



RUTGERS

CS112 Data Structures

Recitation 08

Yu Yang

yy388@cs.rutgers.edu

Office: CoRE 331

Office Hour: 3-5pm, Wed.

2. Two binary trees are *isomorphic* if they have the same shape (i.e. they have identical structures.) Implement the following **recursive** method:

```
public static <T> boolean isomorphic(BTNode<T> T1, BTNode<T> T2) {  
    /* your code here */  
}
```

that returns **true** if the trees rooted at T1 and T2 are isomorphic, and false otherwise. **BTNode** is defined as follows:

```
public class BTNode<T> {  
    T data;  
    BTNode<T> left, right;  
    BTNode(T data, BTNode<T> left, BTNode<T> right) {  
        this.data = data;  
        this.left = left;  
        this.right = right;  
    }  
}
```

SOLUTION

```
public static <T> boolean isomorphic(BTNode<T> T1, BTNode<T> T2) {  
    if (T1 == null && T2 == null) return true;  
    if (T1 == null || T2 == null) return false;  
    if (!isomorphic(T1.left, T2.left)) return false;  
    return isomorphic(T1.right, T2.right);  
}
```

6. * Suppose you are given the following binary tree class definition, which uses the `BTNode<T>` class of the previous exercise.

```
public class BinaryTree<T> {  
    private BTNode<T> root;  
    ...  
}
```

Add the following methods to this class so that applications can do an inorder traversal one node at a time:

```
// returns the first node that would be visited in an inorder traversal  
// null if tree is empty  
public BTNode<T> firstInorder() {  
    /* COMPLETE THIS METHOD */  
}
```

and

```
// returns the next node that would be visited in an inorder traversal;  
// returns null if there is no next node  
public BTNode<T> nextInorder(BTNode<T> currentNode) {  
    /* COMPLETE THIS METHOD */  
}
```

For instance, an application would call these methods like this to do the inorder traversal:

```
BinaryTree<String> strBT = new BinaryTree<String>();  
// insert a bunch of strings into strBST  
...  
// do inorder traversal, one node at a time  
BTNode<String> node = strBST.firstInorder();  
while (node != null) {  
    node = strBST.nextInorder(node);  
}
```

```
// returns the first node that would be visited in an inorder traversal;
// returns null if tree is empty
public BTreeNode<T> firstInorder() {
    if (root == null) {
        return null;
    }
    // left most node in tree is the first node in inorder
    BTreeNode<T> prev=root, ptr=root.left;
    while (ptr != null) {
        prev = ptr;
        ptr = ptr.left;
    }
    return prev;
}

// returns the next node that would be visited in an inorder traversal;
// returns null if there is no next node
public BTreeNode<T> nextInorder(BTreeNode<T> currentNode) {
    if (currentNode == null) { // playing defense here
        return null;
    }
    // if there is a right subtree, then right turn, and left all the way to bottom
    if (currentNode.right != null) {
        BTreeNode<T> ptr=currentNode.right;
        while (ptr.left != null) {
            ptr = ptr.left;
        }
        return ptr;
    }
    // no right subtree
    BTreeNode<T> p = currentNode.parent;
    while (p != null && p.right == currentNode) {
        currentNode = p;
        p = p.parent;
    }
    return p;
}
```

7. Exercise 9.4, page 295 of the textbook.

1. Build a Huffman tree for the following set of characters, given their frequencies:

R	C	L	B	H	A	E
6	6	6	10	15	20	37

2. Using this Huffman tree, encode the following text:

CLEARHEARBARE

3. What is the average code length?

4. If it takes 7 bits to represent a character without encoding, then for the above text, what is the ratio of the encoded length to the unencoded?

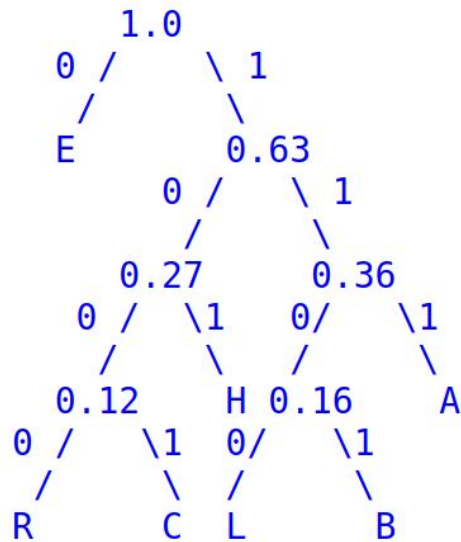
5. Decode the following (the string has been broken up into 7-bit chunks for readability):

1111011 1010111 1101110 0010011 111000

SOLUTION

1. The probabilities of occurrence of the characters, listed in ascending order:

R	C	L	B	H	A	E
0.06	0.06	0.06	0.1	0.15	0.2	0.37



$$3. 1 \cdot 0.37 + 4 \cdot 0.06 + 4 \cdot 0.06 + 3 \cdot 0.15 + 4 \cdot 0.06 + 4 \cdot 0.10 + 3 \cdot 0.20 = 2.54$$

4. Length of unencoded representation using 7 bits per character is $7 \cdot 13 = 91$, while length of representation using Huffman codes is 39. The ratio of encoded to unencoded is $39/91$.