Laporan Praktikum Struktur Data

Semester Gasal 2021/2022

NIM	**71200634**
Nama Lengkap	**Egi Granaldi Ginting**
Minggu ke / Materi	**14 / Binary Search Tree**

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN DUTA WACANA
YOGYAKARTA
2021

BAGIAN 1: Binary Search Tree

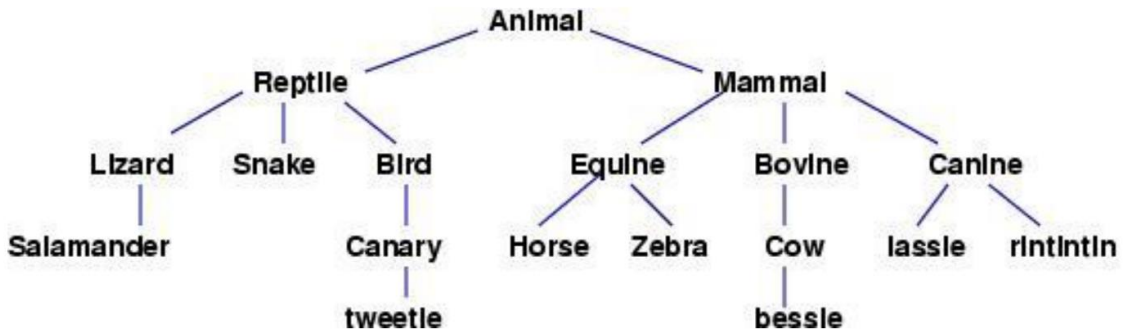
Definisi Binary Search Tree

Tree

Tree merupakan sekumpulan node berisi data sedemikian sehingga antar node terdapat hubungan parent-child yang memenuhi ketentuan sebagai berikut :

- Jika tree tersebut ada isinya, maka terdapat satu node khusus yang tidak memiliki parent. Node khusus tersebut diberi nama root.

- Setiap node (kecuali root) hanya memiliki satu parent



Gambar 1 Tree

Binary Tree

Binary tree merupakan tree yang setiap node-nya memiliki maksimal dua children.

Binary Search Tree

Binary Search Tree merupakan tree yang terurut yang memiliki kelebihan bila disbanding dengan struktur data lain. Diantaranya adalah proses pengurutan dan pencarian dapat dilakukan bila data sudah tersusun dalam struktur data Binary Search Tree.Pengurutan dapat dilakukan Binary Search Tree ditelusuri (Traversed) menggunakan metode In-order.Adapun aturannya adalah sebagai berikut :

- Setiap node child di sebelah kiri nilainya harus lebih kecil dari parent-nya

- Setiap node child di sebelah kanan nilainya harus lebih besar dari parent -nya

- Sub-tree di sebelah kanan dan kiri harus memenuhi ketentuan 1 dan 2.

Jika Traversing dilakukan secara In-order, maka hasilnya adalah data yang terurut.Dengan asumsi Balanced Binary Search Tree :

- Untuk Pencarian secara umum sama dengan binary search pada array yang terurut.

- Untuk operasi Insert dan delete Binary Search Tree akan lebih cepat daripada array yang terurut.

Jika Unbalanced Binary Search Tree :

- Kinerja turun menjadi O(n)

Contoh Aplikasi Binary Search Tree

- Membangun daftar vocabulary yang merupakan bagian dari inverted index (sebuah struktur data yang digunakan oleh banyak mesin pencari seperti Google, Yahoo, dan Ask.

Binary Search Tree memiliki 3 operasi dasar , hampir sama dengan array keliatannya jadi ada 3 yaitu :

- Find(x) : find value x didalam Binary Search Tree (Search)

- Insert(x) : memasukan value baru x ke Binary Search Tree (Push)

- Remove(x) : Menghapus key x dari Binary Search Tree (Delete)

Operasi Search Binary Search Tree

Dengan adanya ciri – ciri atau syarat di dalam Binary Search Tree, maka untuk finding/searching didalam Binary Search Tree menjadi lebih mudah.

Bayangkan kita akan mencari value x

- Memulai pencarian dari root

- Jika root adalah value yang kita cari, maka berhenti

- Jika x lebih kecil dari root maka cari kedalam rekrusif tree sebelah kiri

- Jika x lebih besar dari root maka cari kedalam rekrusif tree sebelah kanan.

