CMPS 1500 Introduction to Computer Science I – Fall 13

11/5/13

9. Homework

Programming portion due **Friday 11/15/13** at 11:55pm on Blackboard.
Written portion (problem 1) due **Friday 11/15/13** at the beginning of class.

Please download the template script `hw9_template.py`, and rename it as `lastName_firstName_hw9.py`. The written portion can be turned in on paper.

In order to receive any credit for the programming portions, you are required to thoroughly comment and test your code.

1. **Huffman coding on paper (6 points)**
   Consider the following input text: `aaabbbbccccddddddd` . This input text is also given in `input1.txt`.

   (a) (1 point) What is the list of symbol frequencies for this text?

   (b) (3 points) On paper, compute the Huffman tree for the input text. Show all the steps of the construction, and identify the final tree.

   (c) (1 point) Encode the input text using the Huffman tree you just constructed.

   (d) (1 point) The input text uses ASCII character encoding. Each ASCII character in a text file is stored using one byte (= eight bits). What is the length of the input text in bits and in bytes? What is the length of the Huffman-encoded text in bits and in bytes?

2. **Huffman coding implementation (18 points)**
   Please download the file `hw9.zip`; this includes the following files: `input1.txt, input3.txt, trainingInput.txt, output2.txt`. In this project, a `HuffmanTree` node contains the `frequency` associated with this node, the string `chars` of all characters in all leaves in the tree rooted at this node, as well as references to the `left` and `right` subtrees.

   (a) (2 points) The function `construct_HuffmanTree(s)` serves to initialize the HuffmanTree construction: First it processes the string `s` by computing a list `frequencyList` of `(character,frequency)` pairs, which is sorted by increasing frequency. The missing code converts this list into a list of singleton HuffmanTree nodes (with corresponding `character` and `frequency` values). It then calls `construct_HuffmanTree_from_List(...)`.
   Fill in the missing code. Test your code using `input1.txt` and visualize the tree using `print_tree(...)`.

   (b) (5 points) The function `construct_HuffmanTree_from_List(...)` is the main function that constructs the HuffmanTree from a sorted list of HuffmanTrees. It should use the following functions:

   - `combineTrees(tree1,tree2)` takes two HuffmanTrees and returns a combined HuffmanTree with an updated `frequency`, `chars` string, and `left` and `right` references.
   - `insert_HuffmanTree(...)` inserts one new HuffmanTree at the correct position into a sorted list of HuffmanTrees.

FLIP OVER TO BACK PAGE ⟹

Fill in the code for this function. Test your code using `input1.txt` and visualize the tree using `print_tree(...)`.

(c) (4 points) The `encode(...)` function encodes an input string given a HuffmanTree. The result should be a bit string of '0's and '1's. Fill in the missing code. Test your code using `input1.txt`.

(d) (2 points) Test your encoding code with `trainingInput.txt` as the string to construct the HuffmanTree from, and with `input1.txt`, `input3.txt`, as well as `trainingInput.txt` as string inputs to be encoded. For each of these three input strings and encoded strings, compute their length in bits and in bytes.

(e) (4 points) The `decode(...)` function decodes an input bit string given a HuffmanTree. Fill in the missing code. Test your code by decoding a previously encoded string.

(f) (1 point) Test your `decode` function on `output2.txt`. What is the decoded text?