



Tree

# Overview

- Pendahuluan
- Terminologi
- Binary Tree
  - Definisi
  - Deklarasi
  - Pembentukan Binary Tree
- Kunjungan : Metode Traversal
  - Preorder
  - Inorder
  - Postorder

# Pendahuluan

- Linked List, Stack, Queue merupakan struktur data yang bersifat linier
- Tree adalah struktur data tak linier yang memiliki sifat khusus.
- Tree biasanya digunakan untuk menggambarkan hubungan yang bersifat hirarkis antara elemen-elemen yang ada.

# Aplikasi

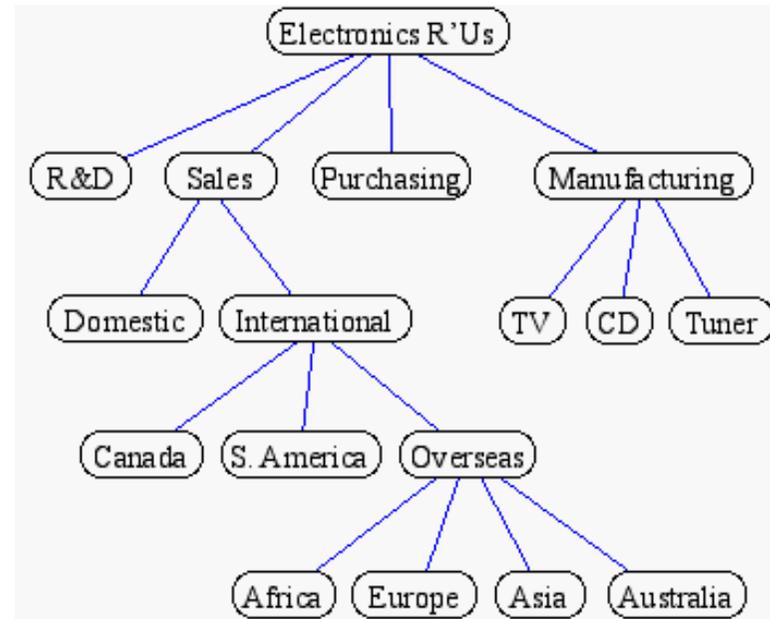
- Struktur organisasi
- Silsilah keluarga
- Sistem pakar
- Ekspresi Aritmatika
- Pertandingan dengan sistem gugur
- dll

# Contoh penggunaan Tree

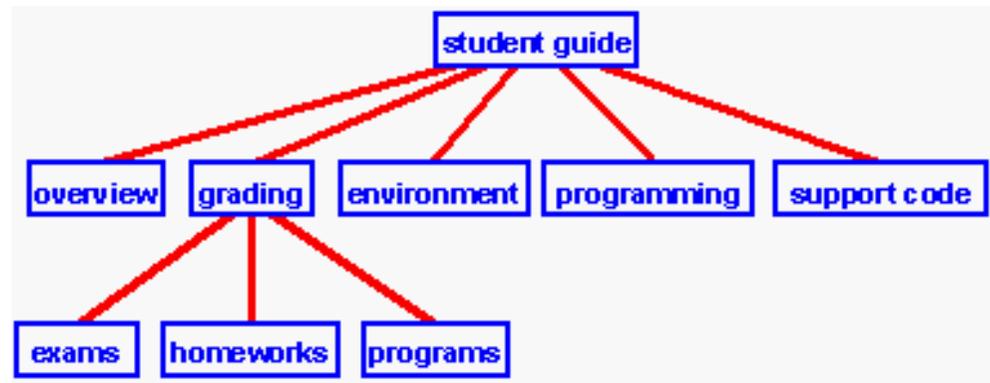
• sebuah **tree** merepresentasikan sebuah hirarki