Operasi Insertion Binary Search Tree

Memasukkan value (data) baru kedalam Binary Search Tree dengan rekursif

Bayangkan kita menginsert value x:

- Dimulai dari root

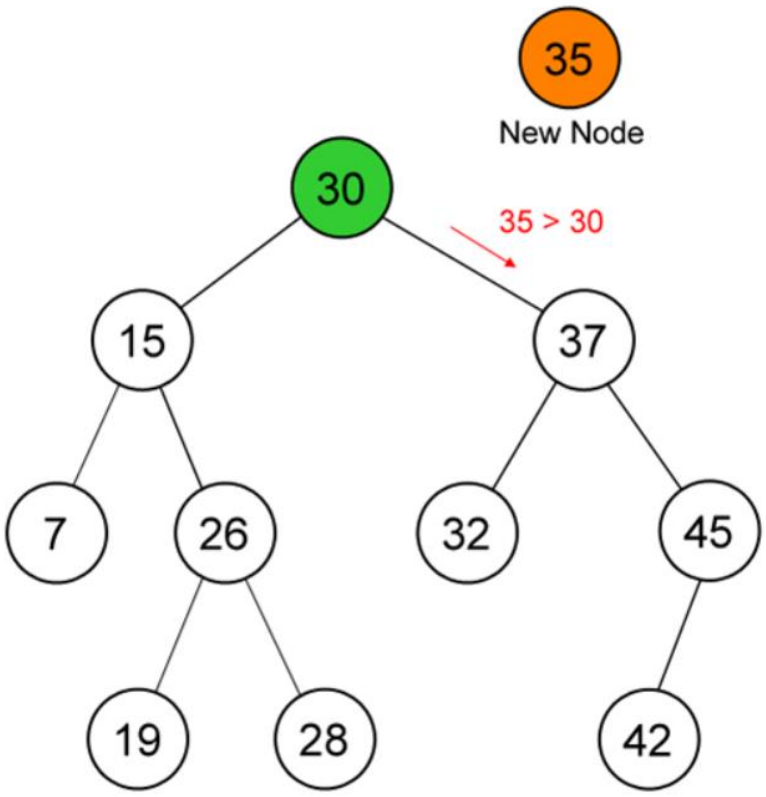
- Jika x lebih kecil dari node value (key) kemudian cek dengan sub-tree sebelah kiri lakukan pengecekan secara berulang (rekursif)

- Jika x lebih besar dari node value (key) kemudian cek dengan sub-tree sebelah kanan lakukan pengecekan secara berulang (rekursif)

- Ulangi sampai menemukan node yang kosong untuk memasukkan value x (X akan selalu berada di paling bawah biasa disebut Leaf atau daun)

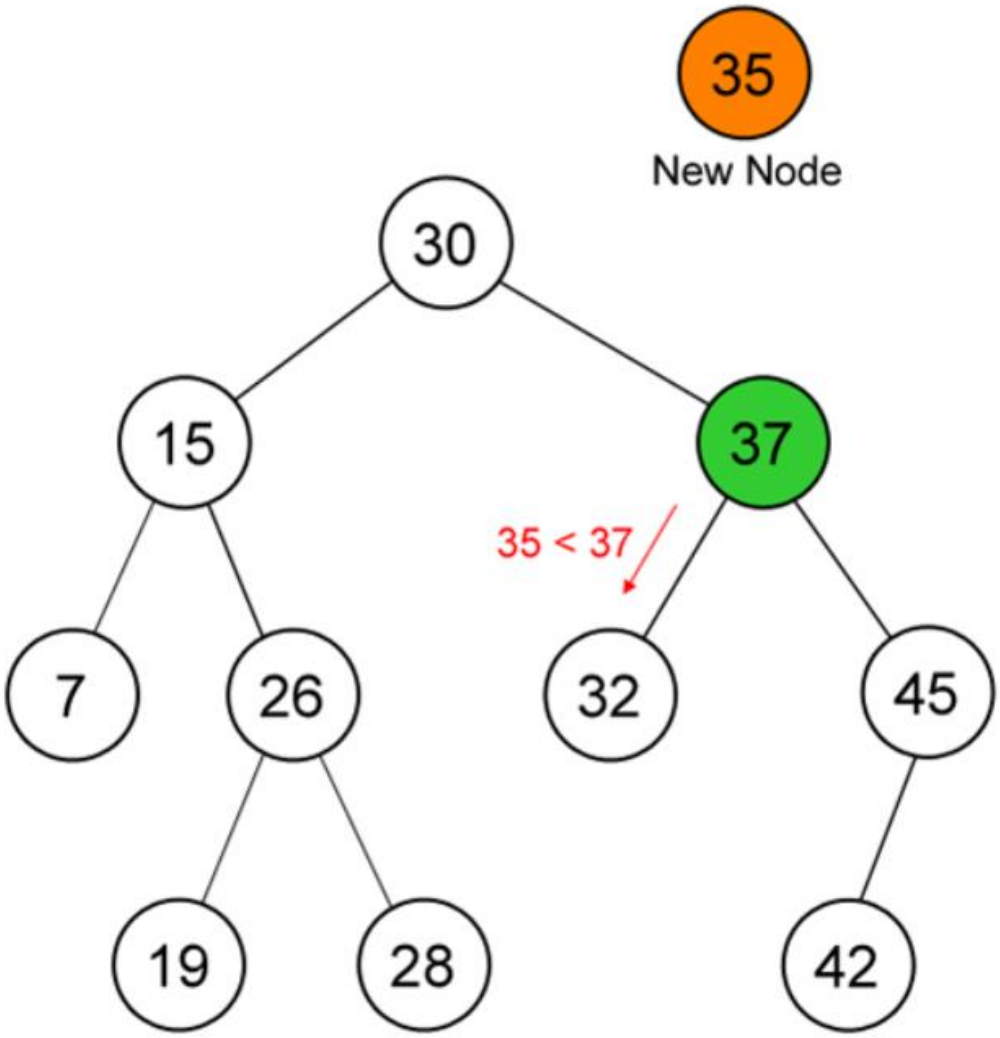
Contoh Insertion :

