Adaptive Huffman Coding

- Each node of the tree is assigned a *weight*.
  - For a leaf node, the weight is the number of times that the symbol corresponding to that leaf has been encountered.
  - For an internal node of the tree, the weight is the sum of the weights of the offspring (children nodes).
- Each node of the tree is also assigned a *number*.
  - If we have an alphabet of size n, there will be $2n-1$ nodes total (n leaf nodes and $n-1$ internal nodes) once the tree reaches its full size (meaning that every symbol in the input alphabet has been encountered at least once).
  - We will number them from 1 to $2n-1$.
  - Sibling nodes will get adjacent numbers.
  - Nodes with higher weight get higher numbers.
  - Parents have higher numbers than their children.

Encoding:

- At the start, the tree at both the encoder and the decoder consists of a single node which corresponds to all symbols *N*ot *Y*et *T*ransmitted (NYT).
- Before encoding begins, encoder and decoder agree on a fixed-length codeword for every symbol in the input alphabet. These codewords will be used for the first time a given symbol appears.
- The very first symbol is encoding using its fixed-length codeword.
- After the very first symbol, each time we have to encode a symbol that is encountered for the first time, we send the code for NYT (obtained by traversing the Huffman tree) and then send the previously agreed upon fixed code for that symbol.
- If a symbol to be encoded has already been encountered, then it has a node in the tree, and we send the codeword for that.
- After each symbol is encoded, the tree is *updated*.

Updating:

- When a symbol is encountered for the first time, the NYT node splits into a new NYT node (with weight 0) and a new leaf node for the new symbol (with weight 1). The old NYT node is given weight 1 (and it becomes an internal node of the tree).
- If a symbol has been encountered before, we check to see if it has the highest node number among all nodes (other than its parent) with the same weight.
  - If yes, we just increment the weight.
  - If no, we exchange this node for that one with the highest number.
- We need to propagate the new weights up the tree.
  - If it was a new leaf, we need to check the parent of the old NYT node.
  - Otherwise, we need to check the parent of the encoded symbol.
- We check whether this parent has the highest node number among all nodes with the same weight.
  - If yes, we just increment the weight.
  - If no, we exchange this node for that one with the highest number.
- Keep on going until you get to the root node, where you just increment the weight.

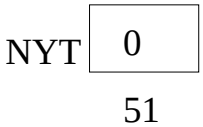
The main concept is this: when you exchange the two nodes, you are taking one that is lower down in the tree (has a longer codeword) and moving it higher up in the tree (has a shorter codeword) reflecting the fact that its counts have become high (it's a higher probability event than we previously realized).

EXAMPLE:

• We wish to encode the messaage [a a r d v a r k i] where our symbol set consists of the 26 lowercase letters.

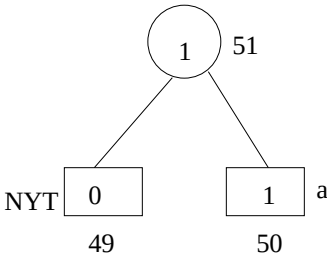
• We assign each of the symbols a 5-bit index. (Actually, since we have only 26 letters, not 32, we could get away with assigning some of them 4-bit indices. But let's ignore that.)

• There will be $2\times 26-1$ = 51 nodes. So, we start numbering them from 51 (and will eventually go down to 1). At the start, there is only a root node:



• The first letter to be transmitted is a. As this is the first symbol, we just send its code.

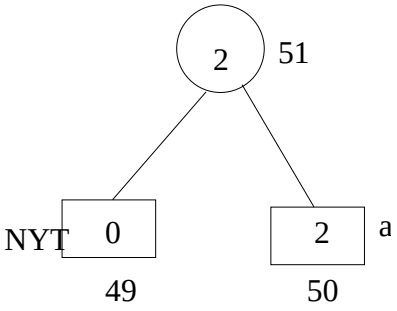
• The NYT node is split into a new NYT node and a node for “a”



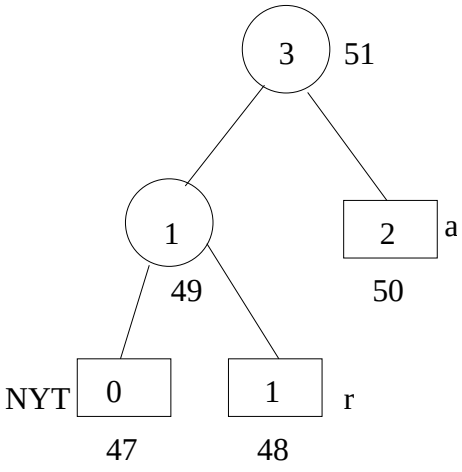
• The weight of the terminal node will be higher than that of the NYT node, so we assign the number 50 to it, and assign 49 to the new NYT node.

• The second letter to be transmitted is also “a”, so we send the code 1 for it.

