JOBSHEET 13

Tree Traversal

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Teknik Informatika	INF1.62.2014	2 x 50 Menit

TUJUAN PRAKTIKUM

1. Mahasiswa memahami konsep penjelajahan data Tree Traversal dan pencarian Binary Tree dalam pemrograman C.
2. Mahasiswa memahami operasi yang ada dalam pencarian Tree Traversal
3. Mahasiswa mampu mengimplementasikan struktur data Tree Traversal inorder, preorder, posrtorder menggunakan pemrograman C dengan IDE.

HARDWARE & SOFTWARE

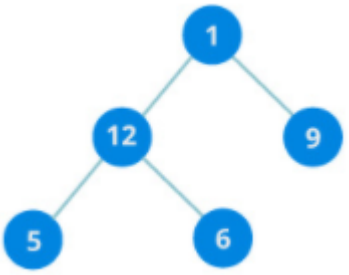
1. Personal Computer
2. DevC++ IDE

TEORI SINGKAT

A. Tree Traversal

Traverse atau Melintasi sebuah pohon data/tree artinya mengunjungi semua simpul yang ada pada pohon tersebut. Ada beberapa hal yang dapat dilakukan contoh jika ingin menambah nilai semua data pada pohon tersebut atau mencari nilai tertinggi diantaranya. Untuk operasi-operasi tersebut, harus dilakukan mengunjungi setiap simpul dari pohon data.

Struktur data linear seperti array, stacks/tumpukan, queues/antrian dan LinkedList hanya memiliki satu jalan untuk membaca sebuah data. Berbeda dengan struktur data yang bentuknya hirakki seperti tree/pohon dapat dijelajahi dengan berbagai cara.



Gambar 1. Contoh Struktur data tree

Cara membaca tree di atas sebagai berikut (dimulai dari atas, kiri, kanan) :

1 -> 12 -> 5 -> 6 -> 9

Sedangkan jika dibaca dari bawah kiri , kanan :

5 -> 6 -> 12 -> 9 -> 1

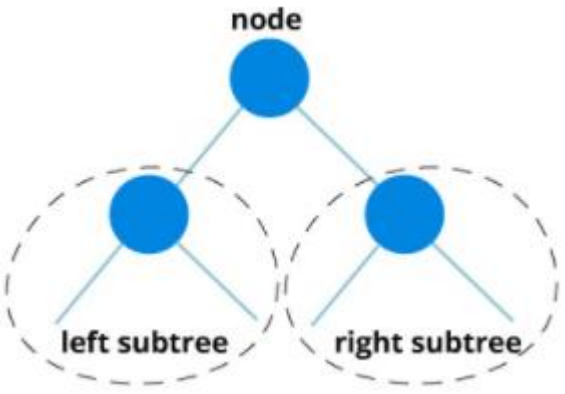
Dalam bahasa pemograman C struktur data dasar dari sebuah tree sebagai berikut :

```
struct node {
    int data;
    struct node* left;
    struct node* right;
}
```

Sebuah simpul struct yang menunjuk/mengarah ke kanan atau ke kiri mungkin memiliki anak di kanan dan kirinya jadi ini dinamakan sub-tree bukan sub-node.

Berdasarkan pernyataan di atas setiap tree adalah kombinasi dari

- Sebuah simpul yang menyimpan data
- 2 sub-tree



Gambar 2. Satu tree memiliki 2 sub-tree di kanan dan kirinya

Tujuan mengunjungi dan mendata setiap simpul/node, yang harus dilakukan adalah mengunjungi semua node/simpul di dalam subtree, mengunjungi root node atau node paling atas dan mengunjungi node yang ada di kanan subtree.

