Worksheet: Huffman Encoding

Huffman encoding was developed in 1952 by David A. Huffman as part of a class assignment on data compression while he was a graduate student at MIT. The algorithm is elegant and efficient, and was adopted first for text and file compression, and later integrated into formats such as ZIP, JPEG, and MP3.

The Huffman Encoding Algorithm

The algorithm assigns variable-length binary codes to symbols based on their frequencies. The most frequent symbols get the shortest codes, while more rare symbols receive longer codes, minimizing the overall encoded message size.

The steps to the encoding algorithm are:

- Count the frequency of each symbol
- Sort the symbols by their frequency
- Construct the Huffman Tree
- Assign binary codes (left edges = 0, right edges = 1)
- Encode the data
- Store the Huffman Tree (the codebook) and the data in a file

We will go through each of these steps using a simple example for compressing text: the made-up word: “ABRACADABRA”.

Count the Frequency of Each Symbol

When manually going through our example, we can make a simple tally chart to count the number of occurrences of each symbol in our data. We now tally the number of each character in the word: “ABRACADABRA”.

Symbol	A	B	R	C	D
Tally	𝍸	II	II	I	I
Total	5	2	2	1	1

Sort the Symbols by Their Frequency

It just so happens that the symbols are arranged from high to low. We wish to arrange them in a ***priority queue***, with a small number of occurrences being higher priority. A ***priority queue*** is a list that is sorted such that the highest priority will be taken out first. We can represent the priority queue as a table:

Symbol	C	D	B	R	A
Total	1	1	2	2	5

The first element in a queue is the *front* of the queue, o*r* ***head*** of the queue. In our example, symbol C is at the head of the queue and is the highest priority element in the queue.

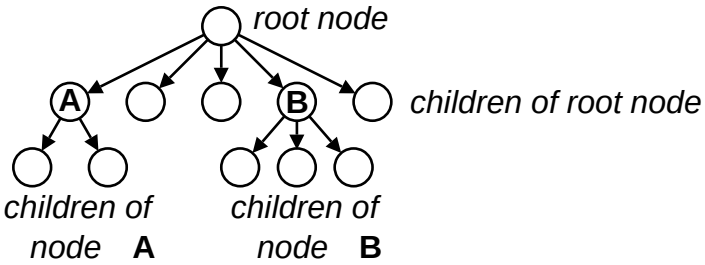
The order will be important in our future steps. We are not only ordering primarily by the frequency, but secondarily, when there are equal numbers of a particular symbol, the order is according to when it was first seen in the data sequence. Since B occurs before R in the data, B comes before R in the priority queue. Similarly C occurs before D in the data, so C comes before D in the priority queue. We can choose any secondary ordering we wish, as long as it is consistent between the tree generated for encoding and decoding the data.

Worksheet: Huffman Encoding

The Binary Tree Data Structure

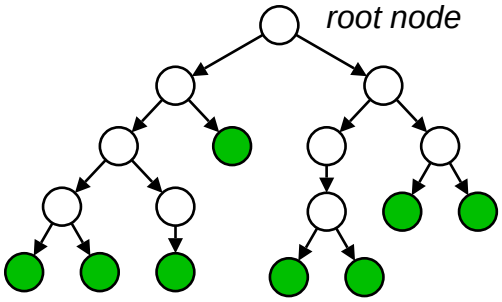
Before we discuss the construction of the Huffman Tree, let us briefly discuss the computer science data structure called a ***binary tree***.

A tree is a data structure made up of any number of ***nodes***. Each node may have any number of ***children***. Each tree has one ***root*** node. The diagram below represents a simple tree. Nodes are represented as circles. Arrows point from a node to its children.



In the diagram, the root node is labeled. The root node has five children, including one labeled child A and another labeled child B. Each of those two labeled nodes has children – node A has two children, while node B has three children. A link between a node and its child is a ***branch***. Any node that does not have any children is called a ***leaf node***.

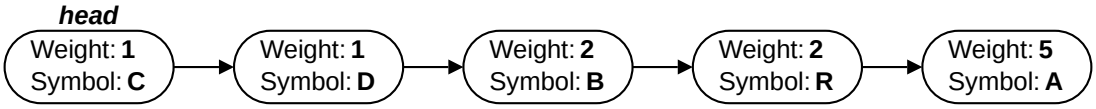
A binary tree is a subset of the tree data structure. A binary tree is a type of tree where any node has a maximum of two children. Commonly the children are referred to as the ***left*** and ***right*** child. Binary trees are an extremely useful data structure, and they are often used for sorting data. The diagram below shows a simple binary tree. The leaves of the tree have been shaded in light green.



