11.2 Applications of trees

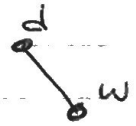
We have already seen an application of trees to hydrocarbons. Trees also naturally appear in computer science and we look next at binary search trees and also prefix codes.

Binary search trees

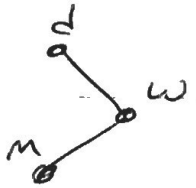
The idea here is to make a structure so that we can efficiently store and find information. The best structure is a binary tree.

To make a tree for the list {d, w, m, b, y, a, p} we label the root with the first letter:

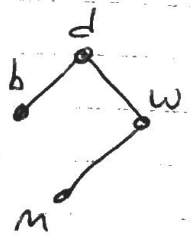
Next we use the alphabetical order to decide if the next letter goes as a child on the left or right. Since w comes after d it goes on right:



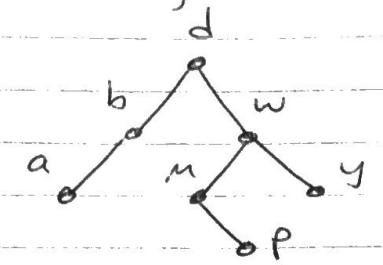
Next compare m with the root label d. It comes after so follow the edge right. It comes before w so



Next compare b. It comes before d:



Repeat this procedure for the remaining elements to get the binary search tree

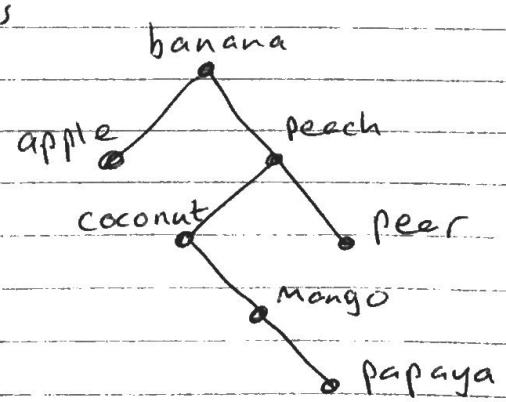


T_1

Each of these letters could be the label of a file. If you are looking for the file labeled by y, for example, you start at the root. Compare y with d. It comes after d so go right. Compare with w next. It comes after so go right. We have found the file y.

Example. Build a binary search tree for the words banana, peach, apple, pear, coconut, mango, papaya. Use the alphabetical (or lexicographical) order.

The answer is



T_2

Note that <u>peach</u> and <u>pear</u> agree on the first three letters. On the fourth they have c and r so pear goes after peach.

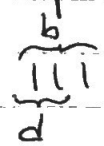
Binary search trees give a very efficient way to store and retrieve data. They become less efficient if they are unbalanced though. In our examples T_1 is balanced but T_2 is not. There are algorithms to rebalance binary search trees.

Prefix codes

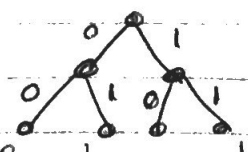
Suppose you want to encode the letters of the alphabet using bit strings. For example

$a = 10$
$b = 111$
$c = 011$
$d = 11$.

If you receive the message 111011 then what were the letters? There is a problem because they could have been "bc" or "dad".
The problem is caused by some bit strings being the first parts (prefixes) of others:



Prefix codes avoid this problem. We construct prefix codes using binary trees. For example

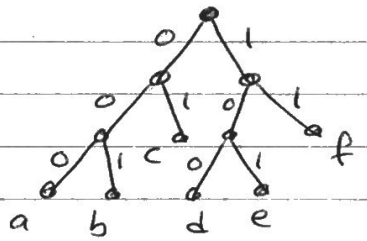


$a = 00$
$b = 01$
$c = 10$
$d = 11$

The edges of the tree are labeled with 0 or 1 (siblings getting different numbers) and the path from the root to the leaf gives the code for that letter.

With this code, the message 111011 has to be "dcd".

Example Use the prefix code given by this binary tree to encode "face".



Answer: 1 1 0 0 0 0 1 1 0 1
f a c e

Huffman coding

This is a method to produce binary prefix codes that use the fewest bits (and allow us to send messages most efficiently.

