lab_huffman

CS 225 Data Structures

Question:

What is the minimum bits required for encoding a message “feed me more food” with prefix-free codes?

prefix-free codes: the bit string (representing some particular symbol) is never a prefix of the bit string (representing any other symbol). (Wikipedia, Huffman coding)

A is 10, B is 1010. What is 101010?

Question:

What is the minimum bits required for encoding a message “feed me more food” with prefix-free codes?

Frequency of each character:

r : 1 | d : 2 | f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

Question:

What is the minimum bits required for encoding a message “feed me more food”? Frequency: r : 1 | d : 2 | f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

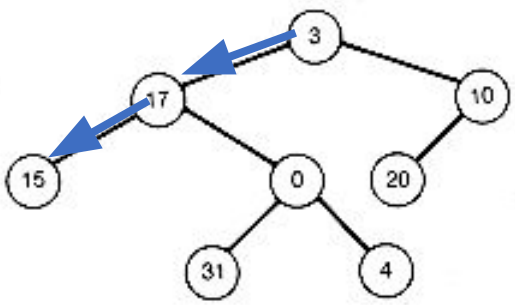
Char	Binary
f	000
e	001
d	010
m	011
r	100
o	110
’SPACE’	101

(A)

Total_Bits(A) = 3*17 = 51 bits

Weighted Path Length (WPL):

Given a binary tree T with internal nodes and leaves (denoted as external nodes), associate weights with each external node. The ***weighted path length*** of T is the sum of the product of the weight and path length of each external node, over all external nodes.



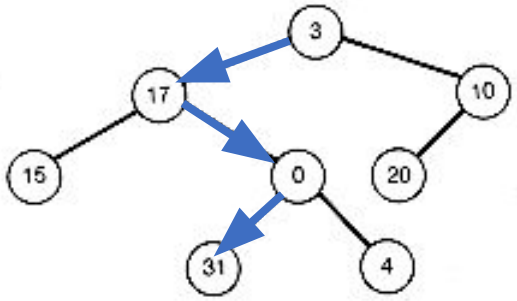
Internal nodes: 3, 17, 10, 0
External nodes: 15, 31, 4, 20

WPL(T) = 15*2 + …

https://planetmath.org/WeightedPathLength

Weighted Path Length (WPL):

Given a binary tree T with internal nodes and leaves (denoted as external nodes), associate weights with each external node. The ***weighted path length*** of T is the sum of the product of the weight and path length of each external node, over all external nodes.



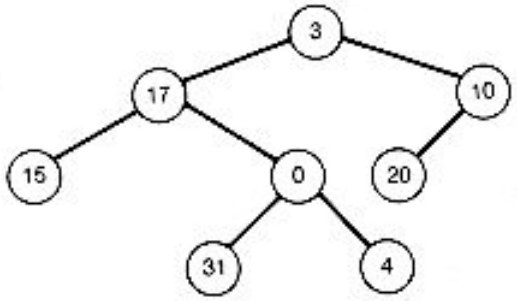
Internal nodes: 3, 17, 10, 0
External nodes: 15, 31, 4, 20

WPL(T) = 15*2 + 31*3 + …

https://planetmath.org/WeightedPathLength

Weighted Path Length (WPL):

Given a binary tree T with internal nodes and leaves (denoted as external nodes), associate weights with each external node. The ***weighted path length*** of T is the sum of the product of the weight and path length of each external node, over all external nodes.



Internal nodes: 3, 17, 10, 0
External nodes: 15, 31, 4, 20

WPL(T) = 15*2 + 31*3 + 4*3 + 20*2 = 175

https://planetmath.org/WeightedPathLength

Weighted Path Length (WPL):

If we regard **external nodes** as our **characters**, **weights of nodes** as **frequency** of each character, **path length** as the **bits** required for encoding each characters, the WPL should give us the total bits to encode the message.

