PRAKTIKUM 27
BINARY SEARCH TREE

A. TUJUAN

Mahasiswa diharapkan mampu :

1. Memahami Konsep Binary Search Tree
2. Mengimplementasaikan Binary Search Tree

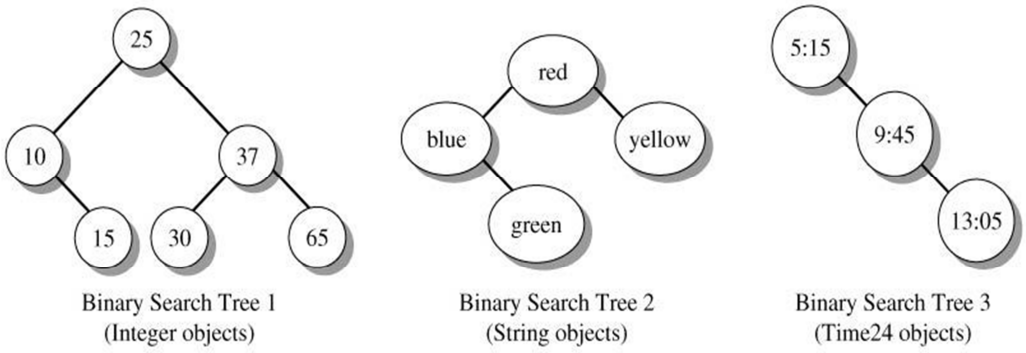
B. DASAR TEORI

Binary search tree adalah salah satu bentuk dari pohon. Di mana masing-masing pohon tersebut hanya memiliki dua buah upapohon, yakni upapohon kiri dan upapohon kanan.Ciri khas yang melekat pada binary search tree ini yang bisa juga dibilang sebagai keunggulan dari binary search tree adalah peletakan isi dari nodenya yang terurut berdasarkan besarnya dari isinya tersebut. Isinya bisa saja berupa integer, karakter, atau apapun sesuai dengan spesifikasi binary search tree yang ada. Operasi dasar dari binary search tree(BST) ini sendiri sangatlah sederhana, yakni hanya fungsi perbandingan dan fungsi rekursif. Di mana anak pohon sebelah kiri node adalah anak pohon yang lebih kecil dari node, sedangkan anak pohon sebelah kanan node memiliki isi yang lebih besar daripada isi node. Biasanya penyimpanan data di dalam binary search tree ini bisa juga berupa record di mana pengurutannya hanya tinggal melihat key dari record tersebut, missal nomor absen, NIM, tanggal, dll. Pada gambar 1 merupakan contoh Binary Search Tree, pada subtree kiri semua node mempunyai value lebih kecil dari root dengan value 25, sedangkan subtree kanan semua node mempunyai value lebih besar dari root dengan value 25. Pada Binary Search Tree, strategi penambahan Node baru adalah sebagai berikut:

1. Jika value dari node baru sama dengan value dari current node, maka mengembalikan nilai false.
2. Jika value dari node baru kurang dari value dari current node maka :
    1) Jika anak kiri current node tidak null, maka ubah current node ke anak kiri tersebut, lakukan langkah 1.
    2) Jika anak kiri current node adalah null, maka tambahkan node baru tersebut sebagai anak kiri dari current node

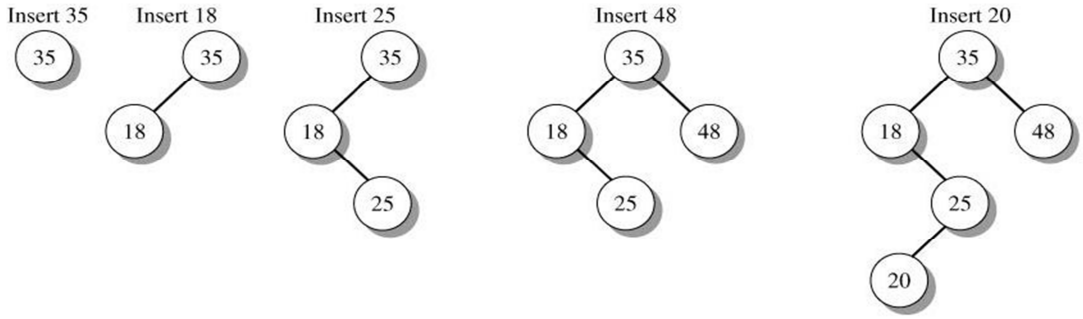
3. Jika value dari node baru lebih besar dari value dari current node maka :