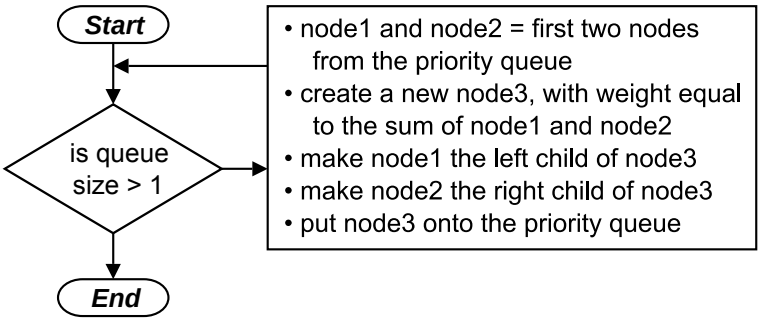
Worksheet: Huffman Encoding

Construct the Huffman Tree

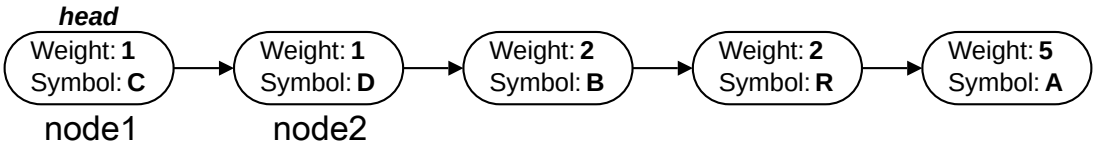
For constructing the *Huffman Tree*, each priority-symbol pair will be a ***node***. In accordance with the vocabulary for a tree, we will call the number of occurrences for each node the ***weight*** of the node. Below is the priority queue from our example ("ABRACADABRA"), represented as a chain of nodes.



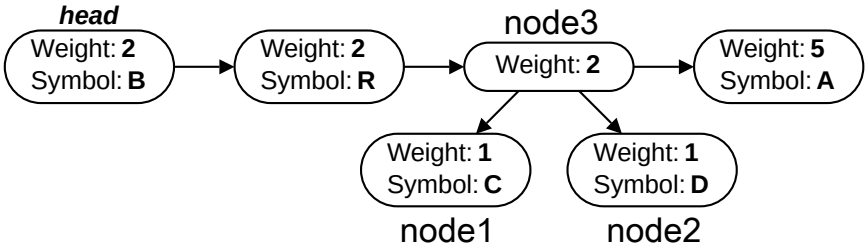
A Huffman Tree is a type of ***binary tree***. The algorithm to construct a Huffman Tree is as follows. We will follow this simple algorithm to construct the tree.



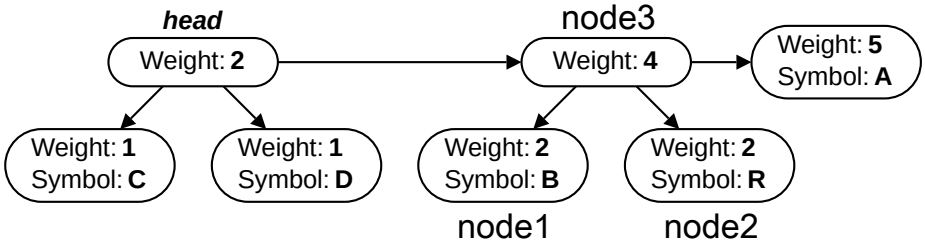
Let's perform the first iteration of the loop for the algorithm. The first node, node1, contains symbol C and a weight of 1. The second node, node2, contains symbol D and a weight of 1.



Thus, we create a new node, node3 with weight 2 (the sum of the weight of node1 and node2), set the left node to node1, the right node to node2, and finally place the new node back on the queue. For this example, when nodes have an equal weighting, the new node will be to the queue after any nodes of equal weight. How the tree differs if we add to the queue earlier than nodes of equal weight will be shown later.