Kita ingin menambahkan value 35 kedalam Binary Search Tree yang sudah ada

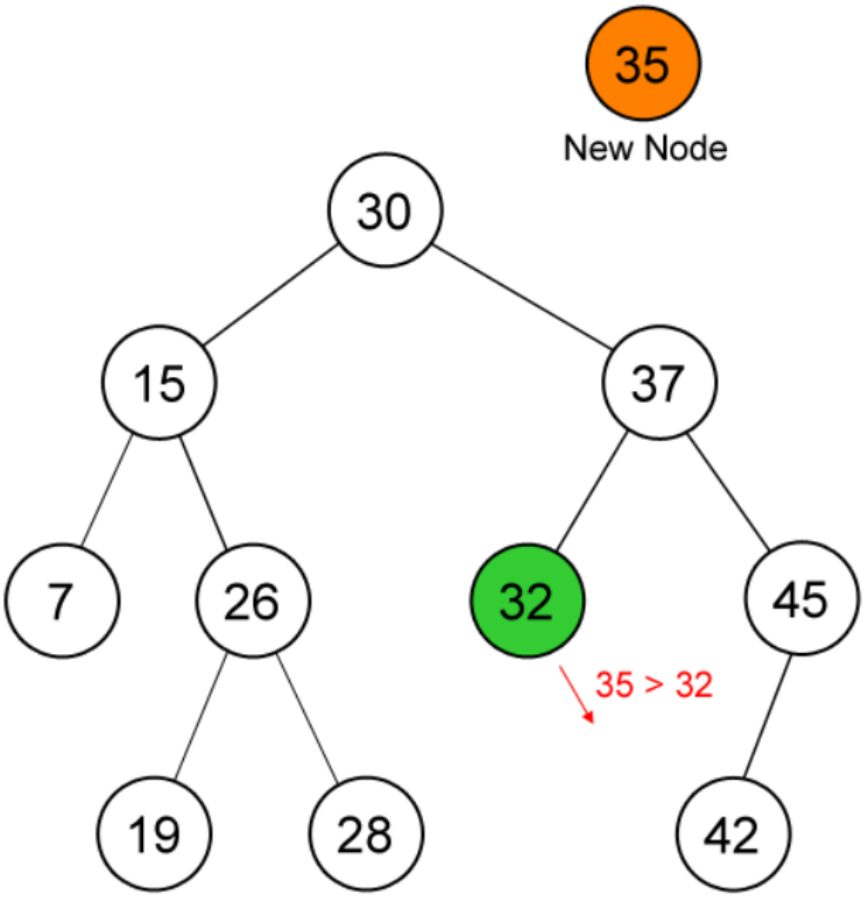


Gambar 2 Insertion Binary Search Tree

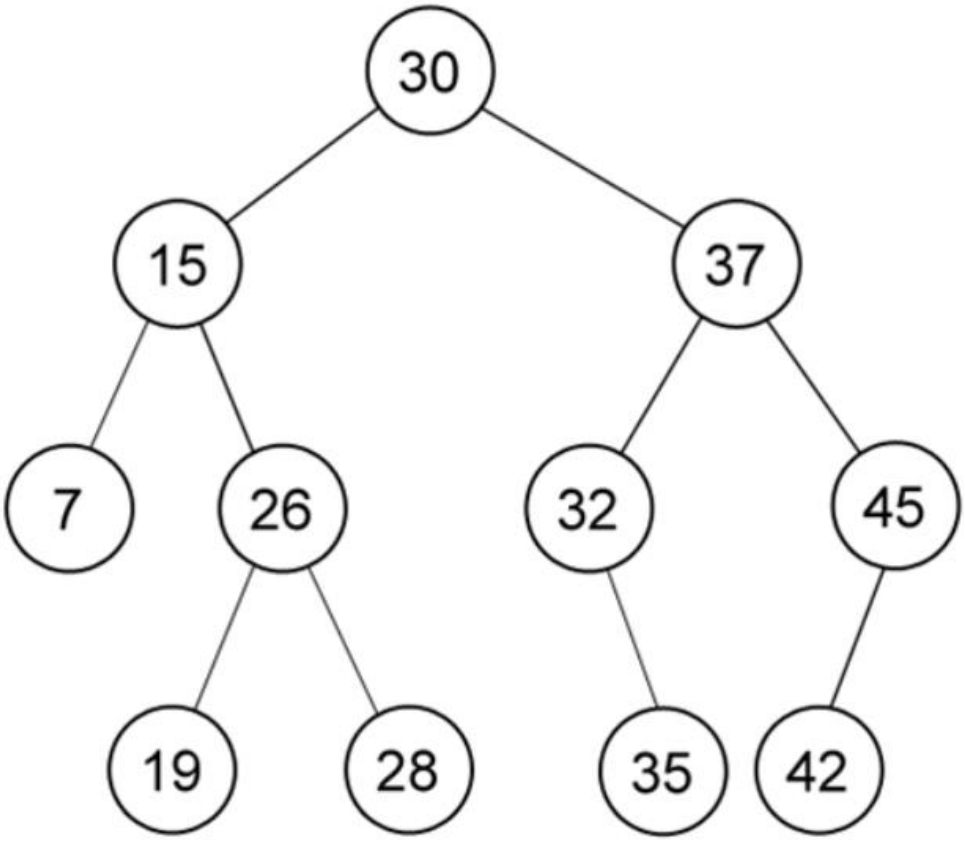
Yang berwarna hijau adalah root , setiap menginsert , mencari , atau delete selalu di mulai dari root.Dan new node berwarna orange yang memiliki value 35, kemudian kita cek dengan root(30), maka 30 .. 35 ? karena 30 < 35 maka , node lebih besar dari root kemudian kita arahkan ke sub-tree yang berada di kanan dan kemudian cek ulang Kembali.



Sekarang kita cek 35 dengan 37 , maka 35 < 37 jadi kita arahkan ke bagian kiri dan kita cek Kembali, karena di bagian kiri sudah ada value yaitu 32



Kemudian kita cek 32 dengan 35 , ternyata 35 > 32 jadi kita masukan ke kanan , dan ternyata kita sudah menemukan tempat kosong untuk mengisi new node(35) jadi kita masukan 35 di sebelah kanan dari node(32).

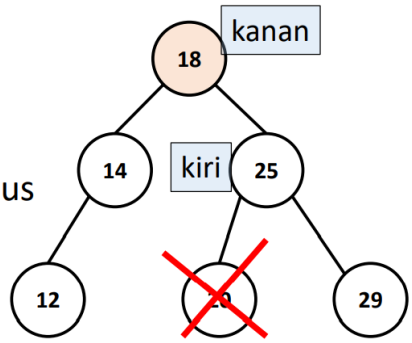


Operasi Delete pada Binary Search Tree

- Berdasarkan posisi node yang dihapus, terdapat beberapa kemungkinan :
  - Yang dihapus adalah leaf node
  - Yang dihapus adalah internal node
  - Yang dihapus adalah root node
- Menghapus leaf node adalah operasi yang paling mudah
  - Node dapat dihapus dan tidak banyak mengubah bentuk tree
- Menghapus internal node atau root node relatif lebih kompleks
  - Akan menyebabkan terjadinya pergeseran/rotasi node, sehingga mengubah bentuk tree
  - Untuk menghapus, perlu melihat apakah memiliki 1 atau 2 child node
- Terdapat tiga macam kondisi penghapusan node:
  - Node yang akan dihapus tidak memiliki child => menghapus leaf node

Hapus 20
Cek dulu apakah ada 20 di dalam BST? -> Ya

Ternyata 20 adalah leaf node, bisa langsung dihapus

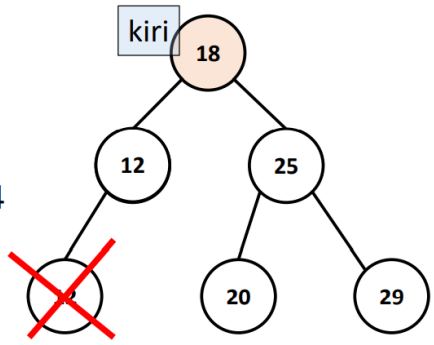


Gambar 3 Operasi Delete yang tidak memiliki Child

- Node yang akan dihapus hanya memiliki satu child node (di kiri atau kanan)

Hapus 14

Cek dulu apakah ada 14 di dalam BST? -> Ya
Ada 1 child node, berarti child node gantikan 14

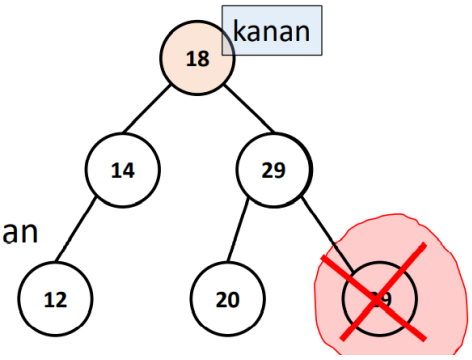


Gambar 4 Operasi Delete yang memiliki 1 Child

- Node yang akan dihapus memiliki dua child nodes

Hapus 25
Cek dulu apakah ada 25 di dalam BST? -> Ya

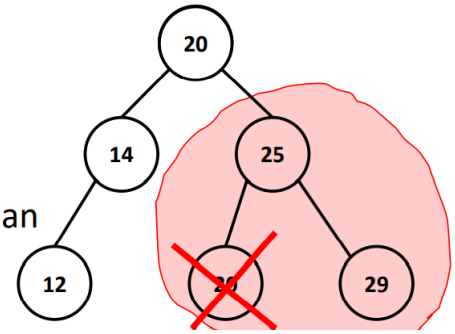
Ada 2 child node, harus cari pengganti, yaitu
node dengan nilai terkecil di subtree sebelah kanan



Gambar 5 Operasi Delete yang memiliki 2 child nodes

Hapus 18
Cek dulu apakah ada 18 di dalam BST? -> Ya

Ada 2 child node, harus cari pengganti, yaitu
node dengan nilai terkecil di subtree sebelah kanan



Gambar 6 Operasi Delete yang memiliki 2 child nodes

Variasi bentuk dan Jenis Tree

Trie

Trie merupakan tree yang khusus digunakan untuk pencarian string terdekat.Setiap nodenya terisi 1 karakter (kecuali root).

Binary Heap

Binary Heap merupakan complete binary tree yang memiliki syarat tambahan yaitu nilai pada node root harus lebih besar dari node childrennya (heap property).Binary heap bisa digunakan untuk priority queue (min-heap), priority ada di root.

MATERI 2

Penjelasan materi 2, dst… sesuai format ini.

BAGIAN 2: SOAL UNGUIDED

No 1 – 4

```
class Node:
    def __init__(self, data, parent):
        self._data = data
        self._parent = parent
        self._left = None
        self._left2 = None
        self._right = None
        self._right2 = None
    def insert_ganjil(self,data):
        if self.left() is None and data < self.operator():
            if self.left() is None:
                self._left = Node(data, self)
            else:
                self.left().insert_ganjil(data)
        elif data > self.operator():
            if self.left2() is None:
                self._left2 = Node(data, self)
            else:
                self.left2().insert_ganjil(data)
        else:
            return False
        return True

    def insert_genap(self,data):
        if self.right() is None and data < self.operator():
            if self.right() is None:
                self._right = Node(data, self)
            else:
                self.right().insert_genap(data)
        elif data > self.operator():
            if self.right2() is None:
                self._right2 = Node(data, self)
            else:
                self.right2().insert_genap(data)
        else:
            return False
        return True

    def insert(self, data):
        if data % 2 != 0:
```

```
            return self.insert_ganjil(data)
        elif data % 2 == 0:
            return self.insert_genap(data)

    def operator(self):
        return self._data

    def parent(self):
        return self._parent

    def left(self):
        return self._left

    def left2(self):
        return self._left2

    def right(self):
        return self._right

    def right2(self):
        return self._right2

class BinarySearchTree:
    def __init__(self):
        self._root = 0

    def add(self, data):
        if self._root == 0:
            self._root = Node(0, self)
            self._root.insert(data)
        else:
            self._root.insert(data)

    def node_kiri(self):
        print("Nilai Data kiri yaitu :",end="")
        self.inorder_kiri(self._root)
    def inorder_kiri(self, node):
        if node is not None:
            self.inorder_kiri(node.left())
            print(node.operator(), end = ' ')
            self.inorder_kiri(node.left2())

    def node_kanan(self):
        print("Nilai Data kanan yaitu:",end="")
        self.inorder_kanan(self._root)
```

```
    def inorder_kanan(self, node):
        if node is not None:
            self.inorder_kanan(node.right())
            print(node.operator(), end = ' ')
            self.inorder_kanan(node.right2())


bst = BinarySearchTree()
data = [5, 4, 3, 9, 8, 6, 7, 11,10]
for i in data:
    bst.add(i)
bst.node_kiri()
print()
bst.node_kanan()
```

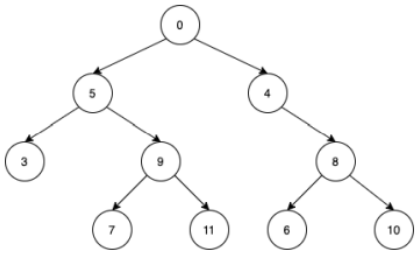
Output :

```
PS C:\Users\M S I> & "C:/Users/M S I/AppData/Local/Programs/Python/Python39/python.exe"
Nilai Data kiri yaitu :0 3 5 7 9 11
Nilai Data kanan yaitu:0 4 6 8 10
PS C:\Users\M S I>
```

Penjelasan :

Pada unguided kali ini kita membuat Binary Tree dengan beberapa aturan, yaitu :

1. Root selalu bernilai 0
2. Sisi kiri (left) dari root dan turunannya hanya berisi nilai ganjil, sedangkan sisi kanan (right) dari root hanya berisi nilai genap
3. Seluruh node dari sisi kiri maupun sisi kanan dari root disimpan secara terurut ascending menurut aturan ordered tree pada umumnya.
4. Apabila diinputkan data (tanpa root) dengan urutan sebagai berikut: (5, 4, 3, 9, 8, 6, 7, 11, 10), maka hasilnya akan seperti gambar tree di atas.



Langkah awal yang kita lakukan adalah membuat Class node.Setelah itu kita menambahkan parameter parent untuk mendapatkan isi elemen pada class.Lalu kita menambahkan function insert untuk menambah data dan kita akan menginsert bilangan ganjil dan bilangan genap. Setelah itu, kita membuat variabel _data untuk menyimpan nilai data, variabel _parent untuk menyimpan nilai parent, _left untuk menyimpan data yang letaknya di sebelah kiri node (datanya lebih kecil dari nilai parentnya), variabel _right untuk menyimpan data yang letaknya di sebelah kanan node (datanya lebih besar dari nilai parentnya), variabel _left2 untuk menyimpan seleruh data yang berada di sebelah kiri root (semua data ganjil), dan terakhir variabel _right2 untuk menyimpan semua data yang berada di sebelah kanan root (semua data genap).Function parent kita tambahkan untuk mendapatkan node Parent.Selanjutnya pada program kita akan membuat function untuk mengecek apakah data yang kita inputkan bernilai ganjil atau genap.Function insert_kiri dan insert_kanan akan kita panggil tergantung nilai datanya.Setelah itu kita membuat class BinarySearchTree.Pada constructornya kita menginisialisasi nilai variabel_rootnya.Lalu kita membuat fungsi node_kanan dan node_kiri untuk mereturn nilai dari fungsi inorder_kiri dan inorder_kanan.Fungsinya adalah untuk menampilkan data secara ascending.Lalu program yang kita tampilkan juga selalu meroot data 0.

DAFTAR PUSTAKA :

https://abdilahrf.github.io/2015/06/pengenalan-binary-search-tree/

http://informatika.unsyiah.ac.id/irvanizam/teaching/SD/bst.pdf