MODUL 7 TREE
(POHON)

7.1. TREE (POHON)

Struktur pada tree (pohon) tidak linear seperti pada struktur linked list, stack, dan queue. Setiap node pada tree mempunyai tingkatan, yaitu orang tua (parent) dan anak (child). Struktur ini sebenarnya merupakan bentuk khusus dari struktur tree yang lebih umum, setiap orang tua hanya memiliki dua anak sehingga disebut pohon biner (binary tree), yaitu anak kiri dan anak kanan. Untuk lebih jelasnya, dibawah ini akan diuraikan istilah-istilah umum dalam tree.

Predecessor : node yang berada di atas node tertentu

Successor : node yang dibawah node tertentu

Ancestor : seluruh node yang terletak sebelum node tertentu dan terletak sesudah pada jalur yang sama

Descendant : seluruh node yang terletak sesudah node tertentu dan terletak sesudah pada jalur yang sama

Parent : predecessor satu level diatas suatu node

Child : successor satu level dibawah suatu node

Sibling : node-node yang memiliki parent yang sama dengan suatu node

Subtree : bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut

Size : banyaknya node dalam suatu tree

Height : banyaknya tingkatan/level dalam suatu tree

Root : satu-satunya node khusus dalam tree yang tidak mempunyai predecessor

Leaf : node-node dalam tree yang tidak memiliki successor

Degree : banyaknya child yang dimiliki suatu node

7.2. Jenis-jenis Tree

Full Binary Tree

Jenis binary tree ini tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.

Complete Binary Tree

Jenis ini mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda dan setiap node kecuali leaf hanya boleh memiliki 2 child.

Skewed Binary Tree

Skewed Binary Tree adalah Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.

Implementasi Binary Tree

Binary tree dapat diimplementasikan dalam C++ dengan menggunakan double linkedlist.

7.3. Operasi-operasi pada Binary Tree

Create	Membentuk binary tree baru yang masih kosong
Clear	Mengosongkan binary tree yang sudah ada
Empty	Function untuk memeriksa apakah binary tree masih kosong
Insert	Memasukkan sebuah node ke dalam tree. Ada tiga pilihan
insert :	sebagai root, left child, atau right child. Khusus insert sebagai root, tree harus dalam keadaan kosong
Find	Mencari root, parent, left child, atau right child dari suatu node. (tree tidak boleh kosong).
Update	Mengubah isi dari node yang ditunjuk oleh pointer curret (Tree tidak boleh kosong)
Retrieve	Mengetahui isi dari node yang ditunjuk oleh pointer current (Tree tidak boleh kosong)
DeleteSub	Menghapus sebuah subtree (node beserta seluruh descendantnya) yang ditunjuk current. Tree tidak boleh kosong. Setelah itu, pointer current dakan berpindah ke parent dari node yang dihapus.
Characteristic	Mengetahui karakteristik dari suatu tree, yakni: size, height, serta average length. Tree tidak boleh kosong.

Traverse Mengunjungi seluruh node-node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linear yang tersimpan dalam tree. Ada tiga cara traverse,yaitu PreOrder, InOrder, dan PostOrder.

Langkah-langkah Tranverse :

· PreOrder : cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child

· InOrder : kunjungi Left Child, cetak isi node yang dikunjungi, kunjungi Right Child

· PostOrder : kunjungi Left Child, kunjungi Right Child cetak isi node yang dikunjungi.

7.4. Binary Search Tree

Binary Tree ini memiliki sifat dimana semua left child harus lebih kecil dari pada right child dan parentnya. Semua right child juga harus lebih besar dari left child serta parentnya. Binary search tree dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching / pendarian node tertentu dalam binary tree. Pada dasarnya operasi dalam Binary Search Tree sama dengan Binary Tree biasa, kecuali pada operasi insert, update, dan delete.

Insert

Pada Binary Search Tree insert dilakukan setelah lokasi yang tepat ditemukan (lokasi tidak ditentukan oleh user sendiri).

Update

Update ini seperti yang ada pada Binary Tree biasa, namun di sini update akan berpengaruh pada posisi node tersebut selanjutnya. Bila update mengakibatkan tree tersebut bukan Binary Search Tree lagi, harus dilakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi Binary Search Tree.

Delete

Seperti halnya update, delete dalam Binary Search Tree juga turut mempengaruhi struktur dari tree tersebut.

AVL Tree

AVL Tree adalah Binary Search Tree yang memiliki perbedaan tinggi/ level maksimal 1 antara subtree kiri dan subtree kanan. AVL Tree muncul untuk menyeimbangkan Binary Search Tree. Dengan AVL Tree, waktu pencarian dan bentuk tree dapat dipersingkat dan disederhanakan.

Selain AVL Tree, terdapat pula Height Balanced n Tree, yakni Binary Search Tree yang memiliki perbedaan level antara subtree kiri dan subtree kanan maksimal adalah n sehingga dengan kata lain AVL Tree adalah Height Balanced 1 Tree.

Untuk memudahkan dalam menyeimbangkan tree, digunakan simbol-simbol Bantu :

- (tanda minus) : digunakan apabila Subtree kiri lebih panjang dari Subtree kanan.
+ (tanda plus) : digunakan apabila Subtree kanan lebih panjang dari Subtree kiri.
0 (nol) : digunakan apabila Subtree kiri dan Subtree kanan mempunyai height yang sama.

Latihan 1 program tree.cpp

```
#include <stdio.h>
#include <malloc.h>

      struct nod {
             struct nod *left;
             char data;
             struct nod *right;
      };

      typedef struct nod NOD;
      typedef NOD POKOK;

      NOD *NodBaru(char item) {
             NOD *n;
             n = (NOD*) malloc(sizeof(NOD));
             if(n != NULL) {
                          n->data = item;
                          n->left = NULL;
                          n->right = NULL;
             }
             return n;
      }
      void BinaPokok(POKOK **T) {
             *T = NULL;
      }
      typedef enum { FALSE = 0, TRUE = 1} BOOL;

      BOOL PokokKosong(POKOK *T) {
```

```
	return((BOOL)(T == NULL));
}
void TambahNod(NOD **p, char item) {
	NOD *n;
	n = NodBaru(item);
*p = n;
}
void preOrder(POKOK *T) {
	if(!PokokKosong(T)) {
			printf("%c ", T->data);
			preOrder(T->left);
			preOrder(T->right);
	}
}
void inOrder(POKOK *T) {
	if(!PokokKosong(T)) {
			inOrder(T->left);
			printf("%c ", T->data);
			inOrder(T->right);
	}
}
void postOrder(POKOK *T) {
	if(!PokokKosong(T)) {
			postOrder(T->left);
			postOrder(T->right);
			printf("%c ", T->data);
	}
}

int main()
{
	POKOK *kelapa;
	char buah;
	BinaPokok(&kelapa);
	TambahNod(&kelapa, buah = 'M');
	TambahNod(&kelapa->left, buah = 'E');
	TambahNod(&kelapa->left->right, buah = 'I');
	TambahNod(&kelapa->right, buah = 'L');
	TambahNod(&kelapa->right->right, buah = 'O');
	TambahNod(&kelapa->right->right->left, buah =
'D');
	printf("Tampilan secara PreOrder: ");
	preOrder(kelapa);
	printf("\nTampilan secara InOrder: ");
	inOrder(kelapa);
	printf("\nTampilan secara PreOrder: ");
	postOrder(kelapa);
	printf("\n\n");
	return 0;
}
```