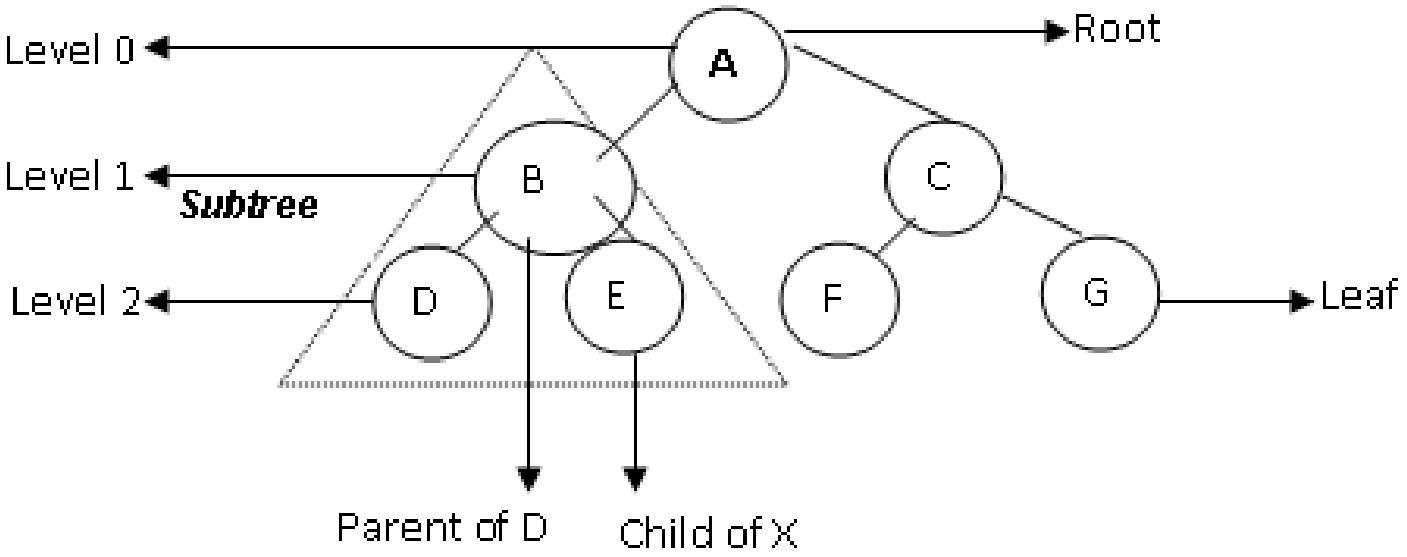
TREE

Struktur Data

TREE

- Merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen.
- Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lainnya (disebut subtree).
- Contoh penggunaan struktur Tree :
- Silsilah keluarga
- Hasil pertandingan yang berbentuk turnamaen
- Struktur organisasi dari sebuah perusahaan