**contoh:**

*struktur organisasi dari sebuah perusahaan*

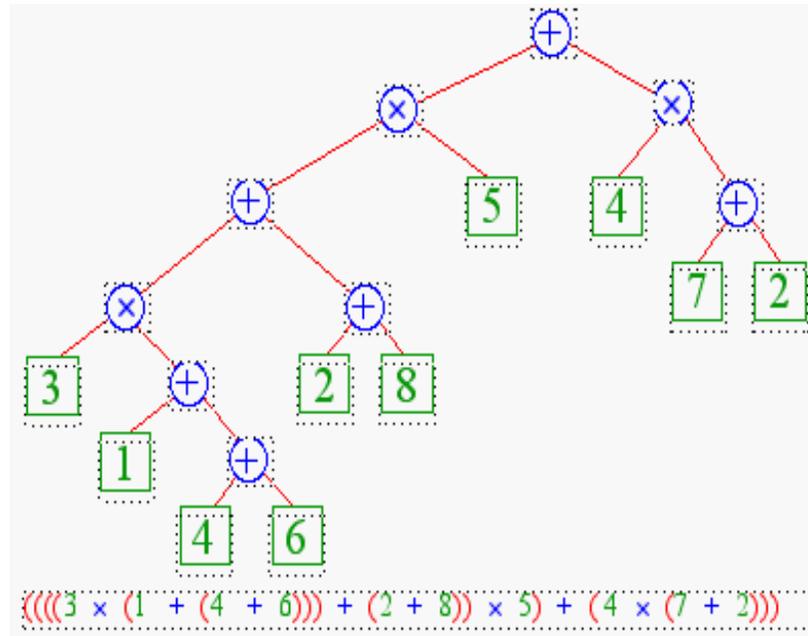


*diagram daftar isi dari sebuah buku*

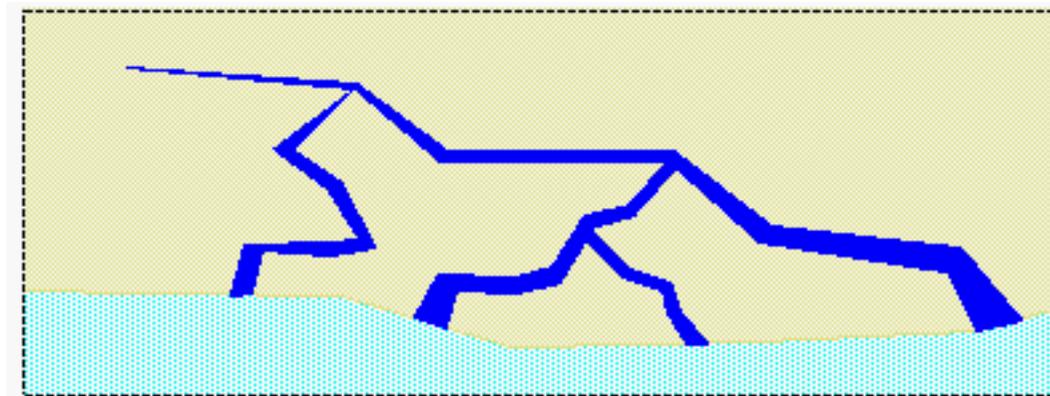


# Contoh penggunaan Tree

- Ekspresi aritmatika

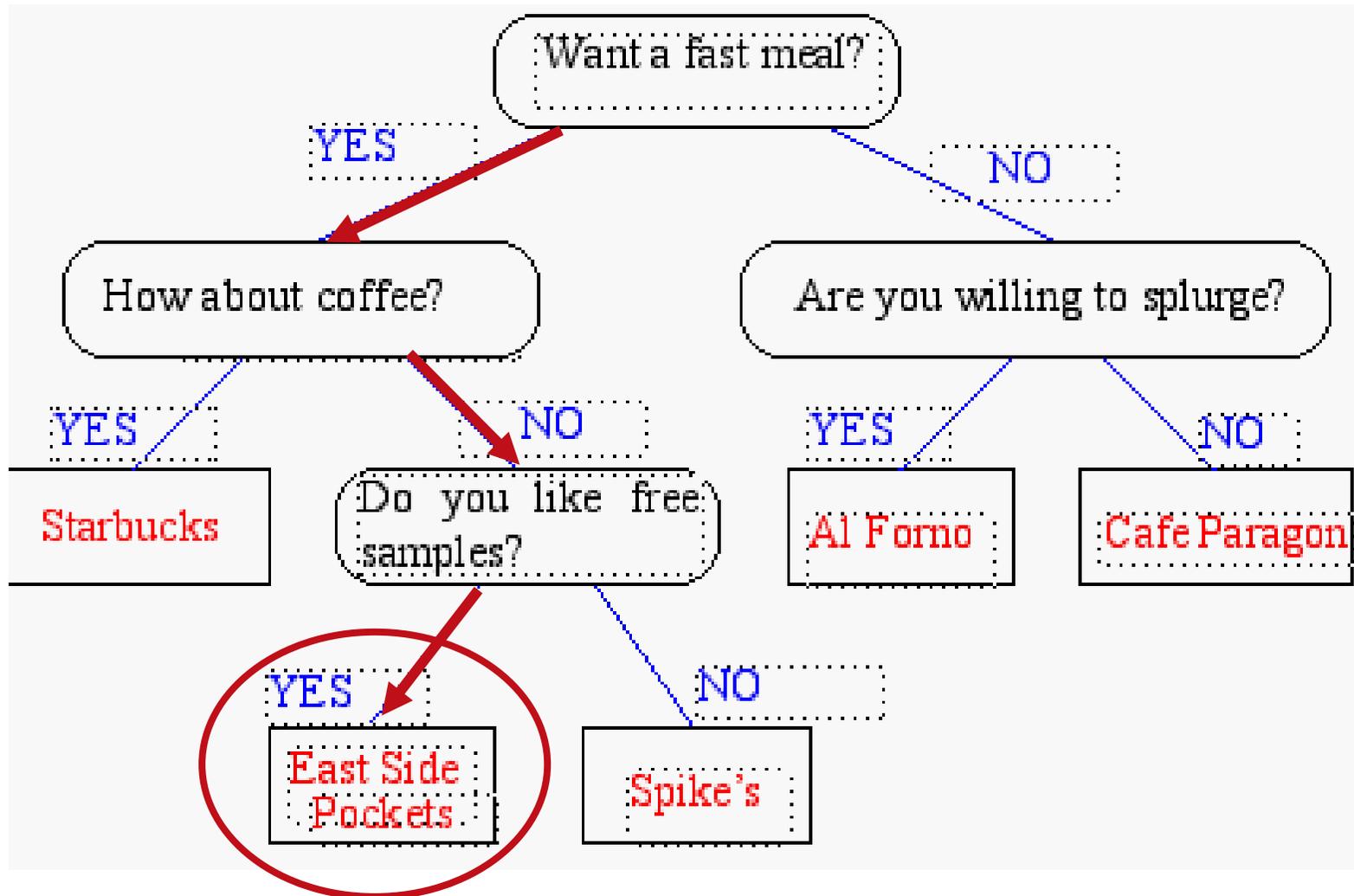


- sungai

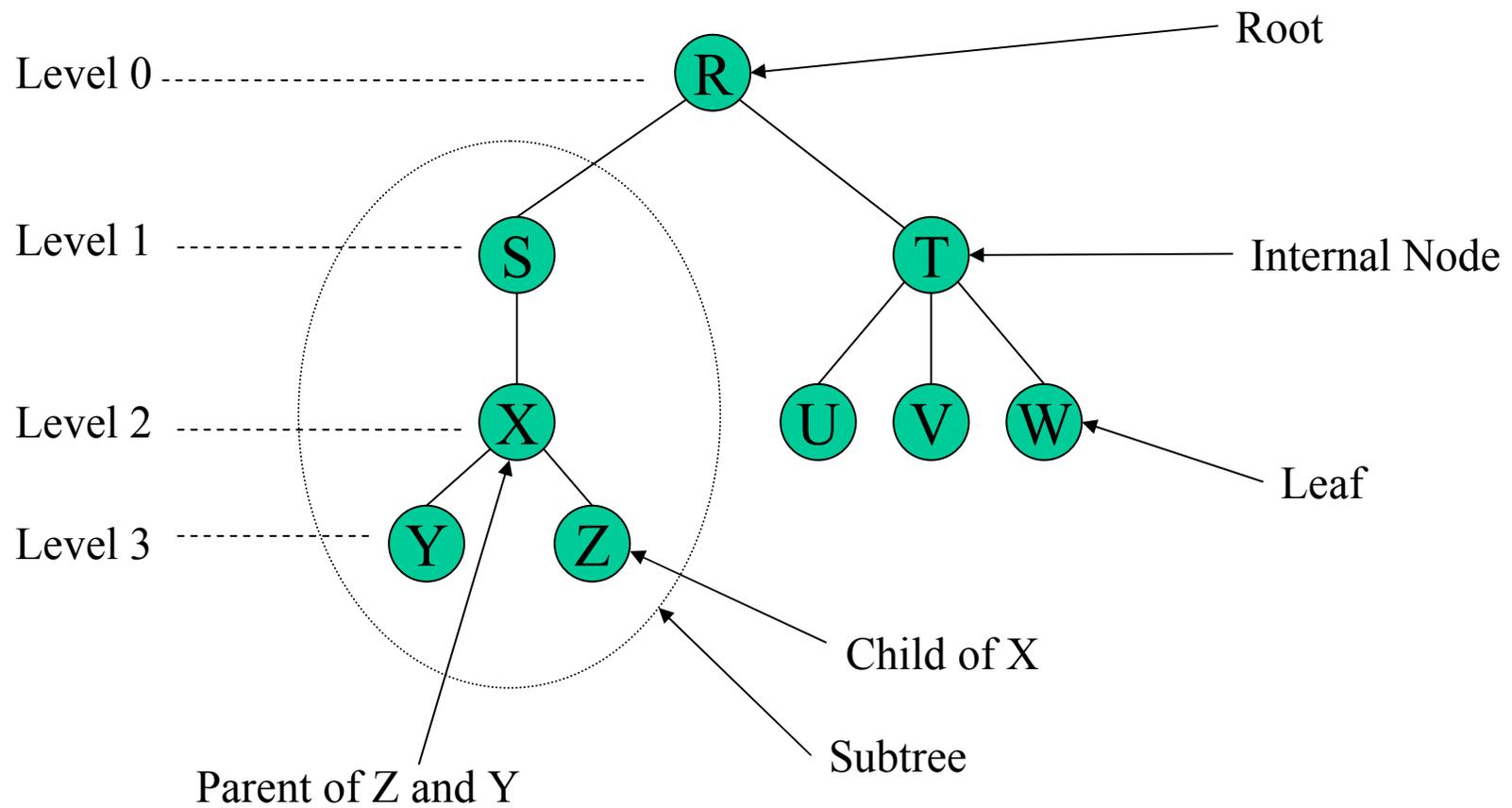


# Contoh penggunaan Tree

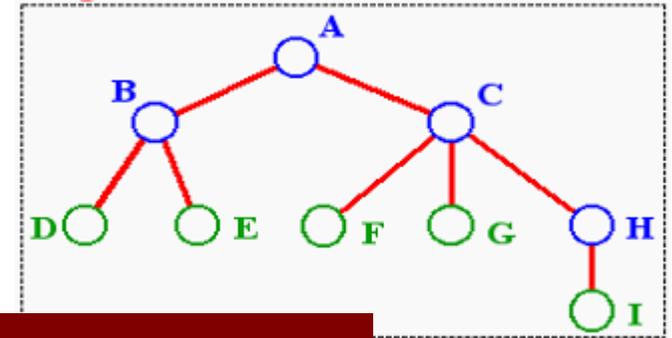
- decision trees → Sistem pakar



# Anatomi Tree



# Terminologi dalam Tree



<b>Term</b>	<b>Definition</b>
<b>Node</b>	Sebuah elemen dalam sebuah tree; berisi sebuah informasi
<b>Parent</b>	Node yang berada di atas node lain secara langsung; B adalah parent dari D dan E
<b>Child</b>	Cabang langsung dari sebuah node; D dan E merupakan children dari B
<b>Root</b>	Node teratas yang tidak punya parent
<b>Sibling</b>	Sebuah node lain yang memiliki parent yang sama; Sibling dari B adalah C karena memiliki parent yang sama yaitu A
<b>Leaf</b>	Sebuah node yang tidak memiliki children. D, E, F, G, I adalah leaf. Leaf biasa disebut sebagai <i>external node</i> , sedangkan node selainnya disebut sebagai <i>internal node</i> . B, A, C, H adalah <i>internal node</i>
<b>Level</b>	Semua node yang memiliki jarak yang sama dari root. A → level 0; B, C → level 1; D, E, F, G, H → level 2; I → level 3
<b>Depth</b>	Jumlah level yang ada dalam tree
<b>Complete</b>	Semua parent memiliki children yang penuh
<b>Balanced</b>	Semua subtree memiliki depth yang sama

