MODUL VII
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

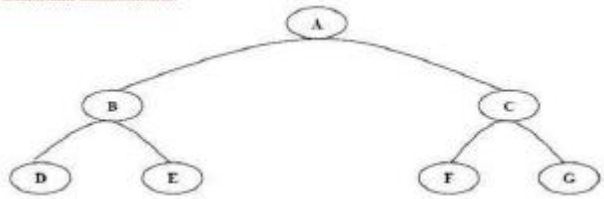
1. PENGERTIAN TREE

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 Algoritma 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

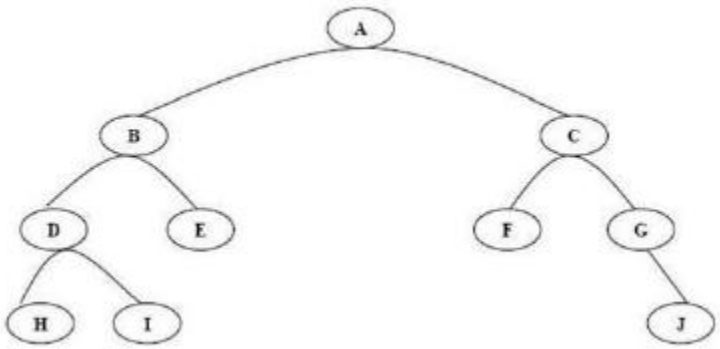
3. JENIS - JENIS

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



4. IMPLEMENTASI ALGORITMA TREE

Insert : menambahkan node 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 insert(int data,int node = 1){

    if(tree[node] == int()){ //int() same NULL
        tree[node] = data;
        curr++;
    }
    else if(data < tree[node]){
        insert(data, 2*node); //do recursion
    }
    else if(data > tree[node]){
        insert(data, 2*node+1); //do recursion
    }
}
```

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

```
void preorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;
    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){
        cout<<tree[node]<<" ";

        if(tree[left] != int()) preorder(2*node); //do recursion
        if(tree[right] != int()) preorder(2*node+1); //do recursion

    }
}
```

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

```
void inorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;

    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){

        if(tree[left] != int()) {
            inorder(2*node); //do recursion
        }

        cout<<tree[node]<<" ";

        if(tree[right] != int()) {
            inorder(2*node+1); //do recursion
        }
    }
}
```

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

```
void postorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;
    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){

        if(tree[left] != int()) {
            postorder(2*node);
        }


        if(tree[right] != int()) {
            postorder(2*node+1);
        }

        cout<<tree[node]<<" ";
    }
}
```

Searching Data : untuk searching data kita tinggal menjelajahi node yang ada di dalam tree, algoritmanya adalah jika data yang di cari sama dengan root maka data di temukan, jika tidak maka akan mengecek kembali apakah data lebih lebih kecil dari root maka secara rekursi akan mencari kekiri, jika data lebih besar dari root maka secara rekursif akan mencari ke kanan

```
void searchTree(int data,int node = 1){
    if(tree[node] == int()){
        cout<<"data tidak di temukan"<<endl;
        return;
    }

    if(data == tree[node]){
        cout<<"data ditemuikan pada node ke : "<<node<<endl;
        return;
    }
    else if(data < tree[node]){
        searchTree(data,node*2);
    }
    else if(data > tree[node]){
        searchTree(data,2*node+1);
    }
}
```

C. LATIHAN

```cpp
#include <iostream>
#include <vector> //library for vector
#include <windows.h>

using namespace std;

vector<int> tree; //like array but size is dynamic
int curr = 0; //size of node

bool empty(){
    if(curr == 0) return true;
    else return false;
}

void clear(){
    curr = 0;
    tree.assign(1000000,int());
}

//function to insert node in tree
void insert(int data,int node = 1){

    if(tree[node] == int()){ //int() same NULL
        tree[node] = data;
        curr++;
    }
    else if(data < tree[node]){
        insert(data, 2*node); //do recursion
    }
    else if(data > tree[node]){
        insert(data, 2*node+1); //do recursion
    }
}

void preorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;
    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){
        cout<<tree[node]<<" ";

        if(tree[left] != int()) preorder(2*node); //do recursion
        if(tree[right] != int()) preorder(2*node+1); //do recursion

