Jiahong Chen, jc4tf, Apr 21, postlab10.pdf

Description of Implementation

In my implementation, I use a heap to store hmnode. The heap is in fact a vector of hmnode pointer. After reading the file to be encoded, I calculate the frequency of each character using a vector of that contains 128 number 0. Each hmnode contains the int frequency that I store in the array and char element, which is the exact letter. Then I created a tree by using the hmnodes in my heap. Every hmtree has at least a root. And the root is unsurprisingly again a hmnode pointer. Everytime I call the method to delete the minimum element in the heap and that create a new node pointer to connect the two deleted nodes. The frequency of the newly created node pointer is the sum of its two children. They push it back to the heap until there is only one node pointer in the heap.

For the pre-lab, which is to encode the messages, after inserting all the required nodes to the heap, I call the buildTree method in the hmtree class. For this encoding part, I also have two other methods. The first one is call opcode. It takes in a node pointer and the string prefix. If the pointer has already reached the left node, then cout the element. Otherwise, if the pointer-> left is not NULL, recursively go to the left pointer and add "0" to the prifix string. And we do the same if the pointer->right is not NULL. But doing so, we traverse the tree for every character and generate the encoded messages. The other method to print out the all the encoded bits is call getCode. It is very similar to the previous one except that we need to pass in a character to see which code to use.

I didn't use recursive method for the inlab part. Instead, I use two for loops in the tree class. I build the tree using by calling makeTree. This method is very simple. For every bits in the prefix, if it hits a 0, created a new node and insert it to the left of the current pointer. We do not need to do the same if that spot is not NULL. And if the bits we get if 1, we do the same to the right. The printCode method takes in the allbits string. It prints the node if it reaches the leaf node and gp back to the root node. When I say node here, I mean hmnode pointer.

Efficiency Analysis

For encoding, we read the file first to get the element(1 byte) and the frequency(4 bytes), create hmnode pointer(itself 8 bytes; other than element and frequency, left/right pointers also 8bytes; total 13 bytes for a hmnode), and insert it to the heap(it's in fact a vector of size 100*8 bytes). All these steps take Big Theta(n) time. The vector that we use to store the frequency of the elements has 128 elements, and so 128*4 bytes. Building the tree is again constant time because the deteleMin and push back methods both take only Big Theta(1) time. But we need to do this

to all the node pointers so it becomes linear time. The space for the tree depends on how many hmnode pointers it holds. So the worst case running time is n^2.

For decoding, we need to again read the file and make the Tree, which takes linear time. So the worst case running time is n. The data structure we use here is the same as those for the encoding part.