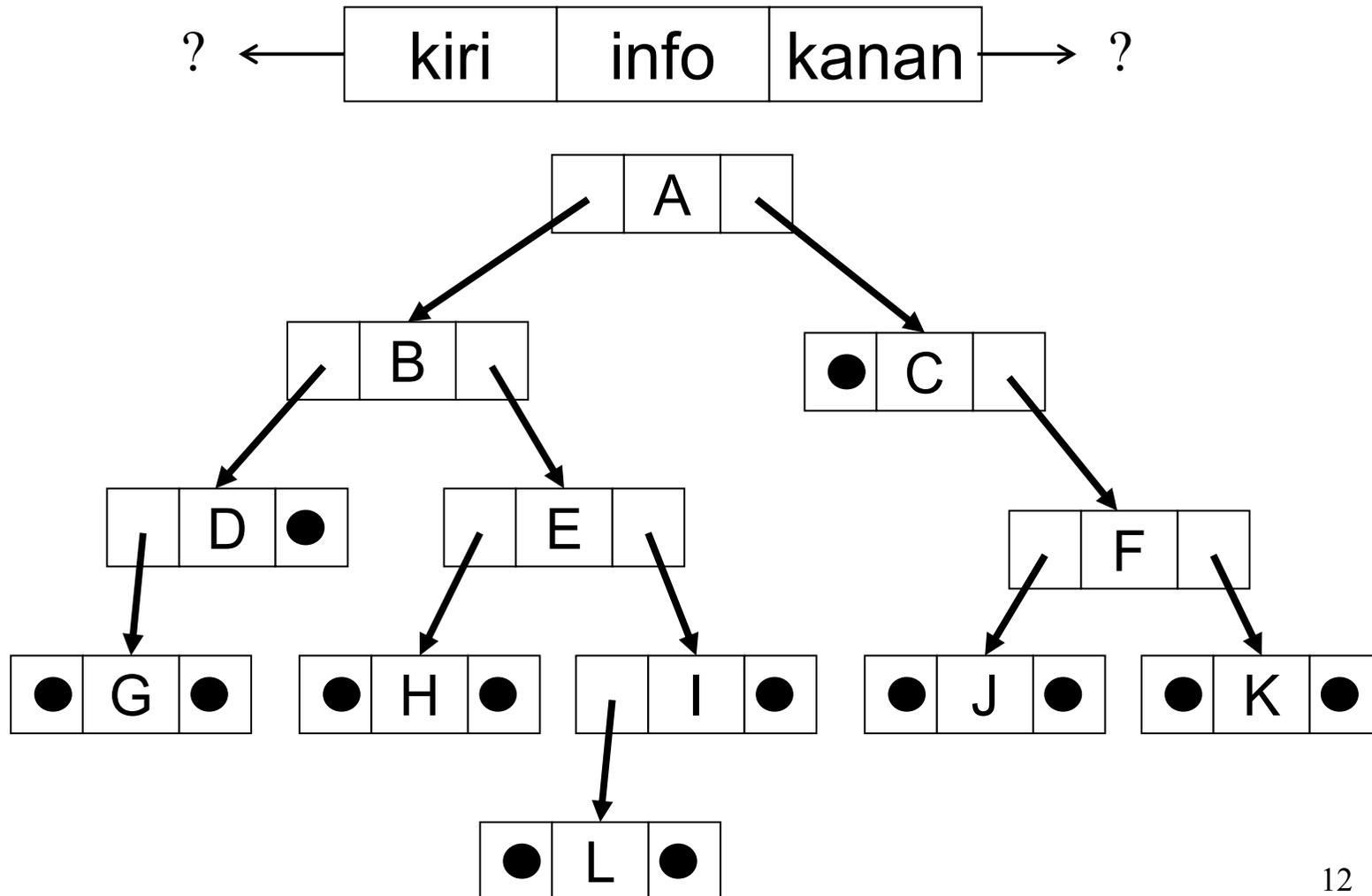
# Fakta-fakta Tree

- Setiap node, kecuali root, memiliki tepat hanya satu parent
- Sekali sebuah link dari sebuah parent ke sebuah child diikuti, tidaklah mungkin untuk kembali ke parent-nya dengan mengikuti link yang lain (tidak ada siklus dalam sebuah tree)
- Kumpulan child-child dari sebuah node, mereka sendiri juga merupakan sebuah tree → disebut sebagai subtree

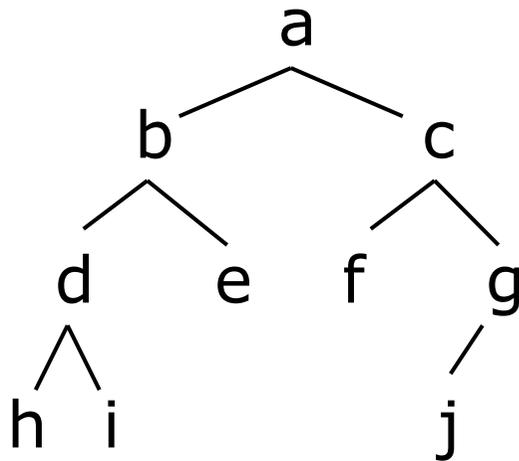
# Binary Tree

- Binary tree adalah sebuah pengorganisasian secara hirarki dari beberapa buah node; masing-masing node tidak mempunyai child lebih dari 2.
- Implementasi binary tree bisa dilakukan menggunakan struktur data linked list; masing-masing node terdiri atas tiga bagian yaitu sebuah data/info dan dua buah pointer yang dinamakan pointer kiri dan kanan.

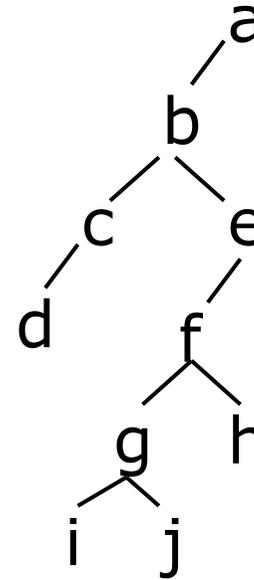
# Binary Tree



# Balance



Balanced binary tree



Unbalanced binary tree

- Suatu binary tree dikatakan sebagai *Balanced binary tree* jika setiap level di atas level yang paling rendah terisi penuh
- Sedangkan dikatakan *Unbalanced binary tree* jika tidak memenuhi kaedah diatas. Dan kebanyakan aplikasi menginginkan suatu tree yang seimbang (balanced).

# Deklarasi Binary Tree

- Node dalam sebuah binary tree disajikan sebagai berikut :



- Sehingga, deklarasi struct untuk sebuah node dalam tree adalah :

