Huffman code tree - Solution

H1. Short answer:

a) (3pts) Huffman coding is a Dynamic Programming problem. True/False. **False**

H2. Assume that the numbers given below represent counts of letters in the hundreds from a file (similar to the CLRS example). For example, in the file there will be exactly 20 * 100 occurrences of the letter 'a' , 11*100 occurrence of the letter 'c', etc. **a: 20, c:11, d:2, e: 10, o:15, m:8, s:10, t:22, u: 2**

a) What is an optimal Huffman code based on the following set of frequencies?
   1) Draw the tree. Show your work at every step.
   2) Fill in the table on the right the ***Huffman*** encoding for each letter.
   3) We encode the file using the Huffman codes produced above. How much memory will the file require with this encoding?

b) Fixed-length encoding:

   1) Fill in the table the ***fixed-length*** encoding for each letter.
   2) What will be the file size when the fixed-length encoding is used?

Letter	Encoding	
	Huffman	Fixed length
a		
c		
d		
e		
o		
m		
s		
t		
u		

Answer: see next pages.

H3.

A)(5 pts) In 3 to 5 bullet points or SHORT sentences, write the algorithm for building the Huffman tree. Graded both for correctness and CLARITY.

a) ***sort*** pair letter-frequency ***in increasing order of frequencies*** (they will be leaves in the tree)
b) repeat until it is all one tree
   i) ***get the 2 trees with smallest count and make a new tree with them*** as children a new node as the root. ***Value of new node*** = *sum of the connected nodes. The smaller tree will be the left child.*
   ii) ***Reinsert the new tree in the collection*** (put new node in correct place)

Using a MinHeap for the collection of trees: two removeMin operations are done to get the smallest trees for i) and one insert operation for ii)

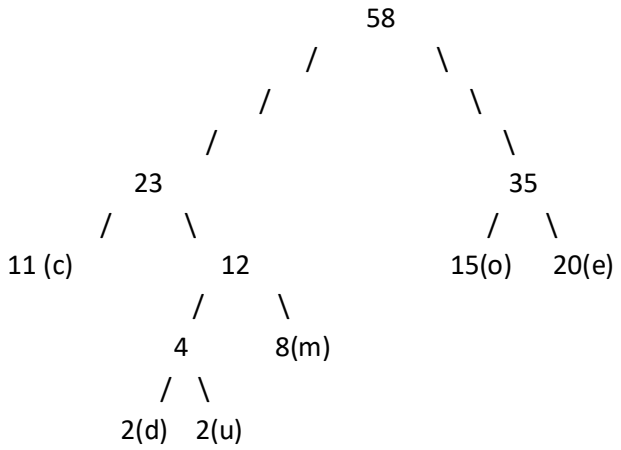
Using a sorted array for the collection of trees: resort the array (or insert in place) at ii). Depending on implementation sliding elements may be needed at step i).

B) What property must Huffman codes have to ensure we can decode the encoded text?

The code for one symbol cannot be a prefix of the code for another symbol.

H4.

a) Given a Huffman tree produce the tree encoding (to store the tree) as done for the homework (0/1/symbol).



001c001d1u1m01o1e

b) Given a tree encoding (0/1/symbol) reconstruct the tree. E.g. given 001c001d1u1m01o1e reconstruct the tree.
The tree from a)

c) Given the Huffman tree from a) and decode the text 0010010011 **code**

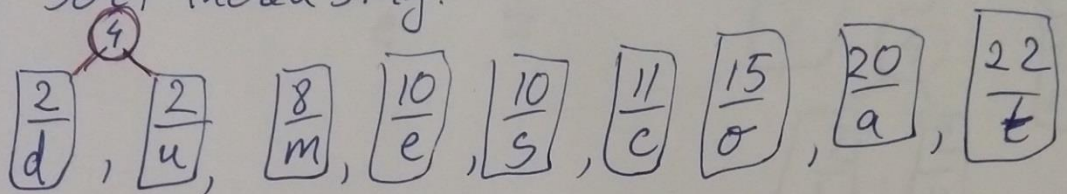
d) Given the Huffman tree from a) encode the text *mood*. **01110100100**

