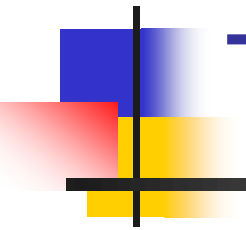
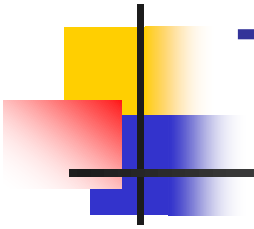


Tree

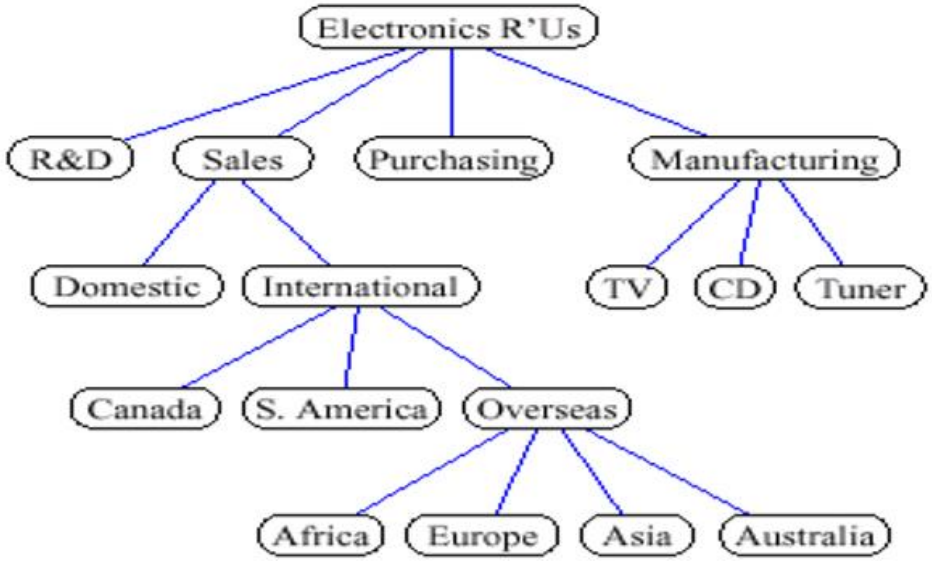


Tujuan

- Memahami definisi dan terminologi mengenai tree secara umum.
- Mengenali aplikasi tree.
- Mengetahui cara melakukan operasi untuk tiap-tiap element pada tree (tree traversal)