Berdasarkan urutan yang akan dilakukan, terdapat 3 cara melintasi tree tersebut :

1. Inorder Traversal,

   langkah-langkahnya :

   1) Kunjungi semua nodes/simpul di sebelah kiri subtree.
   2) Kemudian kunjungi root node/simpul paling atas
   3) Kunjungi semua simpul yang ada di kanan subtree

```
inorder(root->left)
display(root->data)
inorder(root->right)
```

2. Preorder traversal

   1) Visit root node
   2) Visit all the nodes in the left subtree
   3) Visit all the nodes in the right subtree

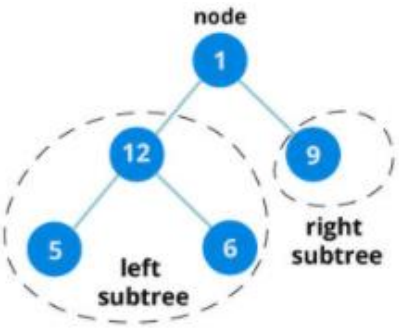
```
display(root->data)
preorder(root->left)
preorder(root->right)
```

3. Postorder Traversal

   1) Visit all the nodes in the left subtree
   2) Visit all the nodes in the right subtree
   3) Visit the root node

```
postorder(root->left)
postorder(root->right)
display(root->data)
```

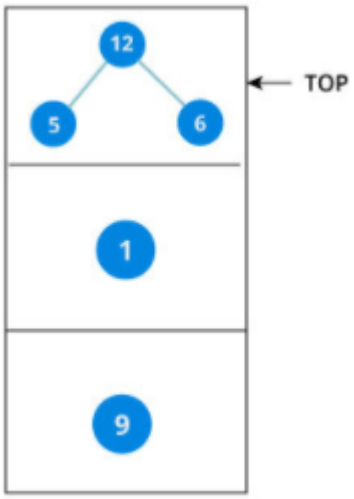
Jika divisualisasikan akan seperti di bawah ini :



Gambar 3. Contoh Visualisasi subtree kiri dan kanan

Jelajahi subtree yang kiri terlebih dahulu. Kemudian kunjungi root node lalu subtree sebelah kanan maka selesai tree traversal yang kita lakukan

Jika digambarkan dalam sebuah tumpukan sebagai berikut :



Gambar 4. Stack 1

Langkah selanjutnya jelajahi subtree yang terletak paling atas tumpukan, menggunakan aturan inorder

```
Left subtree -> root -> right subtree
```

Setelah menjelajah subtree paling atas maka urutan tumpukan menjadi

Gambar 5. Stack 2

Karna node 5 tidak memiliki subtree, maka dapat dicetak langsung. Setelah itu root atau prent 12 dan kemudian anak sebelah kanan 6.

```
5 -> 12 -> 6 -> 1 -> 9
```

We don't have to create the stack ourselves because recursion maintains the correct order for us