TREE ANATOMY



TERMINOLOGI DALAM TREE

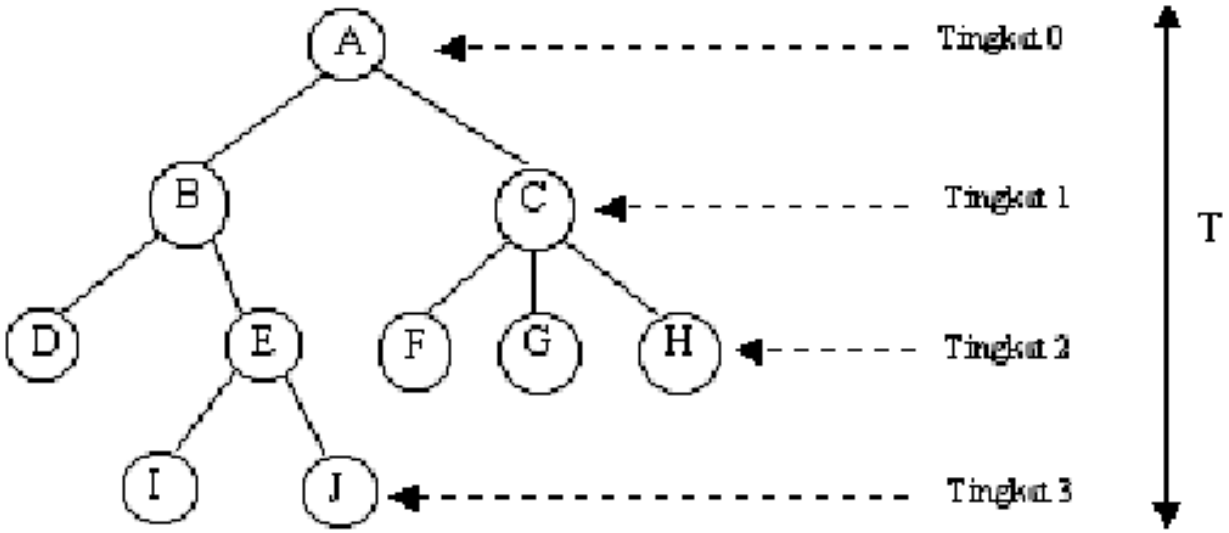
- **Istilah-istilah umum dalam Tree**

Prodecessor	: node yang berada diatas node tertentu.
Successor	: node yang berada di bawah node tertentu.
Ancestor	: seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.
Descendant	: seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.
Parent	: predecssor satu level di atas suatu node.
Child	: successor satu level di bawah suatu node.
Sibling	: node-node yang memiliki parent yang sama dengan suatu node.
Subtree	: bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
Size	: banyaknya node dalam suatu tree.
Height	: banyaknya tingkatan/level dalam suatu tree.
Root	: satu-satunya node khusus dalam tree yang tak punya predecssor.
Leaf	: node-node dalam tree yang tak memiliki seccessor.
Degree	: banyaknya child yang dimiliki suatu node

- Ascestor (F) = C,A
- Descendant (C) = F,G
- Parent (D) = B
- Child (A) = B,C
- Sibling (F) = G
- Size = 7
- Height = 3
- Root = A
- Leaf = D,E,F,G
- Degree (C) = 2

REPRESENTASI TREE

Representasi Tree



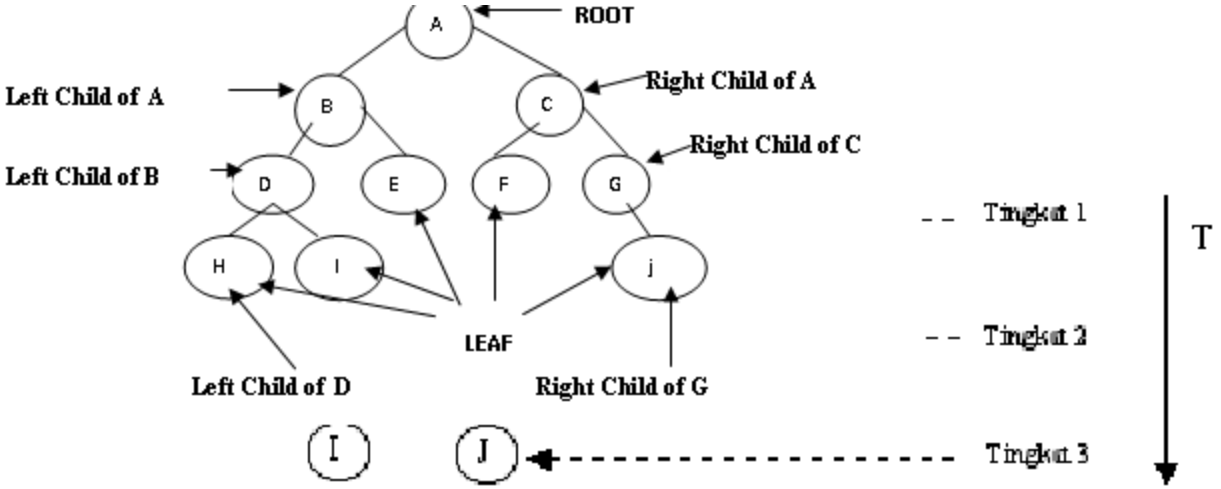
Gambar 6.1. Contoh sebuah Pohon beserta Tingkatnya

JENIS TREE

- 1. Binary Tree

  Binary tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah.

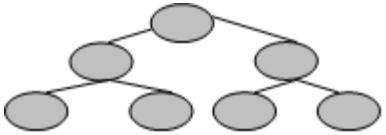
  Sesuai dengan definisi tersebut, maka tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.