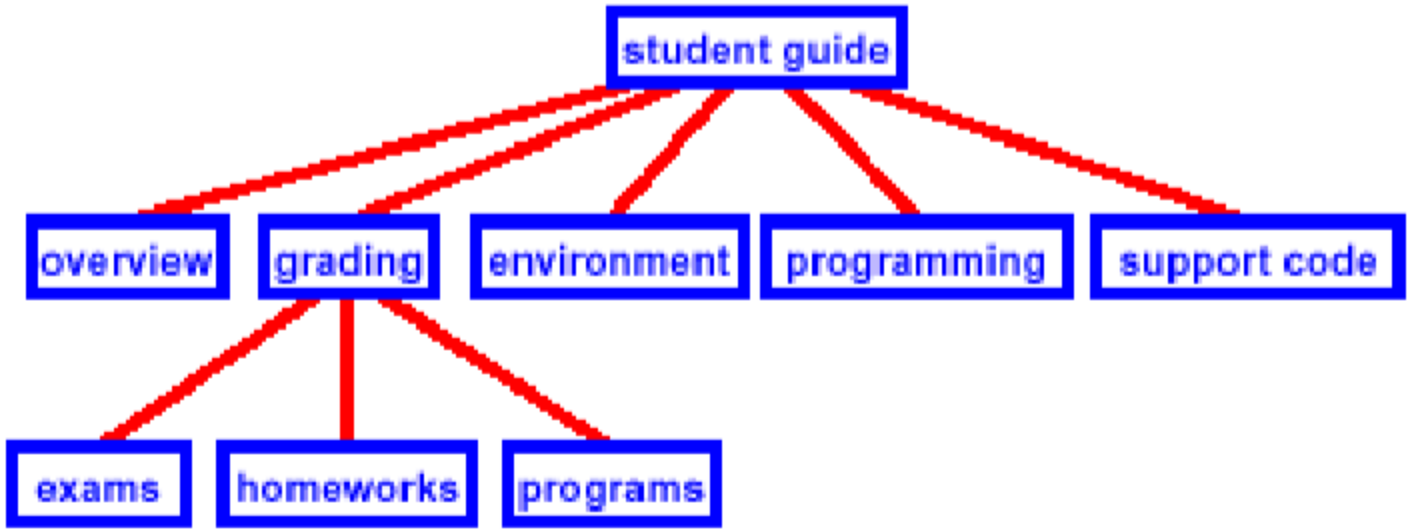
Definisi Tree

- Sebuah tree merepresentasikan sebuah hirarki
  - Mis.: Struktur organisasi sebuah perusahaan



