5.2 Minimum Variance Huffman Code

By performing the sorting procedure in a slightly different manner, we could have found a different Huffman code. The generated codes are identical in terms of their redundancy. However, the variance of the length of the codewords is significantly different, but even more importantly, the Huffman code is not unique. For example, the average size of the generated code (0, 10, 111, 1101, and 1100) shown in Figure2.16a is 0.4×1+0.2×2+0.2×3+0.1×4+0.1×4 = 2.2 bits/symbol. Some of the steps were chosen arbitrarily, since there were more than two symbols with smallest probabilities. Figure 2.16b shows how the same five symbols can be combined differently to obtain a different Huffman code (11, 01, 00, 101, and 100). The average size of this code is 0.4×2+0.2×2+0.2×2+0.1×3+0.1×3 = 2.2 bits/symbol, the same as the previous code.

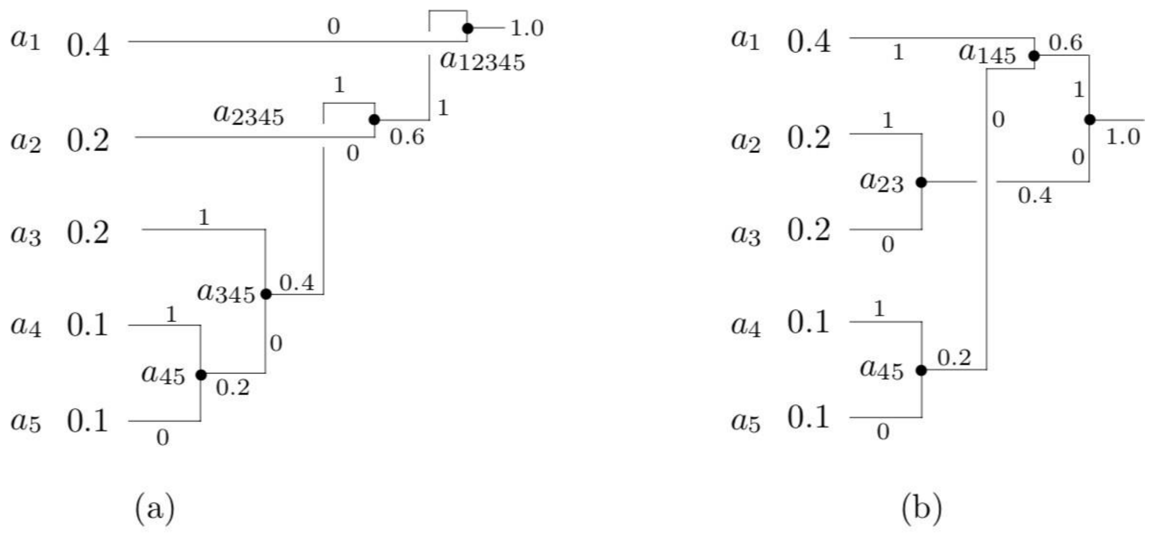


Figure 2.16: Huffman Codes.

It turns out that the arbitrary decisions made in constructing the Huffman tree affect the individual codes but not the average size of the code. Still, we have to answer the obvious question, which of the different Huffman codes for a given set of symbols is best? The answer, while not obvious, is simple: The best code is the one with the smallest variance. The variance of a code measures how much the sizes of the individual codes deviate from the average size. The variance of code 2.16a is

$0.4(1 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(3 - 2.2)^2 + 0.1(4 - 2.2)^2 + 0.1(4 - 2.2)^2 = 1.36$,

while the variance of code 2.16b is

$0.4(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.1(3 - 2.2)^2 + 0.1(3 - 2.2)^2 = 0.16$.

Code 2.16b is therefore preferable. A careful look at the two trees shows how to select the one we want. In the tree of Figure 2.16a, symbol a_{45} is combined with a_3, whereas in the tree of 2.16b it is combined with a_1. The rule is: When there are more than two smallest-probability nodes, select the ones that are lowest and highest in the tree and combine them. This will combine symbols of low probability with ones of high probability, thereby reducing the total variance of the code.

