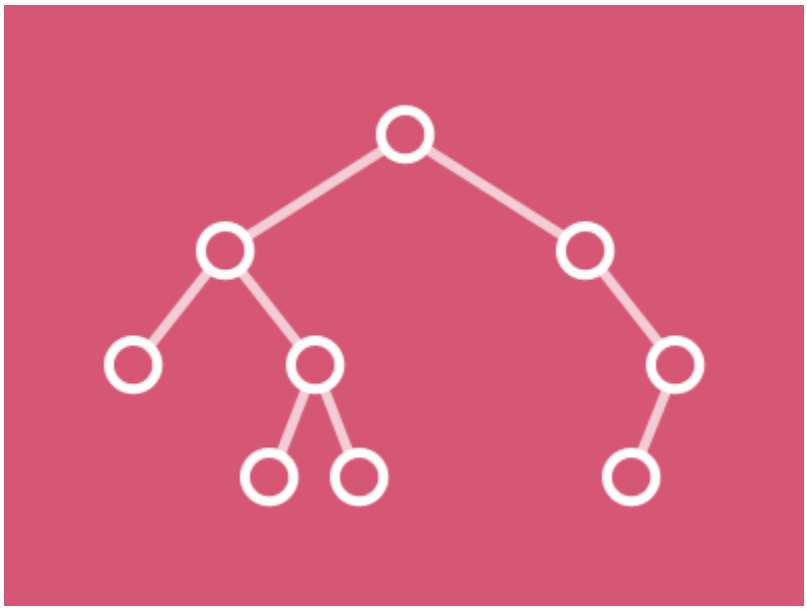
Modul 11

Binary Tree



Praktikum Informasi dan Struktur Data

Semester Genap TA. 2022

Program Studi Informatika

Universitas Atma Jaya Yogyakarta

TUJUAN

- Praktikan mampu memahami konsep dasar dari Binary Tree beserta pengimplementasian nya dalam bentuk kode pada Bahasa Pemrograman C
- Praktikan mampu mengimplementasikan pemahamannya tentang Binary Tree dalam penyelesaian kasus kasus tertentu
- Praktikan mampu memahami dan mengimplementasikan modul-modul sebelumnya dalam modul Binary Tree

MATERI

Tree merupakan sebuah struktur data non-linear yang merepresentasikan struktur hierarki. Dengan kata lain, Tree memiliki ‘atasan’ atau biasa disebut root dan ‘bawahan’ atau biasa disebut child. Tidak seperti Array, Stack, Queue, dan Linked List yang mana termasuk kedalam struktur data linear, Tree memiliki beberapa kelebihan karena sifatnya yang non-linear, namun kita akan bahas mengenai beberapa istilah-istilah yang ada pada tree terlebih dahulu.

