RESUME MATERI

Tree Manipulation

Mata Kuliah	:	Struktur Data
Dosen Pengampu	:	Andi Moch Januriana, ST., M.Kom

NAME : Mustopa

NIM : 3337220023

KELAS : C

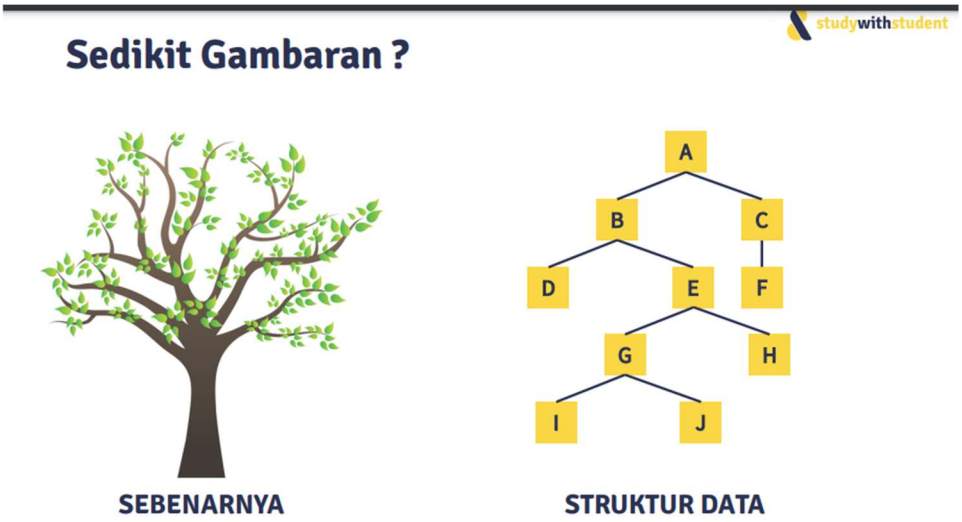
Program Studi Informatika

Fakultas Teknik

Universitas Sultan Ageng Tirtayasa

BINARY TREE PADA STRUKTUR DATA

TREE (POHON)



KOMPONEN TREE

- Node (Simpul)
- Predecessor (Pendahulu)
- Successor (Penerus)
- Ancestor (Leluhur)
- Descendant (Keturunan)
- Parent (Orang tua)
- Child (Anak)
- Sibling (Saudara)
- Subtree
- Size
- Height
- Root (Akar)
- Leaf (Daun)
- Degree

- Forest (Hutan)

- Depth (Kedalaman)

BINARY TREE

- Binary adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree harus terpisah.

OPERASI PADA TREE

- **Create** : digunakan untuk membentuk binary tree baru yang masih kosong.

- **Clear** : digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.

- **Empty** : digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.

- **Insert** : digunakan untuk memasukkan sebuah node kedalam tree.

- **Find** : digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.