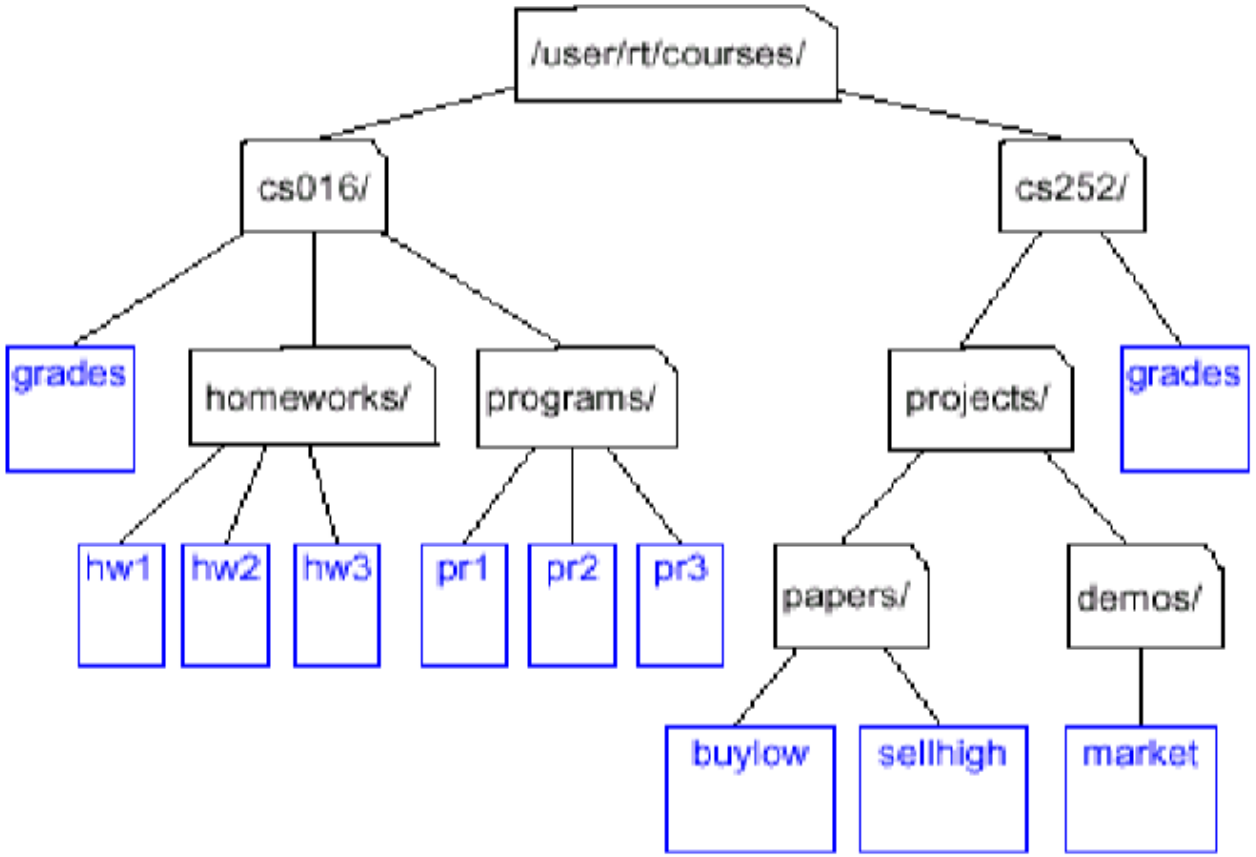
Definisi Tree

- Daftar isi sebuah buku

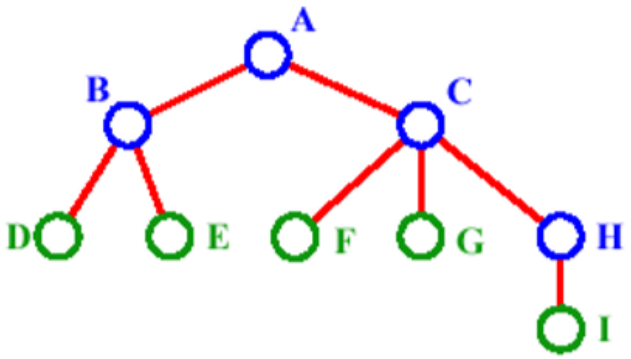


Definisi Tree

- File System



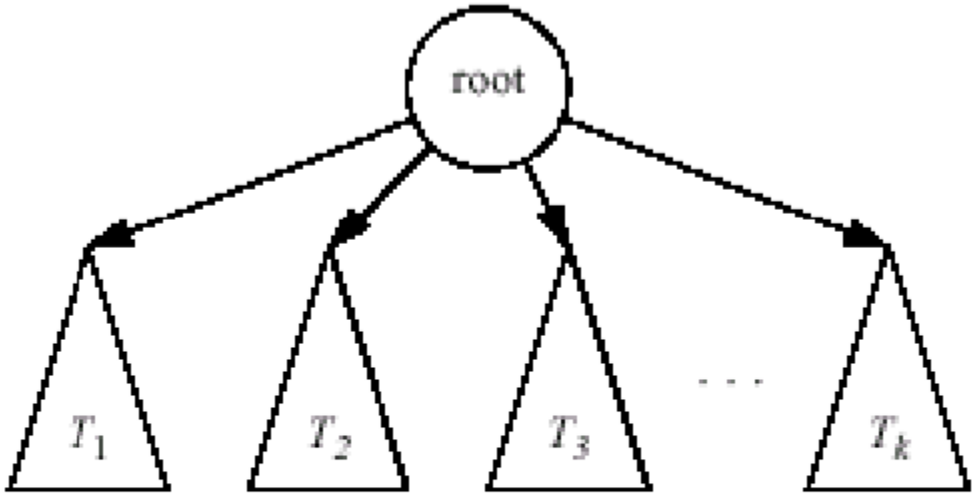
Istilah Umum pada Tree



- *A* is the *root node*
- *B* is the *parent* of *D* and *E*
- *C* is the *sibling* of *B*
- *D* and *E* are the *children* of *B*
- *D, E, F, G, I* are *external nodes*, or *leaves*
- *A, B, C, H* are *internal nodes*
- The *depth/level/path length* of *E* is 2
- The *height* of the tree is 3
- The *degree* of node *B* is 2
- B,D,E adalah subtree

Property:
(# edges) = (#nodes) - 1

Tree dilihat secara rekursif



- Setiap *sub-tree* adalah juga sebuah *tree*!