5.3 Ternary (Nonbinary) Huffman Codes

The Huffman code is not unique. Moreover, it does not have to be binary! The Huffman method can easily be applied to codes based on other number systems. Figure 2.26a shows a Huffman code tree for five symbols with probabilities 0.15, 0.15, 0.2, 0.25, and 0.25. The average code size is

$2\times0.25 + 3\times0.15 + 3\times0.15 + 2\times0.20 + 2\times0.25 = 2.3$ bits/symbol.

Figure 2.26b shows a ternary Huffman code tree for the same five symbols. The tree is constructed by selecting, at each step, three symbols with the smallest probabilities and merging them into one parent symbol, with the combined probability. The average code size of this tree is

$2\times0.15 + 2\times0.15 + 2\times0.20 + 1\times0.25 + 1\times0.25 = 1.5$ trits/symbol. Notice that the ternary codes use the digits 0, 1, and 2.

Furthermore, given seven symbols with probabilities .02, .03, .04, .04, .12, .26, and .49, the construction of binary and ternary Huffman code-trees are shown in Figure 2.26c and Figure 2.26d, respectively.

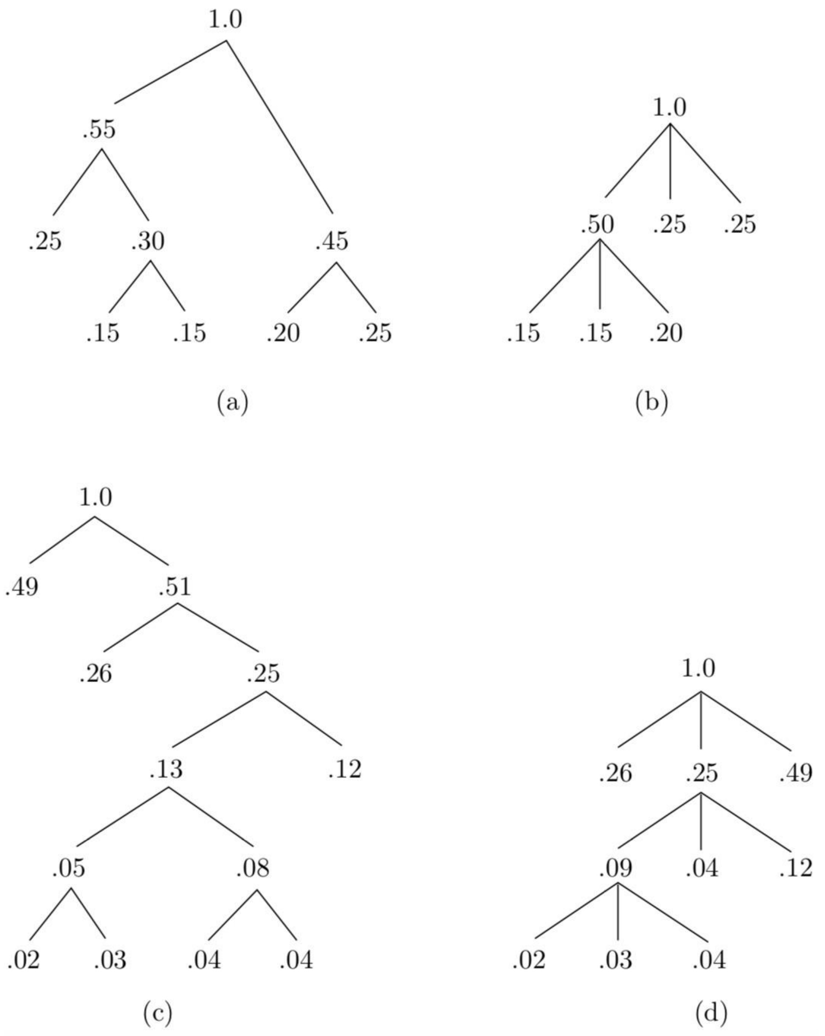


Figure 2.26: Binary and Ternary Huffman Code-Trees.

5.4 Adaptive Huffman Coding

Huffman coding requires knowledge of the probabilities of the source sequence. If this knowledge is not available, Huffman coding becomes a two-pass procedure: the statistics are collected in the first pass, and the source is encoded in the second pass. In order to convert this algorithm into a one-pass procedure, Faller and Gallagher independently developed adaptive algorithms to construct the Huffman code based on the statistics of the symbols already encountered. These were later improved by Knuth and Vitter. Theoretically, if we wanted to encode the (k+1)-th symbol using the statistics of the first k symbols, we could recompute the code using the Huffman coding procedure each time a symbol is transmitted. However, this would not be a very practical approach due to the large amount of computation involved—hence, the adaptive Huffman coding procedures.