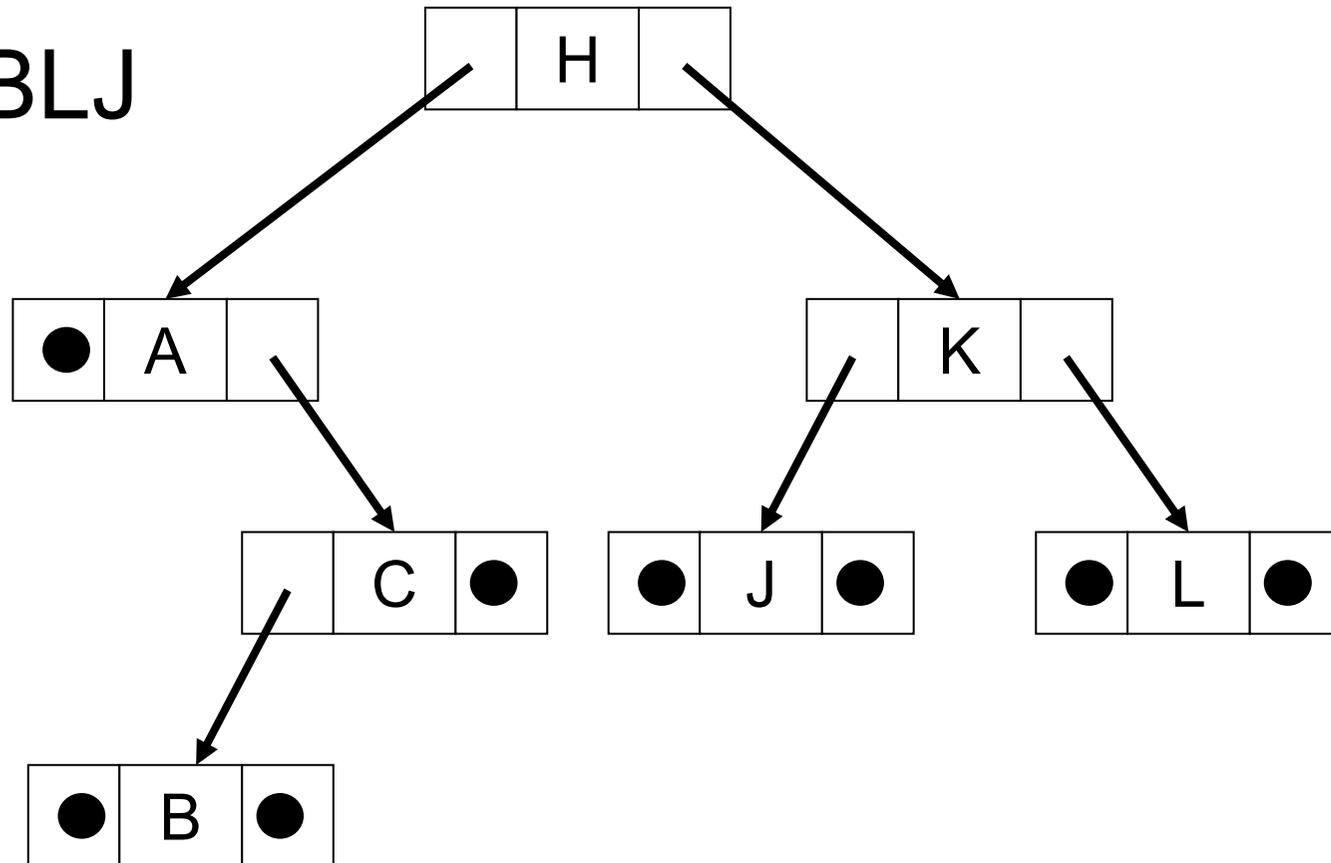
```
typedef char typeInfo;
typedef struct Node tree;
struct Node {
    typeInfo info;
    tree *kiri;          /* cabang kiri */
    tree *kanan;        /* cabang kanan */
};
```

# Pembentukan Binary Tree

- Dapat dilakukan dengan dua cara : rekursif dan non rekursif
- Perlu memperhatikan kapan suatu node akan dipasang sebagai node kiri dan kapan sebagai node kanan.
- Misalnya ditentukan, node yang berisi info yang nilainya “lebih besar” dari parent akan ditempatkan di sebelah kanan dan yang “lebih kecil” di sebelah kiri.

# Bentuk Binary Tree

HKACBLJ



# Langkah-langkah Pembentukan Binary Tree

## 1. Siapkan node baru

- alokasikan memory-nya
- masukkan info-nya
- set pointer kiri & kanan = NULL

## 2. Sisipkan pada posisi yang tepat

- **penelusuran** → utk menentukan posisi yang tepat; info yang nilainya lebih besar dari parent akan ditelusuri di sebelah kanan, yang lebih kecil dari parent akan ditelusuri di sebelah kiri
- **penempatan** → info yang nilainya lebih dari parent akan ditempatkan di sebelah kanan, yang lebih kecil di sebelah kiri

# Algoritma

## Pembentukan Binary Tree

1. Buat node baru (baru)
2. Cek apakah `root = NULL`,  
jika ya, maka `root = baru`, melompat ke langkah 9  
jika tidak, maka lakukan langkah-langkah berikut
3. Mencari posisi yang tepat untuk baru, tentukan `P = root`, `Q = root`
4. Kerjakan langkah 5 dan 6 selama (`Q <> NULL`) dan (`baru->info <> P->info`)
5. Tentukan `P = Q`
6. Cek apakah `baru->info < P->info`  
jika ya, (teruskan ke cabang kiri), tentukan `Q = P->kiri`  
jika tidak, (teruskan ke cabang kanan), tentukan `Q = P->kanan`
7. Cek apakah `baru->info = P->info`  
jika ya, (tidak perlu disisipkan), tampilkan pesan duplikasi, lompat ke langkah 9  
jika tidak, (sisipkan), kerjakan langkah 8
8. Cek apakah `baru->info < P->info`  
jika ya, (sebagai cabang kiri) `P->kiri = baru`  
jika tidak, (sebagai cabang kanan) `P->kanan = baru`
9. Selesai

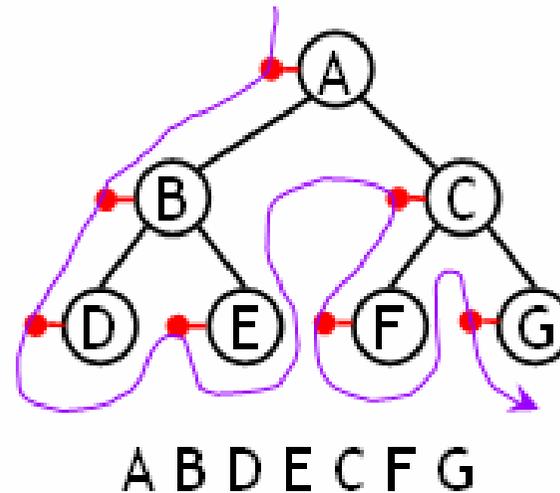
# Metode Traversal

- Salah satu operasi yang paling umum dilakukan terhadap sebuah tree adalah kunjungan (traversing)
- Sebuah kunjungan berawal dari root, mengunjungi setiap node dalam tree tersebut tepat hanya sekali
  - *Mengunjungi artinya memproses data/info yang ada pada node ybs*
- Kunjungan bisa dilakukan dengan 3 cara:
  1. Preorder
  2. Inorder
  3. Postorder
- Ketiga macam kunjungan tersebut bisa dilakukan secara rekursif

