John Zheng

jz5pt

4/18/19

postlab10.pdf

For the prelab, I followed the steps outlined in the lecture slides. First, I created a frequency table of characters. For the frequency table, I used the map data structure. It uses a key and value pair to store information. In this case, the key is the character and the value is the count of that character. I also initiated a vector of char named ‘chars’ to print the code prefixes in the print section. In the while loop, while the file pointer isn’t at the end of the file, I push the character to the back of the vector and check if the character is in the map. If the character isn’t in the map, I insert the character with the count initiated to 1. If the character is in the map, I simply increment the count by 1 of that character. Next step outlined in the lecture slide is to build a min-heap sorted by frequency. I implemented the heap using Huffman nodes. The nodes have the members key, count, left, and right. The key is the character the node is storing, the count keeps track of the frequency of that character, and the two children nodes, left and right. To iterate through the map, I created a map iterator and implemented a for-loop. Inside the for-loop, I initiated an int variable named ascii which holds the ascii code of the character. The point of this variable is to check if the character is a printable character. If it is, I then create a new Huffman node and inserted it into the heap. Next step is building the Huffman tree. In the while loop, as long as the heap size is greater than one, I delete the min from the heap and store it in min1 and the next min and store it in min2. I initiate a new Huffman node and set that node’s children node to min1 and min2 and insert it to the heap. Next step is creating the prefix table.

First I delete the min from the heap and store it in a Huffman node pointer, root. Then I create another map of char and string, prefix. Finally, I call makePrefix. makePrefix is a function that takes in a root parameter, empty string, and the address of the prefix map. It is a recursive function that checks if the children of the current node is null, if it is then it inserts the character of the node and the string into the map. Otherwise, it changes the root to either the left child and concatenates a 0 to the string or the right child and concatenates a 1 to the string.

For the inlab, to build the Huffman tree I initiated a Huffman node pointer, root, a map iterator, and declared a Huffman node, cur. The first for-loop iterates through the map of prefixes. The second for-loop iterates through the second to last character of the prefix code of the current character. If the prefix is 0 and the left child is null, it creates a new node at cur's left. It then sets cur to cur's left. It does the same thing if the prefix is 1, but reverse. If the last character is 0, a new Huffman node is inserted to the cur's left and vice versa. To decode the message, I initiated an empty string. The for-loop iterates through the allbits. If the character is 0, go left, vice versa. If the children nodes are null, concatenate the character to the decode string.

Time complexity of building the frequency table is n because it has to iterate through n character of the file. To build the min-heap, as well as the prefix table, it is constant because there are only so many characters available. Time complexity to decode n because it depends on how many bits are in allbits which depends on the file size. Space complexity for the prelab is constant. There are only so many character to insert into the frequency table.