7_1-binaryTree

1 Tree dan Binary Tree

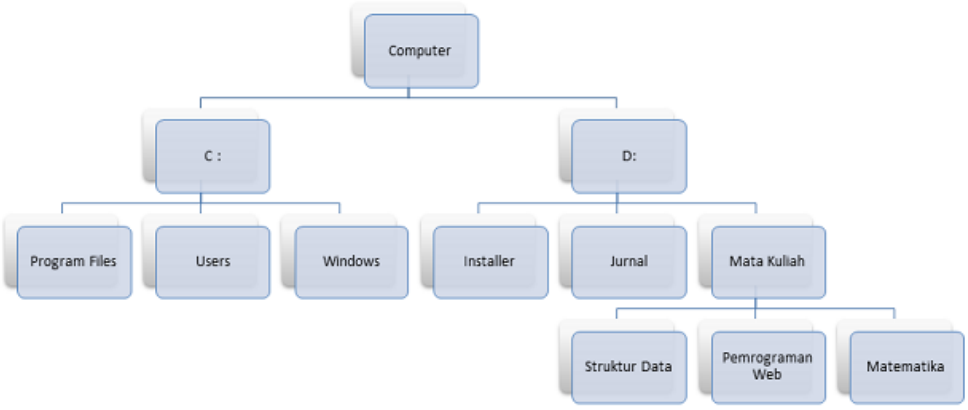
Konsep dasar Tree dan Binary Tree ***

Tree merupakan struktur data yang berbentuk non linear, berbeda dengan struktur data *stack*, *queue*, ataupun *deque*. Bentuk *Tree* di *computer science* hampir sama dengan *tree* yang terdapat pada ekosistem biologi, yaitu keduanya memiliki *root*, *leaves*, dan *branches*. Hanya saja, jika pada ekosistem biologi, akar terletak di bagian paling bawah dan daun ataupun batang terletak diatasnya, maka *tree* pada computer science agak sedikit berbeda, yaitu, akar terletak pada bagian paling atas, sedangkan daun dan cabangnya berada di bawahnya.

Deskripsi mengenai Tree ini akan dibagi menjadi dua bagian, yaitu: 1. Section 1.1 2. Section 1.2

1.1 Tree

Pada *computer science*, pencarian solusi dimulai dari root, bergerak ke bawah mengikuti cabangnya atau *edge*, sampai dengan *leaves* atau daun. Informasi yang terdapat pada daun inilah yang merupakan solusi yang dicari. Contoh *tree* dapat dapat dilihat pada Gambar 1 berikut:

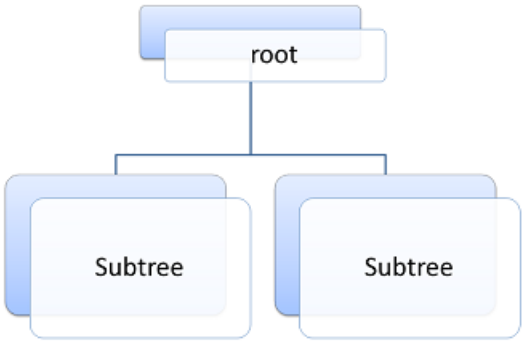


Gambar 1. Bentuk Tree Pada Gambar 1 tersebut, *tree* yang menggambarkan directory pada suatu computer. Jika ingin menuju folder 'struktur data', maka harus mengikuti path sebagai berikut : computer, D:, Mata Kuliah, dan terakhir Struktur Data.

Beberapa istilah pada *Tree* yang harus diketahui adalah sebagai berikut : - **Node**, merupakan bagian dari tree yang menyimpan informasi atau juga disebut sebagai *key* - **Edge**, yang menghubungkan antara node yang satu dengan yang lain. Hanya ada satu edge yang masuk ke

dalam node, dan satu atau lebih edge yang keluar dari suatu node - **Root**, node yang terletak di paling atas, tidak ada edge yang masuk ke dalam node root ini, tapi root mempunyai satu atau lebih dari satu edge yang keluar. - **path**, merupakan urutan node dari root sampai tujuan, misalkan computer --> D: --> Mata Kuliah --> Struktur Data - **children**, merupakan node-node yang memiliki edge yang masuk dari node yang sama. Misalkan program files, users, dan windows merupakan children, krn memiliki edge masuk yang sama-sama berasal dari node C: - **Parent**, node yang memiliki edge yang keluar dari node tersebut, misalkan node C: adalah parent dari program files, users, dan windows - **Siblings**, adalah node-node yang berasal dari parent yang sama - **subtree**, kumpulan dari node dan edge yang terdiri dari parent dan semua node setelah parent - **leaf node**, node yang tidak memiliki edge yang keluar, node yang tidak memiliki children - **level**, tingkatan dari node. Root merupakan level ke-0, semakin kebawah maka level dari node tersebut semakin besar, misalkan struktur data berada di level ke-3 - **height**, merupakan tinggi tree, nilai dari height ini adalah level maksimum dari tree

