EECS 233 Programming Assignment #2

Due Thursday, October 18, 2012 (23:59:59 EST)

In this assignment, you will implement the Huffman encoding, which uses a combined linked list and binary tree data structure. You will define a node class with the following fields:

- `character`: the character denoted by the node. This is meaningful only for the leaf nodes of a generated Huffman tree.
- `frequency`: the frequency of occurrences of all characters in the subtree of the generated Huffman tree. For a leaf node, this frequency is the frequency of the character in the leaf node; for an interior node, the frequency is the sum of all frequency values in the leaves of the subtree.
- `next`: the link field used by the sorted linked list. The sorted linked list is the input of the Huffman encoding algorithm. The linked list is generated after a text file (to be compressed) is scanned to obtain the frequency values of all characters. Note in the generated final Huffman tree this link field should be null (see the algorithm) in all nodes.
- `left`: left child of a node in the Huffman tree
- `right`: right child of a node in the Huffman tree
- `code`: this will contain the encoding for the particular character.

The public interfaces should be methods defined as follows:

```
public static void Huffman_coder(String input_file, String output_file);
```

`Huffman_coder` reads an input text file `input_file` and outputs a compressed file in `output_file`. The output file will be a binary fine containing the binary bits produced by your Huffman coder.

You may need to implement several helper methods, for example (1) one method to scan a text file to generate the initial linked list (you will need to access a file and define a local table in this method to remember the frequency), (2) one method to run the Huffman encoding algorithm to produce the Huffman tree, (3) one method to traverse the Huffman tree to output the character encoding, (4) a method to write a Huffman code to a file, and so on. You should think through the high-level organization of your code so that you have a clear organization and well-chosen methods.

You can use whatever method of computing the character frequencies. However, once you have them, you do need to generate the initial linked list and then produce the Huffman tree, rather than assign variable-length codes to characters in some ad-hoc manner.

As a test of your program, you need to run it on a real sizable text file. Please pick a **plain-text** literary piece at ***http://www.gutenberg.org/browse/scores/top*** and use that piece for (a) generating the Huffman tree and (b) encoding the text using the generated encoding. (Again, make sure you use a plain-text version of your piece.) Please compute the space reduction you would achieve on your selected text. You can either measure this directly by comparing the file lengths or keep track of the lengths inside your code. **Important: Please either find a text up to around 100K or truncate the text and use only the first 100K of it for this step! (Otherwise people running the blackboard will be really upset with me…)**

Deliverables:

A zip file containing:

- All source codes, including sufficient comments for the TAs to understand what you have done.
- README file with instructions on how to run your program. This can also explain some of the logic and organization of your code.
- The text file you used to test your program.
- The Huffman encoding you generated using the above file.
- The encoded text file.
- The calculated (or actual) amount of space savings your encoding has achieved on your text file.

Implementation details. Note these should be considered the minimal functions required, however define other methods if required.

- create class HTreeNode
  - data: character, frequency, next, left, right, code (as explained above)
  - methods: whatever you think is required.
- create class HTree
  - data: HTreeNode root (which is the pointer to the root node)
  - methods: whatever you think is required.
- create class HLinkedList
  - data: HTreeNode head (which is the head of the list)
  - methods:
    HTreeNode getHeadNode();
    void insertIntoPosition(HTreeNode node,int position)
    HTreeNode find (HTreeNode node)
    String toString() → a comma separated sequence of the elements;
- create a class called HuffmanLibrary.
  - The class should have the following methods (apart from any other you use for convenience)
    public static String readFileAsString(String fileName);
    public static HLinkedList createHList(String fileContents);
    public static HLinkedList getSortedLinkedList (HLinkedList);
    public static HTree createHuffmanTree(HLinkedList list);
    -- Note: should work for both sorted and unsorted data
    public void updateCodeValues (HLinkedList,HTree);
    public static void Huffman_coder(String input_file, String output_file);

Strategy & hints:

- Firstly read the input file into a string.
- Use the string to create the linked list of nodes.
- Get each character from the string and add them to the linked list if it is not already there, else just increase the frequency.

- When the list is ready, sort the list with respect to the frequencies.
- The easiest way to sort is create another list and insertion sort it with the elements from the already existing one (use the insertIntoPosition function).
- Use the sorted linked list and use it to create a Huffman tree.
- Again remove head and insert into position are all the functions that you will need once the tree is created
- Update the code values of each character in the linked list that we had created before.
- Go through the Huffman tree and assign the code values to each node in the list. Substitute these codes for the actual characters and obtain a string representation of the entire character sequence. This is what should be the binary representation of our “zipped” file.

[See next page]

You can use and modify the following code for a class called BinaryFileWriter to create the binary "zipped" file. It takes the args filename and the boolean sequence that you generate using the Huffman encoding.

```
import java.io.File;
import java.io.FileOutputStream;

public class BinaryFileWriter {
	public static void writeBinaryFile(String fileName,byte[] b) throws Exception{
		FileOutputStream f = new FileOutputStream (new File(fileName));
		f.write(b);
		f.close();
	}
	public static void createBinaryFile(String seq,String fileName) throws Exception{
		writeBinaryFile(fileName,toByteSequence(seq));
	}

	public static byte[] toByteSequence(String data) throws Exception{
		int j=-1;
		int byteArrSize = data.length()/8;
		if (data.length()%8 != 0)
			byteArrSize++;
		byte[] byteSeq = new byte[byteArrSize];
		for(int i=0;i< data.length();i++){
			if (i%8 == 0){
				j++;
				byteSeq[j] = 0x00;
			}
			byte tmp ;
			if (data.charAt(i) == '1'){
				tmp = 0x01;

			}
			else if (data.charAt(i) == '0') {
				tmp = 0x00;
			}
			else
				throw new Exception ("error in format");
			byteSeq[j] = (byte) (tmp | ( byteSeq[j] << 1));

		}

		return byteSeq;

	}
}
```