LAPORAN RENCANA PRAKTIKUM
STRUKTUR DATA

NAMA : INDRA FIQI RIPANI

NIM : 213010503002

KELAS : F

MODUL : IV (*BINARY TREE*)

Program Studi S1 Teknik Informatika

Fakultas Teknik

Universitas Palangka Raya

Palangka Raya, Kalimantan Tengah

2022

MODUL IV
BINARY TREE

I. TUJUAN PRAKTIKUM

1. Mempelajari variasi bagian-bagian dari tree sebagai suatu bentuk struktur tak linier.
2. Mempelajari beberapa hubungan fakta yang direpresentasikan di dalam sebuah tree, sehingga mampu merepresentasikan tree dalam permasalahan aslinya.
3. Memahami bagaimana menulis program untuk tree, dan bagaimana menga-rtikannya kembali dalam bentuk permasalahan aslinya

II. LANDASAN TEORI

Pohon biner adalah sebuah tree yang pada masing-masing simpulnya hanya dapat memiliki maksimum 2 (dua) simpul anak. Tidak boleh lebih. Pada pohon biner, umumnya kedua node anak disebut dengan posisinya, yaitu kiri dan kanan. Beberapa istilah pada pohon biner:

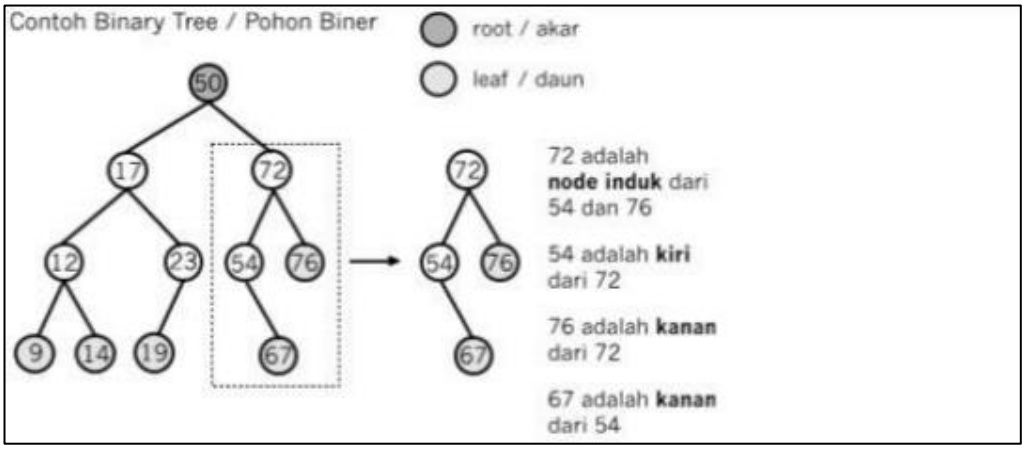
– Size (ukuran): jumlah total node yang terdapat pada pohon biner tersebut.
– Depth (kedalaman): panjang jalur yang menghubungkan sebuah node sampai ke node anaknya yang paling ujung (leaf). Depth biasa juga disebut height.

Jenis-jenis pohon biner:

- Full Binary Tree (Pohon Biner Penuh) adalah pohon biner yang setiap nodenya pasti memiliki 0 atau 2 node anak.
- Perfect Binary Tree (Pohon Biner Sempurna) adalah pohon biner yang semua node leafnya berada pada kedalaman yang sama dari node root. Juga disebut sebagai Complete Binary Tree (Pohon Biner Lengkap).
- Almost Complete Binary Tree (Pohon Biner Hampir Lengkap) adalah pohon biner yang setiap nodenya dapat memiliki 0 node anak, atau memiliki

kiri, atau jika memiliki kanan harus memiliki kiri. Tidak boleh memiliki kanan saja.