Suatu *tree* juga dapat dilihat sebagai suatu struktur data yang memiliki dua buah komponen, yaitu *root* dan *subtree*, dimana sebuah tree dapat terdiri dari sebuah root dan nol atau lebih dari satu subtree. *Subtree* sendiri juga merupakan sebuah tree. Root dari subtree dihubungkan dengan edge ke root dari tree. Hal ini dapat dipandang sebagai definisi rekursif. Definisi ini dapat dilihat pada Gambar 2, sebagai berikut :



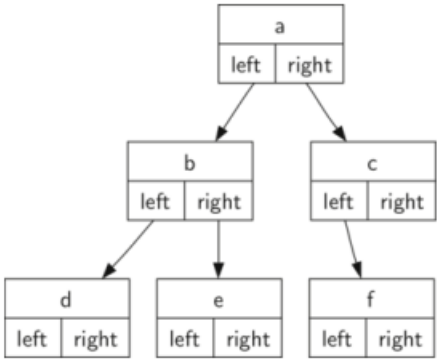
Gambar 2. Tree dan Subtree

Section 1

1.2 Binary Tree

Binary tree merupakan *tree* dimana setiap *parent* hanya memiliki dua buah atau satu *node children* saja.

Karena didalam binary tree terdiri node dimana setiap node memiliki informasi mengenai nilai node dan edge (method untuk menambahkan node baru pada tree), maka *tree* ataupun *binary tree* ini akan lebih mudah diimplementasikan dengan menggunakan tipe data class.
Representasi binary tree dapat dilihat pada Gambar 3 berikut.



Gambar 3. Representasi Binary Tree

1.3 Code

Berikut implementasi class *binary Tree*, dimana pada class ini terdapat tiga buah *property* atau *state*, yaitu *key*, *leftChild*, dan *rightChild*. *Key* digunakan untuk informasi nilai yang terdapat pada *node*, *lefchild* dan *rightchild* digunakan sebagai pointer untuk menunjukkan *child* sebelah kiri dan kanan dari *node*.

```
In [1]: class BinaryTree:
            def __init__(self, root):
                self.key = root
                self.leftChild = None
                self.rightChild = None

In [2]: myTree=BinaryTree(10)
        print(myTree.leftChild)

None
```

1.4 Code

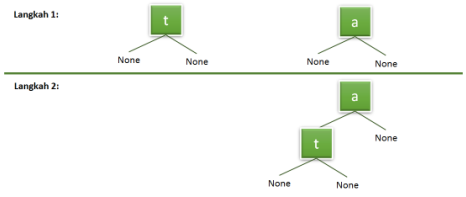
Pada class yang sudah dibuat tersebut, terdapat suatu *constructor*, yaitu node akan dibuat setiap sebuah object didefinisikan dengan menggunakan tipe data class BinaryTree ini. Class BinaryTree ini harus memiliki dua method utama yaitu, **insertLeft** dan **insertRight**. Pada method insertLeft ini, sebuah subtree akan dibuat, dan dilekatkan terhadap tree utama dari edge root yang sebelah kiri, begitu juga insertRight.

```
In [3]: class BinaryTree:
            def __init__(self, root):
                self.key = root
                self.leftChild = None
                self.rightChild = None
            def insertLeft(self, new_node):
                if self.leftChild == None:
                    self.leftChild = BinaryTree(new_node)
```

```
        else:
            t = BinaryTree(new_node)
            t.leftChild = self.leftChild
            self.leftChild = t
    def insertRight(self, new_node):
        if self.rightChild == None:
            self.rightChild = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.rightChild = self.rightChild
            self.rightChild = t
```

Pada method *insertLeft* tersebut perlu dilakukan pemeriksaan terlebih dahulu, apakah child kiri dari node yang akan disisipi subtree baru masih kosong, jika iya maka subtree tersebut dimasukkan pada leftChild. Ilustrasi *insertLeft* pada suatu binary tree yang pointer leftchild masih kosong, dapat dilihat pada Gambar 3. Misalkan tree 't' akan disisipkan pada tree 'a', dimana leftchild dari tree a, masih kosong, sehingga dapat langsung diperintahkan :