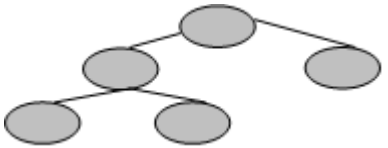
Gambar 6.1. Contoh sebuah Pohon beserta Tingkatnya

JENIS-JENIS BINARY TREE

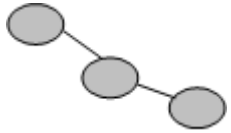
- Full Binary Tree
  - Binary Tree yang tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.



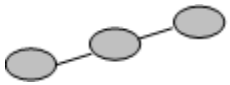
- Complete Binary Tree
  - Mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda. Node kecuali leaf memiliki 0 atau 2 child.



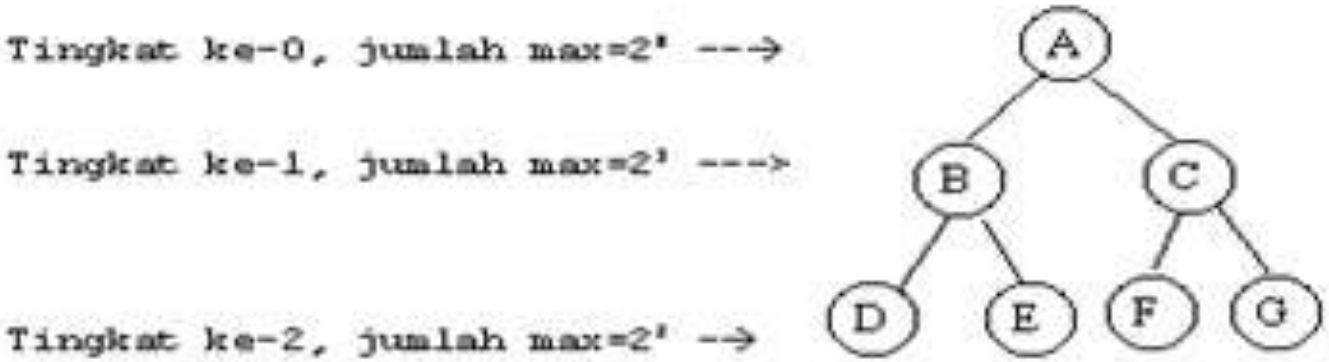
- Skewed Binary Tree
  - Yakni Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.



atau



Note : Jumlah Maksimum Node pada setiap tingkat adalah 2^n



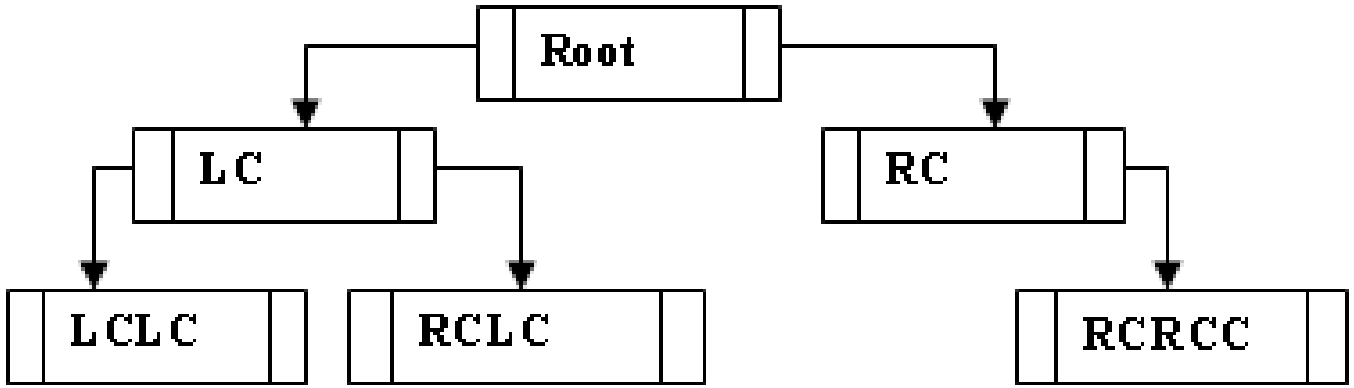
Gambar 6.4. Pohon Biner Tingkat 2 Lengkap

IMPLEMENTASI BINARY TREE

- Binary Tree dapat diimplementasikan dalam Pascal dengan menggunakan double Linked List. Untuk nodenya, bisa dideklarasikan sebagai berikut :

