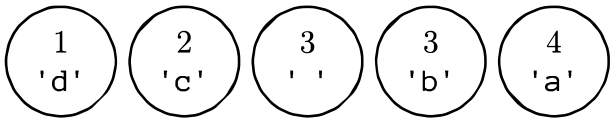
Huffman Coding
CPE 202

Suppose you have a text file with the contents `"abcd abc ab a"`. We want to turn each character into a code composed of `'0'`s and `'1'`s such that the more frequently a character occurs, the shorter its code. So, let's start by looking at how frequently each letter occurs!

Character	`'a'`	`'b'`	`'c'`	`'d'`	`' '`
Frequency	4	3	2	1	3

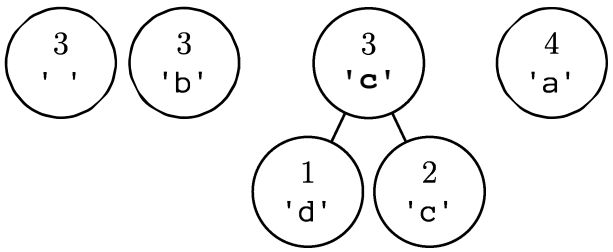
Now let's start building a tree! There are multiple ways to do this but ours will use a sorted list, sorted in ascending order of frequency (i.e., the lowest frequencies will come first). For our implementation, ties in frequency will be broken by looking at the smallest character in the tree (as determined by ASCII value). The smallest value makes for the smaller (when breaking ties in frequency).

1. Create a leaf node for each symbol and add it to sorted list.

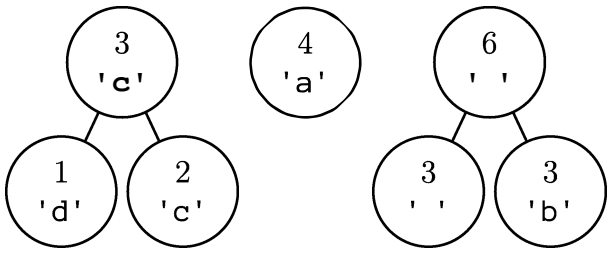


The `' '` is before the `'b'` because it has a smaller ASCII value.

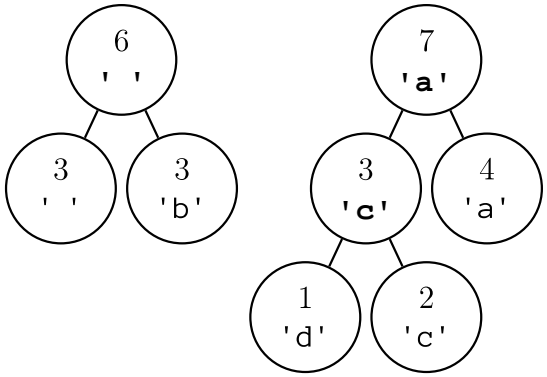
2. While there is more than one node in the list:
   (a) Remove two nodes from the beginning of the list (lowest frequency)
   (b) Create a new internal node with these two nodes as children (for our implementation, the lesser of the two will go on the left) and with frequency equal to the sum of the two nodes' frequencies
   (c) Add the new node to the sorted list (recall that this is a *sorted* list and as such it must be reinserted in the correct location)



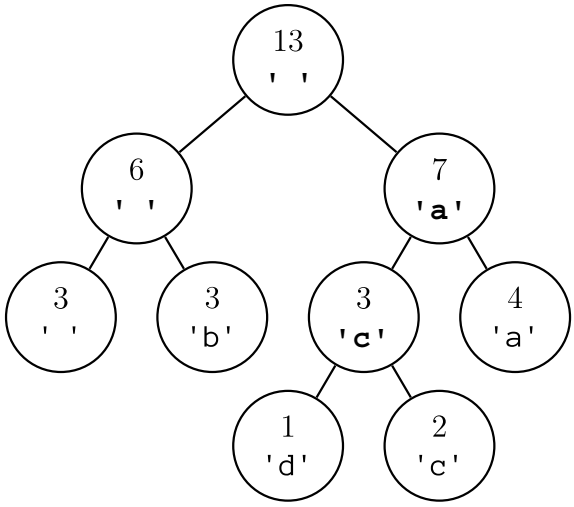
The `'d'` and `'c'` were removed, connected, and re-inserted. (The resulting node is placed after the `' '` and the `'b'` because those characters have smaller ASCII values than `'c'`.)



The `' '` and `'b'` were removed, connected, and re-inserted.



The `'d'`/`'c'` and `'a'` were removed, connected, and re-inserted.



The remaining trees were removed, connected, and re-inserted.

3. The remaining node is the root node and the tree is complete.

When determining codes in our implementation, a `'0'` represents following the *left* child and a `'1'` represents following the *right* child. This means that we have the codes:

Character	`' '`	`'b'`	`'d'`	`'c'`	`'a'`
Code	`"00"`	`"01"`	`"100"`	`"101"`	`"11"`

If we were to compress the entire file from earlier (`"abcd abc ab a"`), we would get the string of codes `"1101101100001101101001101001 1"`. One thing you may notice is that there are no spaces separating the codes. How will we find the breaks when decompressing??? We'll know that we've found the character corresponding to a code when we hit a leaf in the tree!

So, to decompress the string `"1101101100001101101001101001 1"`, let's traverse the tree. A `'1'` will mean to go right, a `'0'` will mean to go left. Every time we hit a leaf node, we've decompressed one character. We'll then move back to the root and keep going.

11	01	101	100	00	11	01	101	00	11	01	00	11
`'a'`	`'b'`	`'c'`	`'d'`	`' '`	`'a'`	`'b'`	`'c'`	`' '`	`'a'`	`'b'`	`' '`	`'a'`

And thus, we've decoded `"1101101100001101101001101001 1"` back to the string `"abcd abc ab a"`.