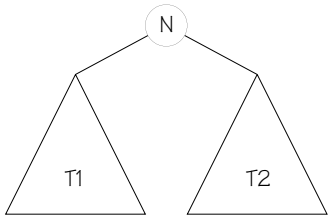
Binärt sökträd



Ett binärt träd kallas ett binärt sökträd om det för varje nod gäller att om noden har nyckeln N så är

- alla nycklar i T1 mindre än N
- alla nycklar i T2 är större än N

Binära sökträd

```
class TreeNode {
  ComparisonKey     key;
  TreeNode          llink;
  TreeNode          rlink;
}

interface ComparisonKey {

  int compareTo(ComparisonKey value);
  String toString();

} //end interface

class StringKey implements ComparisonKey {

  private String key;

  StringKey(String value) {
    key = value;
  }

  public String toString() {
    return key;
  }

  public int compareTo(ComparisonKey value) {
    String a = this.key;
    String b = ((StringKey)value).key;
    return a.compareTo(b);
  }

} //end StringKey class
```


Binära sökträd (forts.)

```
class BinarySearchTree {

  private TreeNode root;

  void insert(ComparisonKey K) {
    root=insertKey(root,K);
  } //end insert()

  void insert(String K) {
    root=insertKey(root, new StringKey(K));
  } //end insert()

  TreeNode insertKey(TreeNode T,ComparisonKey K) {
    // Se OH 19
  } //end insertKey()

  TreeNode find(String K) {
    return find(new StringKey(K));
  } //end find()

  TreeNode find(ComparisonKey K) {
    // Se OH 20
  } //end find()

} // End BinarySearchTree
```


InsertKey

```
TreeNode insertKey(TreeNode T,ComparisonKey K) {

  if (T == null) {
    TreeNode N = new TreeNode();
    N.key = K;
    return N;
  } else {
    if ( K.compareTo(T.key) < 0 ) {
      T.llink = insertKey(T.llink, K);
      return T;
    } else {
      T.rlink = insertKey(T.rlink, K);
      return T;
    }
  }

} //end insertKey()
```

Find

```
TreeNode find(ComparisonKey K) {

  TreeNode T = root;
  int result;

  while (T != null) {
    if ((result = K.compareTo(T.key)) < 0) {
      T = T.llink;
    } else if (result == 0) {
      return T;
    } else {
      T = T.rlink;
    } //end if
  } // end while

  return T;

} //end find()
```


Balanserade träd

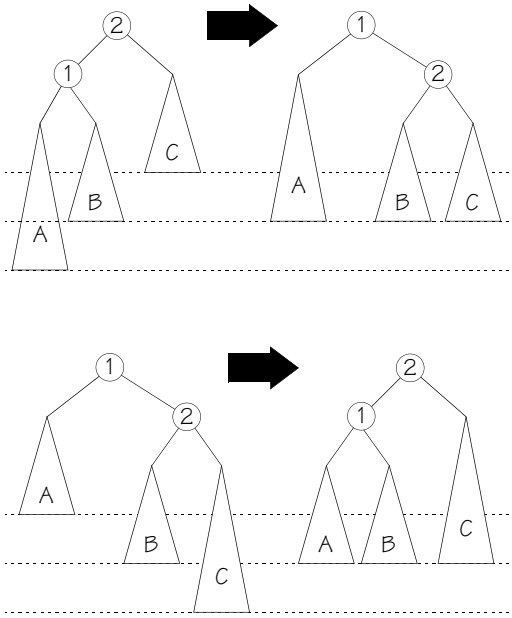
- Ett binärt träd är *perfekt balanserat* om det för varje nod gäller att antalet noder i de två subträden skiljer med max 1.

- Ett binärt träd är *balanserat* om det för varje nod gäller att höjden på de två subträden skiljer med max 1.

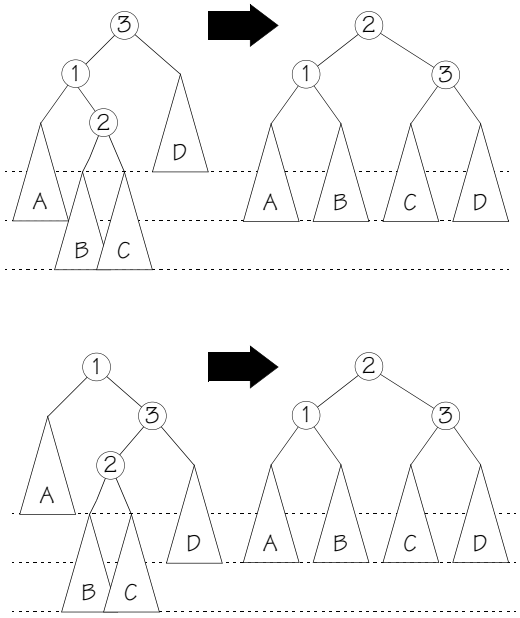
AVL-träd

Ett AVL-träd är ett binärt träd som hela tiden hålls balanserat.

Enkla rotationer

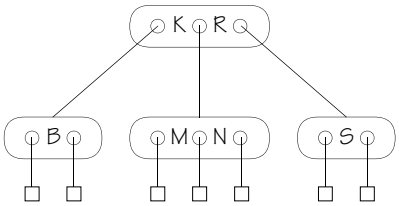


Dubbla rotationer



Två-tre-träd

Insert K
Insert R
Insert S
Insert B
Insert M
Insert N



Tries

- staty
- stencil
- stek
- anka
- andas
- brev
- bred
- bok