Checkpoint

command: ./compress checkpoint1.txt and compressed1.txt

header encode:

The table represents lines form 1-256 that contain numbers other than 0 in file compressed2.txt

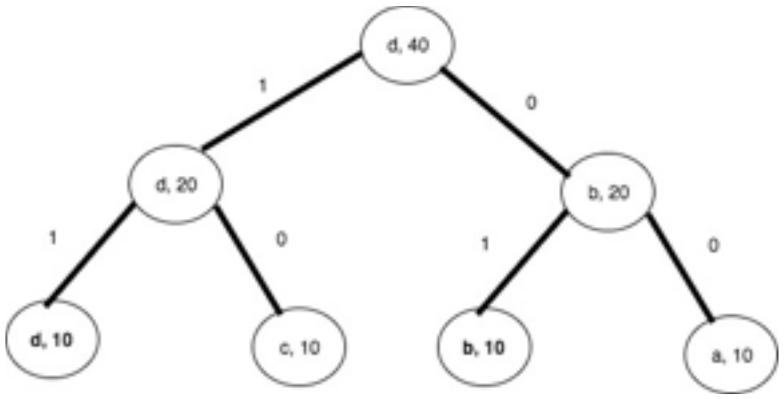
Line	Content
98	10
99	10
100	10
101	10

endfile encode:

000110110001101100011011000110110001101100011011000110110001101100011011 00011011

Manual Construction of Huffman coding tree

checkpoint1.txt input : abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd



First I start with going through the input to find a set of letters in the input to find how many unique nodes I would have to make. Then I counted the frequency for each unique letter. Next, I created a single node consisting its symbol and counts for each unique letter. Usually the node with lower frequency would be on the left of the parent node. However, all nodes have the same frequency so I break tie by examining the ASCI value of the node's symbol. The higher ASCI value symbol will be prioritized. To find the code for each byte, I follow down the path from root to the symbol that the byte represents. Each time I travel from one node to another, I record the binary code for the path.

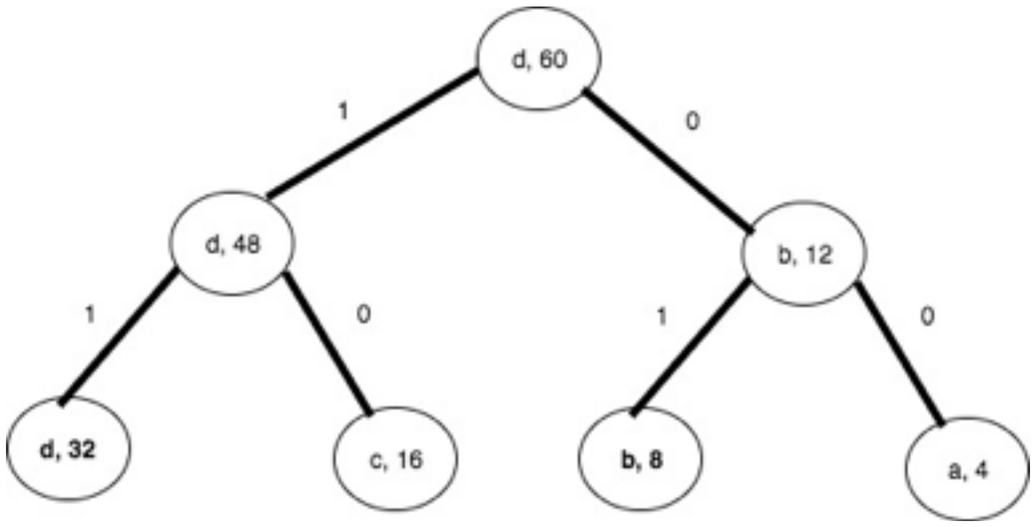
Manual encoding results: 00011011 00011011 00011011 00011011 00011011 00011011 00011011 00011011 00011011 00011011

The results of my manual encode aligns with the compressor in my program

command: ./compress checkpoint2.txt and compressed2.txt
header encode:
The table represents lines form 1-256 that contain numbers other than 0 in file compressed2.txt

Line	Content
98	4
99	8
100	6
101	12

endfile encode:
11111111011011011010101010101010100000000000000000000000000000000010101
0101010101011011011011011 1111



Manual Construction of Huffman coding tree
checkpoint2.txt input : aabbbbccccccccddddddddddddddddddddddccccccccbbbbaa

First I start with going through the input to find a set of letters in the input to find how many unique nodes I would have to make. Then I counted the frequency for each unique letter. Next, I created a single node consisting its symbol and counts for each unique letter. I out them in order of decreasing frequency from left to right. Select the first two lowest frequency, which was b and a. The create proper path to those nodes. To find the code for each byte, I follow down the path from root to the symbol that the byte represents. Each time I travel from one node to another, I record the binary code for the path