan 1 jika kedua bit yang di AND tersebut 1
|	OR adalah menghasilkan 1 jika salah satu bit yang di OR tersebut 1
^	XOR adalah menghasilkan 1 jika salah satu bit yang di XOR tersebut 1
>>	Geser kanan sebanyak yang diberitahu
<<	Geser kiri sebanyak yang diberitahu

❖ **C++ Streams**

Ada cara untuk input/output yang baru dalam object-oriented programming yaitu input/output itu disebut dengan streams. Sebetulnya I/O tersebut terdiri dari class-class yang sudah dideklarasikan dari pembuatnya. I/O functions ini sangat flexibel dan mudah pemahamannya. Stream tersebut fungsinya sama dengan printf( ) dan fungsi lainnya. Ada 4 predefined stream objects yaitu :

- Cin : standard input (sama seperti stdin)
- Cout : standard output (sama seperti stdout)
- Cerr : standard error (sama seperti stderr)
- Clog : buffered versi dari cerr

Semua fungsi I/O tersebut dideklarasikan di sebuah header file disebut iostream library. Untuk mengakses fungsi I/O tersebut maka ditambahkan pada awal program , misalnya #include <iostream.h>.

2.3.1 Pernyataan if dan if...else

- Pernyataan if

Pernyataan if mempunyai bentuk umum : **If (kondisi)**

Penyataan

Arti dari perintah if, jika kondisi bernilai benar, maka pernyataan dikerjakan. Dengan kata lain pernyataan tidak dijalankan bila kondisi bernilai salah. Kondisi dapat berupa sembarang ungkapan sedangkan pernyataan dapat berupa pernyataan tunggal, majemuk, atau kosong.

- Pernyataan if…else

  Pernyataan if…else memilik bentuk umum sebagai berikut :

  If (kondisi)

  Pernyataan-1

  Else

  Pernyataan-2

Arti dari pernyataan if..else adalah jika kondisi benar, maka pernyataan-1 dijalankan dan bila kondisi salah, maka penyataan-2 dijalankan.

2.3.2 Arrays, Character and String

Dalam menangani string atau character, sangat fleksibel. String sebetulnya adalah suatu deretan dari character. Untuk memberi suatu deretan kata ke dalam variabel string tersebut kita harus menggunakan fungsi strcpy atau yang lainnya dimana fungsi-fungsi tersebut sudah tersedia dalam kompilernya sendiri dan terdapat pada file string.h .

Array adalah suatu deretan memory yang telah dialokasikan untuk data item atau structure, misal dipunyai sebuah array dimana tipe datanya adalah int,

ini berarti bahwa variabel array tersebut akan menyimpan deretan angka integer dalam memory. Syntax untuk pendeklarasian sebuah array adalah sebagai berikut:

Tipe data nama_variabel[jumlah];

Tipe data adalah data types (int, float, dan lainnya), dan jumlah adalah jumlah memory yang diinginkan, misal contohnya : int coba [10]

2.3.3 Loop

Dalam C++ juga ada instruksi untuk mengulang sebuah atau beberapa instruksi sampai kondisi tertentu terpenuhi, misal akan mencetak kata "hello,world" sebanyak 10 kali, maka kalau tidak ada instruksi pengulangan berarti harus mengetik kata print 10 kali. Terdapat 2 statemen dalam C++ untuk membuat looping yaitu :

- While

Pernyataan while merupakan suatu bentuk loop berdasarkan suatu tes kondisi. Syntax dari while adalah sebagai berikut :

```
While (condition testing )
 {
      < instruksi>
 }
```

condition testing menggunakan relational operations.

- For

Sebuah instruksi pengulangan juga seperti while, tetapi bedanya for ini akan mengeksekusi instruksi sebanyak yang diberikan oleh programmer itu sendiri.

Syntax dari for loop adalah sebagai berikut :

For (var = constanta; var relational_operator constanta ; ekspresi);

Var adalah variabel yang digunakan dan konstanta adalah nilai awal untuk variabel var. Ekspresi adalah bagaimana ekspresi untuk memanipulasi variabel var (ditambah, dikali, dan lainnya).

2.3.4 Procedure dan Functions

Untuk membuat program yang serba terstruktur, harus dapat memecah program menjadi procedure dan function

- **Procedure**

Procedure adalah sekumpulan instruksi dari program kapan saja diperlukan. Procedure juga dapat dipanggil untuk memanipulasi variabel. Untuk dapat membuat sebuah procedure, sebaiknya nama procedure beserta parameter-parameternya dideklarasikan sebelum fungsi main( ).

Syntax dari procedure adalah sebagai berikut :

Void nama_procedure (type var_nama, type var_nama1,…);

Void adalah reserve words untuk memberitahu bahwa ini adalah sebuah procedure, type adalah tipe data untuk variabel yang dikirim.

- **Functions**

Procedure adalah subroutine yang tidak mengembalikan konstanta maupun apapun, tetapi beda dengan functions, functions mengembalikan konstanta. Pendeklarasian functions sama dengan procedure, perbedaannya hanya pada void, yaitu kalau di function dideklarasikan bukan void melainkan tipe data structure yang ingin dikembalikan.