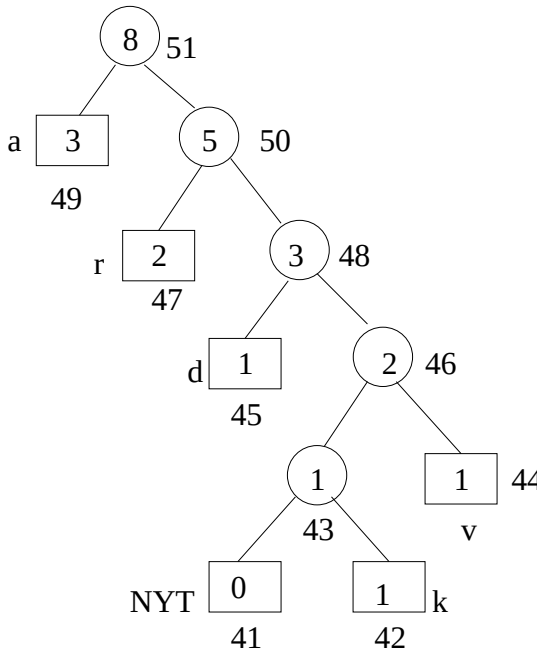
• This node already has the highest node number among all nodes (other than its parent) with the same weight. So, we don’t swap. We just increment.



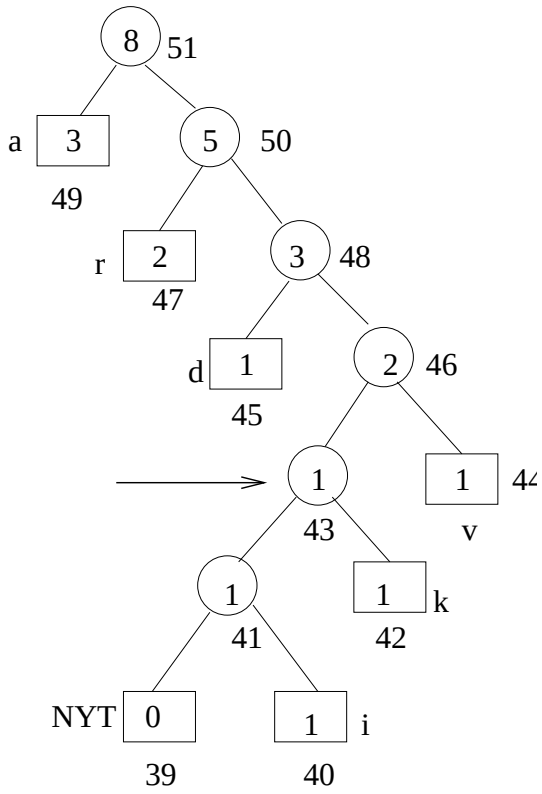
• The next letter to be transmitted is “r” so we send the codeword for NYT, followed by the index for r. The NYT node splits into a new NYT and a leaf for “r”. Weights are incremented.



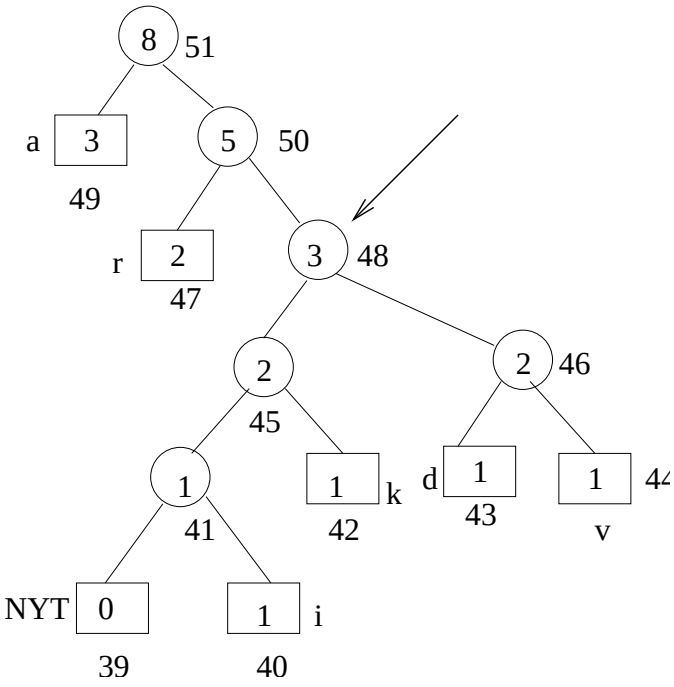
• Skipping ahead a few steps, after the k has been transmitted, the tree looks like this:



• When we transmit the i, the NYT node splits, and we add a new node for i and increment the old NYT node, and then we have to consider its parent (arrow):



• This parent does not have the highest node number among all nodes of the same weight. Number 45 does. So we have to swap it with 45, and then we can increment its weight, and move on to its parent (arrow). Swapping it will mean that it will have a short codeword because it will be higher up in the tree.



• This parent also does not have the highest node number among all nodes of the same weight. Number 49 does. So we have to swap it with 49, and then we can increment its weight. When we move on to its parent, we're at the root node, so we can just increment it: