Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №11
13.05.2010г.

ТЕМА: Двоични дървета. Итератори

```
import java.util.Iterator;

/*
 * Abstract structure Binary Tree.
*/
public class BinaryTree<E> {

	protected E value;
	protected BinaryTree<E> parent;
	protected BinaryTree<E> left;
	protected BinaryTree<E> right;

	public static final BinaryTree EMPTY_TREE = new BinaryTree(); // Empty tree

	/**
	 * Constructs an empty tree
	 */
	private BinaryTree() {
		value = null;
		parent = null;
		left = right = this;
	}

	/**
	 * Constructs an one-element tree
	 */
	public BinaryTree(E value) {
		this.value = value;
		parent = null;
		left = right = EMPTY_TREE;
	}

	/**
	 * Constructs a tree with given subtrees
	 */
	public BinaryTree(E value, BinaryTree<E> left, BinaryTree<E> right) {
		this(value);
		setLeft(left);
		setRight(right);
	}
```

```
	. . . . . .

	/**
	 * Returns a in-order iterator of the tree
	 */
	public Iterator<E> inorderIterator() {
		return new BTreeInorderIterator<E>(this);
	}

	/**
	 * Returns a pre-order iterator of the tree
	 */
	public Iterator<E> preorderIterator() {
		return new BTreePreorderIterator<E>(this);
	}

	/**
	 * Returns a post-order iterator of the tree
	 */
	public Iterator<E> postorderIterator() {
		return new BTreePostorderIterator<E>(this);
	}

	/**
	 * Returns a level-order iterator of the tree
	 */
	public Iterator<E> levelorderIterator() {
		return new BTreeLevelorderIterator<E>(this);
	}
}


import java.util.Iterator;
import java.util.Stack;
import java.util.NoSuchElementException;

class BTreeInorderIterator<E> implements Iterator<E> {

	BinaryTree<E> root;
	Stack<BinaryTree<E>> stack;

	BTreeInorderIterator(BinaryTree<E> root) {
		this.root = root;
		stack = new Stack<BinaryTree<E>>();
		BinaryTree<E> current = root;
		while(!current.isEmpty()) {
			stack.push(current);
```

```
                current = current.left();
            }
        }

        public boolean hasNext() {
            return !stack.isEmpty();
        }

        public E next() {
            if(stack.isEmpty())
                throw new NoSuchElementException();
            BinaryTree<E> node = stack.pop();
            E result = node.value();
            if(!node.right().isEmpty()) {
                BinaryTree<E> current = node.right();
                do {
                    stack.push(current);
                    current = current.left();
                } while (!current.isEmpty());
            }
            return result;
        }

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

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Stack;

class BTreePreorderIterator<E> implements Iterator<E> {
        BinaryTree<E> root;
        Stack<BinaryTree<E>> stack;

        BTreePreorderIterator(BinaryTree<E> root) {
            this.root = root;
            stack = new Stack<BinaryTree<E>>();
            if(root != null)
                stack.push(root);
        }

        public boolean hasNext() {
            return !stack.isEmpty();
        }

        public E next() {
```

```
            if(stack.isEmpty())
                throw new NoSuchElementException();
            BinaryTree<E> node = stack.pop();
            E result = node.value();
            if(!node.right().isEmpty())
                stack.push(node.right());
            if(!node.left().isEmpty())
                stack.push(node.left());
            return result;
        }

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

import java.util.Iterator;
import java.util.Stack;
import java.util.NoSuchElementException;

class BTreePostorderIterator<E> implements Iterator<E> {

        BinaryTree<E> root;
        Stack<BinaryTree<E>> stack;

        BTreePostorderIterator(BinaryTree<E> root) {
            this.root = root;
            stack = new Stack<BinaryTree<E>>();
            BinaryTree<E> current = root;
            while(!current.isEmpty()) {
                stack.push(current);
                if(!current.left().isEmpty())
                    current = current.left();
                else
                    current = current.right();
            }
        }

        public boolean hasNext() {
            return !stack.isEmpty();
        }

        public E next() {
            if(stack.isEmpty())
                throw new NoSuchElementException();
            BinaryTree<E> node = stack.pop();
            E result = node.value();
            if(!stack.isEmpty()) {
```

```
                BinaryTree<E> current = node;
                BinaryTree<E> parent = stack.peek();
                if(current == parent.left()) {
                        current = parent.right();
                        while(!current.isEmpty()) {
                                stack.push(current);
                                if(!current.left().isEmpty())
                                        current = current.left();
                                else
                                        current = current.right();
                        }
                }
        }
        return result;
    }

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

import java.util.Iterator;
import java.util.Queue;
import java.util.LinkedList;

class BTreeLevelorderIterator<E> implements Iterator<E> {

    protected BinaryTree<E> root;
    protected Queue<BinaryTree<E>> queue;

    public BTreeLevelorderIterator(BinaryTree<E> root) {
        this.root = root;
        queue = new LinkedList<BinaryTree<E>>();
        if(!root.isEmpty())
            queue.add(root);
    }

    public boolean hasNext() {
        return !queue.isEmpty();
    }

    public E next() {
        BinaryTree<E> current = queue.remove();
        E result = current.value();
        if(!current.left().isEmpty())
            queue.add(current.left());
        if(!current.right().isEmpty())
            queue.add(current.right());
```

```
            return result;
        }

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

Задача: Да се напише на езика Java клас за представяне на ***n***-арно дърво. Да се напише метод за проверка дали две дървета са изоморфни.
Две дървета са изоморфни, ако данните в корените им равни, корените на двете дървета имат равен брой поддървета и всяко поддърво на първия корен има изоморфно измежду поддърветата на втория корен и обратно.