The Huffman code can be described in terms of a binary tree similar to the ones explained previously. The codeword for a symbol can be obtained by traversing the tree from the root to the leaf corresponding to the symbol, where 0 corresponds to a left branch and 1 corresponds to a right branch. In order to describe how the adaptive Huffman code works, we add two other parameters to the binary tree: the weight of each leaf, which is written as a number inside the node, and a node number. The weight of each external node is simply the number of times the symbol corresponding to the leaf has been encountered. The weight of each internal node is the sum of the weights of its offspring. The node number y_i is a unique number assigned to each internal and external node. If we have an alphabet of size n, then the $2n-1$internal and external nodes can be numbered as $y_1, \dots, y_{2n-1}$ such that if x_j is the weight of node y_j, we have $x_1 \leq x_2 \leq \cdots \leq x_{2n-1}$. Furthermore, the nodes y_{2j-1} and y_{2j} are offspring of the same parent node, or siblings, for $1 \leq j < n$, and the node number for the parent node is greater than y_{2j-1} and y_{2j}. These last two characteristics are called the sibling property, and any tree that possesses this property is a Huffman tree

In the adaptive Huffman coding procedure, neither transmitter nor receiver knows anything about the statistics of the source sequence at the start of transmission. The tree at both the transmitter and the receiver consists of a single node that corresponds to all symbols not yet transmitted (NYT) and has a weight of zero. As transmission progresses, nodes corresponding to symbols transmitted will be added to the tree, and the tree is reconfigured using an update procedure. When a symbol is encountered for the first time, the code for the NYT node is transmitted, followed by the fixed code for the symbol. A node for the symbol is then created, and the symbol is taken out of the NYT list. Both transmitter and receiver start with the same tree structure. The updating procedure used by both transmitter and receiver is identical. Therefore, the encoding and decoding processes remain synchronized.

5.4.1 Update Procedure

The update procedure requires that the nodes be in a fixed order. This ordering is preserved by numbering the nodes. The largest node number is given to the root of the tree, and the smallest number is assigned to the NYT node. The numbers from the NYT node to the root of the tree are assigned in increasing order from left to right, and from lower level to upper level. The set of nodes with the same weight makes up a block. Figure 3.6 is a flowchart of the updating procedure.