    1) Jika anak kanan current node tidak null, maka ubah current node ke anak kanan tersebut, lakukan langkah 1.
    2) Jika anak kanan current node adalah null, maka tambahkan node baru tersebut sebagai anak kanan dari current node



Gambar 1. Contoh Binary Search Tree

Pada gambar 2 merupakan step by step penambahan node baru pada Binary Search Tree. Pertama ditambahkan node dengan value 35. Karena root masih null, maka node 35 menjadi root. Selanjutnya ditambahkan node 18. Pengecekan dimulai dari root, node 18 lebih kecil dibandingkan dengan node 35, dan node 35 tidak mempunyai anak kiri, sehingga node 18 menjadi anak kiri dari node 35. Selanjutnya ditambahkan node 25. Pengecekan dimulai dari root, node 25 lebih kecil dibandingkan dengan node 35, sehingga pengecekan selanjutnya ke anak kiri dari node 35 yaitu node 18. Node 25 lebih besar dibandingkan dengan node 18, karena node 18 tidak mempunyai anak kanan, sehingga node 25 menjadi anak kanan dari node 18. Selanjutnya ditambahkan node 48. Pengecekan dimulai dari root, node 48 lebih besar dibandingkan dengan node 35, karena node 35 tidak mempunyai anak kanan, sehingga node 48 menjadi anak kanan dari node 35. Selanjutnya ditambahkan node 20. Pengecekan dimulai dari root, node 20 lebih kecil dibandingkan dengan node 35, sehingga pengecekan selanjutnya ke anak kiri dari node 35 yaitu node 18. Node 20 lebih besar dibandingkan dengan node 18, sehingga pengecekan selanjutnya ke anak kanan dari node 18 yaitu node 25. Node 20 lebih kecil dibandingkan dengan node 25, karena node 25 belum mempunyai anak kiri, maka node 20 menjadi anak kiri dari node 25.



Gambar 2. Strategi Penambahan Node Baru pada Binary Search Tree

Search Method

Metode search adalah penentu paling dalam program atau bisa dibilang nyawa dari program database. Kenapa search sangat penting untuk efisien.?Bayangkan saja apabila ada miliaran data yang ada di dalam database pada sebuah komputer server dan harus diolah dan selalu harus diupdate setiap hari ataupun diakses pengguna untuk dicari isi data yang diinginkanya. Lalu, yang kita gunakan adalah struktur data yang tidak efisien, di mana suatu saat computer server akan mengalami minimal hang atau yang paling parah adalah server down sehingga tidak bisa diakses dan akhirnya justru membuang waktu bahkan lebih parahnya bisa menyebabkan data dalam harddisk server menghilang. Tentunya kita sangat menghindari hal semacam itu terjadi bukan.

Metode Search sendiri ada banyak, seperti yang sudah sempat disinggung sedikit pada pendahuluan. Bahwa search macamnya ada linear search, binary search, binary search tree, dll. Di sini hanya akan dibahas 3 contoh tersebut. Linear search merupakan tipe search yang paling mendasar dalam dunia search. Pada dasarnya linear search ini sangat sederhana. Yakni memulai pencarian dari elemen pertama kemudian bergeser terus ke elemen berikutnya hingga menemukan isi elemen yang dicari. Apabila elemen yang dicari tidak ada, maka linear search akan terus melakukan traversal hingga elemen terakhir. Mari kita bayangkan apabila terdapat banyak sekali data yang harus ditraversal dan data yang dicar itidak ada di dalam database(worst case possibility). Efektivitas dari linear search ini adalah dengan O(n) worst case tergantung dari jumlah elemen yang ada di dalam database tersebut. Metode search yang berikutnya yakni binary search, bedanya binary search dari linear search adalah dengan pembagian jumlah elemen menjadi dua bagian. Sehingga pencarian bisa lebih efektif dengan cara membagi pencarian ke

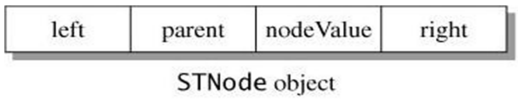
dua arah. Worst case dari binary search ini adalah O(log n) sehingga cara ini bisa dibilang cukup efisien.