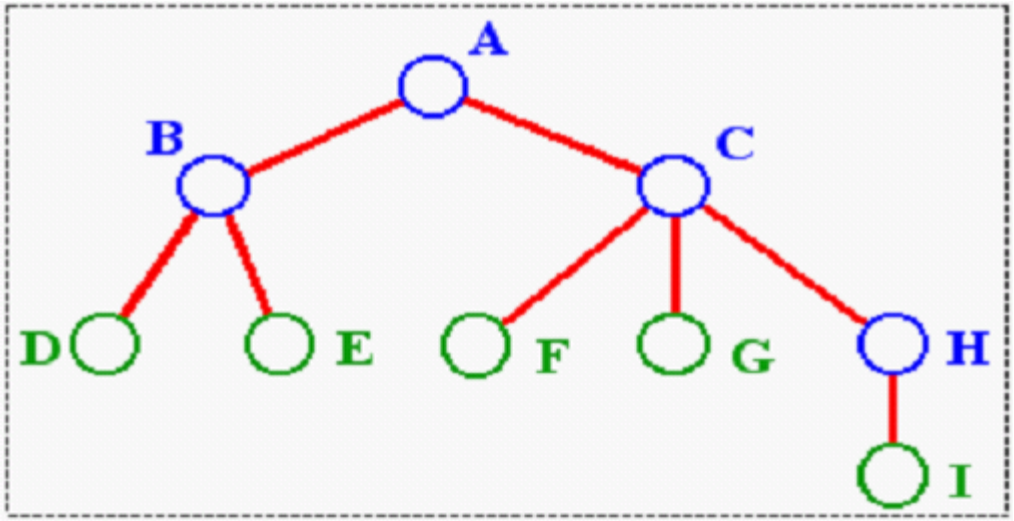
Respresentasi Tree

- Mungkinkah setiap node bisa memiliki link ke setiap anaknya?
  - Bisa tetapi karena jumlah anak sangat bervariasi menjadikan link langsung tidak feasible, jika dipaksakan akan terjadi pemborosan ruang tak terpakai.
- Solusi ▪metoda “first child/next sibling”.
  - Setiap node dengan dua pointer: kiri dan kanan.
  - Anak-anak dari suatu node (relasi sibling) dirangkai dalam linked-list dari anak terkiri ke terkanan dengan pointer kanan.
  - Anak terkiri dari setiap node dirangkai dengan pointer kiri.

Binary Tree

- sebuah pengorganisasian secara hirarki dari beberapa buah simpul, dimana masing-masing simpul tidak mempunyai anak lebih dari 2.
- Simpul yang berada di bawah sebuah simpul dinamakan anak dari simpul tersebut.
- Simpul yang berada di atas sebuah simpul dinamakan induk dari simpul tersebut.

Binary Tree

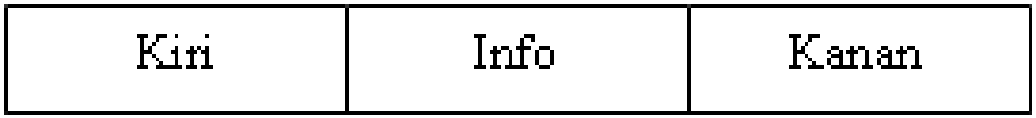


Istilah dalam Tree

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

Struktur Binary Tree

- Masing-masing simpul dalam binary tree terdiri dari tiga bagian yaitu sebuah data dan dua buah pointer yang dinamakan pointer kiri dan kanan.