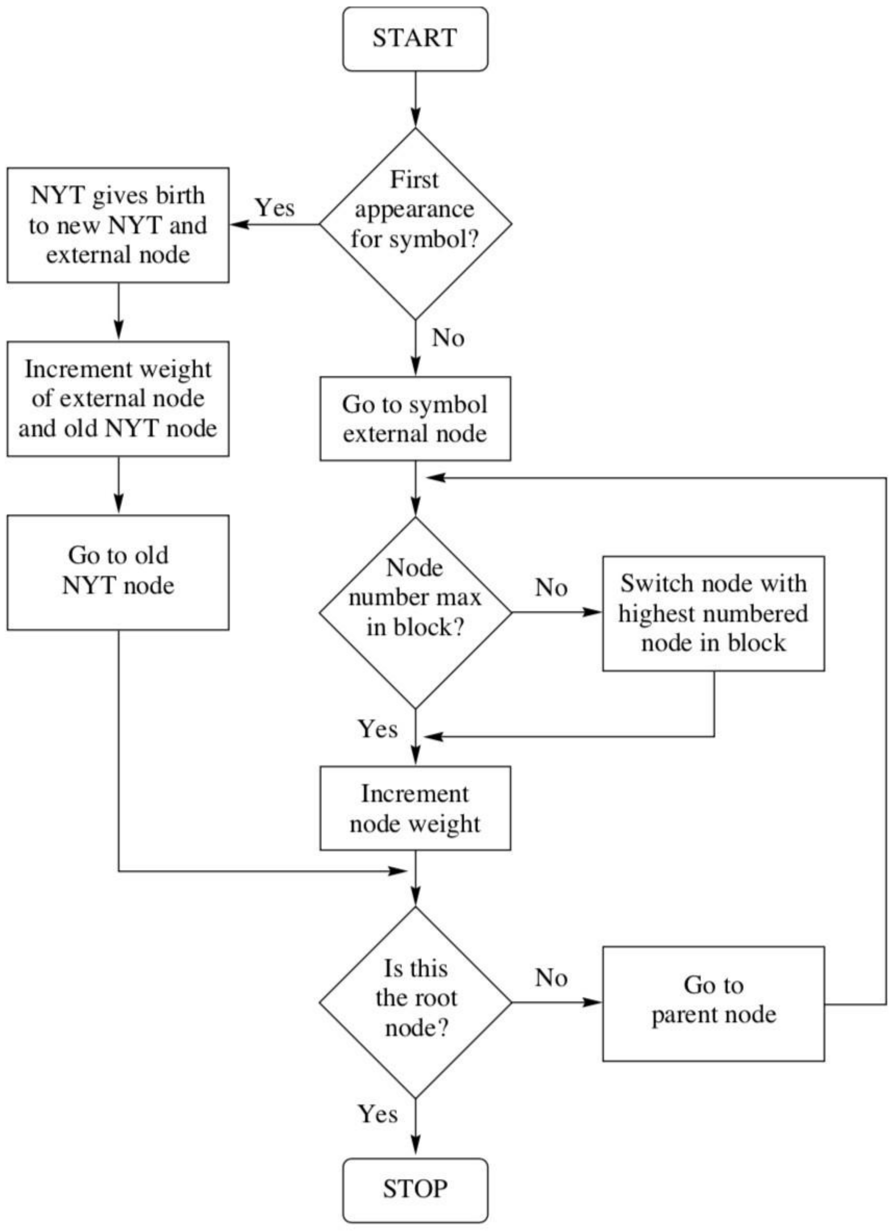

START
First appearance for symbol?
Yes
NYT gives birth to new NYT and external node
No
Increment weight of external node and old NYT node
Go to symbol external node
Go to old NYT node
Node number max in block?
No
Switch node with highest numbered node in block
Yes
Increment node weight
Is this the root node?
No
Go to parent node
Yes
STOP

FIGURE 3. 6 **Update procedure for the adaptive Huffman coding algorithm.**

The function of the update procedure is to preserve the sibling property. In order that the update procedures at the transmitter and receiver both operate with the same information, the tree at the transmitter is updated after each symbol is encoded, and the tree at the receiver is updated after each symbol is decoded. The procedure operates as follows:

After a symbol has been encoded or decoded, the external node corresponding to the symbol is examined to see if it has the largest node number in its block. If the external node does not have the largest node number, it is exchanged with the node that has the largest node number in the block, as long as the node with the higher number is not the parent of the node being updated. The weight of the external node is then incremented. If we did not exchange the nodes before the weight of the node is incremented, it is very likely that the ordering required by the sibling property would be destroyed.

Once we have incremented the weight of the node, we have adapted the Huffman tree at that level. We then turn our attention to the next level by examining the parent node of the node whose weight was incremented to see if it has the largest number in its block. If it does not, it is exchanged with the node with the largest number in the block. Again, an exception to this is when the node with the higher node number is the parent of the node under consideration. Once an exchange has taken place (or it has been determined that there is no need for an exchange), the weight of the parent node is incremented. We then proceed to a new parent node and the process is repeated. This process continues until the root of the tree is reached.

If the symbol to be encoded or decoded has occurred for the first time, a new external node is assigned to the symbol and a new NYT node is appended to the tree. Both the new external node and the new NYT node are offspring of the old NYT node. We increase the weight of the new external node by one. As the old NYT node is the parent of the new external node, we increase its weight by one and then go on to update all the other nodes until we reach the root of the tree.

Example: Update procedure

Assume we are encoding the message [a a r d v a r k], where our alphabet consists of the 26 lowercase letters of the English alphabet. The updating process is shown in Figure 3.7. We begin with only the NYT node. The total number of nodes in this tree will be $2 \times 26 - 1 = 51$, so we start numbering backwards from 51 with the number of the root node being 51.

