Lucas Blumhardt A12020745

The output to ./compress checkpoint1.txt output1.txt is

```
0
...
0
10
10
10
10
0
...
0
0
0
11100100111001001110010011100100111001001110010011100100111001001110010011100100
0100
```

The first 10 appears on line 98 (signifies ASCII char ‘a’)

The output to ./compress checkpoint2.txt output2.txt is

```
0
…
0
4
8
16
32
0
…
0
000000001001001001010101010101010101111111111111111111111111111111111110101010101
010101001001001001000000
```

The 4 appears on line 98 (signifies ASCII char ‘a’).

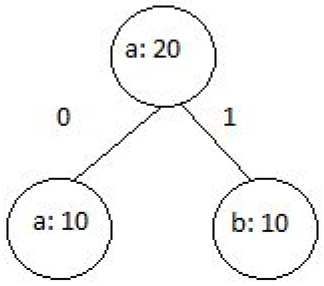
Manually encoding:
Checkpoint1.txt: abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd

Frequencies:
‘a’: 10
‘b’: 10
‘c’: 10

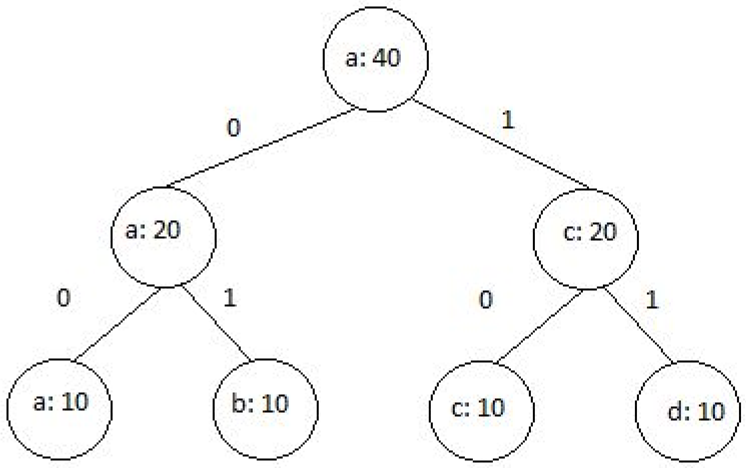
‘d’: 10

Four nodes would be constructed storing the symbols and their frequencies.
(a: 10), (b: 10), (c: 10), (d: 10). These nodes are pushed into a min heap. Since their frequencies are equal, their symbols are compared. a would be the min, followed by b, c, and then d.

The two smallest nodes (a and b) are popped from the min heap and combined by a parent node. This parent node takes the lesser symbol, and combines the frequencies/counts of the two children nodes. The smallest node takes the position as 0 child, the larger one takes the 1 child. The first iteration would look like this



The parent node is then pushed back into the min heap and sorted. We repeat this process until only 1 node is left on the heap. At that point the tree will look like this

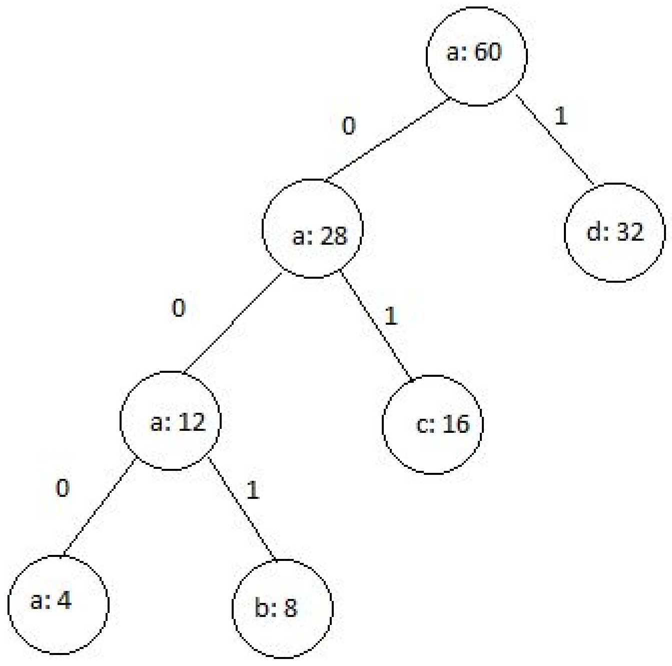


The encodings for each char would be
a: 00
b: 01
c: 10
d: 11

checkpoint1.txt would be encoded to:
0001101100011011000110110001101100011011000110110001101100011011000110110001
1011

checkpoint2.txt: aabbbbccccccccdddddddddddddddddddddddddddddddccccccccbbbbaa
Frequencies:
‘a’: 4
‘b’: 8
‘c’: 16
‘d’: 32

We create four nodes and follow the above algorithm, pulling the two smallest at a time. The huffman tree will look like this



Char encodings would be as follows:
a: 000
b: 001
c: 01
d: 1

Checkpoint2.txt could be encoded as:
000000001001001001010101010101010111111111111111111111111111111111111010101010
101011100100100100100000 0

Manually encoding checkpoint1.txt came out with a different encoding scheme than the algorithm. I wish I could explain why, but let me tell you why I’m confused. My comparator method compares the two node’s symbols if their counts are the same. It returns this->symbol < other.symbol. That’s the same thing I used to do this by hand. Second, my build method sets the smaller node to the 0 child, and the larger node to the 1 child. The parent node adopts the symbol of the smaller node. Those are the same properties I followed when doing this by hand. I doubt you would be able to explain this to me without seeing my code, but I’d greatly appreciate it if you have any ideas as to why this may be.

Checkpoint2.txt is encoded the same way both by hand and by the program. This makes sense because I followed the same properties as the code.