“feed me more food”

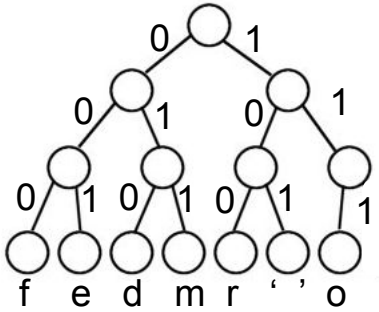
Frequency of each character:

r : 1 | d : 2 | f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

Weighted Path Length (WPL):

Frequency (Weight) of characters: r : 1 | d : 2 | f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

Char	Binary
f	000
e	001
d	010
m	011
r	100
o	111
’SPACE’	101

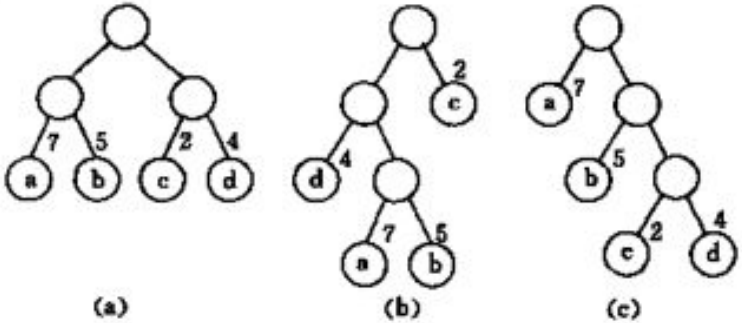


WPL(T) = 17 *3 = 51

Huffman Tree

Given a set of symbols and their weights. {a, 7}, {b, 5}, {c, 2}, {d, 4}

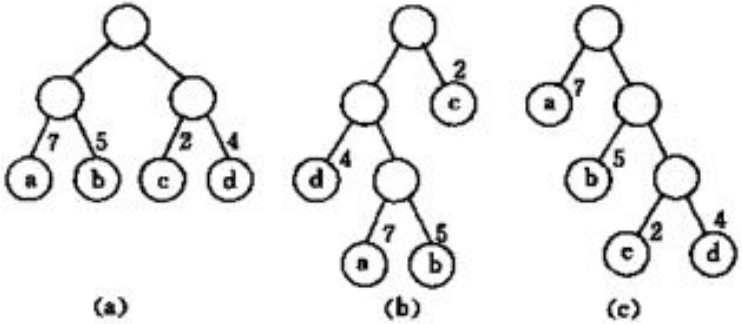
Find a binary tree with ***minimum weighted path length*** from the root.



Huffman Tree

Given a set of symbols and their weights. {a, 7}, {b, 5}, {c, 2}, {d, 4}

Find a binary tree with ***minimum weighted path length*** from the root.

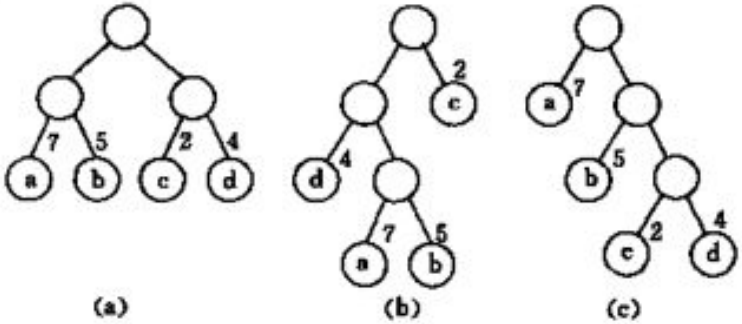


a: WPL = 7×2 + 5×2 + 2×2 + 4×2 = 36;
b: WPL = 7×3 + 5×3 + 2×1 + 4×2 = 46;
c: WPL = 7×1 + 5×2 + 2×3 + 4×3 = 35;

Huffman Tree

Given a set of symbols and their weights. {a, 7}, {b, 5}, {c, 2}, {d, 4}

Find a binary tree with ***minimum weighted path length*** from the root.