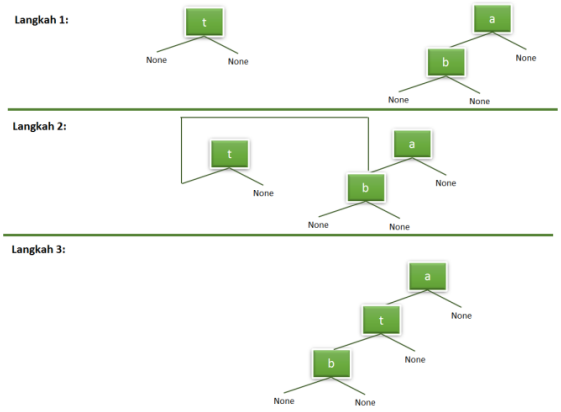
```
self.leftChild=BinaryTree(new_node)
```



Gambar 3. InsertLeft pada tree dimana leftchild tree masih kosong

Jika penyisipan dilakukan pada suatu binary tree, dimana leftchild tree tersebut tidak kosong. Misalkan terdapat tree 'a', dimana tree 'a' tersebut, memiliki leftchild, yaitu 'b'. Jika ingin dilakukan penyisipan tree baru, yaitu 't', maka tahapan yang harus dilakukan adalah :

```
1. t.leftChild = self.leftChild
2. self.leftChild = t
```



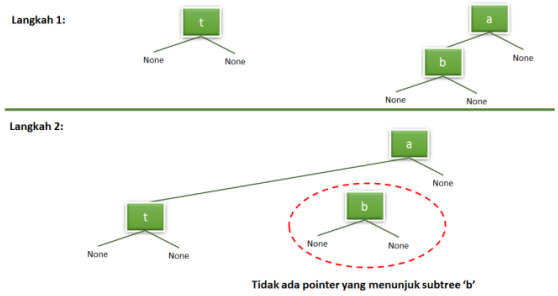
Ilustrasi penyisipan ini dapat dilihat pada Gambar 4.

Gambar 4. InsertLeft pada tree dimana tree tersebut memiliki leftchild

Perintah pada baris pertama digunakan agar node yang terletak pada left child tetap ada dan tidak hilang. Jika perintah tersebut dibalik menjadi :

```
1. self.leftChild = t
2. t.leftChild = self.leftChild
```

Maka node yang merupakan leftchild dari node utama akan hilang, karena tidak ada pointer yang menunjuk ke leftchild tersebut. Ilustrasi ini dapat dilihat pada Gambar 5.



Gambar 5. Kesalahan syntax pada method insertLeft

Penyisipan melalui right child juga melalui tahapan yang sama.

1.5 Code

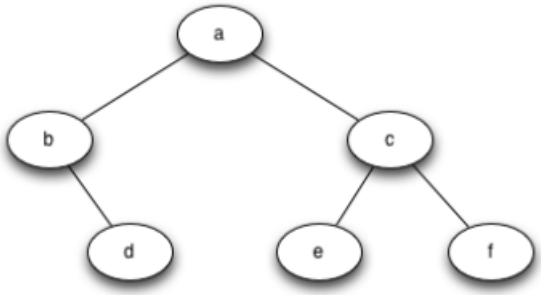
Berikut tambahan method pada class BinaryTree, untuk mendapatkan informasi mengenai left child, right child, dan root dari suatu binary tree.

```
In [1]: class BinaryTree:
            def __init__(self, root):
                self.key = root
                self.leftChild = None
                self.rightChild = None
            def insertLeft(self, new_node):
                if self.leftChild == None:
                    self.leftChild = BinaryTree(new_node)
                else:
                    t = BinaryTree(new_node)
                    t.leftChild = self.leftChild
                    self.leftChild = t

            def insertRight(self, new_node):
                if self.rightChild == None:
                    self.rightChild = BinaryTree(new_node)
                else:
                    t = BinaryTree(new_node)
                    t.rightChild = self.rightChild
                    self.rightChild = t

            def getRightChild(self):
                return self.rightChild
            def getLeftChild(self):
                return self.leftChild
            def setRootVal(self, obj):
                self.key = obj
            def getRootVal(self):
```

```
        return self.key
```

1.6 Latihan

Buatlah binarytree dari class tersebut diatas sesuai dengan gambar berikut



:

Section 1