R 53 Huffman Coding - Reading

Huffman coding is a **lossless** method of **compression**. It looks at the **frequency** at which a **data item** occurs and is particularly useful in **text compression**. In English, letters such as 'e' are frequently used, whereas those like 'z' are rarely used. This method of compression uses fewer bits to **encode** frequently occurring letters than those that are rare.

Example: Encode the text "*bookkeeper*" using Huffman coding

Answer: We make a **Huffman tree**. Each **node** (rounded rectangle) in the tree stores the frequency (a number). It will also store a letter if it represents a letter in the tree.

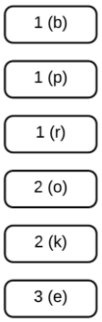
First we find the frequency at which each letter occurs (shown to the right).

Letter	Frequency
b	1
p	1
r	1
o	2
k	2
e	3

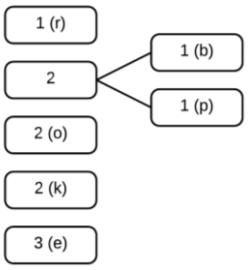
To build the Huffman tree we use the following algorithm:

1. **Arrange nodes in order (smallest at top)**
2. **Combine nodes at the top together and place in the correct position**
3. **Repeat until only one node is on the left-hand column**

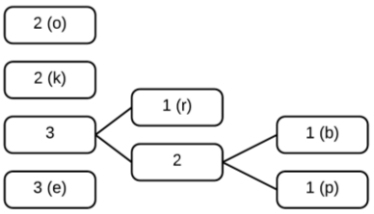
Step 1: Put nodes into order – lowest at top



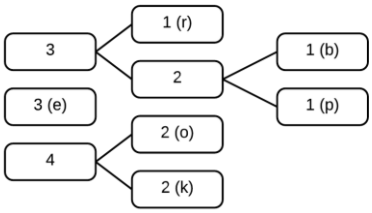
Step 2: Take the two nodes at top and combine as two new branches on the tree



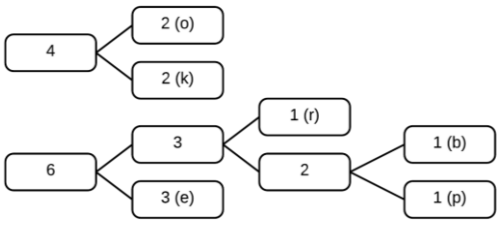
Step 3: Take the top two nodes (2 and 'r') and create a new node of frequency 3. Reorder.



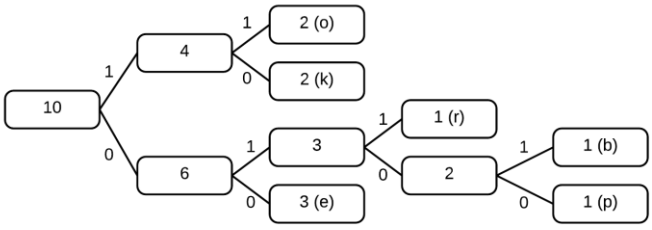
Step 4: Take top two nodes (o,k) and combine to a node of 4. Reorder.



Step 5: Combine the two top nodes (3,3) to make a node of 6. Reorder.



Step 6: Combine the nodes 4 and 6 to make a node of 10. We now only have one node on the left-hand column so can finish. Now add 1 and 0 to each branch.



Character	Huffman coding
b	0101
p	0100
r	011
o	11
k	10
e	00

By assigning 1 and 0 to each branch on the Huffman tree we can now create a Huffman code for each character (left). The word *bookkeeper* can now be converted to binary.

b	o	o	k	k	e	e	p	e	r
0101	11	11	10	10	00	00	0100	00	011

This requires 25 bits to store. If we had 3 bits for each letter (the minimum needed without Huffman coding) we would need 3*10=30 bits. We have therefore compressed the data by (30-23)/30 = 23%

The final binary for *bookkeeper* would be: 010111111010000001000011

Licensed to knole Academy, #10193259