- Type Tree = ^node;
- Node= **record**
- Isi : TipeData;
- Left,Right : Tree;
- **end**;

- Contoh ilustrasi Tree yang disusun dengan double linked list :



- *(Ket: LC=Left Child; RC=Right Child)*

OPERASI-OPERASI PADA BINARY TREE

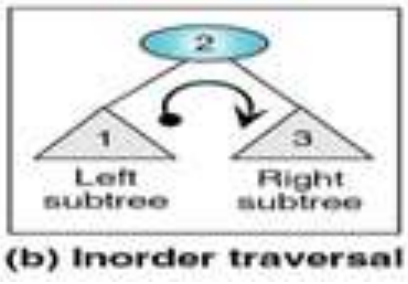
- Create : Membentuk binary tree baru yang masih kosong.
- Clear : Mengosongkan binary tree yang sudah ada.
- Empty : Function untuk memeriksa apakah binary tree masih kosong.
- Insert : Memasukkan sebuah node ke dalam tree. Ada tiga pilihan insert: sebagai root, left child, atau right child.
  Khusus insert sebagai root, tree harus dalam keadaan kosong.
- Find : Mencari root, parent, left child, atau right child dari suatu node. (Tree tak boleh kosong)
- Update : Mengubah isi dari node yang ditunjuk oleh pointer current. (Tree tidak boleh kosong)
- Retrieve : Mengetahui isi dari node yang ditunjuk pointer current. (Tree tidak boleh kosong)
- DeleteSub : Menghapus sebuah subtree (node beserta seluruh descendantnya) yang ditunjuk current. Tree tak boleh kosong. Setelah itu pointer current akan berpindah ke parent dari node yang dihapus.
- Characteristic : Mengetahui karakteristik dari suatu tree, yakni : size, height, serta average lengthnya. Tree tidak boleh kosong. (Average Length = [jumlahNodeLvl1*1+jmlNodeLvl2*2+…+jmlNodeLvln*n]/Size)
- Traverse : Mengunjungi seluruh node-node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linier yang tersimpan dalam tree. Ada tiga cara traverse : Pre Order, In Order, dan Post Order.

LANGKAH-LANGKAH TRAVERSE

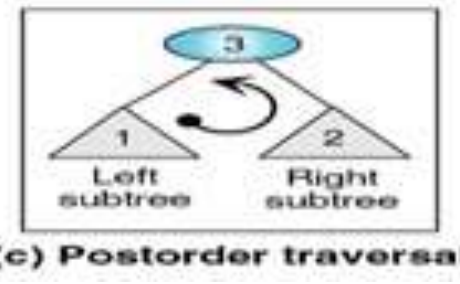
- **PreOrder** : Cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child.
- **InOrder** : Kunjungi Left Child, Cetak isi node yang dikunjungi, kunjungi Right Child
- **PostOrder**: Kunjungi Left Child, Kunjungi Right Child, cetak isi node yang dikunjungi

ILUSTRASI KUNJUNGAN

(a) Preorder traversal



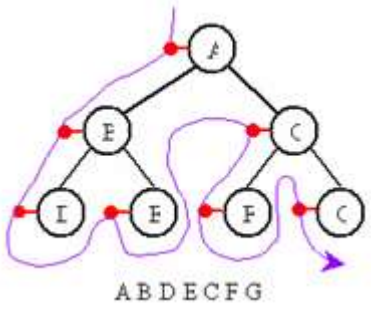
(b) Inorder traversal



(c) Postorder traversal

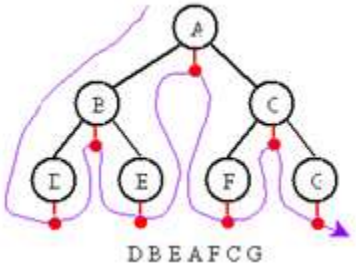
Contoh :

Preorder



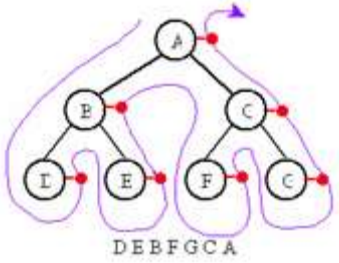
ABDECFG

inorder



DBEAFCG

postorder



DEBFGCA

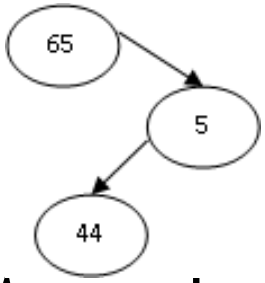
- Untuk lebih jelasnya perhatikan contoh operasi-operasi pada Binary Tree berikut ini :
- **Insert (Root,65)**

Memasukkan sebuah node ke dalam Tree yang masih kosong (Sebagai Tree)

- **Insert (RightChild,5)**

Menambahkan sebuah node sebagai right child dari Root.

