MODUL IX
TREE (POHON)

A. TUJUAN

- Mahasiswa mampu menjelaskan mengenai algoritma Tree
- Mahasiswa mampu membuat dan mendeklarasikan struktur algoritma Tree
- Mahasiswa mampu menerapkan dan mengimplementasikan algoritma Tree

B. DASAR TEORI

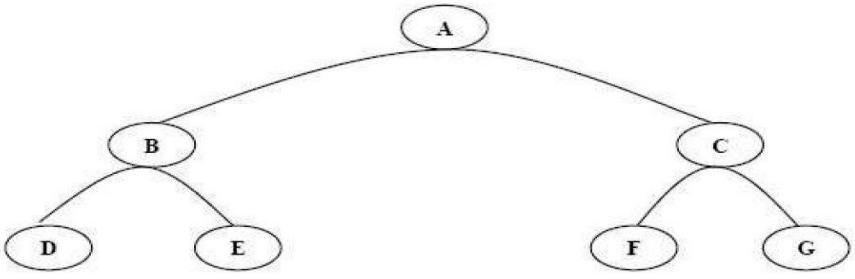
1. PENGERTIAN TREE

Kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layakya struktur sebuah pohon. Struktur pohon adalah suatu cara merepresentasikan suatu struktur hirarki (one-to-many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah. Suatu struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one-to-many) dan tidak linier antara elemen-elemennya.

2. ISTILAH DALAM TREE

Predecesor	Node yang berada diatas node tertentu.
Successor	Node yang berada dibawah node tertentu.
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node.
Child	Successor satu level di bawah suatu node.
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendantnya.
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan dalam suatu tree
Root	Node khusus yang tidak memiliki predecessor.
Leaf	Node-node dalam tree yang tidak memiliki successor.
Degree	Banyaknya child dalam suatu node

Ilustrasi alogaritma tree



Ancestor (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 = 2

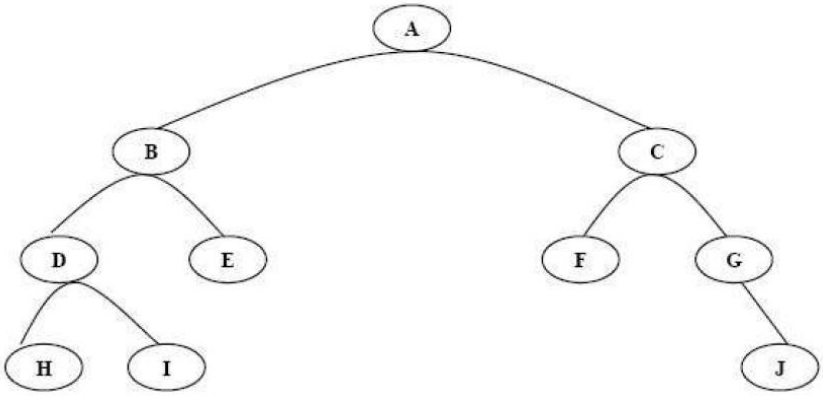
3. JENIS-JENIS TREE

BINARY TREE

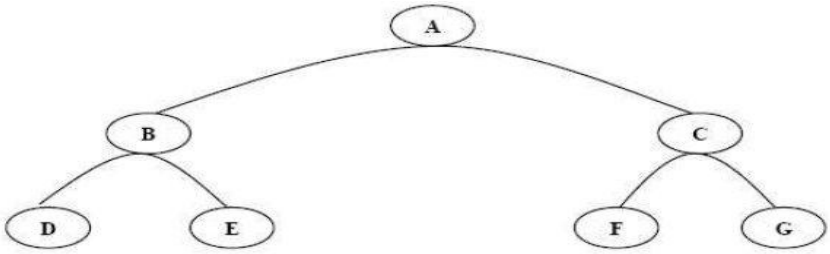
Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua sub pohon dan kedua subpohon harus terpisah.

Kelebihan struktur Binary Tree :

- Mudah dalam penyusunan algoritma sorting
- Searching data relatif cepat
- Fleksibel dalam penambahan dan penghapusan data

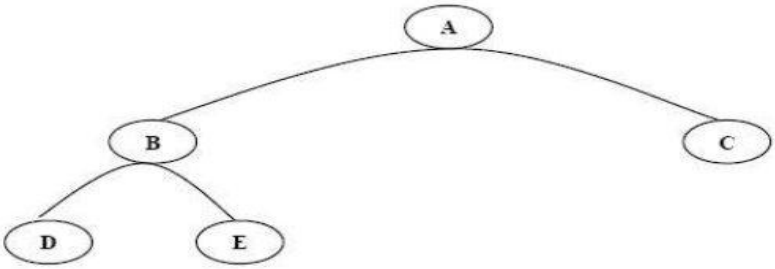


FULL BINARY TREE



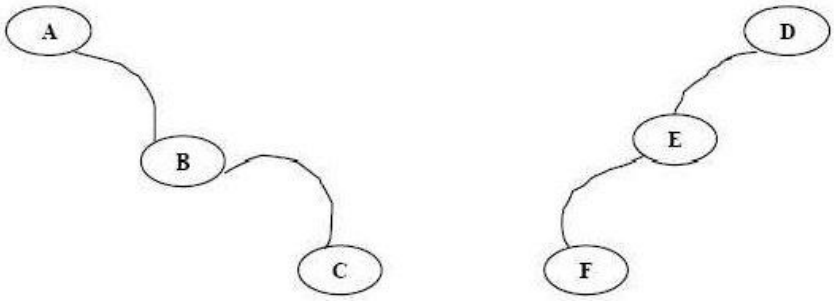
Semua node, kecuali leaf pasti memiliki 2 anak dan tiap subpohon memiliki panjang path yang sama.