Metode search yang terakhir adalah binary search tree (sebenarnya bukan merupakan metode search). binary search tree memiliki kompleksitas algoritma O(log n) yang tidak ada bedanya dengan metode binary search. Tetapi seharusnya metode search pada binary search tree bisa lebih singkat, karena jalur yang dipilih sudah tergolong spesifik, sehingga tidak perlu memeriksa semua elemen untuk mencari elemen tertentu.

Implementasi dari Binary Search Tree

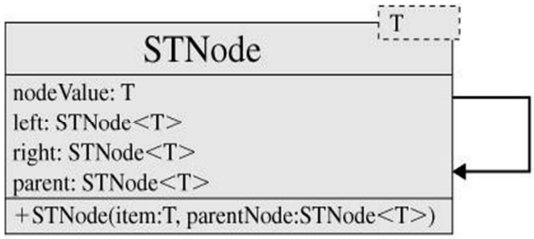
Sebuah Node pada Binary Search Tree direpresentasikan dengan sebuah objek dari Class STNode seperti ditunjukkan pada gambar 3. Sebuah Node terdiri dari 4 bagian yaitu

- Data value disebut nodeValue
- Variabel reference left sebagai anak kiri
- Variabel reference right sebagai anak kanan
- Variabel reference parent sebagai parent dari Node tersebut



Gambar 3. Objek dari Class STNode

Pada gambar 4 adalah UML dari Class STNode<T>. Pada gambar 5 terdapat contoh bentuk tree dan bentuk Node-Node Tree yang direpresentasikan dengan objek STree.



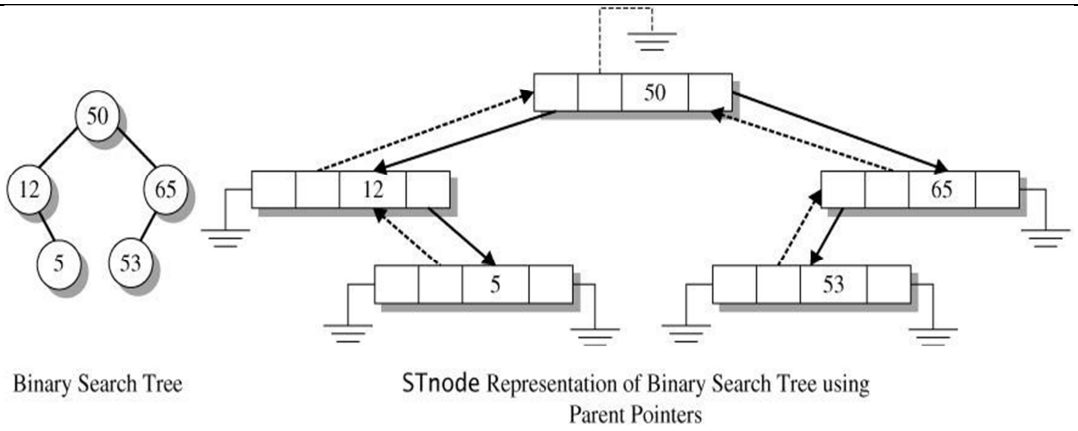
Gambar 4. UML dari Class STNode<T>

```
public class STNode<T> {
   public T nodeValue;
   public STNode<T> left, right;
   public STNode<T> parent ;
```

```
    public STNode(T item)
    {
       nodeValue = item;
       left = right = null;
    }

    public STNode(T item, STNode<T> parent)
    {
       nodeValue = item;
       left = right = null;
       this.parent = parent ;
    }

    public STNode (T item, STNode<T> left, STNode<T> right)
    {
       nodeValue = item;
       this.left = left;
       this.right = right;
    }
}
```



Gambar 5. Representasi STNode dari Binary Search Tree

C. TUGAS PENDAHULUAN

1. Buatlah resume 1 halaman mengenai **Binary Search Tree**
2. Buatlah Binary Search Tree dari node – node 30, 15, 17, 36, 67, 30, 69.

D. PERCOBAAN

Pada percobaan ini buatlah class Binary Search Tree seperti tabel 1.

