CSE 100 PA3 Report

Robin Heinonen

November 11, 2017

The compressor's output for check1.txt was as follows:

```
line 98 10
line 99 10
line 100 10
line 101 10
Encoded string: 11100100111001001110010011100100111001001110010011100100111001001110010011100100
```

The compressor's output for check2.txt was as follows:

```
line 98 4
line 99 8
line 100 16
line 101 32
Encoded string: 11111111011011011010101010101010100000000000000000000000000000000010101010101010101101101101101111111
```

We construct the following Huffman coding tree from the check1.txt header:

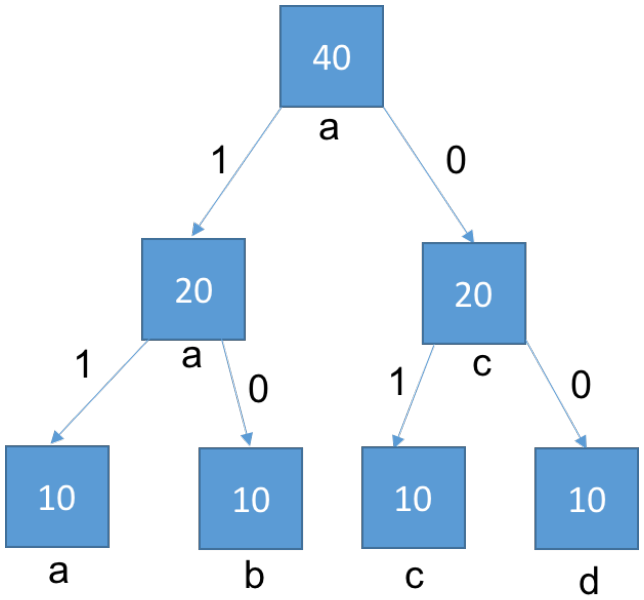

Figure 1: Huffman tree for check1.txt

In this figure, each node is a box with its count in the center. The symbol corresponding to the node is displayed below it, though this symbol only really has meaning for the leaves;

for non-leaves, the symbol is just inherited from the 1-child, and is used to break ties. For this particular tree, the tiebreaking procedure is important, since all the symbols have equal frequency.

The Huffman coding tree for check2.txt is as follows:

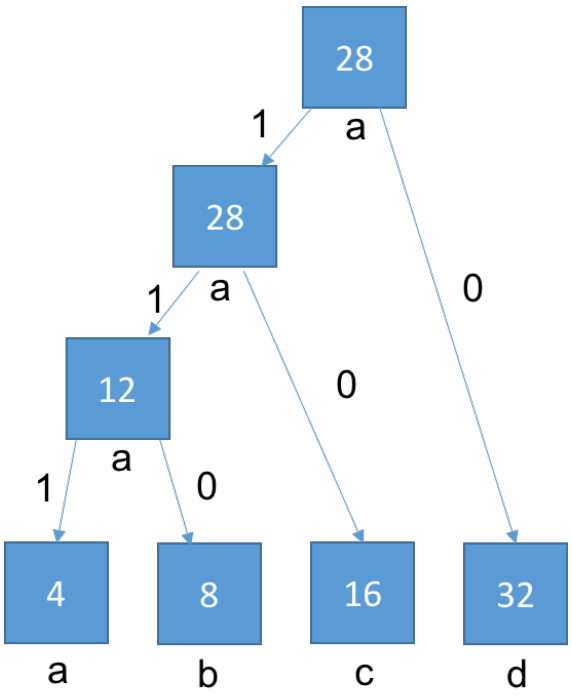

Figure 2: Huffman tree for check2.txt

The trees were constructed by creating a leaf with the corresponding frequency for each symbol appearing in the message, and then while there are still more than 1 node without parents, the two nodes with the smallest frequency are repeatedly selected, and a new node with the sum of the two frequencies is created as its parent. The parent's less frequent child is the 1-child, and the more frequent child is the 0-child; ties are broken by choosing the smaller (lexicographically speaking) symbol to be the 1 child. The parent inherits the 1-child's symbol.

This procedure allows us to assign codes to each symbol by simply following the Huffman tree from the root to the symbol's leaf, concatenating the bits that are encountered. For example, in the first tree, a=11, b=10, c=01, and d=00, whereas in the second tree, a=111, b=110, c=10, and d=0.

check1.txt's message "abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd" thus becomes "11100100" repeated 10 times, in agreement with the output of the compressor. check2.txt's message "aabbbbccccccccddddddddddddddddddddddddddddddddddddcccccccbbbbaa" becomes "111111 1101101101101010101010101010000000000000000000000000000000001010101010101010110110110110 111111," also in agreement with the output of the compressor.

I made my header more efficient by encoding the basic Huffman tree structure rather than the frequencies. Specifically, I traverse the tree via preorder, outputting a '0' for each nonleaf node and a '1,' followed by the corresponding character, for each leaf node. The tree

structure is terminated by a newline character (this isn't ambiguous because the last node traversed must be a leaf node; any other newline character must be preceded by 0). I then write the number of meaningful bits in the last coded byte, which I compute in advance using the character frequencies (this information is necessary to parse the final byte, for the total number of bits need not be a multiple of 8).

The preorder traversal is enough information to uniquely construct the Huffman tree; the algorithm to do so simply involves reading in the characters one by one and constructing the tree from the top, descending to the leaves and reascending as necessary when a branch is filled.

Each character was written as a 1 byte unsigned char; the Huffman coding tree for a message with n distinct characters has n leaf nodes and $n-1$ non-leaf nodes, so this header requires $16n + 8(n-1) + 16 = 24n + 8$ bits. This could be improved upon by writing the header bitwise, but who has the time?