Teacher guide: Huffman code

This resource supports the delivery of the Data Compression section of or GCSE Computer Science (8520) specification. It will help you to understand the process of using a tree to represent Huffman code.

Overview

Huffman coding is a compression technique used to reduce the number of bits needed to send or store a message. Huffman coding is based on the frequency of occurrence of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently.

Preparation

Consider the sentence: *he ties the tether*

The frequency of each character (including the space character) is:

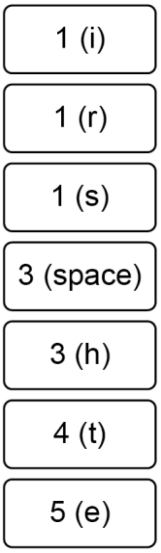
character	frequency
i	1
r	1
s	1
space	3
h	3
t	4
e	5

A minimum of 3 bits is needed to encode the seven different characters (i, r, s, space, h, t, e) in the sentence if each character is represented using the same number of bits.
The total number of characters is 18.
Using a fixed-length character encoding, the sentence would require a minimum of 3 x 18 = 54 bits.

A Huffman coding can be used to reduce the number of bits needed.

Method

Create a node (nodes are the rounded rectangles as shown in the image below) for every different character and label each node with the frequency of the character as well as the character itself, shown in brackets. Make a list of these nodes and arrange them in order of frequency from lowest to highest.

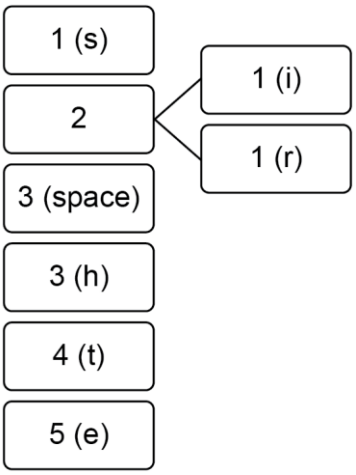


Use the following algorithm:

1. Take the top two nodes out and join them together to make a new node. The label for this new node is the combined frequency of the two nodes that were taken out.
2. Place this new node back, ensuring the list of nodes is still in order.
3. Repeat until there is only one node left.

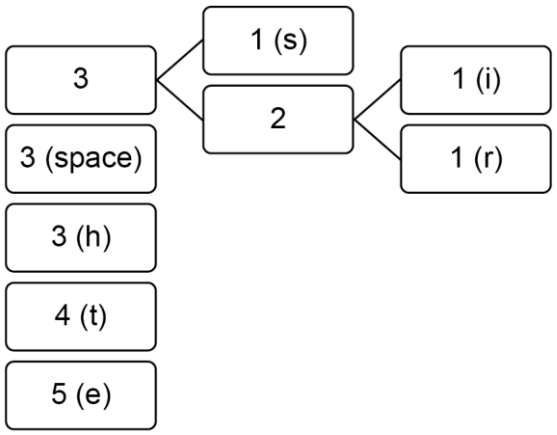
Example: Step 1

Create a new node using the top two nodes 1 (i) and 1 (r). The combined frequency of these two nodes is 2, so a new node with the label 2 is created and the original nodes are linked to this with two lines. This node will be placed in the list between the remaining 1 (s) and 3 (space) nodes to keep the list in order.



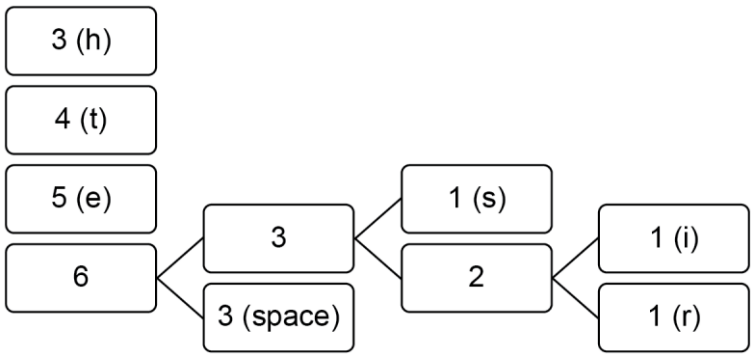
Step 2

Take the top two nodes 1 (s) and 2 and create a new node with the label 3, as this is the sum of 1 and 2. Put this back at the top of the list (although it doesn't matter if it goes ahead, in between or after the next two nodes as they also have the label 3).



