Rakshith Raghu, rr5de, huffman code pdf

Encoder:

I used a min heap(very similar to the one provided by the lecture) but modified for treenode structures. Each treenode is 2 values, an int for frequency and a char for name, along with pointers to two children. Two vectors were used to collect names and frequencies. Worst case time complexity for this collection is O(n) iterations where n is the number of chars. Worst case space complexity is O(2n*8) for the two vectors.

When inserted(they are inserted as vectors), a for loop is used to create a new treenode and to put it into a vector. Worst case time complexity is again O(n_0) where n_0 is the number of unique chars. I then run another for loop to percolate down. This for loop has a worst case runtime of O(n_n*log(n_n)) where n_n is the number of elements of the vector. This assumes that each element has to be moved through the whole min heap.

After the minheap is completed, I consolidate the nodes into one tree. This is done in a while loop that involves at least n_n-1 iterations. Each consolidation involves the create of 3 nodes, the deleting of 2 other nodes, and the insertion of the new tree into the minheap. Space complexity added will, in the worst case be n_n*32 (8 for int, 8 for char, 16 for pointers). Each insertion also has to percolate up the list as much as possible, which would take log(n_n) for each iteration. Thus total time complexity in worst case would be O(n_n*log(n)).

After this step, I create a map with keys and values. Each key is given by a leaf node with the value being given by the route to that leaf node. This travel will take log(n_t) where n_t is the number of nodes in the tree. Given that this is done for every leaf, this will take, in time complexity: O(log(n_t) leaves). In terms of space complexity a new data structure with a key and value are added to the memory. This would be 2*leaves*8 bytes in the worst case.

Using this map structure, I print this map and use two doubles (32 bytes total) to get the cost. Looping through to print the code would take O(n_0) where n_0 is the number of unique chars. I would also be including 4 ints for the final printer, which would require 4*8 space.

Decoder:

The methodology here is generally simpler. I use two strings to read in the char and the code itself. Each string would take 8 bits for space complexity. Time complexity is locked at the number of chars in the code. Each char is then inserted into the huffman tree. Each insertion involves at least one created treenode, which involves the char and pointers to two children that are NULL. This is 8 + 16 in terms of space complexity. In terms of time complexity, O(n*route). Here, n is the number of chars in the code, and the route is the length of each code. Since route is variable in size, I cannot directly say its max, though I can place a harsh limit of o(n^2). Now the number of nodes is also difficult to calculate. At the very minimum, it would require n nodes for number of leaves. Max number of nodes would be 2n + 1. In terms of memory, that would be (8 + 16) * 2n+ 1 where n is the number of unique chars.

After creating the huffman tree, it then reads into the bits themselves to be decoded and then printed. This is a while loop that can be infinite in size depending on the amount of characters in encoded txt. A string is used to get a set of bits to be decoded. That is 8 bits reused through the whole while loop. Getting decoded char itself would require going through the tree using the route given by the code. I use a recursive structure here, where I reiterate on the children of the node depending on which direction is given by the string. This would at most be O(d) where d is the depth of the tree.