Ilustrasi Binary Tree:



Gambar 1.1 Ilustrasi Binary Tree

Implementasi

Implementasi dalam pemrograman, dalam pokok bahasan ini akan dibicarakan untuk pohon biner saja. Asumsi awal adalah data yang hendak dimasukkan ke dalam node, bertipe data integer.

1. Deklarasi Tree

   Karena tree tersusun oleh node-node, maka yang perlu kita deklarasikan adalah komponen node itu sendiri. Dalam contoh dibawah, akan kita namai Node. Sebelumnya perlu kita lihat bahwa untuk mewujudkan implementasi node ke dalam bahasa pemrograman, diperlukan sebuah struktur yang memiliki susunan berikut ini:

kiri	data	kanan
pointer	int	pointer

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

```
};
```

Variabel data digunakan untuk menyimpan nilai angka node tersebut, sedangkan kiri dan kanannya, bertipe pointer, masing-masing mewakili vektor yang akan menunjuk ke node anak kiri dan kanan.

2. Inisialisasi Tree

Untuk pertama kali, saat kita akan membuat sebuah pohon biner, asumsi awal adalah pohon itu belum bertumbuh, belum memiliki node sama sekali, sehingga masih kosong. Oleh karena itu perlu kita tambahkan kode berikut pada baris awal fungsi Main:

```
Node *pohon;
pohon = NULL;
```

Kita mendeklarasikan sebuah pointer yang akan menunjuk ke akar pohon yang kita buat, dengan nama *pohon. Pointer ini ditujukan untuk menunjuk struktur bertipe Node, yang telah dibuat pada bagian 1. Karena pohon tersebut sama sekali belum memiliki node, maka pointer *pohon ditunjukkan ke NULL.

3. Menambahkan Node Pada Tree

Karena pohon yang kita buat merupakan sebuah pohon biner, maka untuk menambahkan sebuah node, secara otomatis penambahan tersebut mengikuti aturan penambahan node pada pohon biner:

a. Jika pohon kosong, maka node baru ditempatkan sebagai akar pohon.
b. Jika pohon tidak kosong, maka dimulai dari node akar, dilakukan proses pengecekan berikut:
    1) Jika nilai node baru lebih kecil dari nilai node yang sedang dicek, maka lihat ke kiri node tersebut. Jika kiri node tersebut kosong (belum memiliki kiri), maka node baru menjadi kiri node yang sedang dicek. Seandainya kiri node sudah terisi, lakukan kembali pengecekan a dan b terhadap node kiri tersebut. Pengecekan ini dilakukan seterusnya hingga node baru dapat ditempatkan.

2) Jika nilai node baru lebih besar dari nilai node yang sedang dicek, maka lihat ke kanan node tersebut. Jika kanan node tersebut kosong (belum memiliki kanan), maka node baru menjadi kanan node yang sedang dicek. Seandainya kanan node sudah terisi, lakukan kembali pengecekan a dan b terhadap node kanan tersebut. Pengecekan ini dilakukan seterusnya hingga node baru dapat ditempatkan.

Proses penambahan ini diimplementasikan secara rekursif pada fungsi berikut:

```
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;
        }
        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!");

}
```

Variabel **root menunjukkan node mana yang sedang dicek saat ini, untuk itu saat pemanggilan fungsi ini, variabel **root kita beri nilai pointer yang menunjuk ke node akar, yaitu pohon.

```
tambah(&pohon,data);
```

4. Membaca dan Menampilkan Node Pada Tree

Untuk membaca dan menampilkan seluruh node yang terdapat pada pohon biner, terdapat 3 macam cara, atau yang biasa disebut kunjungan (visit). Semua kunjungan diawali dengan mengunjungi akar pohon. Karena proses kunjungan ini memerlukan perulangan proses yang sama namun untuk depth (kedalaman) yang berbeda, maka ketiganya diimplementasikan dengan fungsi rekursif.

a. Kunjungan Pre-Order

Kunjungan pre-order dilakukan mulai dari akar pohon, dengan urutan:

1) Cetak isi (data) node yang sedang dikunjungi
2) Kunjungi kiri node tersebut,
   - Jika kiri bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kiri tersebut.
   - Jika kiri kosong (NULL), lanjut ke langkah ketiga.
3) Kunjungi kanan node tersebut,
   - Jika kanan bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kanan tersebut.
   - Jika kanan kosong (NULL), proses untuk node ini selesai, tuntaskan proses yang sama untuk node yang dikunjungi sebelumnya.