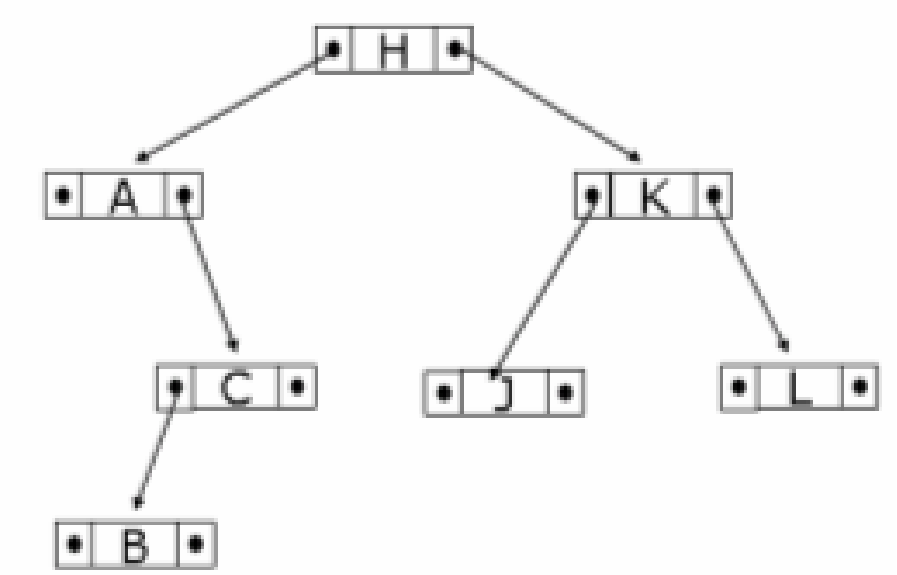
Deklarasi Tree

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

Pembentukan 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.
- Sebagai contoh jika kita memiliki informasi “HKACBLJ” maka pohon biner yang terbentuk

Pembentukan Tree



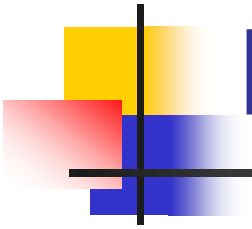
Pembentukan Tree

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

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 dan non rekursif



Preorder

- Kunjungan preorder, juga disebut dengan *depth first order*, menggunakan urutan:

  Cetak isi simpul yang dikunjungi

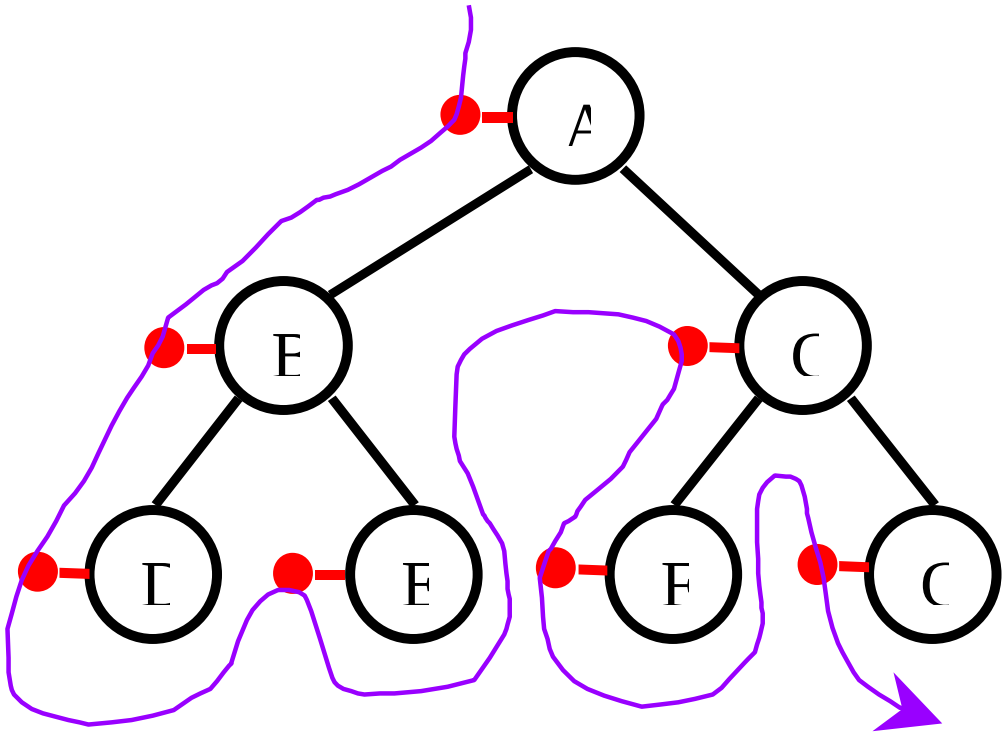
  Kunjungi cabang kiri

  Kunjungi cabang kanan

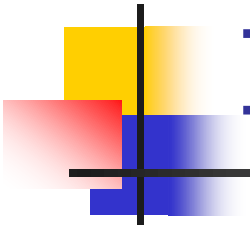
Preorder

```
void preorder(pohon ph)
{
   if (ph != NULL)
   {
     cout<<ph->info;
     preorder(ph->kiri);
     preorder(ph->kanan);
   }
}
```

Preorder



A B D E C F G



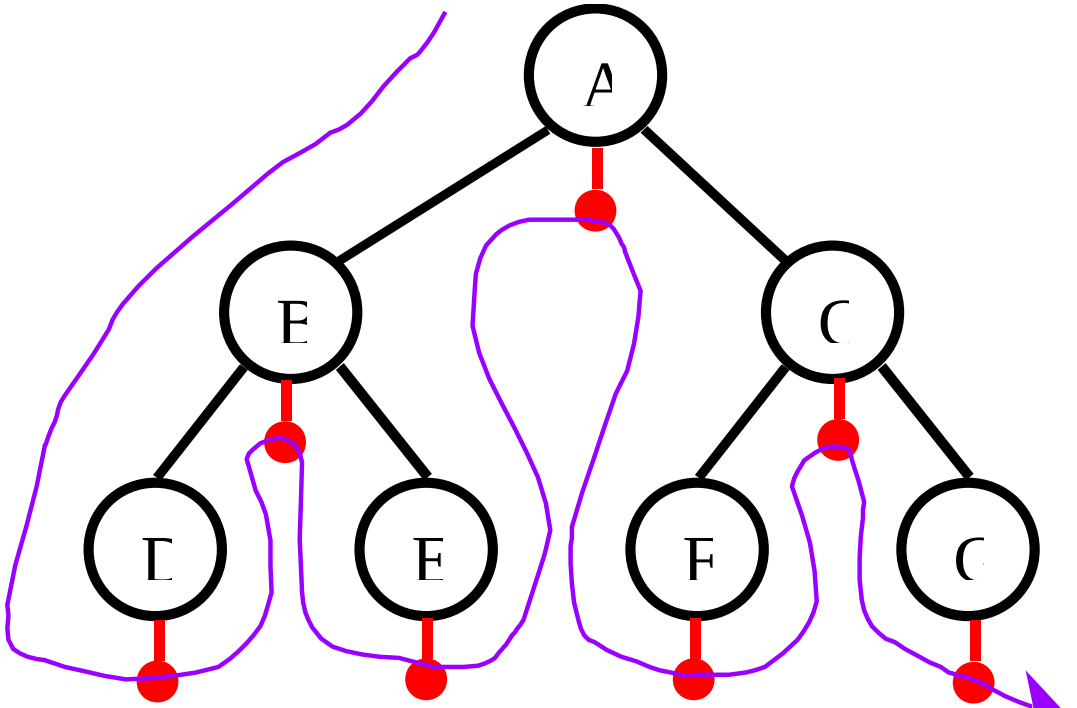
Inorder

- Kunjungan secara inorder, juga sering disebut dengan *symmetric order*, menggunakan urutan:
- Kunjungi cabang kiri
- Cetak isi simpul yang dikunjungi
- Kunjungi cabang kanan

