PRAKTIKUM 28
BINARY SEARCH TREE 2

A. TUJUAN

Mahasiswa diharapkan mampu :

1. Memahami konsep menghapus node pada Binary Search Tree. Node yang dihapus adalah node root, node leaf, node yang mempunyai satu anak dan node yang mempunyai dua anak.
2. Mengimplementasikan menghapus node pada Binary Search Tree dengan menggunakan bahasa java

B. DASAR TEORI

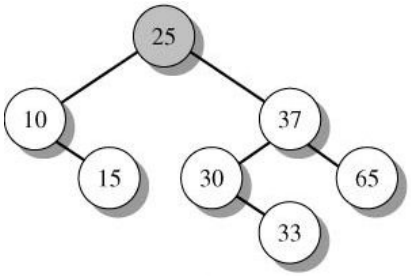
Binary Search Tree (BST)

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.

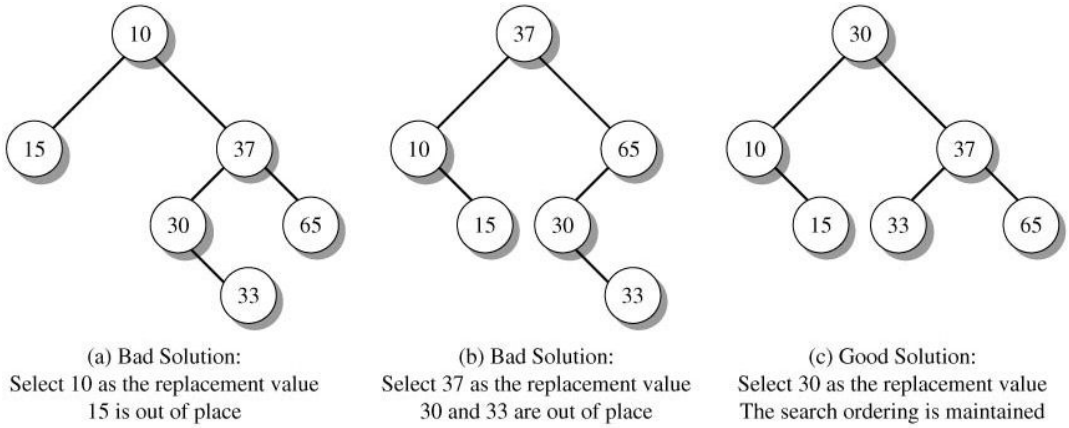
Menghapus Node pada Binary Search Tree

- Aturan dalam menghapus node pada Binary Search Tree
- Root pada BST tidak dapat dihapus, karena terdapat dua subtree pada node root. Tapi dapat diganti dengan node yang tepat yang terdapat pada BST.

Sebagai contoh pada gambar 1, akan menghapus node root 25. Pada gambar 2(a), jika node 25 diganti dengan node 10, bukan merupakan solusi yang tepat, karena ada node yang tidak sesuai yaitu node 15. Pada gambar 2(b), jika node 25 diganti dengan node 37, bukan merupakan solusi yang tepat, karena ada node yang tidak sesuai yaitu node 30 dan 33. Pada gambar 2(c), jika node 25 diganti dengan node 30, merupakan solusi yang tepat, karena semua node mengikuti aturan dari BST.

