Middle East Technical University   Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2014-2015

Programming Assignment 2

Due date: 5 April 2015, Sunday, 23:55

1 Objectives

This homework aims to help you get familiar with functional programming and concept of haskell programming language.

2 Problem Definition

In this homework you are going to implement some functions to work with Huffman Codes. You will implement Huffman Trees, you will encode characters and you will decode encoded words.

2.1 Huffman Coding

Huffman Coding is a type of data compression technique. When you have a long string of characters for example it is wiser to compress them and let this string occupy less space. This algorithm finds an optimal prefix code for each of the characters. Prefix code is a type of code system which requires that there is no code word in the system that is a prefix (initial segment) of any other code word in the system. This property helps a lot while decoding the given encoded segments. To encode and decode we need a Huffman Tree to work on. Huffman Tree is basically a binary tree which is built from the characters in the string and the frequencies of each characters. We will provide the algorithm to construct a Huffman Tree. The data type you will use to hold Huffman Tree is provided below:

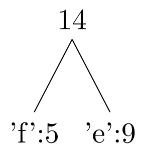
```haskell
data HuffmanTree = Empty | Intermediate Integer HuffmanTree HuffmanTree |
 Leaf Char Integer deriving (Show,Eq)
```

2.2 Huffman Tree

In this section we will describe the Huffman Tree with an example. Lets say you have a set of characters and their frequency of use and want to create a Huffman encoding for them:

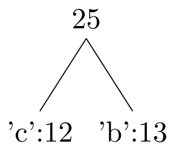
– a:45
– b:13
– c:12
– d:16
– e:9
– f:5

Creating a Huffman Tree is simple. Sort this list by frequency and make the two-lowest elements into leaves, creating a parent node with a frequency that is the sum of the two lower element's frequencies. The only thing you need to be careful is the left children should always have a smaller frequency:

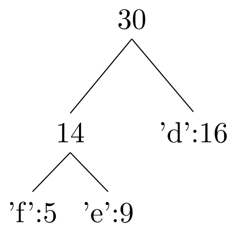


The two elements are removed from the list and the new parent node, with frequency 14, is inserted into the list by frequency. So now the list, sorted by frequency, is:
– c:12
– b:13
– Intermediate:14
– d:16
– a:45

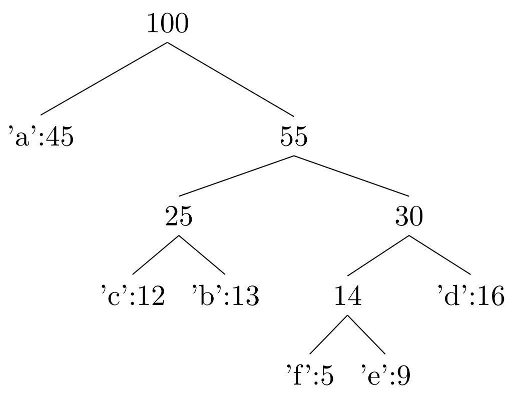
You then repeat the loop, combining the two lowest elements. This results in:



The two elements are removed from the list and the new parent node, with frequency 25, is inserted into the list by frequency. So now the list, sorted by frequency, is:
– Intermediate:14
– d:16
– Intermediate:25
– a:45
You then repeat the loop, combining the two lowest elements. This results in:



Continue the loop until there will be only one element in the list. When you reach there, you will have a Huffman Tree. For this example full Huffman Tree is like the following:



2.3 Methods

2.3.1 frequencies - 15 points

```
frequencies :: [Char] -> [HuffmanTree]
```

This function takes a character list. Actually this list is the list of characters which we will encode the characters. It will take the list and for each character it will find the frequencies. It will return a sorted list of leafs which hold the individual characters and their frequencies. If the frequencies of the both of the characters are same then the character which is the smallest will be smaller. In this part the list provided will have some specific features. For example the character will always be between 'a'and 'z'. They will always be consecutive which means that if 'd' exists in this char list 'a','b','c' are guaranteed to exist in the list. There won't be cases that includes 'a', 'b', 'd' but not 'c'. An example usage is provided below:

```
> frequencies "abacccddcdd"
> [Leaf 'b' 1,Leaf 'a' 2,Leaf 'c' 4,Leaf 'd' 4]
```

