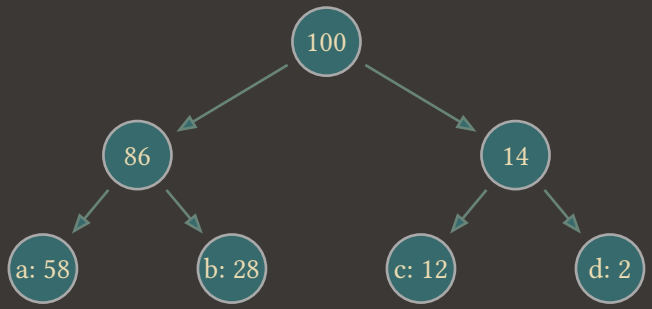
Concept: Binary Tree Representation of Codes

We represent binary codes using a tree as follows, where the code for a letter is the sequence of bits between the root and a leaf.

Note that the constant length coding is a balanced binary tree, whereas variable length coding is not necessarily balanced. Further, if a node has children, it cannot have an associated letter, because it will be a prefix of the children.



Computational Problem: Optimal Coding Scheme

Let C denote our alphabet, and let $f(p)$ denote the frequency of a letter p. Let T be the tree for a prefix code, and let $d_T(p)$ be the depth of p in T. Then the total number of bits needed to encode a file using this code is

$$B(T) = \sum_{p \in C} f(p) d_T(p).$$

We want a code that achieves the minimum possible value of $B(T)$.

Algorithm: Huffman's Algorithm

We can intuitively think of Huffman's algorithm as building the best tree T to represent binary codes.

Initially, each letter is represented by a single node tree, whose weight equals the letter's frequency. We repeatedly choose the smallest tree roots (by weight) and merge them. The new root's weight is the sum of the two children's weights. If there are n letters in the alphabet, there are $n - 1$ merges.

Runtime Analysis

This algorithm runs in $O(n \log n)$ because we initially sort and then do n heap operations.

Huffman's Algorithm

```
1 Q ← C (Q is a priority queue)
2 for i = 1 to n − 1 do
3 |   z ← allocateNode()
4 |   x ← left[z] ← DeleteMin(Q)
5 |   y ← right[z] ← DeleteMin(Q)
6 |   f(z) ← f[x] + f[y]
7 |   Insert(Q, z)
8 return FindMin(Q)
```