    }
}
```

```
51
52  void inorder(int node = 1){
53      int left = 2 * node;
54      int right = 2 * node+1;
55
56      if(empty() == true){
57          cout<<"Node Masih Kosong"<<endl;
58          return;
59      }
60
61      if(tree[node] != int()){
62
63          if(tree[left] != int()) {
64              inorder(2*node); //do recursion
65          }
66
67          cout<<tree[node]<<" ";
68
69          if(tree[right] != int()) {
70              inorder(2*node+1); //do recursion
71          }
72      }
73  }
74
75  void postorder(int node = 1){
76      int left = 2 * node;
77      int right = 2 * node+1;
78      if(empty() == true){
79          cout<<"Node Masih Kosong"<<endl;
80          return;
81      }
82
83      if(tree[node] != int()){
84
85          if(tree[left] != int()) {
86              postorder(2*node);
87          }
88
89
90          if(tree[right] != int()) {
91              postorder(2*node+1);
92          }
93
94          cout<<tree[node]<<" ";
95      }
96  }
```

```
97
98  //function search binary tree
99  void searchTree(int data,int node = 1){
100     if(tree[node] == int()){
101         cout<<"data tidak di temukan"<<endl;
102         return;
103     }
104
105     if(data == tree[node]){
106         cout<<"data ditemuikan pada node ke : "<<node<<endl;
107         return;
108     }
109     else if(data < tree[node]){
110         searchTree(data,node*2);
111     }
112     else if(data > tree[node]){
113         searchTree(data,2*node+1);
114     }
115 }
116
117 int main(){
118     int data,menu;
119     tree.assign(1000000,int()); //provide value to the vector tree
120
121     do{
122         //system("cls");
123         cout<<"MENU"<<endl;
124         cout<<"1. Insert Data"<<endl;
125         cout<<"2. Lihat Pre-Order"<<endl;
126         cout<<"3. Lihat In-Order"<<endl;
127         cout<<"4. Lihat Post-Order"<<endl;
128         cout<<"5. Lihat Jumlah Node "<<endl;
129         cout<<"6. Hapus Semua Node"<<endl;
130         cout<<"7. Cari Data"<<endl;
130         cout<<"7. Cari Data"<<endl;
131         cout<<"8. Keluar"<<endl;
132         cout<<"Pilih Menu : ";
133         cin>>menu;
134         cout<<endl;
135
136         switch(menu){
137             case 1:
138                 cout<<"Masukkan Data :";
139                 cin>>data;
140                 cout<<endl<<endl;
141                 insert(data);
142                 break;
143             case 2:
144                 preorder();
145                 cout<<endl<<endl;
146                 break;
147             case 3:
148                 inorder();
149                 cout<<endl<<endl;
150                 break;
```

```
151            case 4:
152                postorder();
153                cout<<endl<<endl;
154                break;
155            case 5:
156                cout<<curr<<endl<<endl;
157                break;
158            case 6:
159                clear();
160                cout<<"data sudah terhapus"<<endl<<endl;
161                break;
162            case 7:
163                cout<<"Masukkan Data yang akan di cari : ";
164                cin>>data;
165                searchTree(data);
166                cout<<endl;
167                break;
168            default :
169                cout<<"pilihan yang anda masukan salah"<<endl;
170        }
171    }while(menu != 8 );
172 }
```

D. TUGAS RUMAH

1. Buatlah sebuah fungsi atau prosedur yang dapat menghapus suatu node dari struktur data tree. Untuk algoritmanya silahkan lihat di visualgo.net
2. Buatlah sebuah fungsi untuk mencari Parent dan child dari suatu node dalam tree