2.3.2 buildTree - 25 points

```
buildTree :: [HuffmanTree] -> HuffmanTree
```

This function should take an HuffmanTree list (actually the one you have created with the frequencies function). And should create a HuffmanTree with the algorithm that is provided above. An example usage is provided below:

```
> buildTree [Leaf 'f' 5,Leaf 'e' 9,Leaf 'c' 12,Leaf 'b' 13,Leaf 'd' 16,
Leaf 'a' 45]

>Intermediate 100 (Leaf 'a' 45) (Intermediate 55 (Intermediate 25 (Leaf 'c' 12)
(Leaf 'b' 13)) (Intermediate 30 (Intermediate 14 (Leaf 'f' 5) (Leaf 'e' 9))
(Leaf 'd' 16)))
```

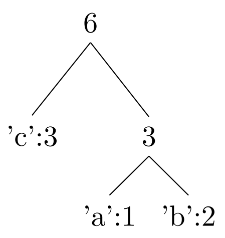
2.3.3 encode - 30 points

```
encode :: HuffmanTree -> [Char] -> [Char]
```

After you generated HuffmanTree you can encode your strings now. Now the list is just one element containing the total frequencies of the string and all the other frequencies are placed in the binary tree. To generate a huffman code you traverse the tree to the value you want starting from the root, outputing a 0 every time you take a lefthand branch, and a 1 every time you take a righthand branch.

For the example provided in the HuffmanTree part the 'a' will be encoded as 0, 'b' is 101, 'c' is 100, 'd' is 111, 'e' is 1101 and 'f' is 1100. To decode you should change each of the characters with its binary encoding.

This function takes two inputs. One is an already built HuffmanTree the second is the string to be encoded. For this function we will provide a smaller set to understand better. The input will be "abbccc". The Huffman Tree of this example would be like the following:



So the value of 'a' should be encoded as '10' , value of 'b' should be encoded as '11' and value of 'c' should be encoded as '0'. So "abbccc" should be encoded as "101111000" .

An example explaining this is provided below:

```
> encode (Intermediate 6 (Leaf 'c' 3) (Intermediate 3 (Leaf 'a' 1) (Leaf 'b' 2)))
"abbccc"

> "101111000"
```

2.3.4 decode - 30 points

```
decode :: HuffmanTree -> [Char] -> [Char]
```

In this function you will have an already built HuffmanTree and a encoded char array. You are expected to return the decoded char array.

An example usage is provided below:

```
> decode (Intermediate 6 (Leaf 'c' 3)(Intermediate 3 (Leaf 'a' 1) (Leaf 'b' 2)) )
"101111000"

> "abbccc"
```

3 Specification Additions

1. If an Intermediate node and Leaf node has the same frequency Leaf node should be at the left node.
2. If only one character is given. You won't go left or right so this case is out of algorithm. Because of that just make it 0.
3. For build tree if two intermediate node comes and they have same frequencies you can make an arbitrary choice. For build tree this case occurs and for this algorithm there are no choice optinions. In test cases both of the solutions will take points.

4 Regulations

1. **Programming Language:** You must code your program in Haskell. Your submission will be tested with hugs interpreter on moodle. You are expected make sure your code loads successfully in hugs interpreter using the run command on the moodle system.
2. **Late Submission:** Late submission is allowed and a penalty of 5 * day * day is applied to the final grade.
3. **Cheating:** In case of cheating, the university regulations will be applied.
4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

5 Submission

Submission will be done via moodle system. You can either download the template file, make necessary additions and upload the file to the system or edit using the editor on the moodle and save your changes.