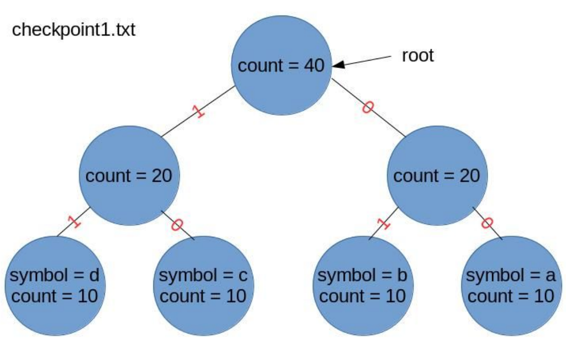
/*
* Name: Megan Chu
* PID: A12814536
* Date: May 10, 2017
* CSE 100 PA3 C++ Huffman
*/

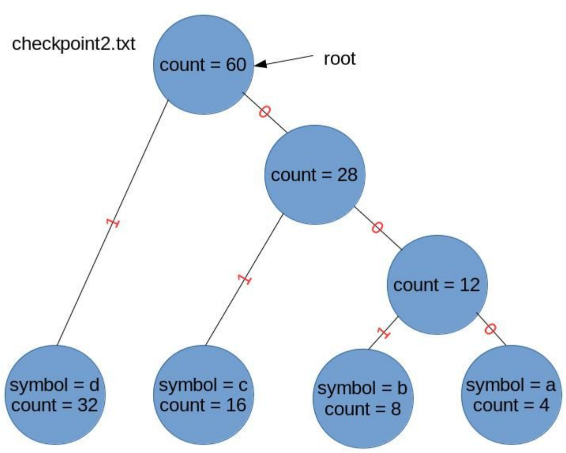
Since the checkpoint submission I have fixed the error in my build method for HCTree.cpp, and completed the previously incomplete uncompress.cpp and this Checkpoint.pdf.

2.
Encoded output of checkpoint1.txt without header
0001101100011011000110110001101100011011000110110001101100011011000110110001
1011

Encoded output of checkpoint2.txt without header
0000010101011011011011011011011011011101101101101101101101101101101101101101
1011011011011011011011011011011011011011011011011011011010110110110110110110
1101010101010000

3.





How I constructed checkpoint1.txt Huffman coding tree. I basically followed what I did in my code to draw the tree. First I created a leaf node for every byte(char) that had frequency > 0 in the input file. Then I ordered these nodes for each byte firstly by their "counts/frequency" and secondly by ASCII key values, thirdly by single node/subtree, and lastly by time of addition to the list. In this example, I firstly got a list in the order a(10), b(10), c(10), d(10). Then I took the first node in the list and set it as the c0 child of a temporary root node (and set the parent to the temporary root), I took the second node in the list and set it as the c1 child of the same temporary root (and set the parent to the temporary root). I then set the temporary root's count to that of the first node's count + the second node's count, removed the first and second nodes from the list, and added the temporary root to the list in the correct order. I ended up with the following list: c(10), d(10), root1(20). I repeated the part in purple and ended up with the following list: root1(20), root2(20). I repeated the part in purple again and got: root3(40). Now that there is only one node in the list, I set the root pointer in the HCTree class to point to that one node and got the resulting Huffman coding tree seen in the picture.

How I constructed checkpoint2.txt Huffman coding tree. Again I followed what I did in my code to draw the tree. First creating a leaf node for every byte with frequency > 0 in the input file. I ordered the nodes by their counts, then ASCII key values, then single node/subtree, and lastly by time of addition to the list. In the checkpoint2.txt example, I firstly got a list in the order a(4), b(8), c(16), d(32). Following the steps in purple from above, I then got: root1(12), c(16), d(32). Following the steps in purple again, I got: root2(28), d(32). Following the steps in purple once more, I got: root3(60). Now that there is one node left in the list, I set the root pointer in the HCTree class to point to this one node, and got the resulting Huffman coding tree seen in the picture above.

To find the "codeword for each byte", I started at the leaf node for that byte, and followed the path up towards the root, pushing the bits for each path onto a stack as I went. Upon reaching the root, I popped all the bits from the stack, resulting in the list of bits telling us which path to take in the order from root node to leaf node for that byte. Let us use the node for byte 'b' in checkpoint1.txt as an example, and pushing/popping from the left(front) of the stack. Starting at the 'b' node, we follow path 1 to reach parent 'count = 20', so we push 1 onto our stack: 1. From 'count = 20' node, we follow path 0 to 'count = 40', so we push 0 onto our stack: 01. Since 'count = 40' node is the root, we pop all the bits from the stack: 01. We first pop 0, the pop 1, so the codeword is '01'. This means from the root, we will need to first traverse the 0 path (c0 path) to the next node. Then from the next node, we will need to traverse the 1 path (c1 path) to the leaf node. This turns out to be the 'b' node, so our algorithm is correct.

4.

I manually encoded the strings from checkpoint1.txt and checkpoint2.txt manually using the method above and this matched the compressor output listed below. I never encounted a situation of a mismatch because I made sure to seriously read through and debug compressor.cpp such that it got results that I assumed were correct before completing this writeup.

checkpoint1.txt

0001101100011011000110110001101100011011000110110001101100011011000110110001
1011

checkpoint2.txt
0000010101011011011011011011011011011011101101101101101101101101101101101101
1011011011011011011011011011011011011011011011011011011011011011011011011011
1101010101010000