- **Update** : digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

- **Retrieve** : digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

- **Delete** Sub : digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

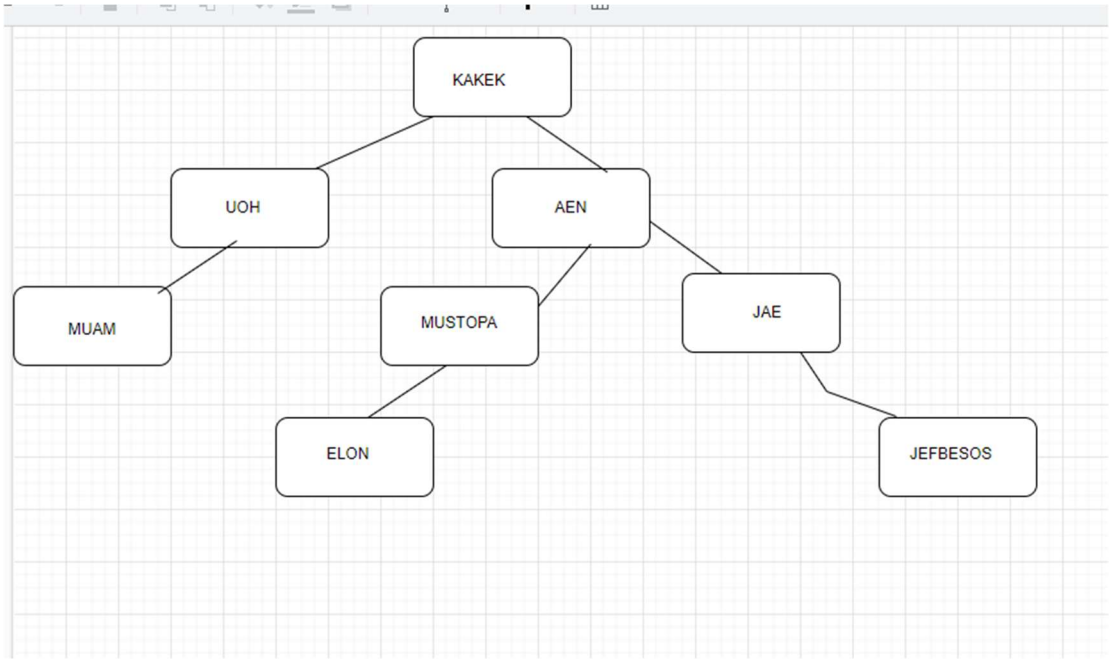
- **Charateristic** : digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average

lenght-nya.

- **Tranverse** : digunakan untuk mengunjungi seluruh node-node pada tree dengan cara tranversal.

IMPLEMENTASI DALAM BAHASA C++

DISINI SAYA MENGGUNAKN STRUKTUR DATA TREE KELUARGA SEBAGAI CONTOH



DEKLARASI

```
// DEKLARASI TREE

struct Keluarga
{
    string data;
    Keluarga *left, *right, *parent;
};
Keluarga *root, *newNode;
```

Membuat struct dengan atribut string data dan varibel pointer local dan global

FUNGSI CREATE

```
// MEMBUAT TREEE
void createTree(string data)
{
    root = new Keluarga();
    root->data = data;
    root->left = NULL;
    root->right = NULL;
    root->parent = NULL;
}
```

Root left,parent right menunjuk ke NULL karena data awal

FUNGSI ISEMPTY

```
// mengecek apakah tree kosong atau tidak
bool isEmpty()
{
    if (root == NULL)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

Jika root == null (kosong)

Return false

FUNGSI INSERTRIGHT

```
// tambah kanan
Keluarga *insertright(string data, Keluarga *bapak)
{
    if (root == NULL)
    {
        cout << "Buat Tree terlebih Dahulu!!!";
        return NULL;
    }
    else
    {
        if (bapak->right != NULL)
        {
            return NULL;
        }
        else
        {
            newNode = new Ke
            newNode->data = data;
            newNode->left = NULL;
            newNode->right = NULL;
            newNode->parent = bapak;
            bapak->right = newNode;
            cout << "\n " << data << " Berhasil di tambahkan Ke Anak Kanan " << newNode->parent->data << endl;
            return newNode;
        }
    }
}
```

std::string data

tambah kanan

Newnode left dan rigt = NULL artinya kosong

Newnode ->parent = bapak artinya parent dari new node bapak

FUNGSI INSERLEFT

```
// tambah kiri
Keluarga *insertkiri(string data, Keluarga *bapak)
{
    if (root == NULL)
    {
        cout << "Buat Tree terlebih Dahulu!!!";
        return NULL;
    }
    else
    {
        if (bapak->left != NULL)
        {
            return NULL;
        }
        else
        {
            newNode = new Keluarga();
            newNode->data = data;
            newNode->left = NULL;
            newNode->right = NULL;
            newNode->parent = bapak;
            bapak->left = newNode;
            cout << "\n " << data << " Berhasil di tambahkan Ke Anak kiri " << newNode->parent->data << endl;
            return newNode;
        }
    }
}
```

Sama seperti right tinggal ganti left / right

FUNGSI UPDATE

