Steps 4:

./compress check1.txt compress1.txt

line 10 1

line 97 10

line 98 10

line 99 10

line 100 10

Encoded string:

1110001101110001101110001101110001101110001101110001101110001101110000110111000

110111000110110

./compress check2.txt compress2.txt

line 10 1

line 97 4

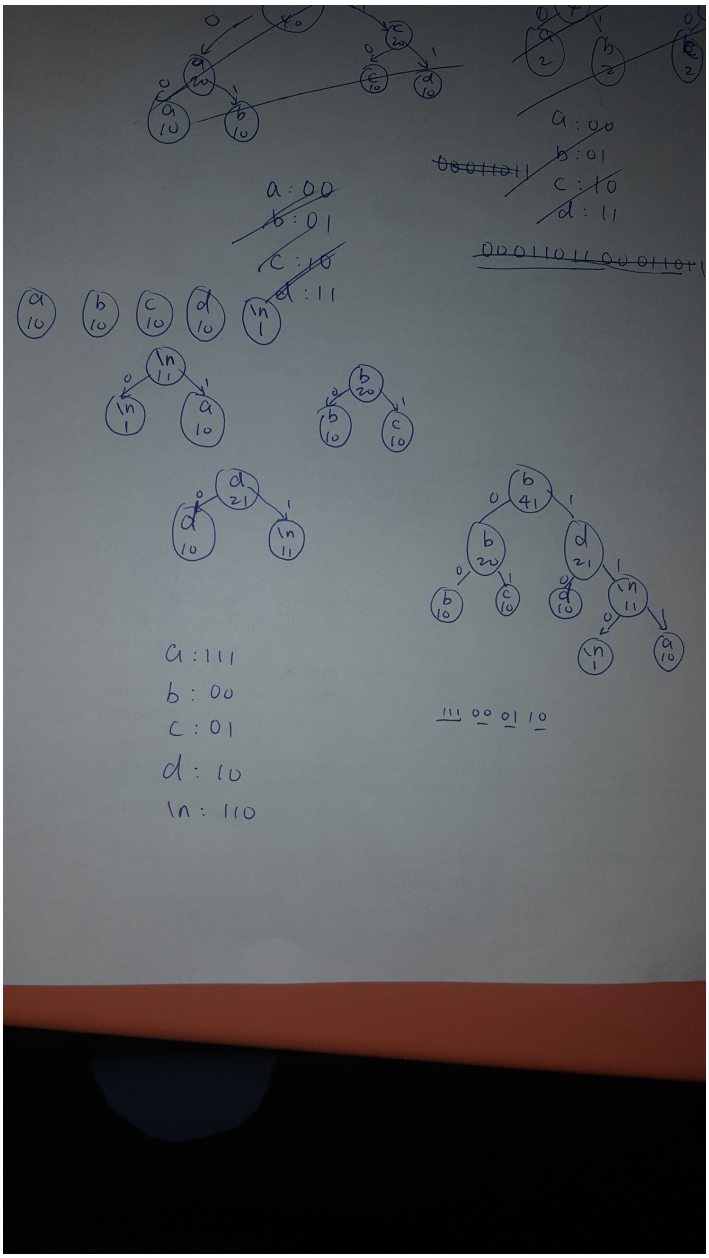
line 98 8

line 99 16

line 100 32

Encoded string:

000100010010010010010101010101010101111111111111111111111111111111110101010101
010101001001001001000100010000



First, I count the frequency of each character in the message, then I make them to be HCNode * which are stored in the leaves vector. Then I take out the two least-frequency nodes (if the frequencies are equal, compare the character), then I make a new node with frequency being the sum of the previous two, and symbol be the symbol of smaller one of previous two nodes, then connect the new node to the least node with 0 and the second least node with 1. Repeat the steps

until there is only one node left. Then I get the tree like the picture above. And by traversing the tree from the root down to the leaves, I get the code for each character. For example, ‘a’ is 111, ‘b’ is 00 etc. And the result for my manually constructed tree matches the result for my compressor output.

Step 6.

Rather than writing the frequencies of all 256 characters line by line, I choose to write the structure of the tree to the file as the write-up suggests. In the beginning of the file, I use one byte to store the total number of unique characters in the file, and use another 4 bytes to store the total number of the characters in the file (for decoding). Then starting from the root, using the pre-order traversal, if the node is not a leaf, I write bit 1 to the file, if it’s a leaf, I write bit 0 to the file. And immediately after 0, I use 8 bit to represent the character. This way, I can get all the necessary information needed to reconstruct the tree when decoding while not wasting memory for the information I don’t need. When decoding, I first read the first byte as the number of unique characters, and next four bytes as total number of characters. Then if I encounter a 0, I know it’s a leaf node, and next 8 bits is the symbol. So I can create a new leaf node from it. If I encounter a 1, then I know it’s not leaf, and at this point, I am able to create another new node with c0 and/or c1. Keeping doing this until I reach the number of unique character count. At this point, it’s the end of the header and I have reconstructed the tree. The rest of the file are encoded message. It turns out that it beats reference solution. For example, warandpeace.txt, refcompress get a compressed file with size 1871653 bytes while my compressor get a file with size 1870740 bytes.