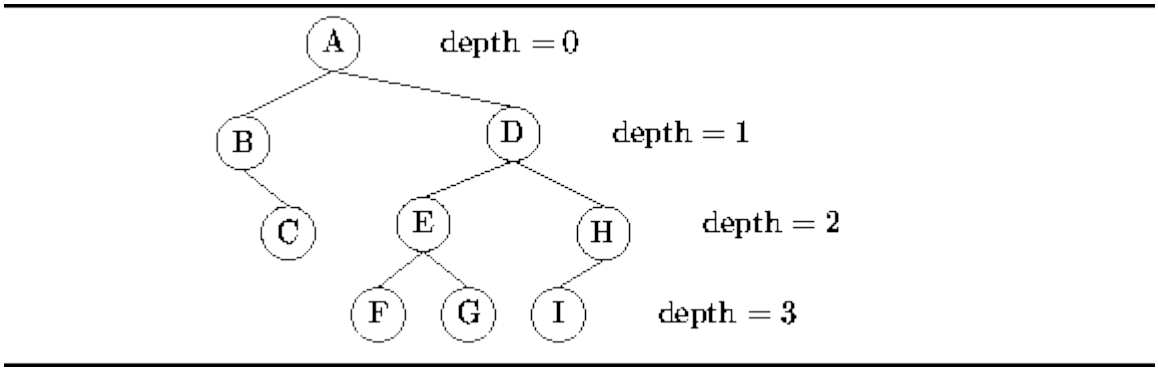
Дисциплина: *СД и ООП* **Тема:** *Двоично дърво - продължение*
Специалност: *КН 1 курс, 3 и 4 групи* **Дата**: *12.05.2010 г.*

Обхождане на непразно двоично дърво:

- *Низходящо (preorder) обхождане на двоично дърво*:
  1. Посещение на корена
  2. Низходящо обхождане на лявото поддърво
  3. Низходящо обхождане на дясното поддърво
- *Възходящо (postorder) обхождане на двоично дърво*:
  1. Възходящо обхождане на лявото поддърво
  2. Възходящо обхождане на дясното поддърво
  3. Посещение на корена
- *Смесено (inorder) обхождане на двоично дърво*:
  1. Смесено обхождане на лявото поддърво
  2. Посещение на корена
  3. Смесено обхождане на дясното поддърво
- *Обхождане по нива (breadth-first)*

***Пример*:** За дървото T:



последователността от символи, която се получава при

1.Низходящото обхождане е ***A, B, C, D, E, F, G, H, I***
2.Възходящото обхождане е ***C, B, F, G, E, I, H, D, A***
3.Смесеното обхождане е ***B, C, A, F, E, G, D, I, H***
4.Обхождането по нива е ***A, B, D, C, E, H, F, G, I***

Итератори:

1.Класът **BTreeInorderIterator** представя итератор за *смесено обхождане на двоично дърво с елементи от тип* ***Object***:

```
public class BTreeInorderIterator implements java.util.Iterator {
        //Data
        java.util.LinkedList list;
        java.util.Iterator it;

        //Constructor
        public BTreeInorderIterator(BinaryTree tree) {
                list = new java.util.LinkedList();
                inorder(tree);
                reset();
        }

        //Private method
        private void inorder(BinaryTree tree) {
                if (!tree.isEmpty()) {
                        inorder(tree.getLeftSubtree());
```

```
                        list.addLast(tree.getRoot().data());
                        inorder(tree.getRightSubtree());
                }
        }

        //Iterator methods
        public boolean hasNext() { return it.hasNext(); }
        public Object next() { return it.next(); }
        public void reset() { it = list.iterator(); }
        public void remove() { it.remove(); }
}
```

2.Класът **BTreePreorderIterator** представя итератор за *низходящо обхождане на двоично дърво с елементи от тип* ***Object***, съгласно спецификацията:

```
public class BTreePreorderIterator implements java.util.Iterator {
        //Данни

        //Конструктор
        public BTreePreorderIterator(BinaryTree tree) {...}

        //Реализация на методите на интерфейса Iterator
}
```

