Implementacija BST

Drevo je podano kot referenca na vozlišče v korenu:

```
public class BSTree implements Dictionary {
    BSTreeNode rootNode; // referenca na koren
    …
}
```

Vozlišče je definirano kot:

```
public class BSTreeNode {
    Comparable key;      // ključ
    BSTreeNode left;     // referenca na levo poddrevo
    BSTreeNode right;    // referenca na desno poddrevo
    …
}
```

Iskanje elementa v BST

Rekurzivno iščemo v enem od poddreves.
Dva robna pogoja: 1) če elementa ni v drevesu: pridemo v prazno poddrevo.
2) če element je v drevesu: ga najdemo v korenu poddrevesa.

```
private boolean member(Comparable x, BSTreeNode node) {
   if (node == null)
      return false;
   else if (x.compareTo(node.key) == 0)
      return true;
   else if (x.compareTo(node.key) < 0)
      return member(x, node.left);
   else
      return member(x, node.right);
} // member
```

Dodajanje elementa v BST

Rekurzivno dodamo v enega od poddreves.
(Normalni) robni pogoj: prazno poddrevo zamenjamo z listom.
Izredni robni pogoj: element je že v drevesu...

```
protected BSTreeNode insertLeaf(Comparable x, BSTreeNode node) {
   if (node == null)
      node = new BSTreeNode(x);
   else if (x.compareTo(node.key) < 0)
      node.left = insertLeaf(x, node.left);
   else if (x.compareTo(node.key) > 0)
      node.right = insertLeaf(x, node.right);
   else
      ; // duplicate; do nothing or throw exception
   return node;
} // insertLeaf
```

ROTACIJA

```
protected BSTreeNode rightRotation(BSTreeNode node) {
   BSTreeNode temp = node;
   node = node.left;
   temp.left = node.right;
   node.right = temp;
   return node;
}

protected BSTreeNode leftRotation(BSTreeNode node) {
   BSTreeNode temp = node;
   node = node.right;
   temp.right = node.left;
   node.left = temp;
   return node;
}
```

Dodajanje elementa v koren

```
private BSTreeNode insertRoot(Comparable x, BSTreeNode node) {
   if (node == null)
      node = new BSTreeNode(x);
   else if (x.compareTo(node.key) < 0) {
      node.left = insertRoot(x, node.left);
      node = rightRotation(node);
   }
   else if (x.compareTo(node.key) > 0) {
      node.right = insertRoot(x, node.right);
      node = leftRotation(node);
   }
   else
      ; // duplicate; do nothing or throw exception
   return node;
} // insertRoot
```

Rekurzija gre najprej v globino, in element se doda kot list.

Pri vračanju i rekurzije se izvaja zaporedje rotacij, ki dvigne element v koren.

Časovna zahtevnost je sorazmerna višini drevesa.

```
// za prenos minimalnega kljuca iz desnega poddrevesa
private Comparable minNodeKey ;

 public BSTreeNode delete(Comparable x, BSTreeNode node) {
   if (node != null) {
     if (x.compareTo(node.key) == 0) { // delete node
       if (node.left == null) // empty left
         node = node.right;
       else if (node.right == null) //empty right
         node = node.left;
       else {
         node.right = deleteMin(node.right); // delete min from right
         node.key = minNodeKey; // put it into the node
       }
     }
     else if (x.compareTo(node.key) < 0)
       node.left = delete(x, node.left);
     else
       node.right = delete(x, node.right);
   } // if (node != null)
   return node;
 } //delete
```

Brisanje minimalnega

```
private BSTreeNode deleteMin(BSTreeNode node) {
    if (node.left != null) {
        node.left = deleteMin(node.left); // ohranjamo strukturo
        return node;
    }
    else {
        minNodeKey = node.key; // prenos kljuca
        return node.right; // ohrani desno poddrevo
    }
} // deleteMin
```