Lab_huffman Hazardous Huffman

Week #7 – February 28-March 2, 2018

Welcome to Lab Huffman!

Course Website: https://courses.engr.illinois.edu/cs225/labs

Overview

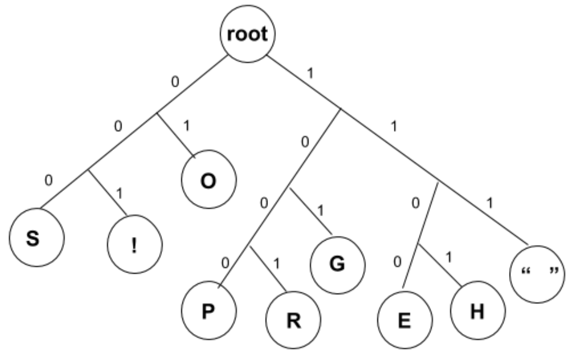
This week's lab will introduce a new application of binary trees: Huffman encoding. There are two parts in this worksheet: **Part 1** will include exercises about Huffman encoding, and **Part 2** will provide a review of Binary Search Trees as a gentle stepping stone to next week's lab: lab_avl.

Part 1: Huffman Encoding

Before attempting this part of the worksheet, please read the introduction to Huffman Encoding provided on lab_huffman's webpage. You can alternatively watch the video for this lab; a link to the youtube video is provided in this lab's webpage. Both the video and the webpage go over the same example of how to construct a huffman tree.

Encoding and Decoding

Now that we know how to construct a Huffman tree from a given text, let's practice how to use the Huffman tree to encode and decode messages. Suppose we are given the following Huffman tree to use:



Exercise 1.1: Using the tree given above, what would be the Huffman encoding of the word "heroes" ?

Exercise 1.2: What does the following string of bits say? Use the Huffman tree above to decode it:

1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1 0 0 1 0 0 0 0 0 1

Save Huffman Tree to File

Suppose you and your friend in UIC want to use Huffman encoding to pass secret messages to each other. But before you can start, you need to send your friend the Huffman tree you created so that they'll be able to encode and decode messages. You would like to save your tree in a file as a string of characters, and send that file to your friend.

Exercise 2.1: The following pseudocode recursively translates a binary tree into a sequence of characters and writes it to a text file; this algorithm uses **1** as a flag to signal a leaf node, and **0** as a flag to signal an internal node. ***Do not confuse these 0's and 1's with the edge labels in a Huffman tree!***
How will the example tree in **Exercise 1** be saved in a file? What would the output file look like?

```
1) Start at the root
2) If the current node is a leaf:
    a) Write a "1" to the output file
    b) Write the character that the leaf node represents to the
       output file
3) Else (the current node is an internal nod):
    a) Write a "0" to the output file
    b) Recurse on the left subtree, then the right subtree
```

	output.txt
1	
2	
3	

<u>Exercise 2.2:</u> Suppose your friend sent you the following file, can you construct the original binary tree that it represents?

```
treeFile.txt
0 1 A 0 1 B 0 1 C 1 D

```

// Draw the original tree here:

Part 2: Binary Search Trees

In order to make binary trees more efficient, we implement them as search structures which we call Binary Search Trees (BST). BSTs have the property that each every node's value is larger than the value of any node in its left subtree, and is smaller than the value of any node in its right subtree. In class, we learned about the operations on BSTs - find, insert, remove. In **Exercise 4**, we will explore the remove function in a BST. Remember there are **three cases** when removing a node in a BST:

1. Remove a leaf node → just cross it out
2. Remove a node with one child → replace with the child
3. Remove a node with two children → replace with IOP (In-Order Predecessor)

```
       3
     /   \
   1      8
    \   /  \
     2  4   11
         \
          5
           \
            7
           /
          6
```

<u>Exercise 4.1:</u> Given the above binary search tree, what will the new tree look like if we remove the node **11** ?

<u>Exercise 4.2:</u> What will the new tree look like if we remove the node **4** ?

<u>Exercise 4.3:</u> What will the new tree look like if we remove the node **8** ?

In the programming part of this lab, you will:

- Complete the implementation of the HuffmanTree class:
- Implement the buildTree() function
- Implement the writeTree() and readTree() functions for writing and reading a binary tree from a file.
- Have fun encoding and decoding!

As your TA and CAs, we're here to help with your programming for the rest of this lab section!