The first letter to be transmitted is a. As a does not yet exist in the tree, we send a binary code 00000 for a and then add a to the tree. The NYT node gives birth to a new NYT node and a terminal node corresponding to a. The weight of the terminal node will be higher than the NYT node, so we assign the number 49 to the NYT node and 50 to the terminal node corresponding to the letter a. The second letter to be transmitted is also a. This time the transmitted code is 1. The node corresponding to a has the highest number (if we do not consider its parent), so we do not need to swap nodes. The next letter to be transmitted is r. This letter does not have a corresponding node on the tree, so we send the codeword for the NYT node, which is 0 followed by the index of r, which is 10001. The NYT node gives birth to a new NYT node and an external node corresponding to r. Again, no update is required. The next letter to be transmitted is d, which is also being sent for the first time. We again send the code for the NYT node, which is now 00 followed by the index for d, which is 00011. The NYT node again gives birth to two new nodes. However, an update is still not required. This changes with the transmission of the next letter, v, which has also not yet been encountered. Nodes 43 and 44 are added to the tree, with 44 as the terminal node corresponding to v. We examine the grandparent node of v (node 47) to see if it has the largest number in its block. As it does not, we swap it with node 48, which has the largest numberin its block. We then increment node 48 and move to its parent, which is node 49.In the block containing node 49, the largest number belongs to node 50. Therefore, we swap nodes 49 and 50 and then increment node 50. We then moveto the parent node of node 50, which is node 51. As this is the root node, all we do is increment node 51.

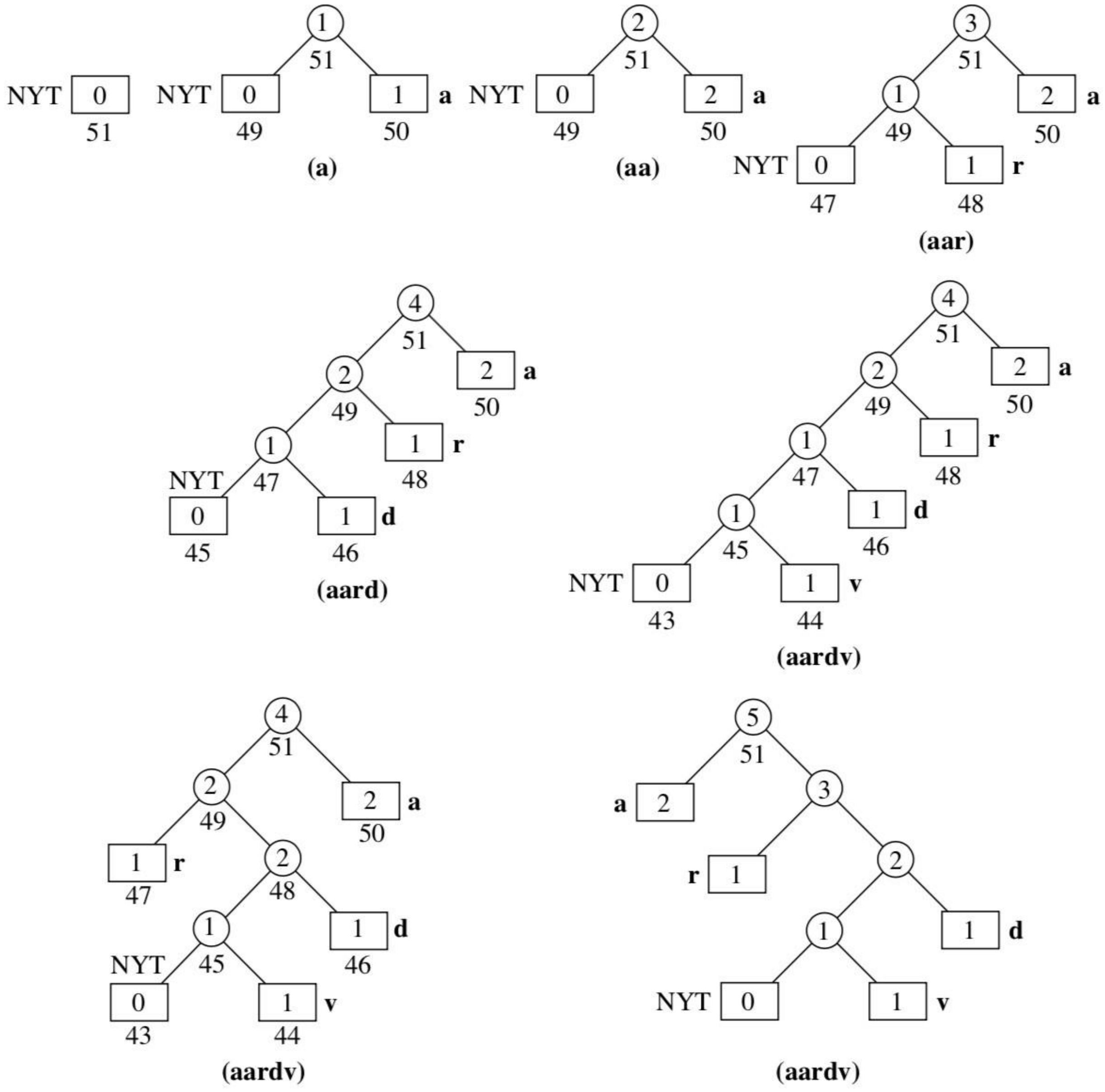
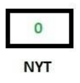
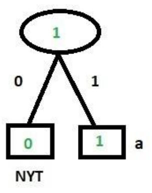
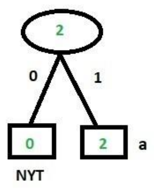