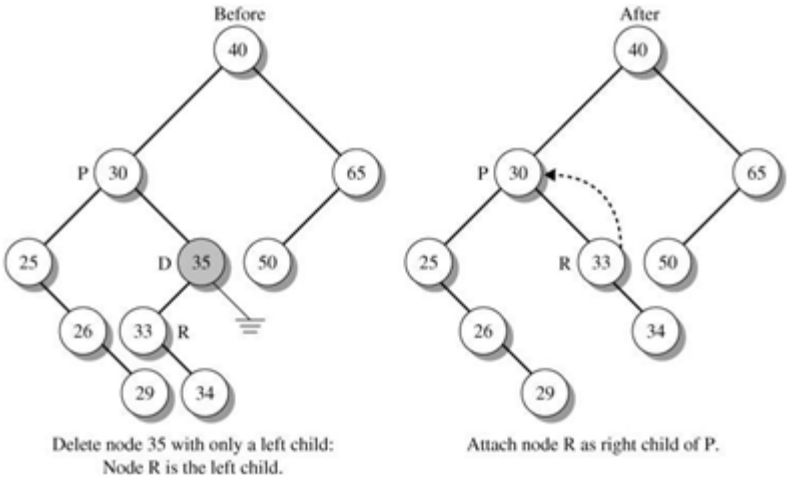


Gambar 1. Menghapus Node Root 25

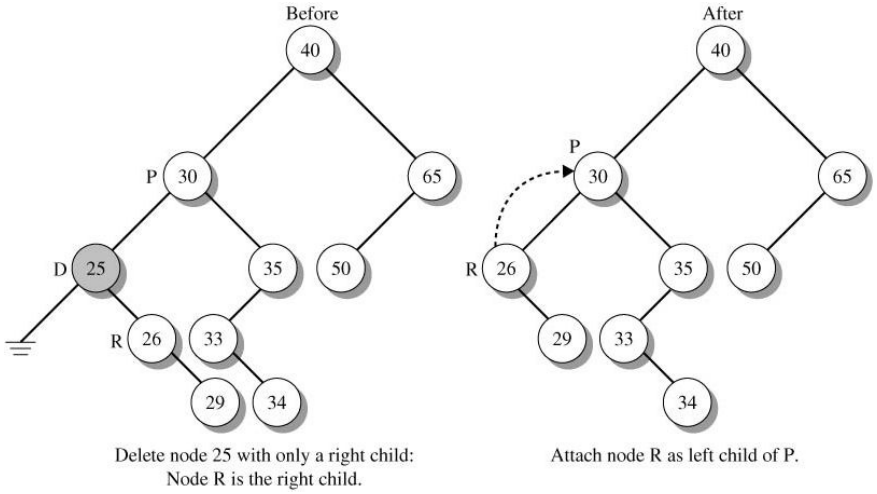


Gambar 2. Mencari Pengganti dengan Nilai yang Tepat

- Menghapus sebuah node yang memiliki satu anak. Pada gambar 3 node 35 bukan node leaf, node 35 mempunyai subtree kosong. Node 35 diberi tanda dengan D, Parent dari node 35 yaitu node 30 adalah P dan anak dari node 35 yaitu node 33 ditandai dengan R. Selanjutnya hapus node 35, dan kaitkan R(node 33) sebagai anak dari P(node 30). Pada gambar 4 node 25 bukan node leaf, node 25 mempunyai subtree kosong. Node 25 diberi tanda dengan D, Parent dari node 25 yaitu node 30 adalah P dan anak dari node 25 yaitu node 26 ditandai dengan R. Selanjutnya hapus node 25, dan kaitkan R(node 26) sebagai anak dari P(node 30).

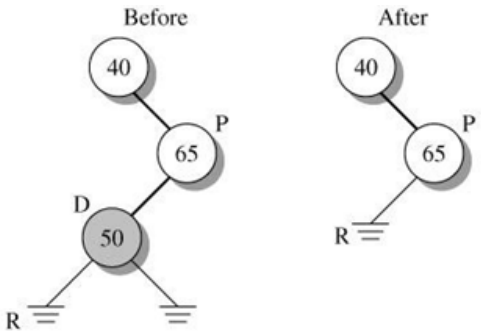


Gambar 3. Contoh 1, menghapus node yang memiliki 1 anak



Gambar 4. Contoh 2, menghapus node yang memiliki 1 anak

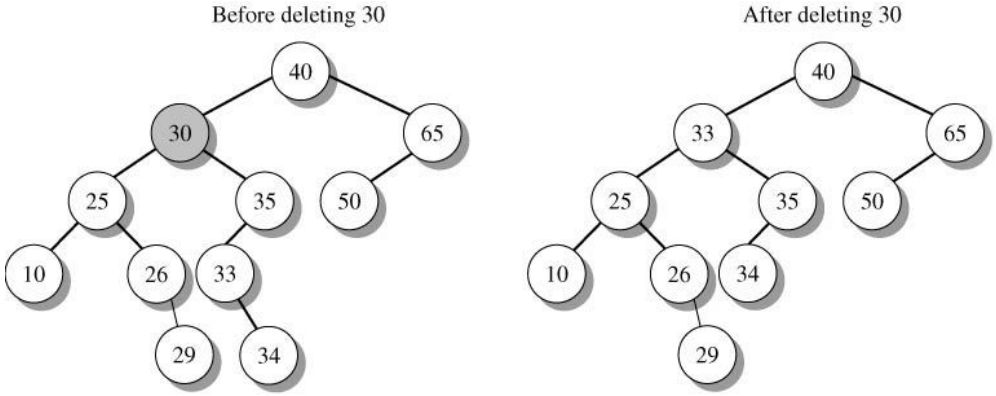
- Menghapus node leaf, menggunakan cara yang sama dengan menghapus node yang bukan leaf memiliki 1 anak. Pada gambar 5, node yang akan dihapus node 50 ditandai dengan D, parent dari node 50 yaitu node 65 ditandai dengan P, dan anak dari node 50 ditandai dengan R. R bernilai null, karena node yang akan dihapus adalah node leaf. Hapus node 50(D), kaitkan R(null) menjadi anak dari node P(65).



