Peter Phan
CSE100wdh
A13042904

Checkpoint

How the Huffman trees were built:

By hand I pushed the nodes into a priority queue and retained all the properties where the node with the smallest freq goes at the top. At every step, I would pop the two smallest nodes. Then, I would create a new node that has the summ of the freq and the symbol of the first node popped. The first node would be the “0” child while the second node popped would be the “1” child of the new node. Then, I would push that new node back into priority queue. I did this until there was only one node left and that the the root of the tree.

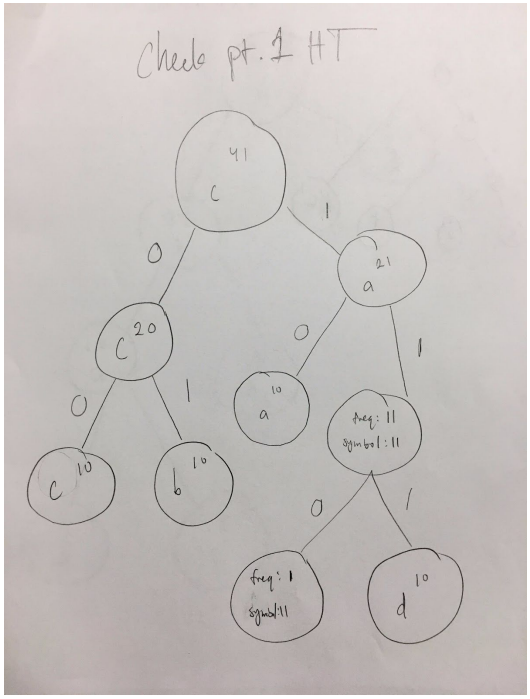
How to find the code for each byte:

By eye, I would start at the root and trace the shortest path to the node with the same byte. Every time I jump from one node to another I would record the bit that encodes that link.

Checkpoint1.txt

Result:
0100100010000010010001000001001000100000100100010000010010001000001001000100
0001001000100000100100010000010010001000001001000100001

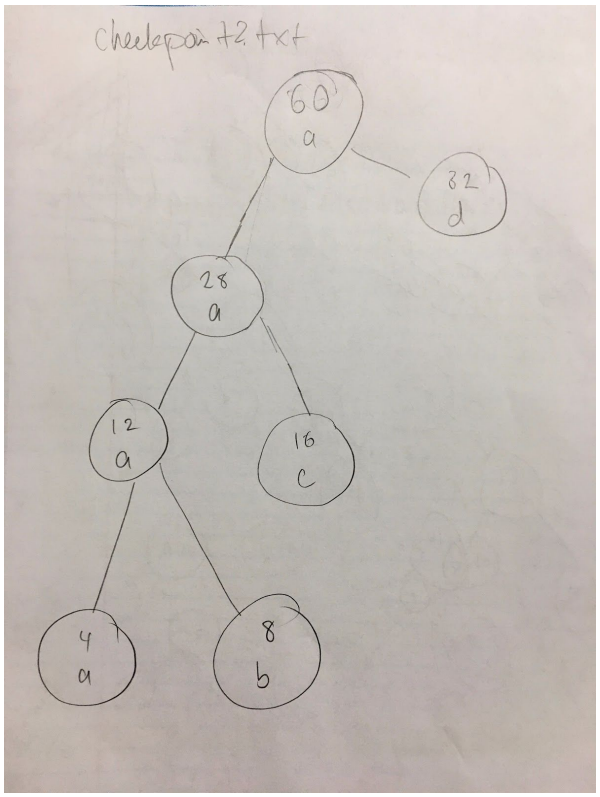
Tree:



My Results:
1001001111001001111001001111001001111001001111001001111001001111001001111001
00111100100111110

Checkpoint2.txt
Result:1101010101001001001001001001001001000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000100100100100100
0100100101010101111

HT:



My result:
0000000010010010010101010101010101111111111111111111111111111111111110101010101
0101010010010010010000000

Comment: Both of these are wrong because the huffman tree comes from a priority queue where the largest is first rather than the smallest is first. So, I changed my code to return the smallest freq first. I learned how to initialize priority queue such that it will use my comparisons when inserting things.