Instituto Superior Técnico – Universidade de Lisboa

2016 / 2017

Binary preﬁx codes and Huffman’s codes

Network Algorithms and Performance

Group nº 13:

* Dominik Gizynski, 85736
* João Caetano, 75218

# Algorithms Explanation and Pseudo-codes

## Procedure GenerateCode(Root, Code)

This procedure receives as input the root to a binary tree, an initially empty string *code and* an also initially empty dictionary (Python type of data structure) *symbols.* It outputs the dictionary where each symbol has a binary code associated.

The function works recursively and is based on the PreOrder traversal print of a binary tree. In this case it saves, instead of printing, only the leaves (symbols). Each time the functions calls itself it appends a 0 or 1 to the string *code* whether it goes to the left or right child of the current node. Every time a leaf is obtained the string *code* is saved to the dictionary of the respective symbol.

*GenerateCode(root, code, symbols)*

*if root.symb != NULL*

*symbols[root.symb] = code*

*else*

*GenerateCode(root.l, code + '0',symbols)*

*GenerateCode(root.r, code + '1',symbols)*

*endif*

*return symbols*

## Procedure Decode(Root, InString, OutString)

This procedure receives as input the root to a binary tree, a string with a binary code to be decoded *InString* and an empty string *OutString.*

While the string *Instring* still has characters to be decoded another function RunTree is called. This function runs the tree until a leaf is found, if the bit is 0 it moves to the left child else it moves to the right. For each bit read it discards the first element of *InString.* When a leaf is obtained the corresponding symbol is saved and outputted. This process seems silly for using two functions but every time the function RunTree returns, the root node returns to his initial state without the need of auxiliary pointers.

## Program PrefixCode

This program reads a set of symbols from a file and saves them as Node structures in a list. Then a random tree where the leaves are the read symbols is created and its root returned. This is done in a function called RandTree. The procedure GenerateCode(Root, Code) is then called which outputs the dictionary containing the binary codes for each symbol. Each symbol and corresponding binary code is then printed to the screen and the program waits for the user to write a binary string to be decoded. This string is inputted in the procedure Decode(Root, InString, OutString) which prints the decoded symbols.

## Procedure HuffmanCode(Symbols, Freq, Code)

This procedure inputs two vectors, one of symbols and one of frequencies. It makes use of a Binary Heap class previously created for better efficiency.

First we create our tree as a Heap and then we insert the symbols as nodes in the Heap. While the Heap has more than a node we choose the two ones with smaller frequency and create a new node that will parent them with his frequency equal to the sum of the children, this node is then inserted in the Heap. The last element present in the Heap will be the root of the newly created Huffman Tree.

The root of the tree will then be inputted in the procedure GenerateCode(Root, Code) which will output the dictionary with the binary codes for each symbol. The symbols with the corresponding code will then be printed for the user.

*HuffmanCode(symbols, freqs)*

*HuffTree := BinHeap()*

*i := 0*

*for symb in symbols*

*HuffTree.insert(Node(freqs[i], symb))*

*i := i +1*

*endfor*

*while HuffTree.currentSize > 1*

*small1 := HuffTree.delMin()*

*small2 := HuffTree.delMin()*

*HuffTree.insert(Node(small1.freq + small2.freq, NULL, small1,small2)*

*endwhile*

*root := HuffTree.heapList[1]*

*code :=GenerateCode(root)*

*return Code // In this case I print the code inside the function*

# Discussion