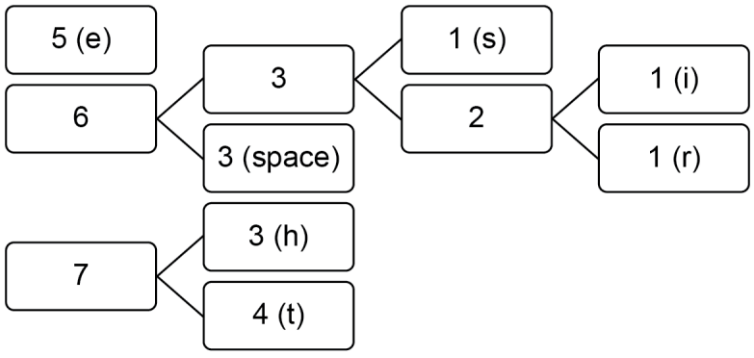
Step 3

Repeat the steps for the top two labels to create a new node with the label 6. This is the highest value label so it will be placed at the bottom of the list.



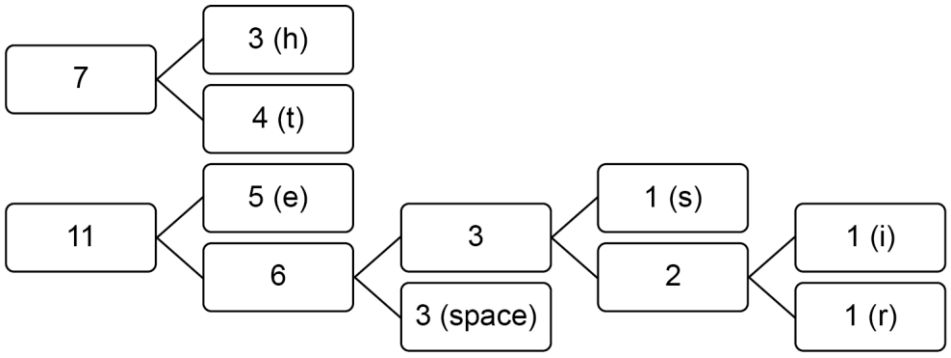
Step 4

The next node is created from the 3 and 4 nodes to make a new 7 node that will again be placed at the bottom as it is the highest value.



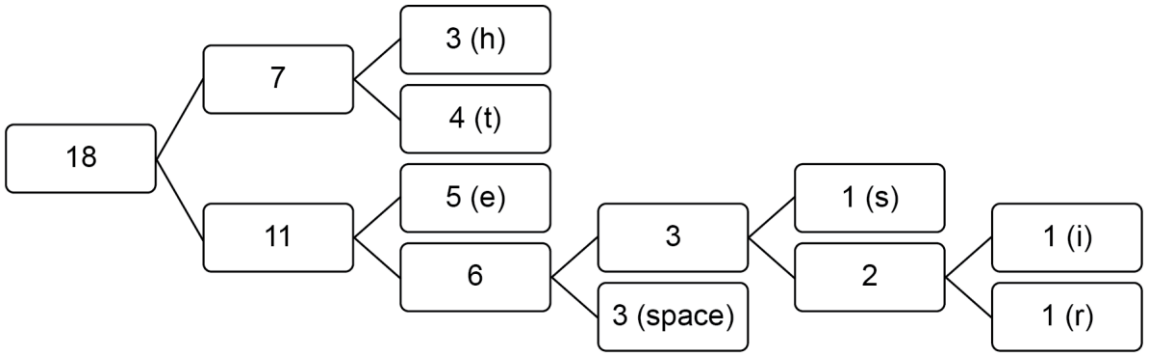
Step 5

The next node is created from the 5 and 6 nodes to make a new 11 node.