PERCOBAAN

1. Tree 1

```
#include <iostream>
#include <conio.h>
#include <stdlib.h> // dibutuhkan untuk system("cls");
#include<iomanip>
using namespace std;

struct tree_node
{
   tree_node* left;
   tree_node* right;
   tree_node* temp;
   tree_node* help;
   tree_node* height;
   string nomor;
   string nama;
```

```
    string nim;
    string email;

};

tree_node* root;

bool isEmpty()
{return root==NULL;}

void insert(string no, string nm,string n,string ml)
{
    tree_node* t = new tree_node;
    tree_node* parent;
    t->nomor = no;
    t->nama = nm;
    t->nim = n;
    t->email= ml;
    t->left = NULL;
    t->right = NULL;
    parent = NULL;
      if(isEmpty())root = t;
      else
      {
         tree_node* curr;
         curr = root;

            while(curr!=NULL)
               {
                  parent = curr;
                     if(t->nomor > curr->nomor){
                        curr = curr->right;
                     }
                     else if(t->nama > curr->nama){
                        curr = curr->right;
                     }
                     else if(t->nim > curr->nim){
                        curr = curr->right;
                     }
                     else if(t->email > curr->email){
                        curr = curr->right;
                     }
                     else curr = curr->left;
               }

      if(t->nomor < parent->nomor){
         parent->left = t;
      }
      else if(t->nama < parent->nama){
         parent->left = t;
      }
      else if(t->nim < parent->nim){
         parent->left = t;
      }
      else if(t->email < parent->email){
```

```
            parent->left = t;
        }
        else
            parent->right = t;
        }
    }

    void inorder(tree_node* p)
    {
        if(p!=NULL)
        {
            if(p->left)
                inorder(p->left);
                cout<<" |"<<p->nomor<<" | "<<p->nama<<"                | "<<p->nim<<"            |   "<<p-
>email<<endl;
                    if(p->right)
                        inorder(p->right);
        }
        else
        return;
    }
    void postorder(tree_node* p)
           {
               if(p!=NULL)
               {
                   postorder(p->left);
                   postorder(p->right);
                   cout<<" |"<<p->nomor<<" | "<<p->nama<<"                | "<<p->nim<<"            |   "<<p-
>email<<endl;
               }

               else
               return;
           }

    void preorder(tree_node* p)
    {
       if(p != NULL)
       {
          cout<<" |"<<p->nomor<<" | "<<p->nama<<"                | "<<p->nim<<"            |   "<<p-
>email<<endl;
          if(p->left)
             preorder(p->left);
          if(p->right)
             preorder(p->right);
       }

       else
          return;
    }

    void print_preorder()
    {
       preorder(root);
    }
```

```
void print_postorder()
{
postorder(root);
}

void print_inorder()
{
inorder(root);
}
int height(tree_node *root)
{
      if(root == NULL)
            return -1;
      else{
            int u = height(root->left);
       int v = height(root->right);
       if(u > v)
         return u + 1;
       else
         return v + 1;
      }
}


int main()
{
root=NULL;
string x, tmp,t,o;
int ch;
int del;

while(1)
{
system("cls");
cout<<endl;
cout<<"Menu Utama Operasi Pohon Biner"<<endl;
cout<<"--------------------"<<endl;
cout<<"1. Insert/Tambah Data"<<endl;
cout<<"2. Kunjungan In-Order"<<endl;
cout<<"3. Kunjungan Pre-Order"<<endl;
cout<<"4. Kunjungan Post-Order"<<endl;
cout<<"5. Exit"<<endl;
cout<<"Pilihan Anda : ";
cin>>ch;
cout<<endl;
switch(ch)
{
case 1 :
cout<<"- NOMOR : ";
cin>>x;
cout<<"- Nama : ";
cin>>tmp;
cout<<"- NIM : ";
cin>>t;
```

```
cout<<"- Email : ";
cin>>o;
insert(x,tmp,t,o);


break;
case 2 : cout<<endl;
cout<<"Kunjungan In-Order"<<endl;
cout<<"---------------"<<endl;
cout<<" ------------------------------------------------------------------------"<<endl;
      cout<<" |No|"<<setw(10)<<"        NAMA "<<setw(15)<<"|"<<setw(5)<<"     NIM
"<<setw(10)<<"|"<<setw(10)<<"e-MAIL"<<setw(10)<<"|";
      cout<<" \n ------------------------------------------------------------------------"<<endl;
print_inorder();getch();
break;
case 3 :
      cout<<" ------------------------------------------------------------------------"<<endl;
      cout<<" |No|"<<setw(10)<<"        NAMA "<<setw(15)<<"|"<<setw(5)<<"     NIM
"<<setw(10)<<"|"<<setw(10)<<"e-MAIL"<<setw(10)<<"|";
      cout<<" \n ------------------------------------------------------------------------"<<endl;

   print_preorder();getch();
break;
case 4 : cout<<endl;
cout<<"Kunjungan In-Order"<<endl;
cout<<"---------------"<<endl;
cout<<" ------------------------------------------------------------------------"<<endl;
      cout<<" |No|"<<setw(10)<<"        NAMA "<<setw(15)<<"|"<<setw(5)<<"     NIM
"<<setw(10)<<"|"<<setw(10)<<"e-MAIL"<<setw(10)<<"|";
      cout<<" \n ------------------------------------------------------------------------"<<endl;
print_postorder();getch();
break;
case 5 : return 0;
break;
default: cout<<"Pilihan yang Anda Masukkan salah!"<<endl;
getch();
break;
}
}
}
```