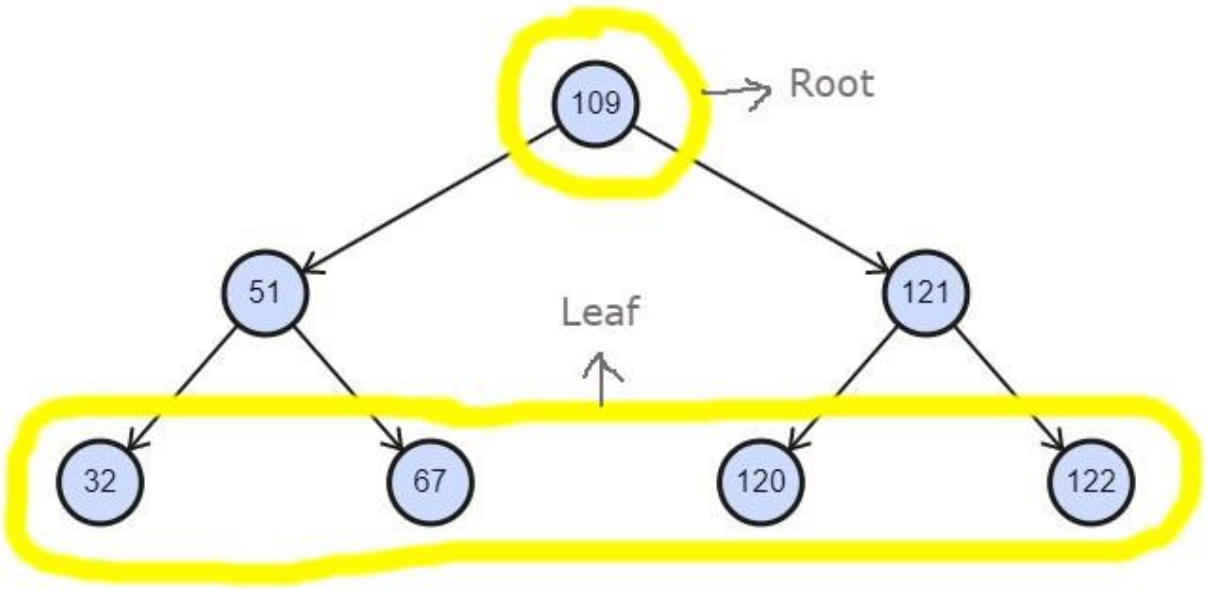
- Root, merupakan sebuah node teratas dari sebuah Tree yang mana tidak memiliki Parent Node dan setiap Tree hanya memiliki satu buah Root. (Lihat gambar 1.1)
- Edge, merupakan sebuah node yang memiliki Parent Node dan Child Node dan berfungsi untuk menghubungkan keduanya. (Lihat gambar 1.2)
- Leaf, merupakan sebuah node yang tidak memiliki Child Node dan bisa dikatakan bahwa Leaf merupakan node terbawah dari suatu Tree. Setiap Tree pasti memiliki minimal 1 Leaf. (Lihat gambar 1.1)
- Subtree, dapat dikatakan sebagai Child Node dari sebuah node. (Lihat gambar 1.2)
- Supertree, adalah Parent Node dari sebuah Subtree atau dengan kata lain merupakan Parent Node dari sebuah Child Node.
- Depth, merupakan jarak dari Root ke sebuah node tertentu. (Lihat gambar 1.3)
- Height, merupakan jarak dari Root ke jalur yang memiliki Edge terbanyak. (Lihat gambar 1.4)
- Weight, merupakan jumlah Leaf dari sebuah Tree.

Binary Tree merupakan turunan dari konsep Tree yang mana pembedanya adalah setiap node pada Binary Tree hanya dapat memiliki maksimal 2 Child Node. Adapun jenis Binary Tree yang akan digunakan pada Praktikum Informasi dan Struktur Data - Modul Binary Tree adalah Binary Search Tree yang mana memiliki sifat yang berpola, yakni seluruh Left Child harus memiliki nilai yang lebih kecil daripada Parent Node dan Right Child.

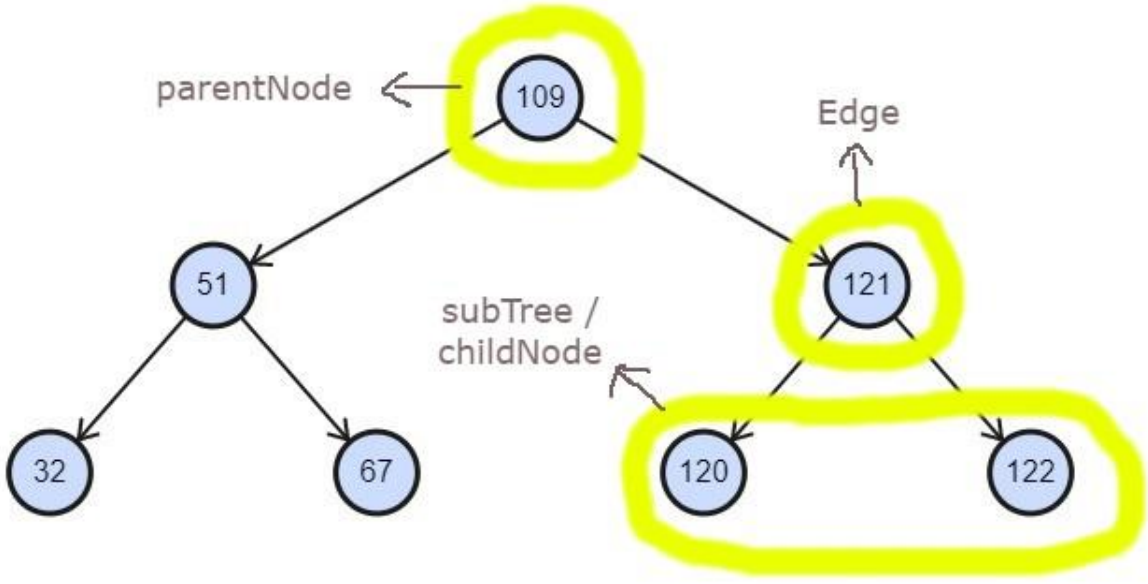
Traversal merupakan cara untuk menelusuri setiap node yang ada pada Binary Search Tree dengan mengimplementasikan konsep Recursive sehingga perulangan yang dilakukan menjadi semakin sederhana. Adapun beberapa jenis Traversal adalah,