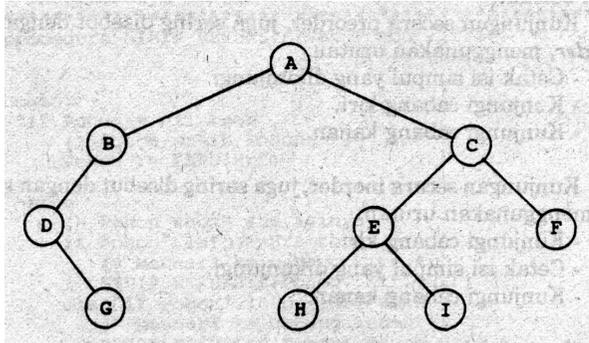
# Preorder Traversal

*(depth first order)*

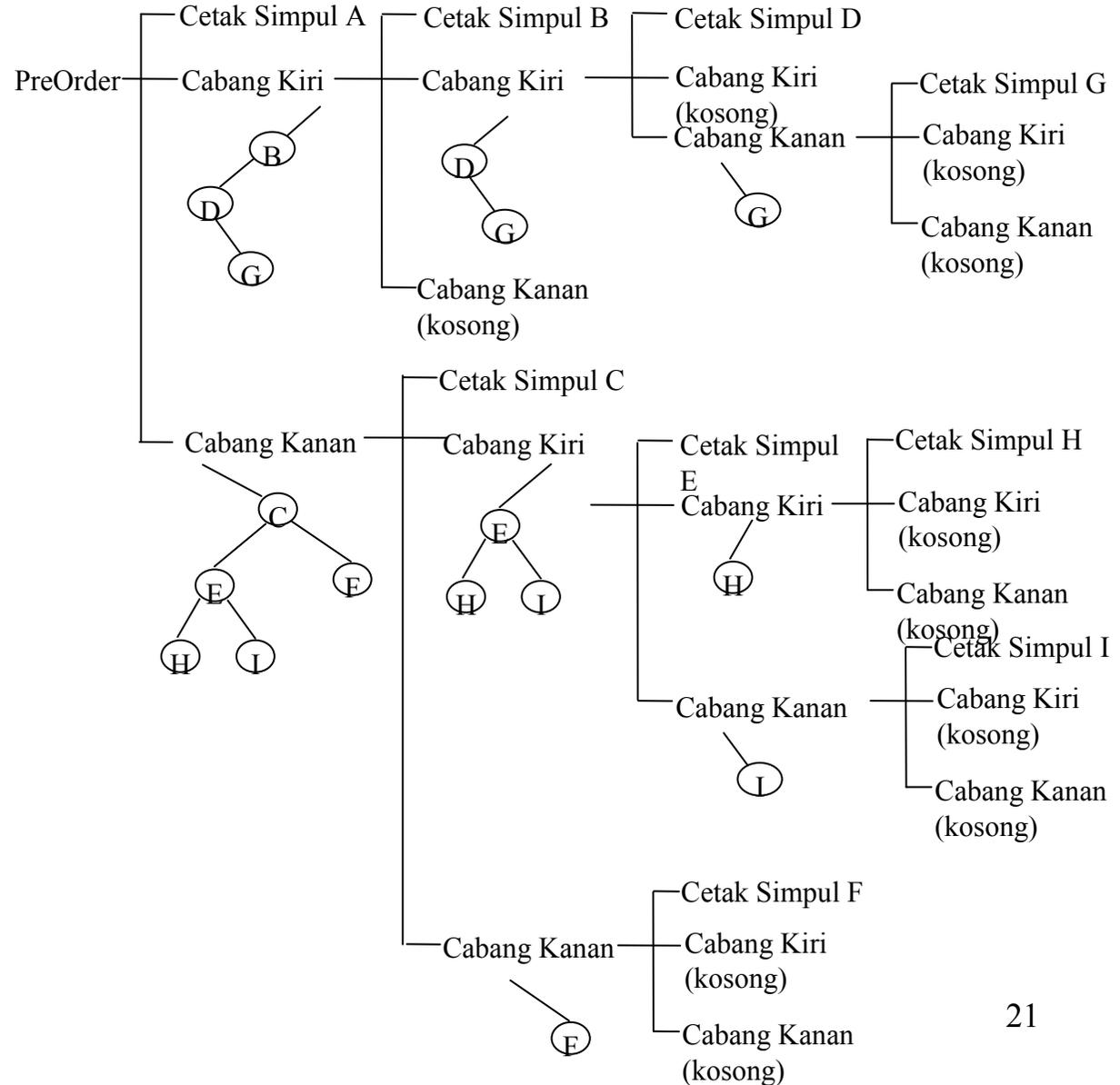
- cetak info pada node yang dikunjungi
- Kunjungi cabang kiri
- Kunjungi cabang kanan



# Preorder Traversal (depth first order)



Hasil penelusuran  
secara preorder :  
A B D G C E H I F



# Algoritma Preorder

*(rekursif)*

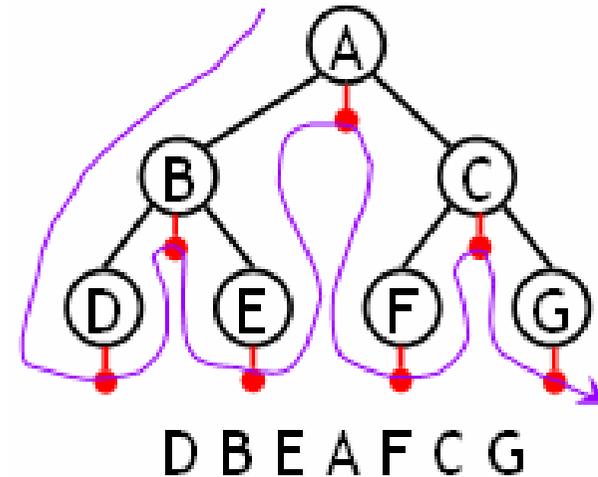
## **preorder (root)**

1. Jika `root <> NULL`, lakukan langkah 2 sampai dengan 4
2. Cetak `root->info`
3. Panggil fungsi : `preorder (root->kiri)`
4. Panggil fungsi : `preorder (root->kanan)`