2. Tree 2

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

struct node {
  int data;

  struct node *leftChild;
  struct node *rightChild;
};

struct node *root = NULL;
```

```
void insert(int data) {
  struct node *tempNode = (struct node*) malloc(sizeof(struct node));
  struct node *current;
  struct node *parent;

  tempNode->data = data;
  tempNode->leftChild = NULL;
  tempNode->rightChild = NULL;

  //if tree is empty
  if(root == NULL) {
    root = tempNode;
  } else {
    current = root;
    parent = NULL;

    while(1) {
      parent = current;

      //go to left of the tree
      if(data < parent->data) {
        current = current->leftChild;

        //insert to the left
        if(current == NULL) {
          parent->leftChild = tempNode;
          return;
        }
      }  //go to right of the tree
      else {
        current = current->rightChild;

        //insert to the right
        if(current == NULL) {
          parent->rightChild = tempNode;
          return;
        }
      }
    }
  }
}

struct node* search(int data) {
  struct node *current = root;
  printf("Visiting elements: ");

  while(current->data != data) {
    if(current != NULL)
      printf("%d ",current->data);

    //go to left tree
    if(current->data > data) {
      current = current->leftChild;
    }
```

```
      //else go to right tree
      else {
        current = current->rightChild;
      }

      //not found
      if(current == NULL) {
        return NULL;
      }
    }

    return current;
}

void pre_order_traversal(struct node* root) {
  if(root != NULL) {
    printf("%d ",root->data);
    pre_order_traversal(root->leftChild);
    pre_order_traversal(root->rightChild);
  }
}

void inorder_traversal(struct node* root) {
  if(root != NULL) {
    inorder_traversal(root->leftChild);
    printf("%d ",root->data);
    inorder_traversal(root->rightChild);
  }
}

void post_order_traversal(struct node* root) {
  if(root != NULL) {
    post_order_traversal(root->leftChild);
    post_order_traversal(root->rightChild);
    printf("%d ", root->data);
  }
}

int main() {
  int i;
  int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

  for(i = 0; i < 7; i++)
    insert(array[i]);

  i = 31;
  struct node * temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }
```

```
    i = 15;
    temp = search(i);

    if(temp != NULL) {
      printf("[%d] Element found.", temp->data);
      printf("\n");
    }else {
      printf("[ x ] Element not found (%d).\n", i);
    }

    printf("\nPreorder traversal: ");
    pre_order_traversal(root);


    printf("\nInorder traversal: ");
    inorder_traversal(root);

    printf("\nPost order traversal: ");
    post_order_traversal(root);

    return 0;
  }
```

3. Tree 3

```
// C program for different tree traversals
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Given a binary tree, print its nodes according to the
  "bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
```

```
    if (node == NULL)
      return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    printf("%d ", node->data);
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    printf("%d ", node->data);

    /* then recur on left sutree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root  = newNode(1);
    root->left          = newNode(2);
    root->right         = newNode(3);
    root->left->left    = newNode(4);
    root->left->right   = newNode(5);

    printf("\nPreorder traversal of binary tree is \n");
```

```
    printPreorder(root);

    printf("\nInorder traversal of binary tree is \n");
    printInorder(root);

    printf("\nPostorder traversal of binary tree is \n");
    printPostorder(root);

    getchar();
    return 0;
}
```

DAFTAR PUSTAKA

1. Kernighan, Brian W, & Ritchie, Dennis M. 1988. The Ansi C Programming Language Second Edition, Prentice-Hall.
2. Cipta Ramadhani. 2015. Dasar Algoritma & Struktur Data. Yogyakarta: ANDI.