LAB 1 - Prefix-free Codes Using Binary Trees

Submission Deadline: Jan. 26, 11:59 pm
Assessment: 5% of the total course mark.

Description:

In this assignment you are required to implement prefix-free codes using binary trees with linked nodes. For this, you have to write the `Java` classes `BinTree` and `TNode` in the same package. Class `BinTree` uses binary trees with linked nodes to represent prefix-free codes. Class `TNode` represents the nodes of the binary tree. You also need to perform the time and space complexity analysis of your algorithms.

You are not allowed to use any predefined `Java` methods other than those defined in the classes `java.util.ArrayList`, `java.lang.String` and `java.lang.Math`.

Definitions:

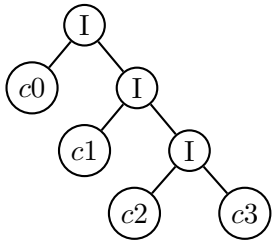
A prefix-free code (used in data compression) is a set of binary sequences (sequences of 0's and 1's) such that none of them is a prefix of another. For instance, the set $\mathcal{A} = \{0, 10, 110, 111\}$ is a prefix-free code. On the other hand, the set $\{0, 10, 100, 111\}$ is not a prefix-free code because the sequence 10 is a prefix of 100. The elements of the prefix-free code are referred to as (binary) **codewords**.

A prefix-free code can be used to **encode** a sequence of symbols from an alphabet $\mathcal{B}$ (i.e., **convert the sequence of symbols into a bitstream**) as follows. Each symbol in alphabet $\mathcal{B} = \{c0, c1, c2, \cdots, cn\}$ is assigned a distinct binary codeword. Then we can encode any sequence of symbols by replacing every symbol by the corresponding binary codeword. For instance, assume that alphabet $\mathcal{B}$ contains 4 symbols. Then they are $c0$, $c1$, $c2$, $c3$. Further, assume that symbol $c0$ is assigned 0, $c1$ is assigned 10, $c2$ is assigned 110 and $c3$ is assigned 111. Consider now the following sequence of symbols over the alphabet $\mathcal{B}$: "$c2$ $c3$ $c1$ $c2$ $c0$ $c0$ $c0$". This sequence is encoded into the bitstream "11011110110000".

Conversely, a bit sequence is **decoded** by dividing it first into non-overlapping codewords and then replacing each codeword with the corresponding alphabet symbol. For instance, the sequence "0011001010111" can be parsed as follows: "0,0,110,0,10,10,111". Then the decoded alphabet sequence is "$c0$ $c0$ $c2$ $c0$ $c1$ $c1$ $c3$". Note that the prefix-free property ensures that the parsing of the binary sequence into codewords is unique.

A prefix-free code can be represented using a binary tree. Note first that any path in the tree, from some node to one of its descendants, can be regarded as a sequence of branches. By replacing any left branch with 0 and any right branch with 1 we obtain a binary sequence that represents the path. In a binary tree representation of a prefix-free code, the paths from the root to the leaves represent the codewords. Thus, the number of leaves equals the number of codewords. **Each leaf stores the alphabet symbol corresponding to the binary codeword that describes the path from the root to that leaf.**

Example: For the prefix-free code $\mathcal{A} = \{0, 10, 110, 111\}$ used to encode the symbols of alphabet $\mathcal{B}$ as described above, the binary tree is the following ("I" stands for "internal node"):



SPECIFICATIONS:

You **have to use** the following `Java` class `TNode`:

```
public class TNode {
   String data;
   TNode left;
   TNode right;

   TNode(String s, TNode l, TNode r){
       data=s;
       left=l;
       right=r;
   }
}
```

For any internal node, the field `data` is null, while for any leaf, `data` stores the corresponding alphabet symbol as a string.

Class `BinTree` **has only one instance field**, namely a reference to the root of the tree (a reference variable of type `TNode`). It has to be `private`.

Class `BinTree` contains the following constructor:

- `public BinTree(String[] a) throws IllegalArgumentException` - constructs a `BinTree` that represents the prefix-free code stored in the `String` array `a`. Each `String` in the array `a` is a binary sequence (contains only 0's and 1's) representing a codeword. The codeword stored in `a[i]` corresponds to the alphabet symbol ci. Thus, the tree leaf corresponding to this codeword must store the `String`: `"c" + i`. If array `a` contains two identical codewords, or a codeword that is a prefix of another one, then the set of codewords is not prefix-free and the constructor should **throw an** `IllegalArgumentException` with the message "Prefix condition violated!". If any element in the array is not a binary string, the method should **throw an** `IllegalArgumentException` with the message "Invalid argument!".

Class `BinTree` contains the following public methods (you may declare additional `private` methods, but not `public` methods):

- `public void printTree()` - visits the tree nodes through an **inorder** traversal and prints "I" for each internal node, and the symbol stored in the leaf (in `data`) for each leaf. It invokes a recursive method `printTree()`. You **have to use** the following `Java` code:

```
public void printTree(){ printTree(root);}

private void printTree(TNode t){
   if(t!=null){
       printTree(t.left);
       if(t.data == null )
           System.out.print("I ");
       else
           System.out.print(t.data + " ");
       printTree(t.right);
   }
}
```

- `public void optimize()`. If the tree is not a full binary tree (i.e., where each internal node has two children), the lengths of some codewords can be reduced by removing some bits, while the prefix-free condition is maintained. This method checks if the tree is full and if it is not, it performs these changes (i.e., it reduced codewords and adapts the tree) until the tree becomes full.
- `public ArrayList<String> getCodewords()` - returns an `ArrayList<String>` object that stores the codewords in lexicographical order[1]. Each item in the list is a codeword (a string of 0's and 1's).
- `public String[] toArray()` - returns an array representation of the binary tree. You have to use the convention given in COMP ENG 2SI3 for the representation of binary trees using arrays. The first array element stores `null`. Any array element corresponding to a missing tree node stores `null`. Any array element corresponding to an internal node stores `"I"`. Any array element corresponding to a leaf stores the string corresponding to that leaf.
- `public String encode(ArrayList<String> a)`. The input list `a` represents a sequence of alphabet symbols. Each list item is a `String` object representing a symbol, i.e., a string consisting of letter 'c' followed by a number that is at most equal to n-1, where n is the number of leaf nodes. This method encodes the input and outputs the corresponding bitstream. You may assume that each string in the list is a valid alphabet symbol.
- `public ArrayList<String> decode(String s) throws IllegalArgumentException`. The input string `s` contains only 0's and 1's). This method outputs the sequence of alphabet symbols obtained by decoding the bit sequence `s`. Each alphabet symbol has to be stored as a separate item in the list. The order of items is important here. Assume that `s` is a concatenation of codewords, so it can be decoded. The

[1]The lexicographical order of two binary strings is the same as their alphabetical order if we replace '0' by 'a' and 1 by 'b'.

method throws an exception if the input stream is not binary or of it is binary, but cannot be parsed int a sequence of codewords.

- `public String toString()` - returns the string representation of the prefix-free code as the sequence of codewords in lexicographical order separated by empty spaces and ending with an empty space. For our example, this string is "0 10 110 111 ".

You have to include meaningful comments at key points in your code.

For each method, you have to indicate in a comment the time and space complexities of the algorithm and include a brief, but clear justification.

Your algorithms have to be efficient.

Submission Instructions: Submit the source code for the `Java` class `BinTree` in a single text file. Include your name and student number in the name of the file. For instance, if your name is "Ellen Davis" and the student number is 12345 then the file should be named "Lab4EllenDavis12345.txt".