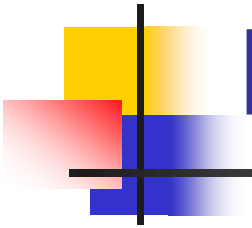
Inorder

```
void inorder(pohon ph)
{
   if (ph != NULL)
   {
        inorder(ph->kiri);
        cout<<ph->info;
        inorder(ph->kanan);
   }
}
```

Inorder

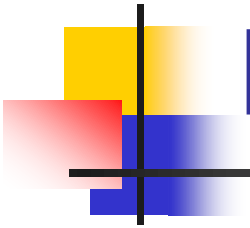


D B E A F C G



Postorder

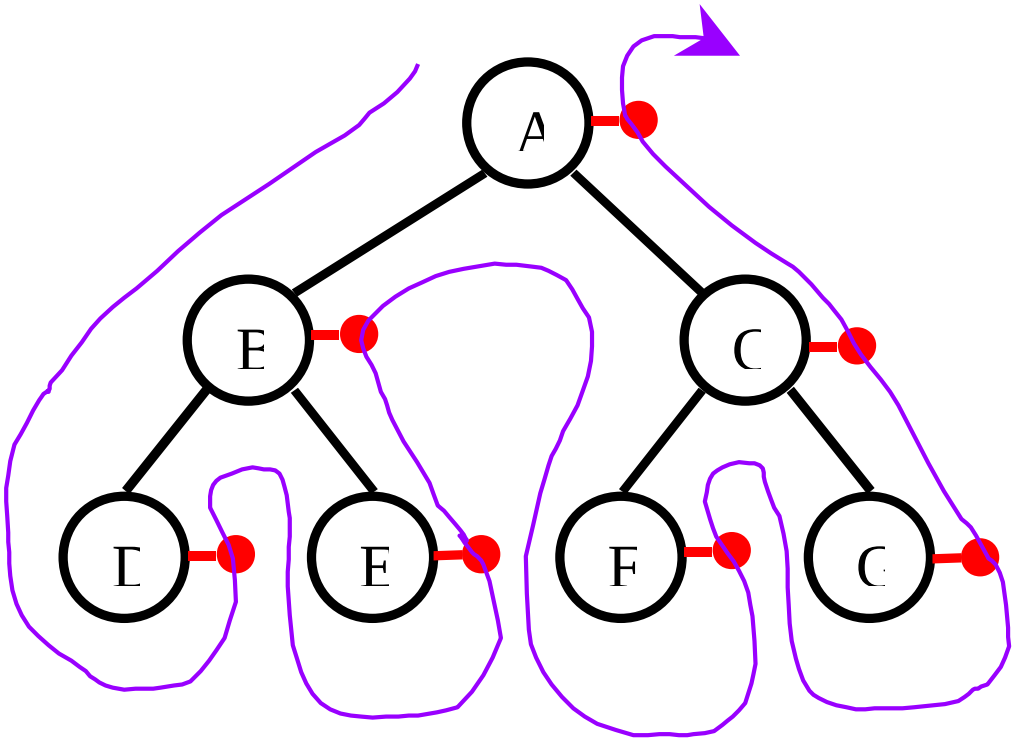
- Kunjungan secara postorder menggunakan urutan:
- Kunjungi cabang kiri
- Kunjungi cabang kanan
- Cetak isi simpul yang dikunjungi

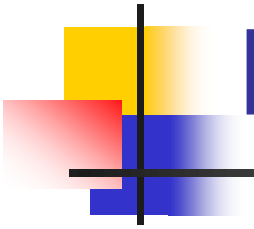


Postorder

```
void postorder(pohon ph)
{
   if (ph != NULL)
   {
        postorder(ph->kiri);
        postorder(ph->kanan);
        cout<<ph->info;
   }
}
```

Postorder





Referensi

- http://lecturer.ukdw.ac.id/anton
- http://www.cs.ui.ac.id/WebKuliah/IKI10100/1998/handout/handout20.html