```
// Mengedit data tree yang diinginkan
void UpdateTree(string newdata, Keluarga *pilihan)
{
    if (root == NULL)
    {
        cout << "Buat Tree terlebih Dahulu!!!";
    }
    else
    {
        string bantu = pilihan->data;
        pilihan->data = newdata;
        cout << "data " << bantu << " Berhasil di ubah Menjadi : " << newdata << endl;
    }
}
```

Disini mebuat parameter newdata dan variable pointer pilihan(buat memilih node yang ingin di update)

Kemudian tinggal ubah isi datanya

FUNGSI RETRIVE

```
// mengetahui isi data dari node yang di tunjuk
void retrive(Keluarga *node)
{
    if (root == NULL)
    {
        cout << "Buat Tree terlebih Dahulu!!!";
    }
    else
    {
        if (node == NULL)
        {
            cout << "node yang ditunjuk tidak ada!!!";
        }
        else
        {
            cout << "data Node : " << node->data << endl;
        }
    }
}
```

FUNGSI FIND

```
// mencari root , parent ,anakkanan , anak kiri, saudara ,  dari node yang di tunjuk
void find(Keluarga *node)
{
    if (root == NULL)
    {
        cout << "Buat Tree terlebih Dahulu!!!";
    }
    else
    {
        cout << " \ndata Node :" << node->data;
        cout << " \nroot Node :" << root->data;

        if (node->parent == NULL)
        {
            cout << "  \nTidak Punya orang tua" << endl;
        }
        else
        {
            cout << "\nParent Node :" << node->parent->data;
        }
        if (node->right == NULL)
        {
            cout << " \n Tidak Punya Anak Kanan" << endl;
        }
        else
        {
            cout << "\nAnak kanan Node :" << node->right->data;
        }
        if (node->left == NULL)
        {
            cout << " \n Tidak Punya Anak kiri" << endl;
        }
        else
        {
            cout << "\nAnak Kiri Node :" << node->left->data;
        }
        if (node->parent != NULL && node->parent->left != node && node->parent->right
 == node)
        {
            cout << " \nSaudara : " << node->parent->left->data << endl;
        }
        else if (node->parent != NULL && node->parent->left == node && node->parent->
right != node)
        {
            cout << "\n Saudara : " << node->parent->right->data << endl;
        }
        else
        {
            cout << "\n Tidak Punya saudara !!!\n";
        }
    }
}
```

FUNGSI TRANVERSAL

```
// mengunjungi node-node pada tree dengan cara tranversal ( preeorder)

void preorder(Keluarga *node = root)
{
    if (root == NULL)
    {
    }
    else
    {

        if (node != NULL)
        {
            cout << node->data << ", ";
            preorder(node->left);
            preorder(node->right);
        }
    }
}

// mengunjungi node-node pada tree dengan cara tranversal ( inorder)

void inOrder(Keluarga *node = root)
{
    if (root == NULL)
    {
        cout << "\nBuat Tree terlebih Dahulu!!!\n";
    }
    else
    {

        if (node != NULL)
        {
            inOrder(node->left);
            cout << node->data << ", ";
            inOrder(node->right);
        }
    }
}
// mengunjungi node-node pada tree dengan cara tranversal ( PostOrder)

void PostOrder(Keluarga *node = root)
{
    if (root == NULL)
    {
        cout << "\nBuat Tree terlebih Dahulu!!!\n";
    }
    else
    {

        if (node != NULL)
        {
            PostOrder(node->left);
            PostOrder(node->right);
            cout << node->data << ", ";
        }
    }
}
```

FUNGSI DELETE