FIGURE 3. 7 **Adaptive Huffman tree after [a a r d v] is processed.**

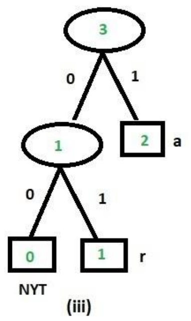




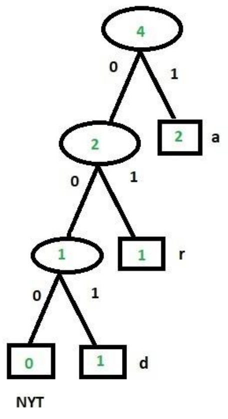
(i)



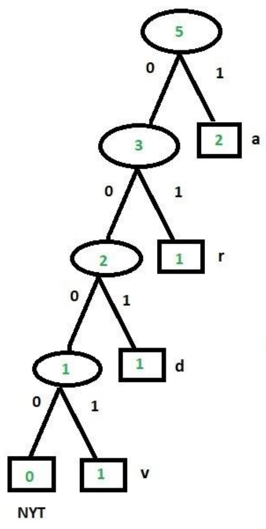
(ii)



(iii)

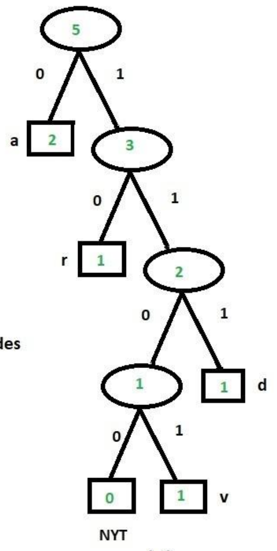


(iv)

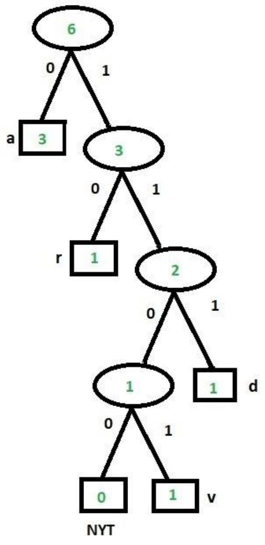


(v)

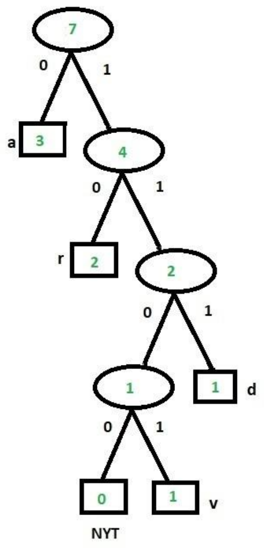
Swapping of nodes



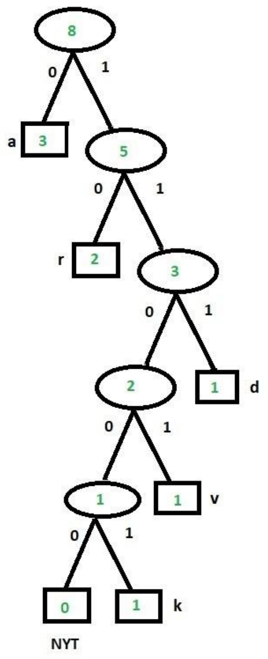
(vi)