- Pre-Order: `Root – Left Child – Right Child`
    1. Root
    2. Kunjungi Left Child secara Pre-Order
    3. Kunjungi Right Child secara Pre-Order
- In-Order: `Left Child – Root – Right Child`
    1. Kunjungi Left Child secara In-Order
    2. Root
    3. Kunjungi Right Child secara In-Order
- Post-Order: `Left Child – Right Child – Root`
    1. Kunjungi Left Child secara Post-Order
    2. Kunjungi Right Child secara Post-Order
    3. Root

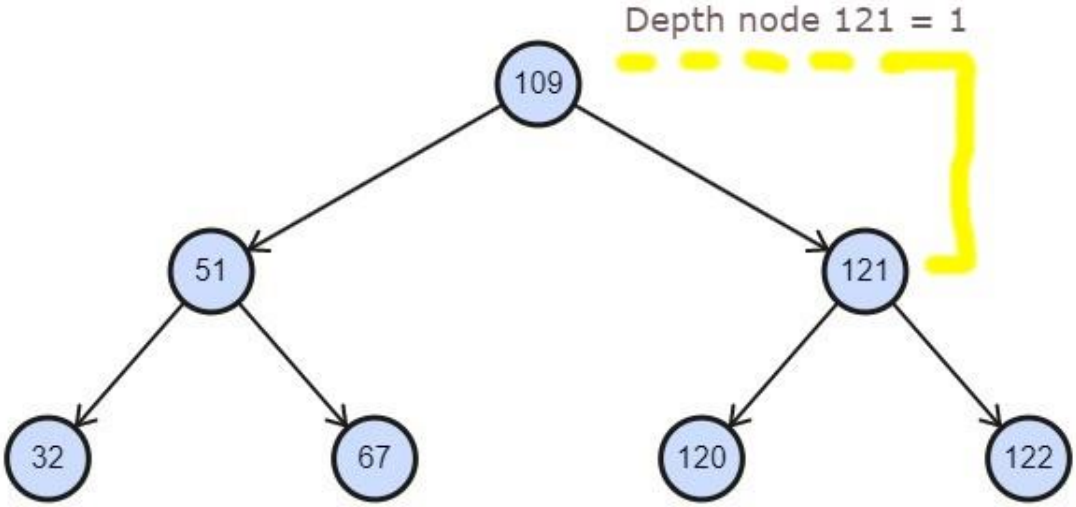
GAMBAR – GAMBAR



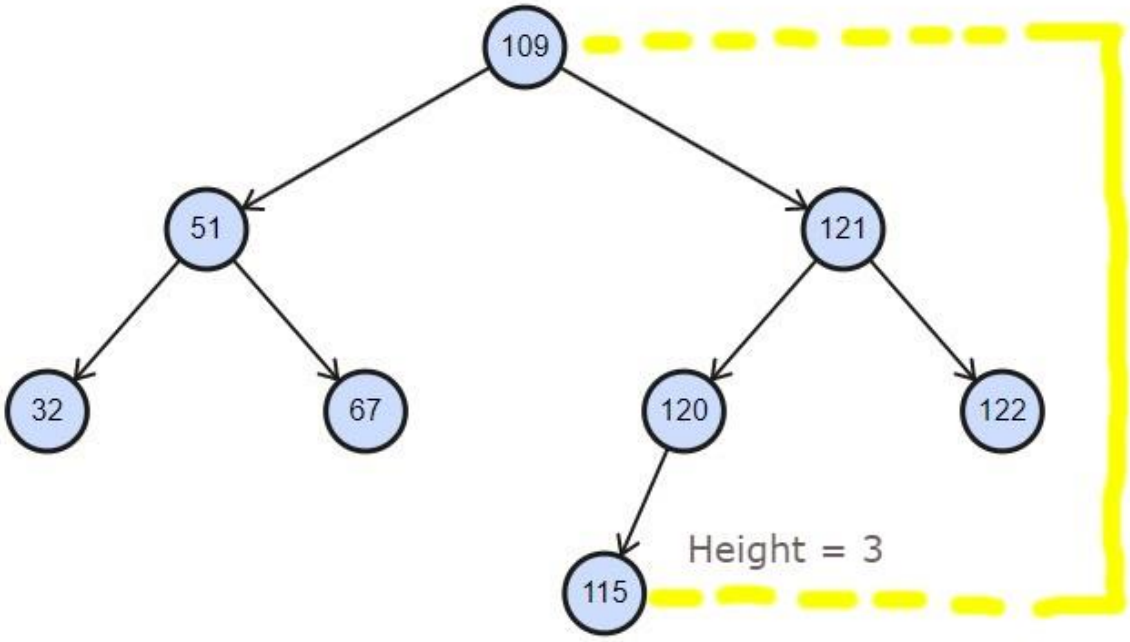
Gambar 1. 1



Gambar 1. 2



Gambar 1. 3



Gambar 1. 4

GUIDED

BINARY TREE

Code pada header.h

```c
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<windows.h>
#include<stdlib.h>
#include <stdbool.h>

typedef struct Node* address;
typedef struct Node* BinaryTree;
typedef char string[100];

typedef struct{
    string jenisPokemon;
    int hp;
}Pokemon;

typedef struct Node{
   Pokemon pokemon;
   address Left;
   address Right;
}Node;

void createEmpty(BinaryTree *BT);

bool isEmpty(BinaryTree BT);
bool isLeaf (BinaryTree BT);
bool isFound (BinaryTree BT,string jenisPokemon);
```

```c
address alokasi(int health, string jenisPokemon);

void insTreeBST (BinaryTree *BT, address P);
void DelTreeAt(BinaryTree *BT, string jenisPokemon);
void DelTree(BinaryTree *BT);

void preOrder (BinaryTree BT);
void inOrder (BinaryTree BT);
void postOrder(BinaryTree BT);

void updateTree(BinaryTree *BT, string jenisPokemon, string update);
```

Code pada main.c

```c
#include "header.h"

int main(int argc, char *argv[])
{
    BinaryTree BT_BST;
    createEmpty (&BT_BST);

    int menu, health;
    string jenisPokemon, ubah, hapus;
    address P, cari;
    do{
        system("cls");
        printf("---- GUIDED BINARY TREE NAMA/NPM ----");
        printf("\n1. Insert data");
        printf("\n2. Print data");
        printf("\n3. Update data");
        printf("\n4. Delete data");
        printf("\n>>> "); scanf("%d", &menu);
        switch(menu){
            case 1:
                printf("Masukkan jenis pokemon: "); fflush(stdin); gets(jenisPokemon);
                printf("Masukkan health pokemon: "); scanf("%d", &health);
                P = alokasi(health, jenisPokemon);
                insTreeBST(&BT_BST, P);
                break;
```

```c
            case 2:
                if(isEmpty(BT_BST)){
                    printf("\nBinary Tree kosong [!]");
                } else{
                    printf("\nPre Order:");
                    preOrder(BT_BST);
                    printf("\n\nIn Order: ");
                    inOrder(BT_BST);
                    printf("\n\nPost Order: ");
                    postOrder(BT_BST);
                    printf("\n\n\n\n");
                }
                break;

            case 3:
                if(isEmpty(BT_BST)){
                    printf("\nBinary Tree kosong [!]");
                } else{
                    printf("\nMasukkan jenis pokemon yang ingin di ubah : "); fflush(stdin); gets(ubah);
                    if(isFound(BT_BST, ubah)){
                        printf("\nMasukkan jenis pokemon yang baru : "); fflush(stdin); gets(jenisPokemon);
                        updateTree(&BT_BST, ubah, jenisPokemon);
                        printf("\nBerhasil mengubah jenisPokemon ");
                    } else{
                        printf("\nNode tidak ditemukan [!]");
                    }
                }
                break;
```

```
            case 4:
                printf("Masukkan jenis pokemon yang ingin dihapus: "); fflush(stdin); gets(hapus);
                if(isFound(BT_BST, hapus)){
                    DelTreeAt(&BT_BST, hapus);
                    printf("\nBerhasil menghapus node");
                } else{
                    printf("\nNode tidak ditemukan [!]");
                }
                break;

            default:
                printf("\nMenu tidak tersedia [!]");
                break;
        }
        getch();
    } while(menu != 0);
    return 0;
}
```