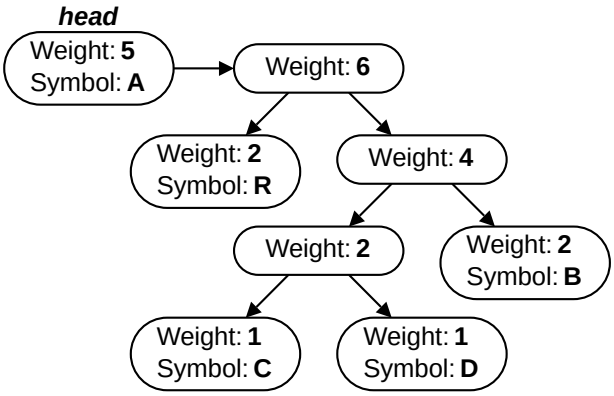


For the second iteration of the loop, the first node, node1, contains symbol B and a weight of 2. The second node, node2, contains symbol R and a weight of 2. We create a new node with weight 4, and set the left and right children – node1 on the left, node2 on the right.

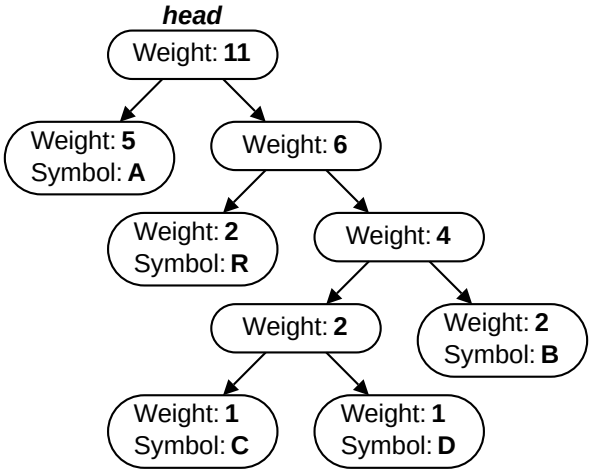


Now the third iteration. The new node will have a weight of 6. The right child will be the node with weight 2 and symbol R. The left node will be the tree with weight 4.

Worksheet: Huffman Encoding



After the fourth and final iteration of the loop, we end up with a single node in the queue, with a weight of 11 – equal to the total number of symbols in our data.



As shown in the flowchart for creating the Huffman Tree, once there is only one node in the queue, the task is complete. The head node of the priority queue now contains the root node of the Huffman Tree.

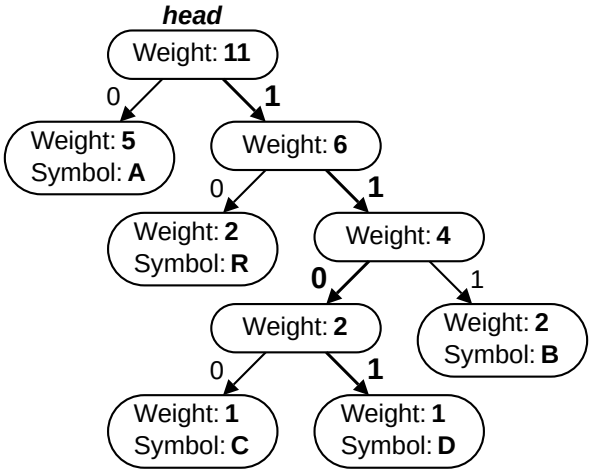
Worksheet: Huffman Encoding

Assigning the Binary Codes

Assigning the values to each symbol is done simply by starting at the root node and traverse the tree until a leaf node is reached. By convention, we assign a value of 0 to left branches and a value of 1 to right branches.

For the symbol A, we start at the root, take the left branch, which gives a value of 0, and we arrive at the node with symbol A. Thus, the symbol A, the most frequently occurring symbol, is encoded by a single digit: 0. We only needed to take one branch to arrive at the leaf node.

The remaining symbols will each require more than a single digit to encode. Use the diagram below to confirm that the symbol D is encoded for by 1101. (It is just a coincidence that both C and D happen to be encoded by the binary codes that also represent those hexadecimal digits.



Here is the final binary encoding for each symbol:

Symbol	A	R	C	D	B
Encoding	0	10	1100	1101	111

Our original word, “ABRACADABRA” would be represented in 23 bits as the bitstream: 01111001100011010111100. The table below shows how this bitstream was generated.

A	B	R	A	C	A	D	A	B	R	A
0	111	10	0	1100	0	1101	0	111	10	0

There are few words that only use these letters but try to decode the following words:

0100111	1100100111	1110101101	11000101101
ARAB	CRAB	BARD	CARD