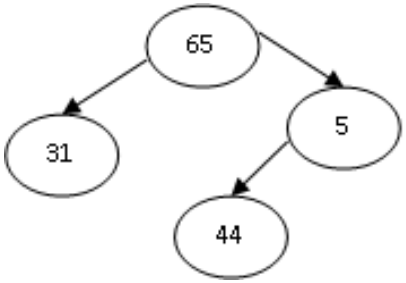
- **Insert (LeftChild,44)**



Menambahkan sebuah node sebagai left child dari node yang sebelunya di-insert.

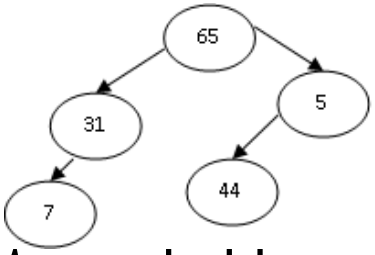
- **Find (Root)**

  Insert (LeftChild,31)



Memindahkan pointer ke Root kemudian menambahkan sebuah node sebagai left child dari root.

- **Insert (LeftChild,7)**

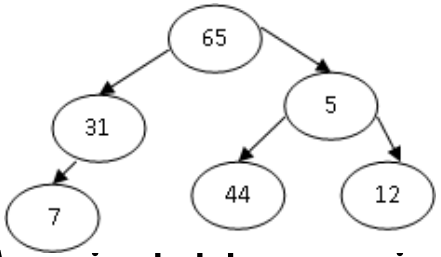


Menambahkan sebuah node sebagai left child dari node yang sebelumnya di-insert.

- **Find (Root)**

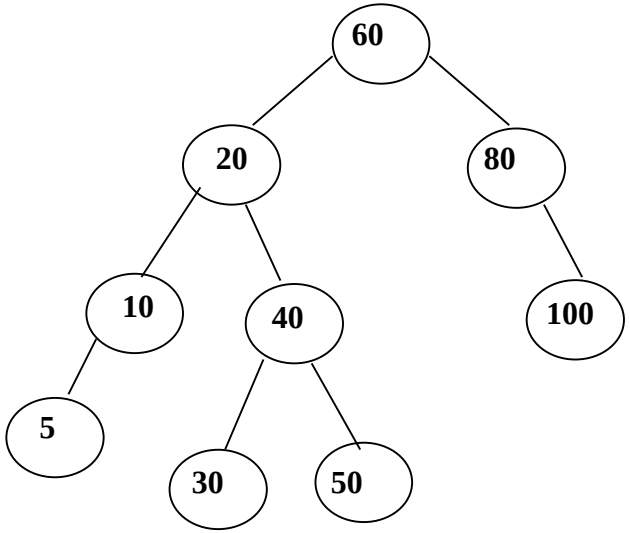
  Find (RightChild)

  Insert (RightChild,12)



Memindahkan pointer ke Root, kemudian pindahkan lagi pointer ke right child dari Root, kemudian masukkan sebuah node sebagai right child dari node yang sedang ditunjuk oleh pointer

- Latihan
- 1. sebutkan istilah tree :Ascestor, descendant, parent, child, Sibling, size, height, root, leaf, degree pada pohon dibawah ini :



- 2 Buatlah binary tree dengan operasi :
  - Insert (root,18)
  - Insert (rightchild,23)
  - Insert (leftchild, 21)
  - Find (root)
  - Insert (leftchild,10)
  - Insert (leftchild, 5)
  - Find (root)
  - Find(rightchild)
  - Insert(rightchlid, 33)
  - Insert (rightchlid,40)
  - Find (root)
  - Find (leftchild)
  - Insert(rightchild, 14)
  - Insert (rightchild,17)