Tabel 1. Variabel dan Method pada Class Binary Search Tree

Class Binary Search Tree<T>	
private STNode<T> root;	Variabel reference ke root

private int treeSize;	Variable untuk mengetahui jumlah node pada binary search tree
public boolean add(T item){}	Method add() untuk menambahkan node baru ke binary search tree. Mengembalikan nilai true jika node baru berhasil ditambahkan, mengembalikan nilai false jika node baru sudah terdapat pada binary search tree.
public STNode<T> findNode(T item)	Method findNode() untuk mencari sebuah node dengan item tertentu dengan mengembalikan alamat dari node tersebut.
public boolean find(T item)	Method find () untuk mencari sebuah node dengan item tertentu dengan mengembalikan nilai true jika ada item tersebut, nilai false jika tidak ada.
public T first() {}	Method first() untuk mendapatkan nilai terkecil dari node-node yang terdapat pada binary search tree.
public T last() {}	Method last() untuk mendapatkan nilai terbesar dari node-node yang terdapat pada binary search tree.
public STNode<T> getRoot() {}	Method getRoot() untuk mendapatkan alamat reference dari root
public int getTreeSize() {}	Method getTreeSize() untuk mendapatkan jumlah node pada Binary Search Tree

Percobaan 1. Membuat class BinarySearchTree<T>, konstruktor dan method add().Method add() untuk menambahkan node baru ke binary search tree. Mengembalikan nilai true jika node baru berhasil ditambahkan, mengembalikan nilai false jika node baru sudah terdapat pada binary search tree.

```
public class BinarySearchTree<T> {

    private STNode<T> root;
    private int treeSize;

    public BinarySearchTree() {
        root=null;
    }
    public boolean add(T item) {

        STNode<T> t = root, parent = null, newNode;
        int orderValue = 0;

        while (t != null) {
            parent = t;
```

```
            orderValue = ((Comparable<T>) item).compareTo(
                    t.nodeValue);
            if (orderValue == 0) {
                return false;
            } else if (orderValue < 0) {
                t = t.left;
            } else {
                t = t.right;
            }
        }

        newNode = new STNode<T>(item, parent);
        if (parent == null)
        {
            root = newNode;

        } else if (orderValue < 0)        {
            parent.left = newNode;

        } else
        {
            parent.right = newNode;
        }

        treeSize++;

        return true;
    }
}
```

Percobaan 2. Membuat method findNode() untuk mencari sebuah node dengan item tertentu dengan mengembalikan alamat dari node tersebut.

```
public class BinarySearchTree<T> {

…
    public STNode<T> findNode(T item) {
        STNode<T> t = root, parent = null, newNode;
        int orderValue = 0;

        while (t != null) {
            orderValue = ((Comparable<T>) item).compareTo(
                    t.nodeValue);

            if (orderValue == 0) {
                return t;
            } else if (orderValue < 0) {
                t = t.left;
            } else {
                t = t.right;
            }
        }
        return null;
    }
```

```
    }
```

Percobaan 3. Mebuat method find () untuk mencari sebuah node dengan item tertentu dengan mengembalikan nilai true jika ada item tersebut, nilai false jika tidak ada.

```
public class BinarySearchTree<T> {

…

   public boolean find(T item) {
        STNode<T> t = findNode(item);
        T value = null;

        if (t != null) {
            value = t.nodeValue;
            return true;
        }

        return false;
    }
}
```

Selanjutlah ujilah program yang sudah dibuat dengan Class Main seperti dibawah ini:

```
public class Main {
    public static void main(String[] args) {
        BinarySearchTree<Integer> bst = new BinarySearchTree<Integer>();

        bst.add(35);
        bst.add(18);
        bst.add(25);
        bst.add(48);
        bst.add(20);

        System.out.println(BinaryTree.inorderDisplay(bst.getRoot()));

        System.out.println(bst.findNode(20));
        System.out.println(bst.find(50));
    }
}
```

Output program adalah

```
18  20  25  35  48
20
false
```

E. LATIHAN

Latihan 1 : Buatlah method first() untuk mendapatkan nilai terkecil dari node-node yang terdapat pada binary search tree.

Latihan 2 : Buatlah method last() untuk mendapatkan nilai terbesar dari node-node yang terdapat pada binary search tree.

Latihan 3 . Buatlah Binary Search Tree dengan menambahkan node dengan value {10, 5,20,2,6,19,25,21}. Kerjakan :

- Gambarkan hasil akhir binary search tree tersebut !
- Bacalah Binary Searh Tree dengan Traversal PostOrder
- Cari node dengan nilai 19.
- Cari node dengan nilai 55.
- Tambahkan pada Binary Tree Node baru 56.
- Dapatkan node dengan nilai terkecil
- Dapatkan node dengan nilai terbesar

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.