COMPLETE BINARY TREE



Tree yang mirip dengan full binary tree, tapi tiap subtree boleh memiliki panjang path yang berbeda dan tiap node (kecuali leaf) memiliki 2 anak.

SKEWED BINARY TREE



Binary tree yang semua nodenya (kecuali leaf) hanya memiliki satu anak

4. IMPLEMENTASI PROGRAM TREE

DEKLARASI STRUCT

```
typedef struct Node{
    int data;
    Node *kiri;
    Node *kanan;
};
```

DEKLARASI VARIABEL

```
Node *pohon;
```

OPERASI

Create: membentuk sebuah tree baru yang kosong.

```
pohon = NULL;
```

Insert: menambah node ke dalam Tree.

Jika data yang akan dimasukkan lebih besar daripada elemen root, maka akan diletakkan di node sebelah kanan, sebaliknya jika lebih kecil maka akan diletakkan di node sebelah kiri. Untuk data pertama akan menjadi elemen root.

```
void tambah(Node **root, int databaru)
{
    if((*root) == NULL){
        Node *baru;
        baru = new Node;
        baru->data = databaru;
        baru->kiri = NULL;
        baru->kanan = NULL;
        (*root) = baru;
        (*root)->kiri = NULL;
        (*root)->kanan = NULL;
        printf("Data bertambah!");
    }
    else if(databaru < (*root)->data)
        tambah(&(*root)->kiri,databaru);
    else if(databaru > (*root)->data)
            tambah(&(*root)->kanan,databaru);
    else if(databaru == (*root)->data)
        printf("Data sudah ada!");
}
```

PreOrder: cetak node yang dikunjungi, kunjungi left, kunjungi right

```
void preOrder(Node *root){
    if(root != NULL){
        printf("%d ",root->data);
        preOrder(root->kiri);
        preOrder(root->kanan);
    }
}
```

InOrder: kunjungi left, cetak node yang dikunjungi, kunjungi right

```
void inOrder(Node *root){
    if(root != NULL){
        inOrder(root->kiri);
        printf("%d ",root->data);
        inOrder(root->kanan);
    }
}
```

PostOrder: kunjungi left, kunjungi right, cetak node yang dikunjungi

```
void postOrder(Node *root){
    if(root != NULL){
        postOrder(root->kiri);
        postOrder(root->kanan);
        printf("%d ",root->data);
    }
}
```

C. LATIHAN

```
#include <stdio.h>
#include <conio.h>
typedef struct Node{
    int data;
    Node *kiri;
    Node *kanan;
};
void tambah(Node **root, int databaru)
{
    if((*root) == NULL){
        Node *baru;
        baru = new Node;
        baru->data = databaru;
        baru->kiri = NULL;
        baru->kanan = NULL;
        (*root) = baru;
        (*root)->kiri = NULL;
        (*root)->kanan = NULL;
        printf("Data bertambah!");
    }
    else if(databaru < (*root)->data)
        tambah(&(*root)->kiri,databaru);
    else if(databaru > (*root)->data)
            tambah(&(*root)->kanan,databaru);
    else if(databaru == (*root)->data)
        printf("Data sudah ada!");
}
void preOrder(Node *root){
    if(root != NULL){
        printf("%d ",root->data);
        preOrder(root->kiri);
        preOrder(root->kanan);
    }
}
void inOrder(Node *root){
    if(root != NULL){
        inOrder(root->kiri);
        printf("%d ",root->data);
        inOrder(root->kanan);
    }
}
void postOrder(Node *root){
    if(root != NULL){
        postOrder(root->kiri);
        postOrder(root->kanan);
        printf("%d ",root->data);
    }
}
```

```
main(){
    int pil,c;
    Node *pohon,*t;
    pohon = NULL;
    do{
        int data;
        printf("MENU\n");
        printf("1. Tambah\n");
        printf("2. Lihat pre-order\n");
        printf("3. Lihat in-order\n");
        printf("4. Lihat post-order\n");
        printf("5. Exit\n");
        printf("Pilihan : ");
        scanf("%d",&pil);
        switch(pil){
            case 1:     printf("Data baru : ");
                scanf("%d", &data);
                tambah(&pohon,data);
                break;
            case 2:     if(pohon!=NULL) preOrder(pohon);
                else printf("Masih kosong!");
                break;
            case 3:      if(pohon!=NULL) inOrder(pohon);
                else printf("Masih kosong!");
                break;
            case 4:      if(pohon!=NULL) postOrder(pohon);
                else printf("Masih kosong!");
                break;
            }
        getch();
    }while(pil!=5);
}
```

D. TUGAS

1. Buatlah program Tree dengan fungsi untuk menghapus sebuah node tertentu dan menghapus seluruh node.
2. Buatlah program Tree dengan fungsi untuk searching node.