Umm Al-Qura University
Computers and Information Systems
College

Computer Science Department
Data Structures & Algorithms
Spring 2016

Laboratory Notes - Session 8
Huffman Decoding

Time allowed	Deadline	Points
One week	Beginning of Week 14's session.	5
Required software	**Deliverables**	
Java editor & compiler	Error-free executable program code [.java] Program demonstration	

Sessions Objectives:

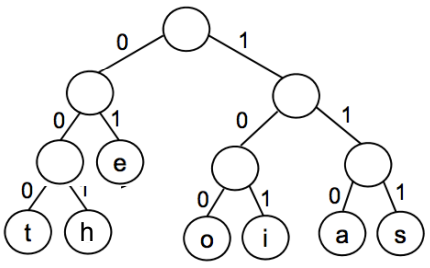
You will implement the initial structures and objects that shall be used in for file compression using Huffman Encoding. This includes: Binary Tree and traversing methods.

Background:

Every file, written in ASCII, can be thought of as a sequence of bytes (values from 0 to 255). Uncompressed files use 8 bits for each possible byte value. The idea of Huffman coding is simply to use an encoding scheme so that the most frequently occurring byte values are represented by a short code (fewer bits) and the less frequently occurring byte values are represented by a longer code (more bits). Huffman encoding significantly reduces the size of files, but it requires that you be able to read or write a bit at a time.

The algorithm for Huffman coding generates a binary tree, similar to the shown figure, which has:

- Right edges are labeled by `0`
- Left edges are labeled by `1`
- The leaves of the tree are the characters



The path from the root to the leaf gives the encoding of the byte represented by that node. For example, from the figure above:

- t is represented by 000
- h is represented by 001
- e is represented by 01
- o is represented by 100
- i is represented by 101
- a is represented by 110
- s is represented by 111

Leaves that are close to the root have short encodings and leaves that are farther away have longer encodings. The trick is to generate a tree where the most frequently occurring byte values are placed in leaves close to the root.

Decode a compressed file:
Read one bit at a time and traverse the tree, starting from the root:

- when you read a bit of 1, go to the right child
- when you read a bit of 0, go to the left child
- when you reach a leaf node, record the character, return to the root, and continue reading bits.

For example, if a message is given as '00101110000' it can be encoded using the tree above into: 'heat' as follows:

00101**110**000 = **h** e **a** t

Exercise Description & Requirements:

In this assignment, you will create a binary tree data structure and manually build a Huffman encoding tree. Then you will use this tree to decode a sequence of bit.

You will be provided with the initial BT node class (`BTNode.java`).

In order to complete this assignment, you have to:

1. Download `BinaryTree.java` from the eLearn system.
2. Create a project with the name `HuffmanDecoder`.
3. You need to define the following data members of `HuffmanDecoder`:
   - `root` is a BTNode object which represent the Huffman Encoding tree.
4. Implement each method in the `BinaryTree.java` inside the `HuffmanDecoder` class.
5. In addition, you need to implement the following methods:
   - `void HuffmanDecoder ()`
     A constructor method to create a new empty binary tree.

   - `void HCTreeExample()`
     Create the tree shown above manually.

     HINT:
     If you want to create the following tree:
     'a' is root node
     'a' has a left node 'b'
     'b' has a right node 'c'
     The following code is used:

```
root.addLeft(new BTNode('a'));
root.left.addLeft(new BTNode('b'));
root.left.left.addRight(new BTNode('c'));
```

- `void HCDecode(String code)`

  Implement a Huffman decoding method using the Huffman encoding tree stored at the `root`. You will decode the string given in argument `code`.

  Use the algorithm given in the background section above.

  Test your code by decoding the following:

  ```
  000001101111
  11101110000
  00010101111
  11101110
  11000001
  11011100101111
  ```

- `void HCDecode(String filename)`

  Implement a Huffman decoding method using the Huffman encoding tree stored at the `root`. You will decode the bits stored in the file `filename`.

  You need to read the file in a binary mode bit by bit.

  Use the algorithm given in the background section above.