3.Класът **BTreePostorderIterator** представя итератор за *възходящо обхождане на двоично дърво с елементи от тип* ***Object***, съгласно спецификацията:

```
public class BTreePostorderIterator implements java.util.Iterator {
        //Данни

        //Конструктор
        public BTreePostorderIterator(BinaryTree tree) {...}

        //Реализация на методите на интерфейса Iterator
}
```

4.Класът **BTreeBreadthFirstIteratorEasy** представя итератор за *обхождане по нива на двоично дърво с елементи от тип* ***Object***:

```
import java.util.*;

public class BTreeBreadthFirstIteratorEasy implements Iterator {
        //Data
        LinkedList list;
        Iterator it;

        //Constructor
        public BTreeBreadthFirstIteratorEasy(BinaryTree tree) {
                list = new LinkedList();
                breadthFirstTraversal(tree);
                reset();
        }

        //Private method
        private void breadthFirstTraversal(BinaryTree tree) {
                LinkedList<TreeNode> queue = new LinkedList<TreeNode>();
                queue.addLast(tree.getRoot());
                while(!queue.isEmpty()) {
                        TreeNode temp = queue.removeFirst();
                        list.addLast(temp.data());
                        if(temp.left() != null)
                                queue.addLast(temp.left());
                        if(temp.right() != null)
```

```
                queue.addLast(temp.right());
            }
        }

        //Iterator methods
        public boolean hasNext() { return it.hasNext(); }
        public Object next() { return it.next(); }
        public void reset() { it = list.iterator(); }
        public void remove() { it.remove(); }
}
```

5.Класът **BTreeBreadthFirstIterator** представя итератор за *обхождане по нива на двоично дърво с елементи от тип* ***Object***:

```
import java.util.*;

public class BTreeBreadthFirstIterator implements Iterator {
        //Data
        Vector<LinkedList> levels;          //Vector of levels
        Iterator it;
        int level;

        //Constructor
        public BTreeBreadthFirstIterator(BinaryTree tree) {
            levels = new Vector<LinkedList>();
            breadthFirstTraversal(tree);
            reset();
        }

        //Private method
        private void breadthFirstTraversal(BinaryTree tree) {
            LinkedList<TreeNode> currentLevel = new LinkedList<TreeNode>();
            if(!tree.isEmpty()) currentLevel.addLast(tree.getRoot());
            while(!currentLevel.isEmpty()) {
                LinkedList data = new LinkedList();
                LinkedList<TreeNode> nextLevel = new LinkedList<TreeNode>();
                Iterator<TreeNode> it = currentLevel.iterator();
                while(it.hasNext()) {
                    TreeNode n = it.next();
                    data.addLast(n.data());
                    if(n.left() != null)
                        nextLevel.addLast(n.left());
                    if(n.right() != null)
                        nextLevel.addLast(n.right());
                }
                levels.add(data);
                level++;
                currentLevel = nextLevel;
            }
        }

        //Methods
        public int numberOfLevels() { return levels.size(); }
        public LinkedList getData(int level) { return levels.get(level); }

        //Iterator methods
        public boolean hasNext() {
            if(it.hasNext()) return true;
            level++;
            if(level < levels.size()) {
                it = levels.get(level).iterator();
                return true;
            }
            return false;
```

```
        }

        public Object next() {
                if(!hasNext()) throw new NoSuchElementException();
                return it.next();
        }

        public void remove() { throw new UnsupportedOperationException(); }

        public void reset() { level = 0; it = levels.get(0).iterator(); }
}
```

Тестване: Програмата **TestBinarySearchTree:**

1.Чете думи от файла testTreeData1.txt и създава двоично дърво от думи

tree = new BinarySearchTree(new NaturalComparator())

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult1.txt.

2.Чете низове от вида: дума@цяло_число от файла testTreeData2.txt и създава двоично дърво от двойки (дума,цяло_число) – обекти на класа **Entry**, като се използва класа **EntryComparator**

tree = new BinarySearchTree(new EntryComparator())

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult2.txt.

3.Чете низове от вида: дума@цяло_число от файла testTreeData3.txt и създава двоично дърво от двойки (дума,цяло_число) – обекти на класа **Entry**, като се използва класа **EntryComparator2**

tree = new BinarySearchTree(new EntryComparator2())

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult3.txt.