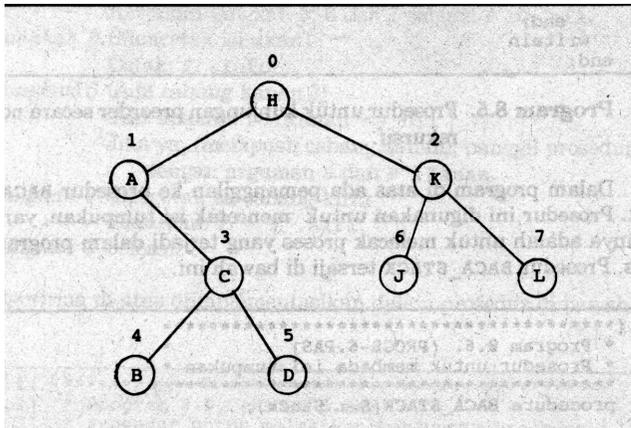
# Inorder Traversal

*(symetric order)*

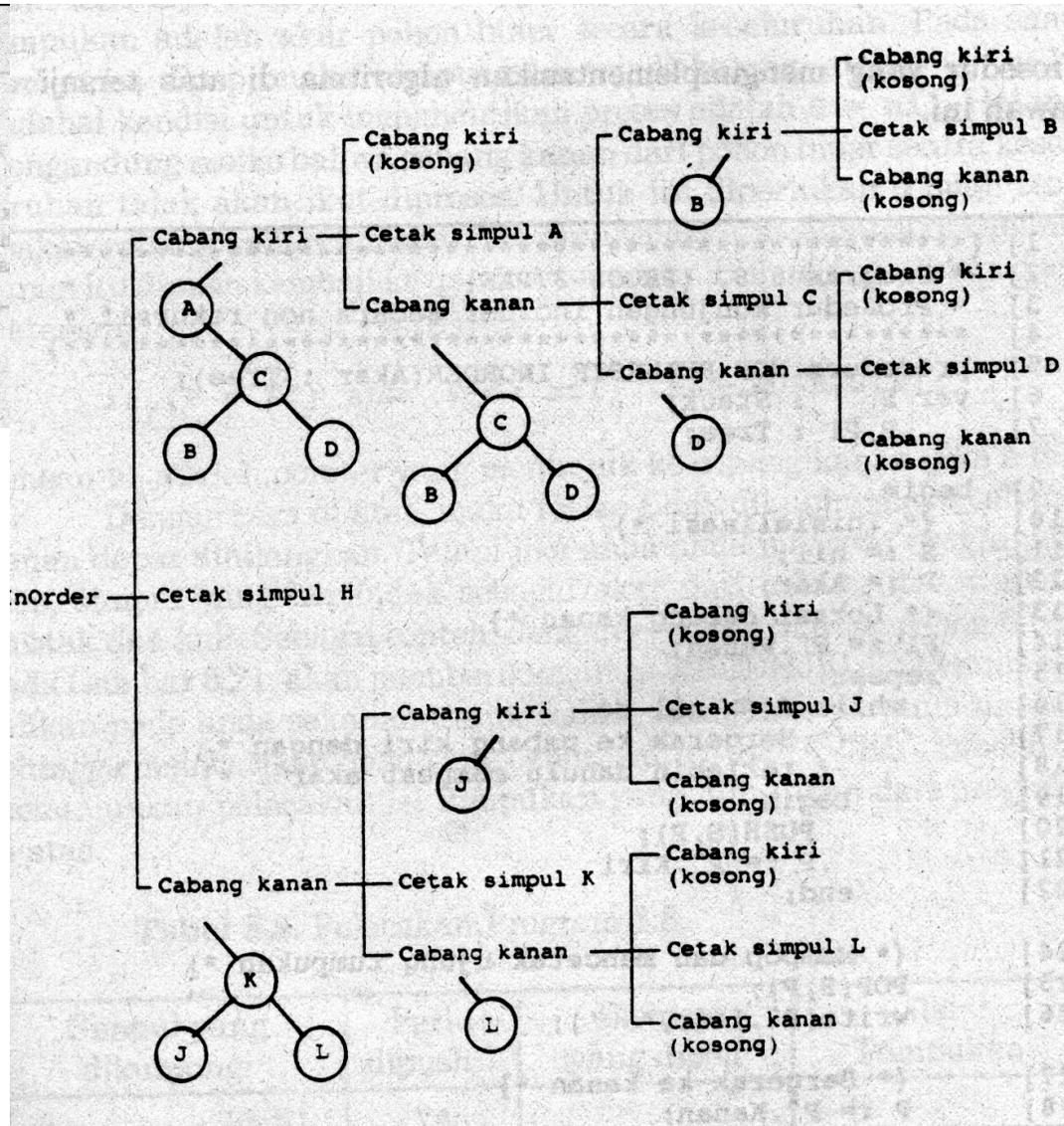
- Kunjungi cabang kiri
- cetak info pada node yang dikunjungi
- Kunjungi cabang kanan