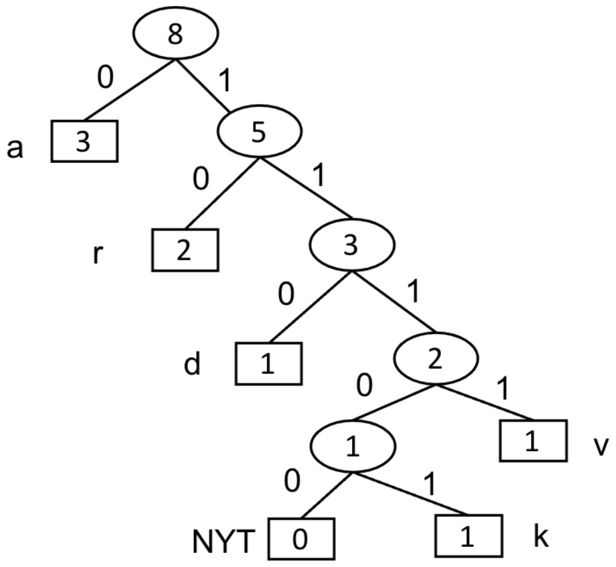
(vii)



(viii)



(ix)



Adaptive Huffman after (a a r d v a r k) is processed

5.4.2 Encoding Procedure

The flowchart for the encoding procedure is shown in Figure 3.8. Initially, the tree at both the encoder and decoder consists of a single node, the NYT node. Therefore, the codeword for the very first symbol that appears is a previously agreed-upon fixed code. After the very first symbol, whenever we have to encode a symbol that is being encountered for the first time, we send the code for the NYT node, followed by the previously agreed-upon fixed code for the symbol. The code for the NYT node is obtained by traversing the Huffman tree from the root to the NYT node. This alerts the receiver to the fact that the symbol whose code follows does not as yet have a node in the Huffman tree. If a symbol to be encoded has a corresponding node in the tree, then the code for the symbol is generated by traversing the tree from the root to the external node corresponding to the symbol. To see how the coding operation functions, we use the same example that was used to demonstrate the update procedure.

Example: Encoding procedure

In this example, we used an alphabet consisting of 26 letters. In order to obtain our prearranged code, we have to find **m** and **e** such that $2^e + r = 26$, where $0 \leq r < 2^e$. It is easy to see that the values of **e = 4** and **r = 10** satisfy this requirement.

The first symbol encoded is the letter a. As a is the first letter of the alphabet, k = 1. As 1 is less than 20, a is encoded as the 5-bit binary representation of k−1, or 0, which is 00000. The Huffman tree is then updated as shown in the figure. The NYT node gives birth to an external node corresponding to the element a and a new NYT node. As a has occurred once, the external node corresponding to a has a weight of one. The weight of the NYT node is zero. The internal node also has a weight of one, as its weight is the sum of the weights of its offspring. The next symbol is again a. As we have an external node corresponding to symbol a, we simply traverse the tree from the root node to the external node corresponding to a in order to find the codeword. This traversal consists of a single right branch. Therefore, the Huffman code for the symbol a is 1.

After the code for a has been transmitted, the weight of the external node corresponding to a is incremented, as is the weight of its parent. The third symbol to be transmitted is r. As this is the first appearance of this symbol, we send the code for the NYT node followed by the previously arranged binary representation for r. If we traverse the tree from the root to the NYT node, we get a code of 0 for the NYT node. The letter r is the 18th letter of the alphabet; therefore, the binary representation of r is 10001. The code for the symbol r becomes 010001. The tree is again updated as shown in the figure, and the coding process continueswith symbol d. Using the same procedure for d, the code for the NYT node, whichis now 00, is sent, followed by the index for d, resulting in the codeword 0000011. The next symbol v is the 22nd symbol in the alphabet. As this is greater than 20,we send the code for the NYT node followed by the 4-bit binary representation of 22 − 10 − 1 = 11. The code for the NYT node at this stage is 000, and the 4-bitbinary representation of 11 is 1011; therefore, v is encoded as 0001011. The nextsymbol is a, for which the code is 0, and the encoding proceeds.