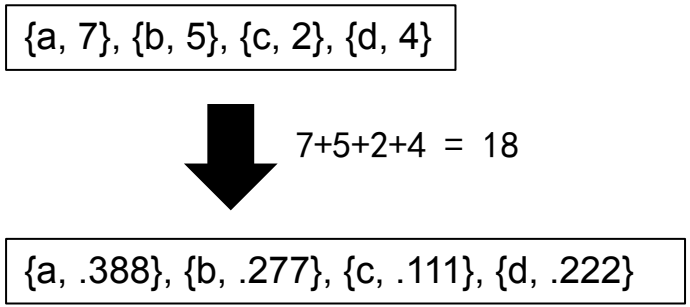


a: WPL = 7×2 + 5×2 + 2×2 + 4×2 = 36;
b: WPL = 7×3 + 5×3 + 2×1 + 4×2 = 46;
c: WPL = 7×1 + 5×2 + 2×3 + 4×3 = 35;

Huffman Tree

Given a set of symbols and their weights (usually proportional to probabilities).

Find a binary tree with ***minimum weighted path length*** from the root.



Building the Huffman Tree

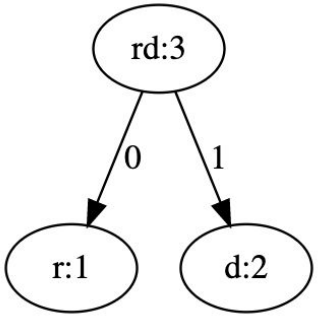
Input: “feed me more food”

Step 1: Calculate frequency of every character in the text, and order by increasing frequency. Store in a queue.

r : 1 | d : 2 | f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

Building the Huffman Tree

Step 2: Build the tree from the bottom up. Start by taking the two least frequent characters and merging them (create a parent node for them). Store the merged characters in a new queue:

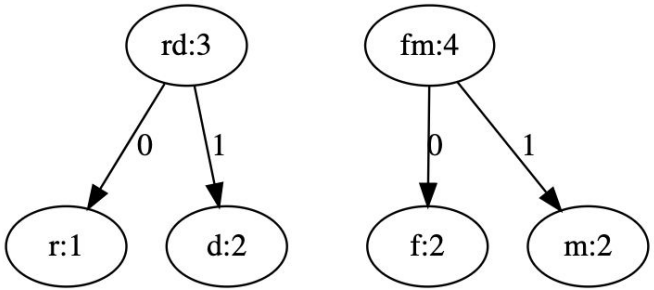


SINGLE: f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

MERGED: rd : 3

Building the Huffman Tree

Step 3: Repeat Step 2 this time also considering the elements in the new queue. ‘f’ and ‘m’ this time are the two elements with the least frequency, so we merge them:

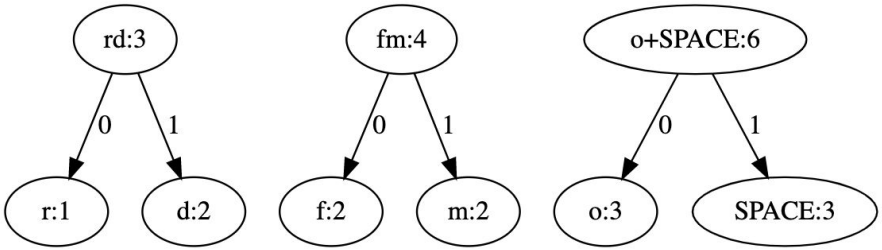


SINGLE: o : 3 | 'SPACE' : 3 | e : 4

MERGED: rd : 3 | fm : 4

Building the Huffman Tree

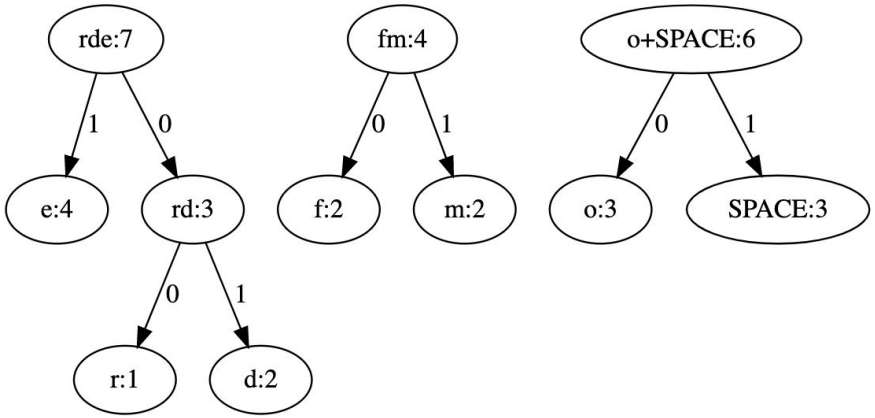
Step 4: Repeat Step 3 until there are no more elements in the SINGLE queue, and only one element in the MERGED queue:



SINGLE: e : 4

MERGED: rd : 3 | fm : 4 | o+SPACE : 6

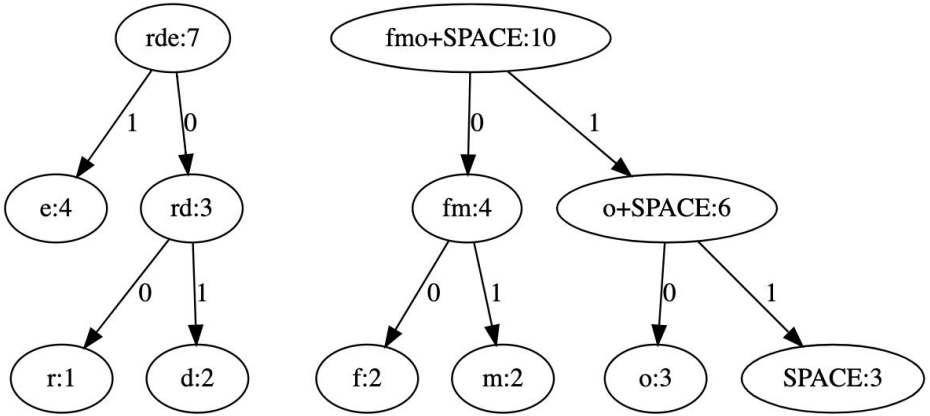
Building the Huffman Tree



SINGLE:

MERGED: fm : 4 | o+SPACE : 6 | rde: 7

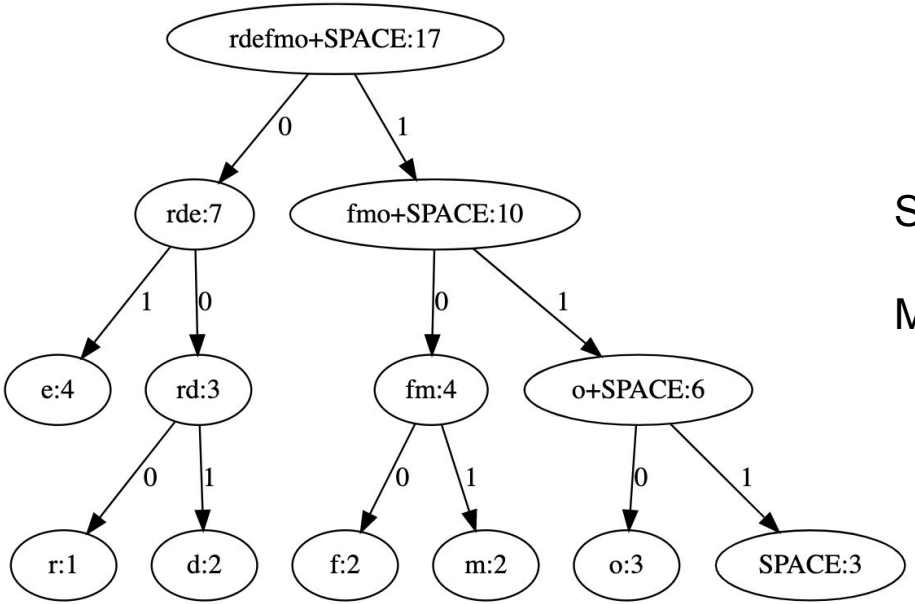
Building the Huffman Tree



SINGLE:

MERGED: rde: 7 | fmo+SPACE: 10

Building the Huffman Tree



SINGLE:

MERGED: rdefmo+SPACE: 17