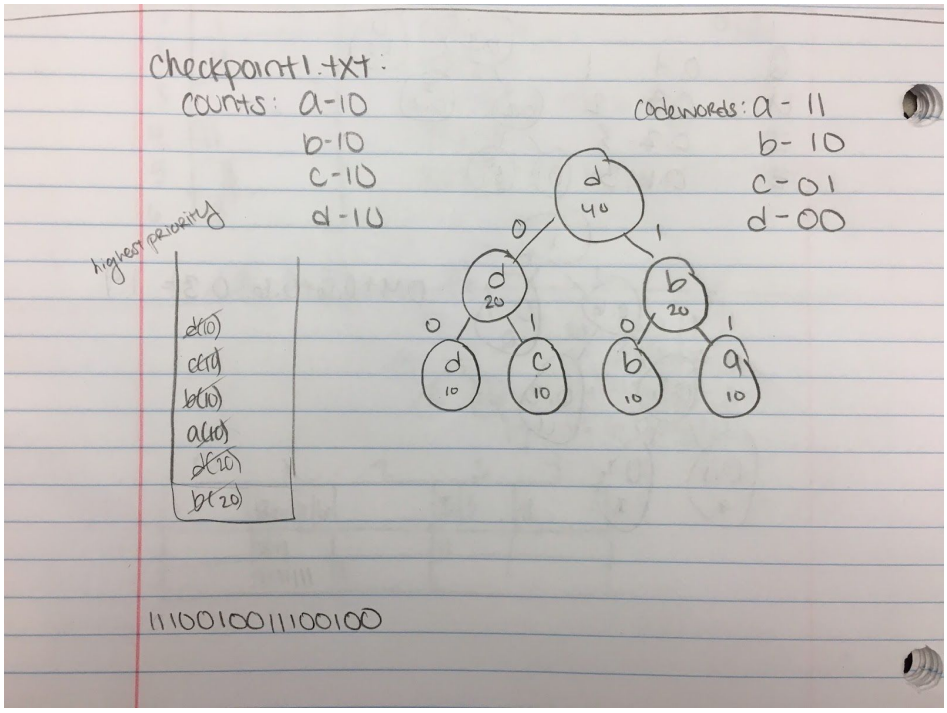
Authors: Allison Reiss, Sarah Gemperle
Date: 5/10/2017

Checkpoint

Encoded output from compressing checkpoint1.txt:
11100100111001001110010011100100111001001110010011100100111001001110010011100100

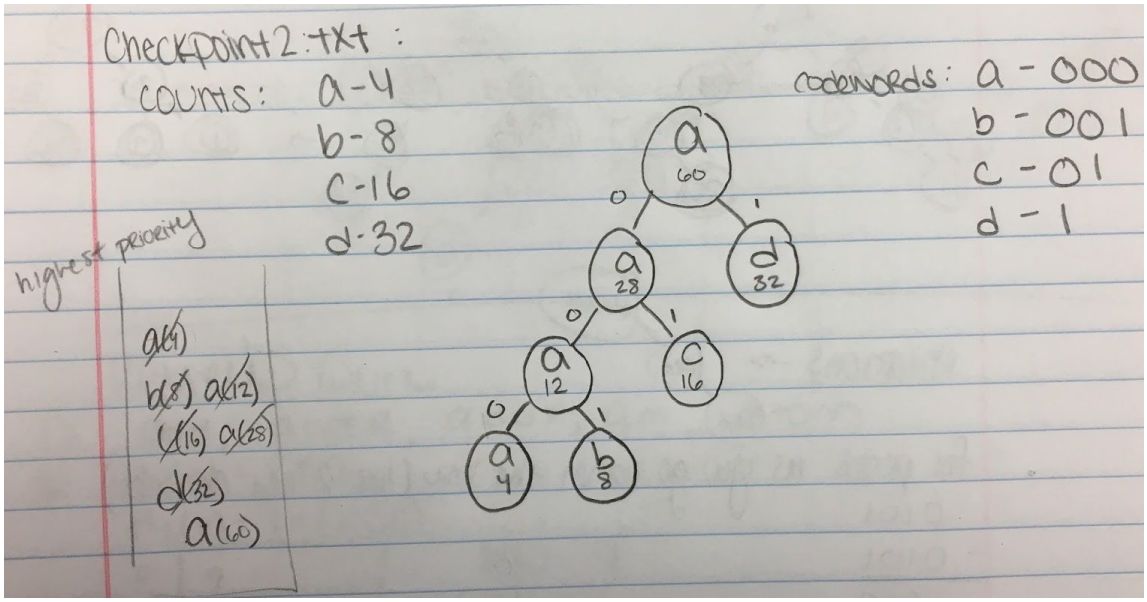


To build the tree, we first counted how many times each character appeared in checkpoint1.txt. Since each character appeared 10 times, we then deferred to the symbol to see which nodes had a higher priority. Since our algorithm gives symbols with larger ASCII values higher priority (the equivalent to having a lower count). Once we had these priorities, we took the symbols with the 2 highest priorities (d and c), and made them into nodes with their respective counts. We then created a new root node with d as its left child and c as its right child, with a count equivalent to the sum of d and c's counts (20). This was placed back in the priority queue and we then took the next 2 highest priority symbols, in this case b and a because they had the lowest counts. We made a new root node for these nodes with b as the left child and a as the right child, with a count of the sum of a and b's counts (20). This was then placed back in the queue. We then took the 2 remaining elements in the queue and created a new root node with d as the left child and b as the right child, with a count of the sum of d and b (40). After putting this node back in the priority queue, it is the only node so that is the final tree. Since the tree is in the form of a binary tree, we labeled each pointer to a left child with a 0, and pointers to the right child as 1. To get the codeword for a given byte, you simply follow the path down to that leaf and concatenate the corresponding 0 or 1 on the labeled path taken between each node.

To encode the input (abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd), we simply replaced each letter with its respective codeword to get the following:
11100100111001001110010011100100111001001110010011100100111001001110010011100100

This is equivalent to our compressor output, so we never encountered the situation of our output being wrong for this case.

Encoded output from compressing checkpoint2.txt:
00000000100100100101010101010101011111111111111111111111111111111101010101010101010010010011001000000



To build the tree, first we counted how many times each character appeared in checkpoint2.txt. Once we had the counts, we took the characters with the 2 lowest counts and made them into nodes with their counts. We then created a new node as the root of that tree, whose left child was a and right child was b, and whose count was the sum of a and b's counts (12). We then took the character with the next lowest count (c) and repeated this process, creating the new root node with the sum of the new a and c's counts (28). We then took the character with the next lowest count (d) and repeated this process, creating the new root node with the sum of the new a and d's counts (60). Since this is now the only node left in the priority queue, our tree is complete. Since the tree is in the form of a binary tree, we labeled each pointer to a left child with a 0, and pointers to the right child as 1. To get the codeword for a given byte, you simply follow the path down to that leaf and concatenate the corresponding 0 or 1 on the labeled path taken between each node.

To encode the input (aabbbbccccccccddddddddddddddddddddddddddddddddddddddddcccccccbbbbaa), we simply replaced each letter with its respective codeword to get the following:
00000000100100100101010101010101011111111111111111111111111111111101010101010101010010010011001000000

This is equivalent to our compressor output, so we never encountered the situation of our output being wrong for this case.