# Inorder Traversal (symetric order)



Hasil penelusuran  
secara inorder :  
A B C D H J K L



# Algoritma Inorder

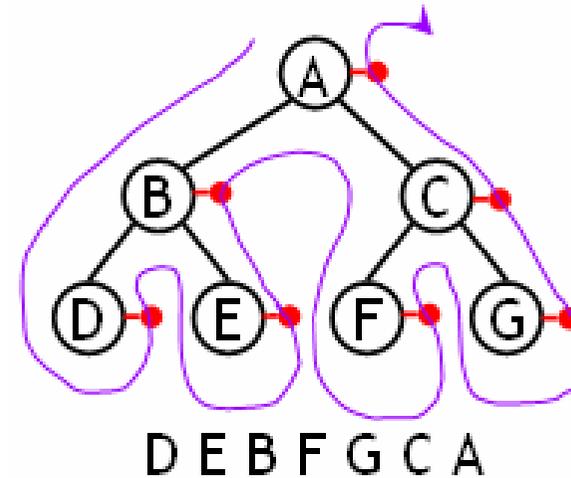
*(rekursif)*

## **inorder (root)**

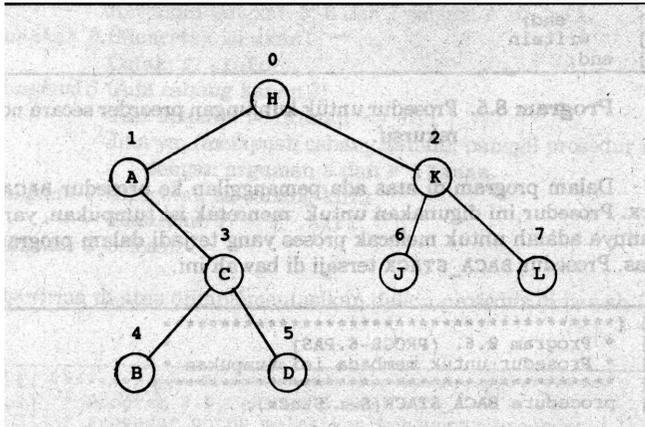
- Jika `root <> NULL`, lakukan langkah 2 sampai dengan 4
- Panggil fungsi : `inorder (root->kiri)`
- Cetak `root->info`
- Panggil fungsi : `inorder (root->kanan)`

# Postorder Traversal

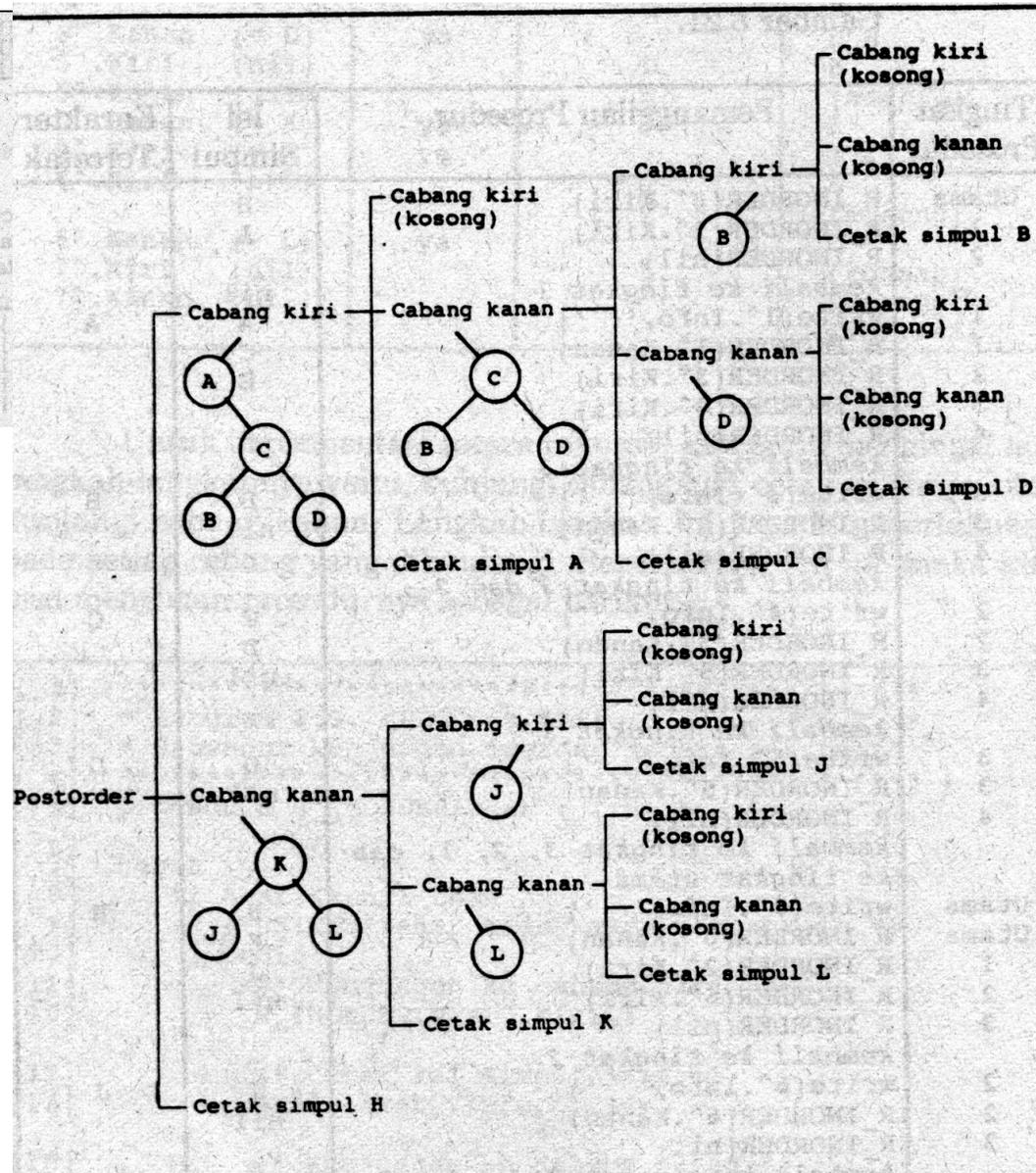
- Kunjungi cabang kiri
- Kunjungi cabang kanan
- cetak info pada node yang dikunjungi



# Postorder Traversal



Hasil penelusuran  
secara postorder :  
B D C A J L K H



# Algoritma Postorder

*(rekursif)*

## **postorder (root)**

1. Jika `root <> NULL`, lakukan langkah 2 sampai dengan 4
2. Panggil fungsi : `postorder (root->kiri)`
3. Panggil fungsi :  
`postorder (root->kanan)`
4. Cetak `root->info`