(a) Delete leaf node 50 Replacement node is null

Gambar 5. Menghapus node leaf

- Menghapus node yang memiliki dua anak. Pilih R (node yang berada di subtree dari node 30) sebagai node dengan nilai terkecil, tapi lebih besar dari nilai node yang dihapus. Pada gambar 6, node 30 sebagai node yang akan dihapus, cari node R yang terkecil, tapi lebih besar dari nilai node yang dihapus (node 30) yaitu node 33. Hapus node 30 dan kaitkan R ke parent dari node yang dihapus.



Replacing deleted value 30 with replacement value 33 and attaching right subtree 34.

Gambar 6. Menghapus node yang memiliki dua anak

C. TUGAS PENDAHULUAN

Jelaskan langkah-langkah menghapus node pada gambar 1 dan 2!

D. PERCOBAAN

Pada percobaan ini tambahkan pada class Binary Search Tree seperti tabel 1.

Tabel 1. Variabel dan Method pada Class Binary Search Tree

Class Binary Search Tree<T>

public void removeNode(STNode<T> dNode)	Method yang digunakan untuk menghapus node pada Binary Search Tree

Percobaan 1. Method untuk menghapus sebuah Node

```
   public class BinarySearchTree<T> {

       private STNode<T> root;
       private int treeSize;

   …

   public void removeNode(STNode<T> dNode) {
           if (dNode == null) {
               return;
           }

           STNode<T> pNode, rNode;

           pNode = dNode.parent;
           // node yang dihapus mempunyai satu anak
           if (dNode.left == null || dNode.right == null) {
               if (dNode.right == null) {
                   rNode = dNode.left;
               } else {
                   rNode = dNode.right;
               }

               if (rNode != null) {
                   System.out.println("Ngeset Parent");
                   rNode.parent = pNode;
               }
               // menghapus node root
               if (pNode == null) {
                   root = rNode;
               } else if (((Comparable<T>)
dNode.nodeValue).compareTo(pNode.nodeValue) < 0) {
                   pNode.left = rNode;
               } else {
                   pNode.right = rNode;
               }
           } // node yang dihapus mempunyai dua anak
           else {
               STNode<T> pOfRNode = dNode;
               rNode = dNode.right;
               pOfRNode = dNode;

               while (rNode.left != null) {
                   pOfRNode = rNode;
                   rNode = rNode.left;
               }
               dNode.nodeValue = rNode.nodeValue;
               if (pOfRNode == dNode) {
                   dNode.right = rNode.right;
```

```
                } else {
                    pOfRNode.left = rNode.right;
                }

                if (rNode.right != null) {
                    rNode.right.parent = pOfRNode;
                }
            }
        }
    }
```

Selanjutnya ujilah program diatas dengan Class Main sebagai berikut:

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

            bst.add(40);
            bst.add(30);
            bst.add(25);
            bst.add(35);
            bst.add(26);
            bst.add(33);
            bst.add(29);
            bst.add(34);
            bst.add(65);
            bst.add(50);

            System.out.println(BinaryTree.inorderDisplay(bst.getRoot()));
            bst.removeNode(new STNode<Integer>(35));

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

Output program adalah sebagai berikut:

```
25  26  29  30  33  34  35  40  50  65
25  26  29  30  34  35  40  50  65
```

E. LATIHAN

Latihan 1 : Kerjakan studi kasus yang terdapat pada gambar 1 sampai dengan gambar 6.

Latihan 2 : Dari studi kasus latihan 1, jelaskan langkah-langkahnya berdasarkan program!

F. LAPORAN RESMI

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