Example. Find the binary code that represents this sentence

"the tree path part there"

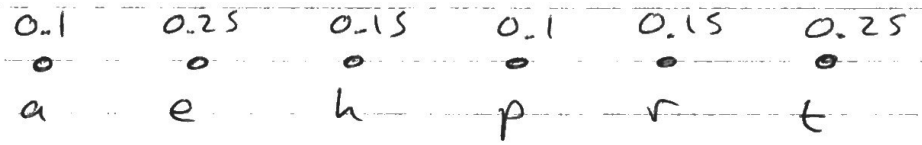
in the fewest bits. What is the average number of bits used to encode a letter of this sentence?

We first work out the relative frequency of each letter:

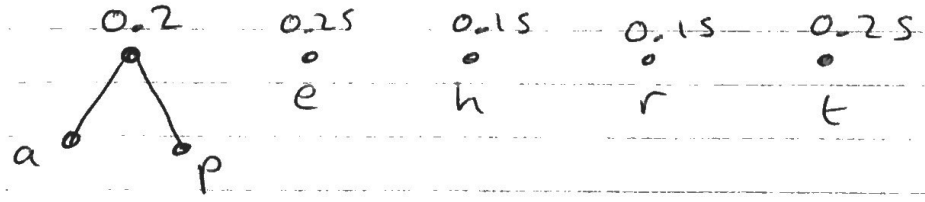
letters	a	e	h	p	r	t	
appear	2	5	3	2	3	5	times
relative frequency	0.1	0.25	0.15	0.1	0.15	0.25	

[$\frac{\text{appears}}{\text{total}=20}$

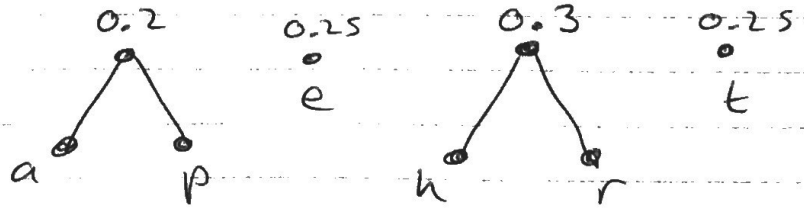
Next we make each letter the root of a tree and give it the weight (number) of its relative frequency.



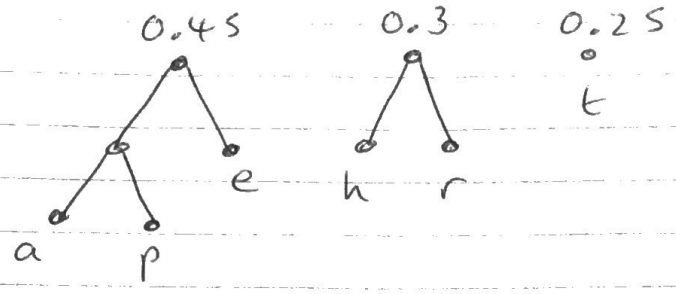
Find the two smallest weights and combine them with a new root. Give this new root a weight which is the sum of these two smallest weights:



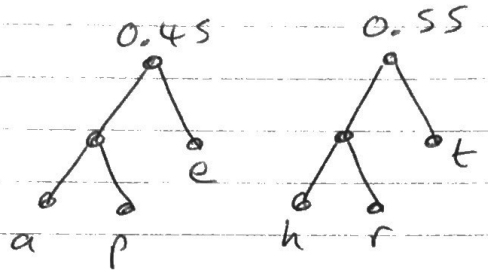
Repeat - now h and r are smallest



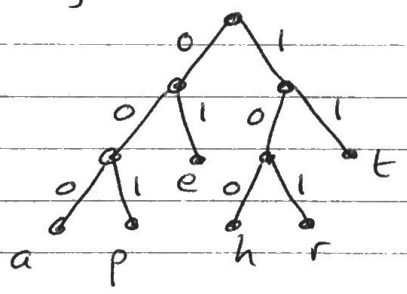
The weights 0.2 and 0.25 are now smallest



Then



and lastly



and add 0s and 1s with 0 on left and 1 on right.

Travelling down from the root we get the codes: ⟶

a = 000
p = 001
e = 01
h = 100
r = 101
t = 11

We see that letters that appear more often are coded with fewer bits. For example

"the tree" ⟶ 11100011110101 01

For the full sentence, the average number of bits used per letter = $\frac{\text{total bits}}{\text{total letters}} = \frac{50}{20}$

= **2.5 bits per letter**

Also = $3(0.1) + 2(0.25) + 3(0.15) + 3(0.1) + 3(0.15) + 2(0.25)$.

(bits for a; rel. freq of a)