The binary string generated by the encoding procedure is:

00000 1 010001 000011 0001011 0 (a a r d v a)

00000 1 010001 000011 0001011 0 10 110001010 (a a r d v a r k)

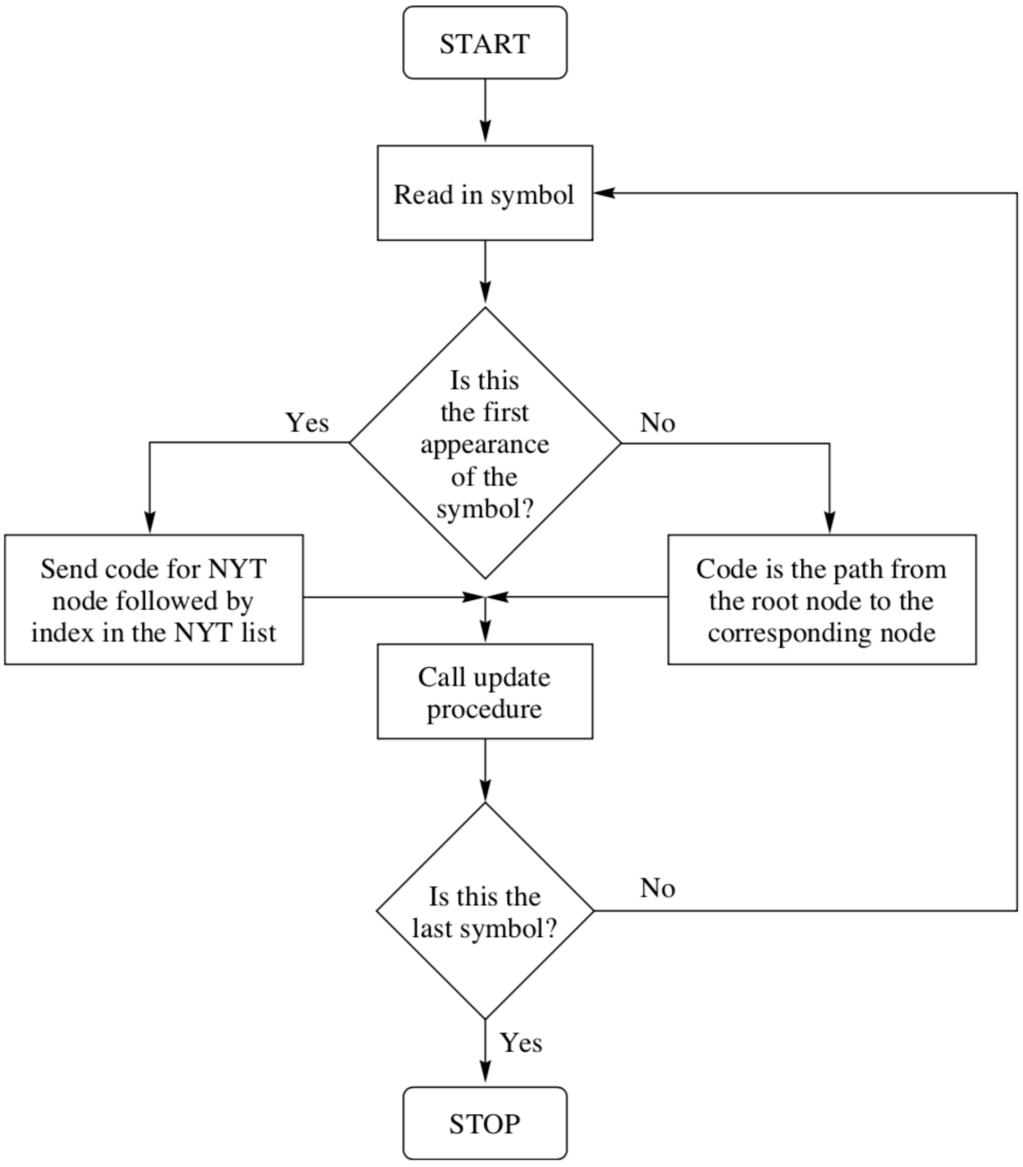

START
Read in symbol
Is this the first appearance of the symbol?
Yes
No
Send code for NYT node followed by index in the NYT list
Code is the path from the root node to the corresponding node
Call update procedure
Is this the last symbol?
No
Yes
STOP

FIGURE 3. 8 **Flowchart of the encoding procedure.**