Code pada source.c

```
#include "header.h"

void createEmpty (BinaryTree *BT)
{
    (*BT) = NULL;
}
bool isEmpty(BinaryTree BT)
{
    return (BT == NULL);
}
bool isLeaf (BinaryTree BT)
{
    return( isEmpty (BT->Left) && isEmpty (BT->Right) );
}

bool isFound (BinaryTree BT, string jenisPokemon)
{
    if(isEmpty(BT))
        return false;
    else
        return ((strcmpi(BT->pokemon.jenisPokemon, jenisPokemon) == 0) ||
        (isFound (BT->Left,jenisPokemon)) || (isFound (BT->Right,jenisPokemon)));
}
```

```
address alokasi(int health, string jenisPokemon){
    address P = NULL;
    P = (Node*) malloc (sizeof (Node));
    if(P == NULL)
    {
        return NULL;
    }else
        strcpy(P->pokemon.jenisPokemon, jenisPokemon);
        P->pokemon.hp = health;
        P->Left = P->Right = NULL;
        return P;
}

void insTreeBST(BinaryTree *BT, address P)
{
    if (isEmpty((*BT)))
        (*BT) = P;
    else
    {
        if (P->pokemon.hp < (*BT)->pokemon.hp)
            insTreeBST(&(*BT)->Left, P);
        else
            insTreeBST(&(*BT)->Right, P);
    }
}
```

```c
void preOrder (BinaryTree BT){
    if (!isEmpty(BT)){
        printf("%d - ", BT->pokemon.hp);
        preOrder (BT->Left);
        preOrder (BT->Right);
    }
}
void inOrder (BinaryTree BT){
    if (!isEmpty(BT)){
        inOrder (BT->Left);
        printf("%d - ", BT->pokemon.hp);
        inOrder (BT->Right);
    }
}
void postOrder (BinaryTree BT){
    if (!isEmpty(BT)){
        postOrder (BT->Left);
        postOrder (BT->Right);
        printf("%d - ", BT->pokemon.hp);
    }
}

void deleteLeaf(address *P){
    if(isLeaf(*P)){
        address A=(*P);
        (*P) = NULL;
        free(A);
    }
}
```

```c
void DelTree(BinaryTree *BT){
    if(isEmpty(*BT)){
        return;
    } else{
        DelTree(&(*BT)->Left);
        DelTree(&(*BT)->Right);
        deleteLeaf(&(*BT));
    }
}
void DelTreeAt(BinaryTree *BT, string jenisPokemon){
    if(strcmpi((*BT)->pokemon.jenisPokemon, jenisPokemon) == 0){
        DelTree(&(*BT));
    }else{
        if(isFound((*BT)->Left, jenisPokemon)){
            DelTreeAt(&(*BT)->Left, jenisPokemon);
        } else{
            DelTreeAt(&(*BT)->Right, jenisPokemon);
        }
    }
}
void updateTree(BinaryTree *BT, string jenisPokemon, string update){
    if(strcmpi((*BT)->pokemon.jenisPokemon, jenisPokemon) == 0){
        strcpy((*BT)->pokemon.jenisPokemon, update);
        return;
    }else{
        if(isFound((*BT)->Left, update)){
            updateTree(&(*BT)->Left, jenisPokemon, update);
        } else{
            updateTree(&(*BT)->Right, jenisPokemon, update);
        }
    }
}
```

Referensi

https://www.programiz.com/dsa/binary-tree

https://www.geeksforgeeks.org/introduction-to-binary-tree-data-structure-and-algorithm-tutorials/

https://www.javatpoint.com/binary-tree

https://www.thegeekstuff.com/2013/02/c-binary-tree/

https://www.educba.com/binary-tree-program-in-c/