Compression – Huffman Coding

Compressing Text

You should already be aware of the reasons why **data compression** may be desirable and the difference between **lossless** and **lossy** compression.

Huffman coding is a **lossless compression** method which can be applied to passages of text. The aim is to reduce the file size needed to store the text.

The method works by looking at the **frequency** of characters in the text – how often each character appears. So, in the text:

"woolloomooloo"

Woolloomooloo is a place in Sydney, Australia.

the frequency of each character is:

w	o	l	m
1	8	3	1

Huffman coding was developed by American computer scientist David A Huffman in the 1950s. Huffman was also known for his work in the field of Mathematical Origami…

You can see straight away that the character 'o' occurs most frequently (8 times), with 'l' appearing 3 times and 'w' and 'm' each once.

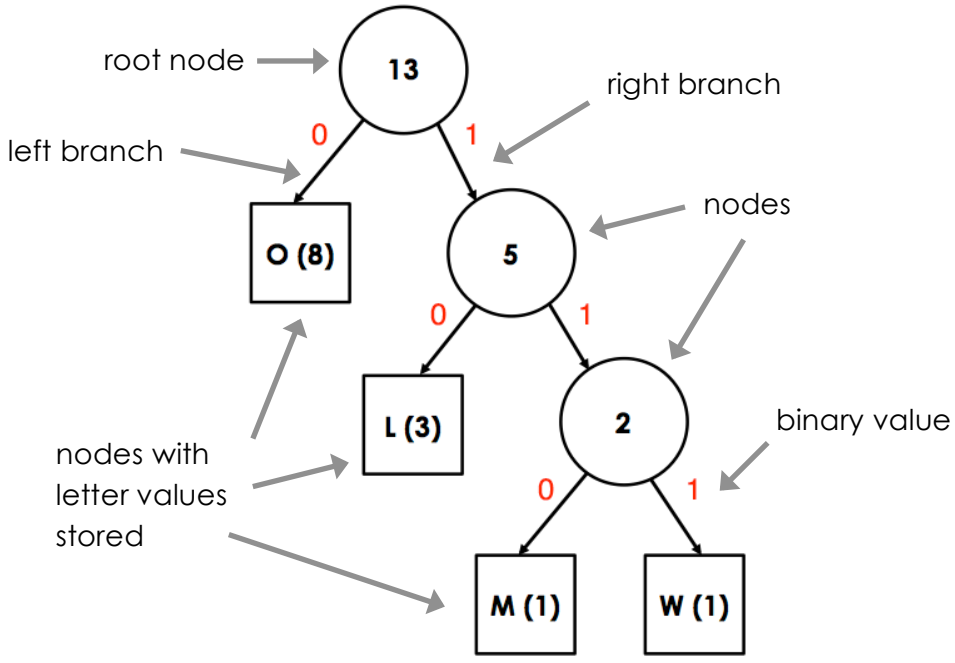
The more frequently a character occurs in the text the more effective the Huffman Coding technique is.

Huffman Trees

Huffman coding works by building a **Binary Tree** for the passage of text. The most common characters appear towards the top of the tree with the rarest characters towards the bottom.

A binary tree is just a way of organising data. It consists of a number of **nodes** (circles and squares). These are connected by **branches**. Each node has two **sub-branches** – so there are two possible values. These can be given the binary values 0 and 1.

In the binary tree diagrams used by the exam board, 0 is always on the left.



For now don't worry about how Huffman Trees are built. Just accept that the most frequently occurring characters are towards the top and the less frequent characters towards the bottom.

Using Huffman Trees

Trees have branches. In Huffman Trees the left branch is always coded 0 and the right branch is coded 1. By adding bits together we can create a reference for each character in the tree.

So, in the tree opposite, 'o' is coded as the single bit 0, 'l' is 10, 'm' is 110 and 'w' is 111. This can be recorded in a table as:

Character	Huffman Coding
o	0
l	10
m	110
w	111

In an exam you might be given a Huffman Tree and asked to complete a table like this.

Notice that letters at the top of the tree are coded using less binary digits (bits). Because these letters appear more frequently in the text this means we'll save storage space.

This means that the original text can be written as:

11100101000110001000

It might make this easier to read if I put the character breaks in using spaces:

111 0 0 10 10 0 0 110 0 0 10 0 0

This requires a total of 20 bits – there are 20 binary digits in the line. So, we're using 20 binary digits to encode a word which is 13 characters long (there are 13 letters in woolloomooloo).

If we were using 7-bit ASCII coding for characters, the same text would require 7 bits per character.

bits needed = 7 bit × 13 characters = 91 bits

So the space saving is:

91 bits – 20 bits = 71 bits saved.

You will need to be able to do calculations like these in exams.

Activity 1:

a) What is the purpose of Huffman Coding?

b) Describe where in a Huffman Tree the most frequently occuring letters are stored.

c) Using the Huffman Tree opposite, the word "moo" can be coded as 110 0 0. This uses 5 bits of memory to store.
Write down the binary coding for the following words using the same Huffman Tree:
(i) "loom" (ii) "owl" (iii) "low" (iv) "moll"

d) For each of the words in c), write down the number of bits required to store the word using Huffman Coding.

e) Using the same Huffman Tree,
(i) encode the word "loom"
(ii) calculate the number of bits needed to encode "loom" using 7-bit ASCII code
(iii) calculate the space saving by encoding "loom" using Huffman Coding