CS 430 Programming Languages
Spring 2016
Module 11 PA

JAMES MADISON UNIVERSITY

Introduction

In this PA will implement functions to encode texts as bit strings and decode bit strings to texts using Huffman coding. I have prepared a template file called Mod11PATemplate.hs. Please download this file and rename it Mod11PA.hs. Then open the file and write your code in it, filling out the stubbed functions.

This template embodies one design for a solution to this problem. If you want to, you can solve the problem differently. If you do, please change the test functions at the bottom of the program to fit your implementation, and let me know so that I will know how to grade the result (our goal is correct encode, decode, and makeHuffmanTree functions, so those cannot change).

The Tests

If you look towards the bottom of Mod11PATemplate.hs you will see a bunch of test functions. You should not change these functions in any way. If you load this file into ghci as is and run the main function, you will see that all the tests fail. The idea is that as you write your code, you can keep running main (or any of the other test functions) and determine whether your code is (probably) correct. I will also run main to see if you code is correct.

The Functions

Several of the functions have preconditions. If these are violated, the behavior of the program is undefined (in other words, it can do anything, including crash).

The functions we are really interested in are encode, decode, and makeHuffmanTree, which are specified as follows.

> makeHuffmanTree :: String -> Tree. This function must take an arbitrary string and create and return a Huffman coding tree for it. Precondition: the string parameter must contain at least two distinct characters.
>
> encode :: String -> String. This function must take an arbitrary string as its parameter, build a Huffman tree from it, use that tree to encode the text as a bit string (just a plain string containing only 0 and 1 characters), and return the resulting bit string. Precondition: the string parameter must contain at least two distinct characters.
>
> decode :: String -> Tree -> Bool. This function must take an arbitrary bit string and a Huffman coding tree and return the text decoded from the tree. If the bit string parameter is the empty string, it must return the empty string. If the tree parameter is the empty tree, it must return the empty string. If the last bits in the bit string do not reach the bottom of the Huffman tree (that is, they do not encode a character), then these last bits must be ignored (that is, they must not result in any characters in the result).

Once we have these three functions, we can encode and decode strings, and decode strings using an arbitrary Huffman tree as well. One way to implement these functions is to implement the

functions stubbed out at the top of the template file. Here are descriptions of them (you can also look at the test cases to see what they are supposed to do, of course).

countChars :: String -> [(Char, Int)]. This function must take a string and return a list of pairs where the first element of each pair is a character from the string and the second is its number of occurrences in the string. The first elements of each pair must occur only once in the list.

makeTreeList :: [(Char, Int)] -> [Tree]. This function must take a list of pairs consisting of characters and counts and return a list of Huffman tree leaf nodes. The leaf nodes must contain the characters and their counts as indicated in the parameter, and appear in the list in the same order as in the parameter.

makeOrderedTreeList :: [(Char, Int)] -> [Tree]. This function must take a list of pairs consisting of characters and counts and return a list of Huffman tree leaf nodes. The leaf nodes must contain the characters and their counts as indicated in the parameter, and appear in the list ordered by the character counts in the nodes, from least to greatest, with ties broken arbitrarily.

treeListInsert :: Tree -> [Tree] -> [Tree]. This function must take two parameters: a Huffman tree node and an ordered list of Huffman tree nodes. It must return the result of inserting the node into the list, with the result still ordered. Preconditions: the tree list does not contain the empty tree and it ordered.

buildTreeFromList :: [Tree] -> Tree. This function must take an ordered list of Huffman tree nodes and return a complete Huffman tree built from the list. Preconditions: the list is ordered and does not contain the empty tree.

makeHuffmanTable :: String -> Data.Map.Map Char String. This function must take a string and return an instance of Data.Map (look it up) that maps each character in the string to its Huffman bit string encoding. Precondition: the string parameter must contain at least two distinct characters. (This Map is useful for encoding strings.)

Deliverable Requirement

Your program must be named Mod11PA.hs. This file must have test code at the bottom unchanged from Mod11PATemplate.hs.

Submit your Haskell file in Canvas by the listed due date.