Greedy

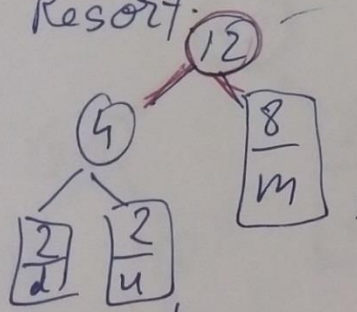
2. Huffman encoding

a:20, c:11, d:2, e:10, o:15, m:8, s:10, t:22, u:2

Sort increasing:

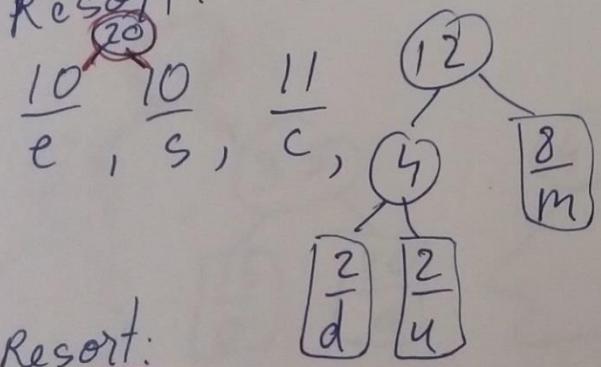


Resort:



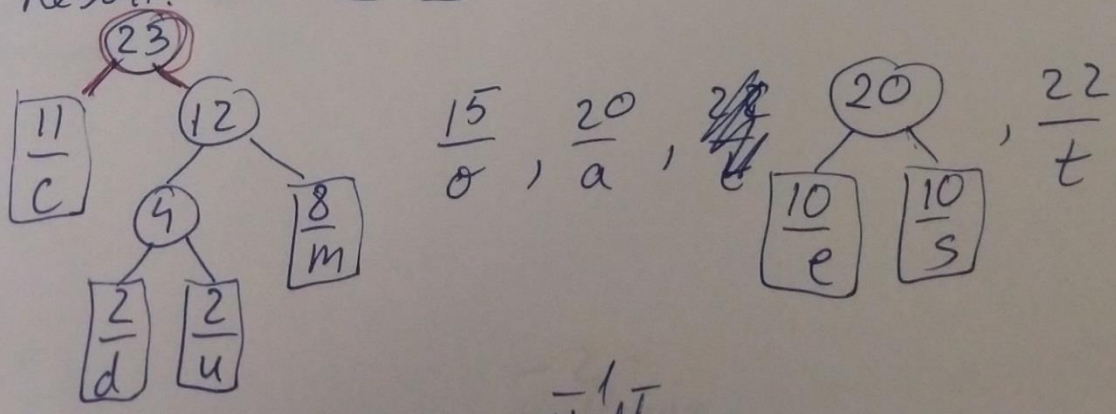
10 e, 10 s, 11 c, 15 o, 20 a, 22 t

Resort:



15 o, 20 a, 22 t

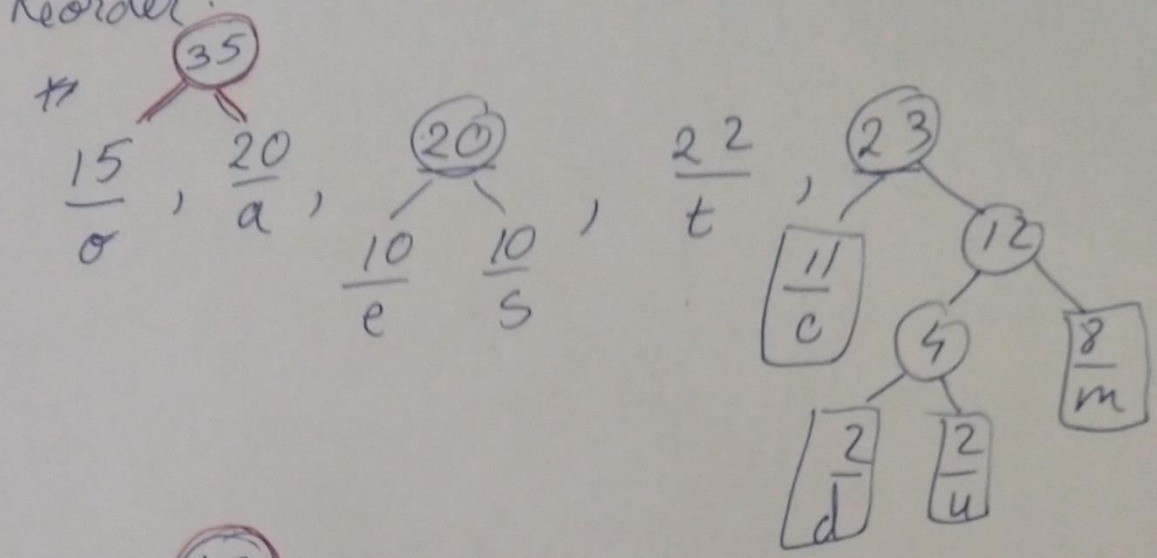
Resort:

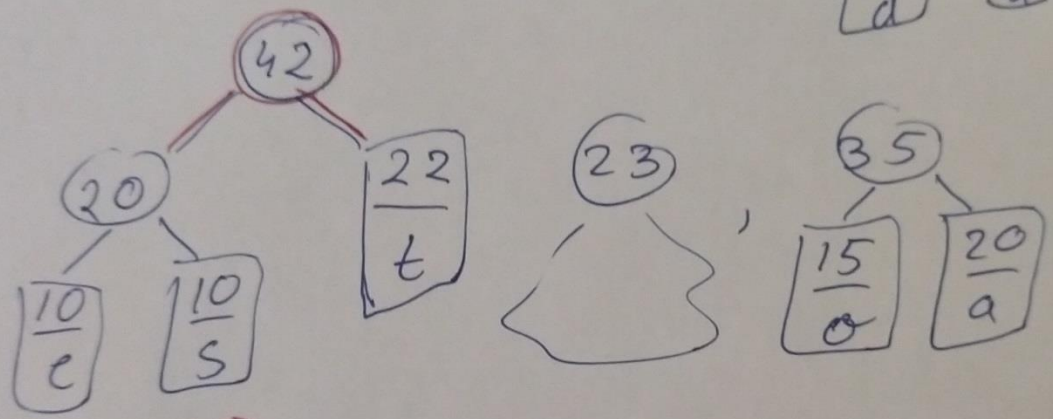


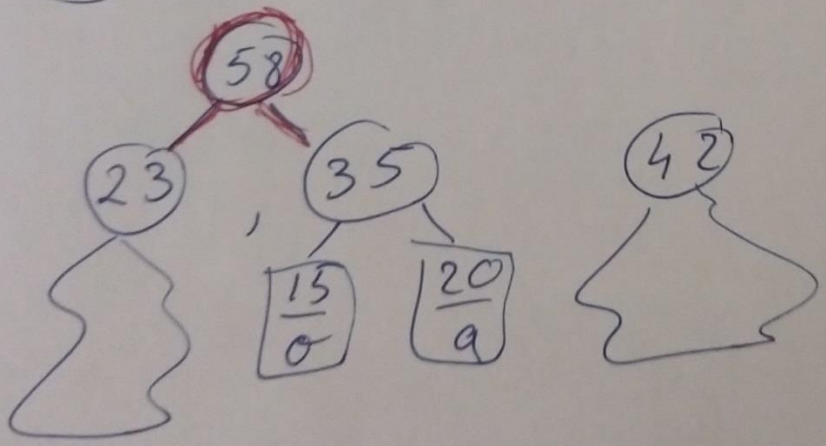
15 o, 20 a, 22 t

Huffman

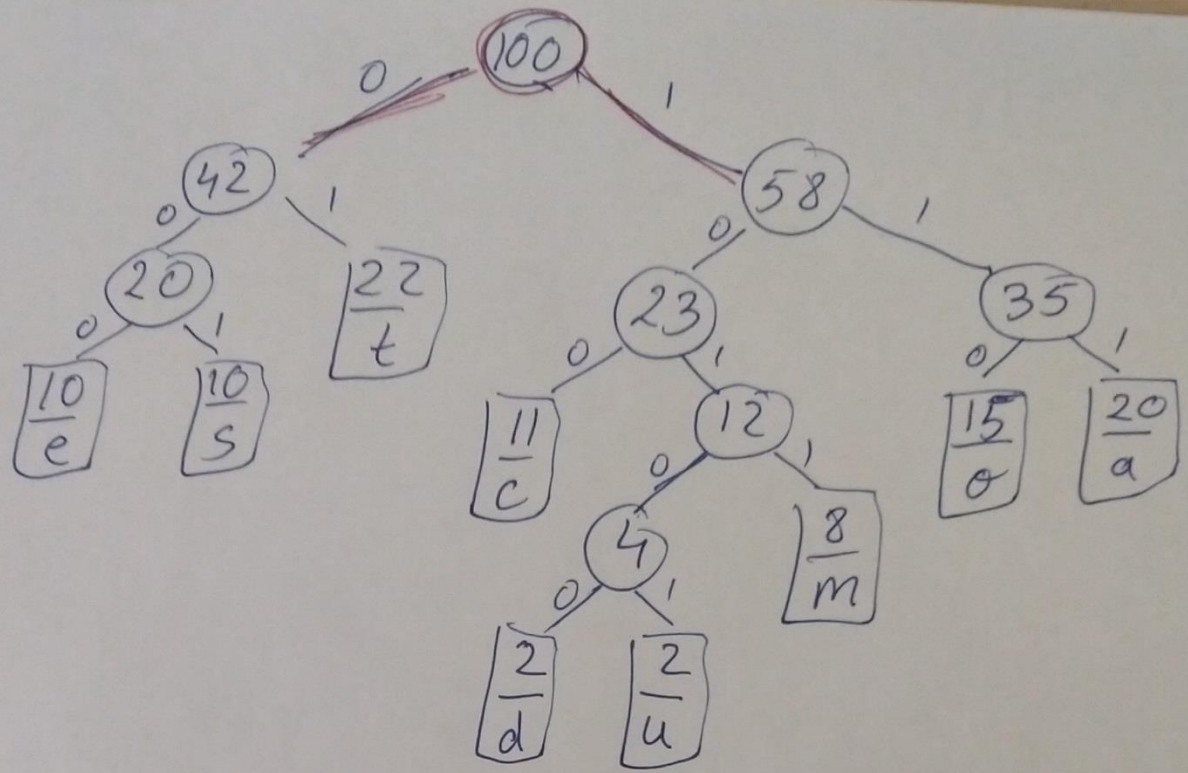
Reorder:







Huffman



Tree is complete. ~~Ref~~

Label 0/1 on branches.

(a) (b) encoding: e : 000 (3bits) => (a) (c)

s : 001

t : 01

c : 100

d : 10100

u : 10101

m : 1011

o : 110

a : 111

Space = $\sum_{\text{letter}}$ count * bits

= 10 · 100 · 3 +
10 · 100 · 3 +
22 · 100 · 2 +
10 · 100 · 3 +
2 · 100 · 5 +
2 · 100 · 5 +
8 · 100 · 4 +
15 · 100 · 3 +
20 · 100 · 3

= 29,400 bits

(Note for 'exe have:
10 · 100 count · 3 bits)

-3
Huffman

(b) Fixed length:

a) 9 symbols => need 9 different encoding
=> need 4 bits

a: 0000
c: 0001
d: 0010
e: 0011
o: 0100
m: 0101
s: 0110
t: 0111
u: 1000

b) Size: $= \left(\sum_{\text{letter}} \text{count}\right) \cdot 4 =$

$= 4(20+11+2+10+15+8+10+22+2) \cdot 100$

$= 4 \cdot 100 \cdot 100 = 40{,}000$ bits

Huffman