```cpp
// menghapus node beserta subtree nya (turunan di bawahnya)
void deleteTree(Keluarga *node)
{

    if (!root)
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else
    {

        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);

            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {

                delete node;
                cout << "\nnode berhasil di hapus";
            }
        }
    }
}

// menghapus subtree node beserta turunan di bawahnya (tidak dengan node nya sendiri)
void deleteSub(Keluarga *node)
{
    if (!root)
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nSubtree node " << node->data << " berhasil dihapus." << endl;
    }
}
```

FUNGSI CLEAR

```cpp
// clear
void clear()
{
    if (!root)
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\nTree berhasil dihapus." << endl;
    }
}
```

FUNGSI CHARATERISTIC (untuk melihat size , tinggi , dan rata rata)

```cpp
int size(Keluarga *node = root)
{
    if (root == NULL)
    {

        return 0;
    }
    else
    {
        if (node == NULL)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->right) + size(node->left);
        }
    }
}

int height(Keluarga *node = root)
{
    if (root == NULL)
    {

        return 0;
    }
    if (node == NULL)
    {
        return 0;
    }
    else
    {
        int heightkiri = height(node->left);
        int heightKanan = height(node->right);
        if (heightkiri >= heightKanan)
        {
            return heightkiri + 1;
        }
        else
        {
            return heightKanan + 1;
        }
    }
}

// karakteristik untuk meliat size , tinggi , dan rata rata
void charateristic()
{
    cout << "\nSize Tree : " << size() << endl;
    cout << "Height Tree : " << height() << endl;
    cout << "Average Node of Tree : " << size() / height() << endl;
}
```

INT MAIN

```
int main()

{
    createTree("Kakek");
    Keluarga *Naen, *Nuoh, *Nmuam, *Nmustopa, *Nelon, *Njae, *Njef;

    Nuoh = insertkiri("uoh", root);
    Nmuam = insertkiri("MUAM", Nuoh);
    Naen = insertright("aen", root);
    Njae = insertright("JAE", Naen);
    Njef = insertright("JEFBESOS", Njae);
    Nmustopa = insertkiri("MUSTOPA", Naen);
    Nelon = insertkiri("ELON", Nmustopa);

    UpdateTree("maryam", Naen);
    retrive(Naen);
    cout << "\n\nhasil find TREE : \n";
    find(root);
    cout << "hasil preorder TREE : \n";
    preorder(root);

    cout << "\n\n karakteristik : ";
    charateristic();

    deleteTree(root);
    preorder(root);
    cout << "\n\n karakteristik : ";
    charateristic();
    cout << "\n\n\nsize TREE :" << size();
}
```

OUTPUT

```
 uoh Berhasil di tambahkan Ke Anak kiri Kakek
 MUAM Berhasil di tambahkan Ke Anak kiri uoh
 aen Berhasil di tambahkan Ke Anak Kanan Kakek
 JAE Berhasil di tambahkan Ke Anak Kanan aen
 JEFBESOS Berhasil di tambahkan Ke Anak Kanan JAE
 MUSTOPA Berhasil di tambahkan Ke Anak kiri aen
 ELON Berhasil di tambahkan Ke Anak kiri MUSTOPA
data aen Berhasil di ubah Menjadi : maryam
RETRIVE  Node DATA  : maryam


hasil find TREE :
data Node :Kakek
root Node :Kakek
Tidak Punya orang tua
Anak kanan Node :maryam
Anak Kiri Node :uoh
 Tidak Punya saudara !!!
hasil preorder TREE :
Kakek, uoh, MUAM, maryam, MUSTOPA, ELON, JAE, JEFBESOS,

 karakteristik :
Size Tree : 8
Height Tree : 4
Average Node of Tree : 2

node berhasil di hapus
node berhasil di hapus

 karakteristik :
Size Tree : 0
Height Tree : 0
```

FULL SOURCE CODE : https://github.com/mustopa17/Strukturdata