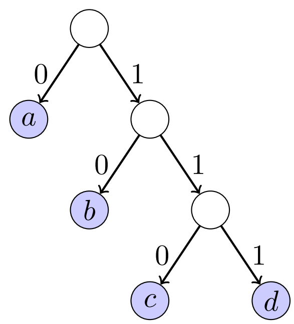
Huffman Codes

1 Encoding

Huffman coding is a lossless ecoding algorithm that encodes an alphabet into a **prefix code**, a mapping of codewords to characters so that no codeword is a prefix of another. The encoded alphabet can be represented as a binary tree with all letters on the leaves.



Optimal Code If $P(x)$ is the probability of letter x and $d_T(x)$ is the depth in the tree of the codeword, for an alphabet C, the expected encoding length for n characters, which we want to minimize, is

$$B(T) = n \sum_{x \in C} p(x) d_T(x) \tag{1}$$

2 Algorithm

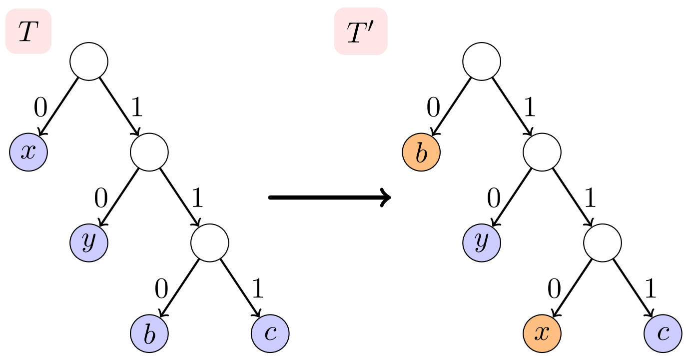
Build the tree bottom-up, merging 2 characters into a meta-character on each step, then recurse on the 1-letter smaller alphabet with the meta-character. When merging, always pick the 2 characters with the lowest probability.

```
for x in C:
    add x to heap Q by p(x)

for i in [1, |C| - 1]:
    z = new internal tree node
    left[z] = x = extract-min(Q)
    right[z] = y = extract-min(Q)
    p(z) = p(x) + p(y)
    insert z into Q

return last element in Q as root
```

3 Proof

Claim For $x, y \in C$ with the lowest probabilities, there exists an optimal tree with these two letters at maximum depth.

Proof By contradiction, take $b, c, x, y \in C \mid p(b) \leq p(c), p(x) \leq p(y)$ with no loss of generality. By the claim that x and y have the lowest probabilities, $p(x) \leq p(b), p(y) \leq p(c)$. Now put b, c at the bottom of the tree so that $d_T(b) \geq d_T(x), d_T(c) \geq d_T(y)$.



Swap the positions of x and b in the tree to obtain the new tree T'. The cost change from T to T' is given by:

$$
\begin{aligned}
B(T') &= B(T) - p(x)d_T(x) + p(x)d_T(b) - p(b)d_T(b) + p(b)d_T(x) && (2)\\
&= B(T) + p(x)(d_T(b) - d_T(x)) - p(b)(d_T(b) - d_T(x)) && (3)\\
&= B(T) - (p(b) - p(x))(d_T(b) - d_T(x)) && (4)
\end{aligned}
$$

In the analysis above, we fixed that $p(x) \leq p(b)$ and $d_T(x) \leq d_T(b)$, so therefore $B(T') \leq B(T)$. By a similar exchange argument for y, this proves that there exists some optimal tree with x and y at the deepest level.

Claim Huffman's algorithm always produces an optimal tree.

Proof Use induction. The base case $n = 1$ is trivially correct, and by the inductive hypothesis we assume correctness on $n - 1$ characters.

Given a tree for n characters such that x, y are the lowest probability letters, we know that they are at the bottom of the optimal solution. Now replace x, y with z so that $p(z) = p(x) + p(y)$, as in the algorithm so that now $|C'| = n - 1$.

Consider that *any* prefix tree T for C' has z as a leaf node (by definition). Replace this leaf node with an internal node branching out to two new leaves x and y. The cost change is:

$$B(T') = B(T) - p(z)d_T(z) + p(x)(d_T(z) + 1) + p(y)(d_T(z) + 1) \tag{5}$$
$$= B(T) + (p(x) + p(y)) \tag{6}$$

By the inductive hypothesis, $B(T)$ is minimal. Because we picked x, y so that their sum would be the smallest possible out of the alphabet, $B(T')$ is optimal as well, which means Huffman's algorithm always produces a minimal prefix code.