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

b. Kunjungan In-Order

1) Kunjungi kiri node tersebut,
   - Jika kiri bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kiri tersebut.
   - Jika kiri kosong (NULL), lanjut ke langkah kedua.
2) Cetak isi (data) node yang sedang dikunjungi
3) Kunjungi kanan node tersebut,
   - Jika kanan bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kanan tersebut.
   - Jika kanan kosong (NULL), proses untuk node ini selesai, tuntaskan proses yang sama untuk node yang dikunjungi sebelumnya.

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

c. Kunjungan Post-Order

1) Kunjungi kiri node tersebut,
   - Jika kiri bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kiri tersebut.
   - Jika kiri kosong (NULL), lanjut ke langkah kedua.
2) Kunjungi kanan node tersebut,
   - Jika kanan bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kanan tersebut.
   - Jika kanan kosong (NULL), lanjut ke langkah ketiga.
3) Cetak isi (data) node yang sedang dikunjungi. Proses untuk node ini selesai, tuntaskan proses yang sama untuk node yang dikunjungi

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

Variabel **root pada setiap fungsi diatas menunjukkan node mana yang sedang dikunjungi saat ini, untuk itu saat pemanggilan, variabel **root kita beri nilai pointer yang menunjuk ke node akar, yaitu pohon.

III. TUGAS

1. Perbaikilah koding program binary tree di bawah.

```
#include <iostream>
#include <conio.h>
using namespace std
struct *node{
node *kiri;
node *kanan;
};
node *akar=NULL;char data;
node *addNode (node *akar, char isi){
if(akar)==NULL){
node *baru;
baru = new node;
Baru = data -> isi;
baru -> kiri = NULL;
baru -> kanan = NULL;
(akar) = baru
}
```

```
}
node *preOrder (node akar){
if(akar != NULL){
preOrder(akar -> kanan);

cin << " " << akar -> data
preOrder(akar -> kiri);
}
}
node *inOrder (node akar){
if(akar != NULL){
inOrder (akar -> kiri);
cout << " " << akar -> data;
inOrder ( akar = kanan);
}
}
node *postOrder (node akar){
if (akar == NULL){
postOrder (akar -> kanan)

postOrder (akar -> kiri);
cout << " " << akar -> data;
}
}
main (){
int abjad;
cout << "\n\n\t Posisi Awal Tree : \n\n"
cout << "\t \A\n\t /\\\n\t B C\n\t /\n\t
D\n\t /\\\n E F\n\n";
addnode(&akar, abjad = 'A');
addnode(&akar -> kanan, abjad = 'B');
```

```
addnode(&akar -> kiri, abjad = 'C');
addnode(&akar -> kanan -> kiri, abjad = 'D');
addnode(&akar -> kiri -> kanan -> kanan, abjad =
'E');
addnode(&akar -> kiri -> kanan -> kanan, abjad =
'F');
Cout "Tampilan PreOrder :
preOrder(akar);
cout << "\nTampilan InOrder :
inOrder(akar);
cout << "\nTampilan PostOrder :
postOrder(akar);
}
```

2. Buatlah program dengan fungsi untuk menghitung jumlah node keseluruhan pada pohon biner. Penghitungan dilakukan dengan menjelajahi isi pohon, bukan dengan menambahkan counter saat setiap data baru dimasukkan!

3. Buatlah program dengan fungsi untuk mencetak nilai node minimum (terkecil) pada pohon biner!