Step 6

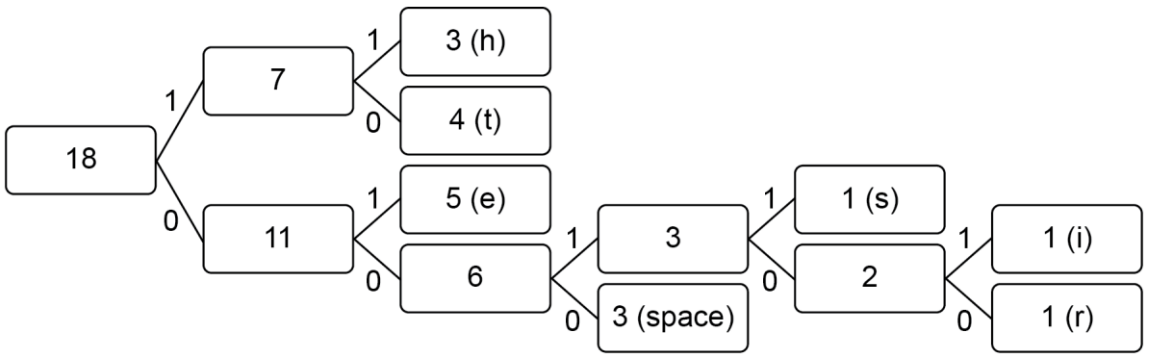
The final node is made up of the last two remaining nodes. At this point the algorithm stops.



You can perform a basic form of error detection on the Huffman code at this point as the label of the leftmost node of the Huffman code should be the same as the total number of characters in the original text.

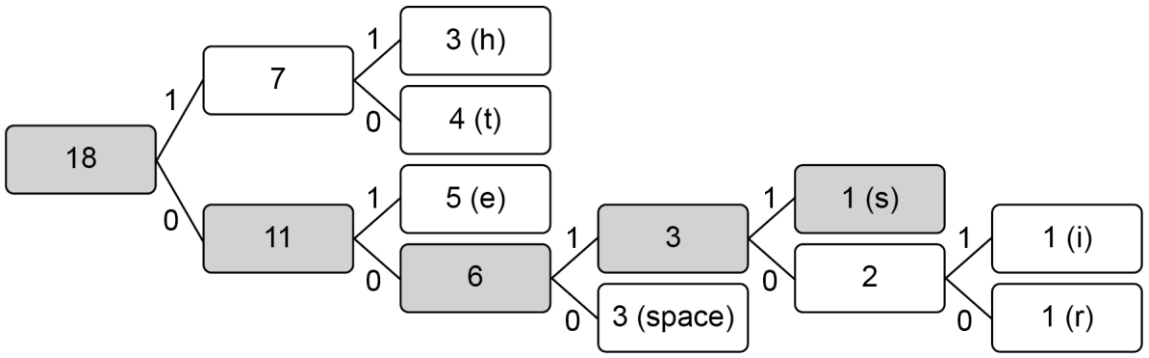
Step 7

Now the nodes are complete, the lines from each node need to be labelled so that the Huffman coding can be read. Each upper line will be labelled with a 1 and each lower line with a 0:



Step 8

The coding is now complete. To read the coding for a character you start at the node on the left and trace the route to the character's node reading off the 1 or 0 on the lines as you go. For example, to read the coding for the character s you would start at the leftmost node and follow the lines to the s node like so:



So the code for the character s is 0011.

The complete Huffman coding is:

character	Huffman coding
i	00101
r	00100
s	0011
space	000
h	11
t	10
e	01

The encoding for he ties would be 110100010001010010011.

Justification

The total encoding for he ties the tether requires 47 bits using our Huffman coding. This compares with the 54 bits necessary for a fixed-length encoding where every character requires 3 bits. We have compressed the original encoding by almost 13% without losing any data.

Summary

In a novel written in English you would expect to see characters such as *e*, t and space far more commonly than *z* and *q*. Using a Huffman coding for this file would compress the information by an even larger percentage than that found for the sentence he ties the tether. In fact if we ignore the space needed to store the list of nodes itself (a Huffman tree), a Huffman coding will always perform as well or better than a fixed-length encoding.