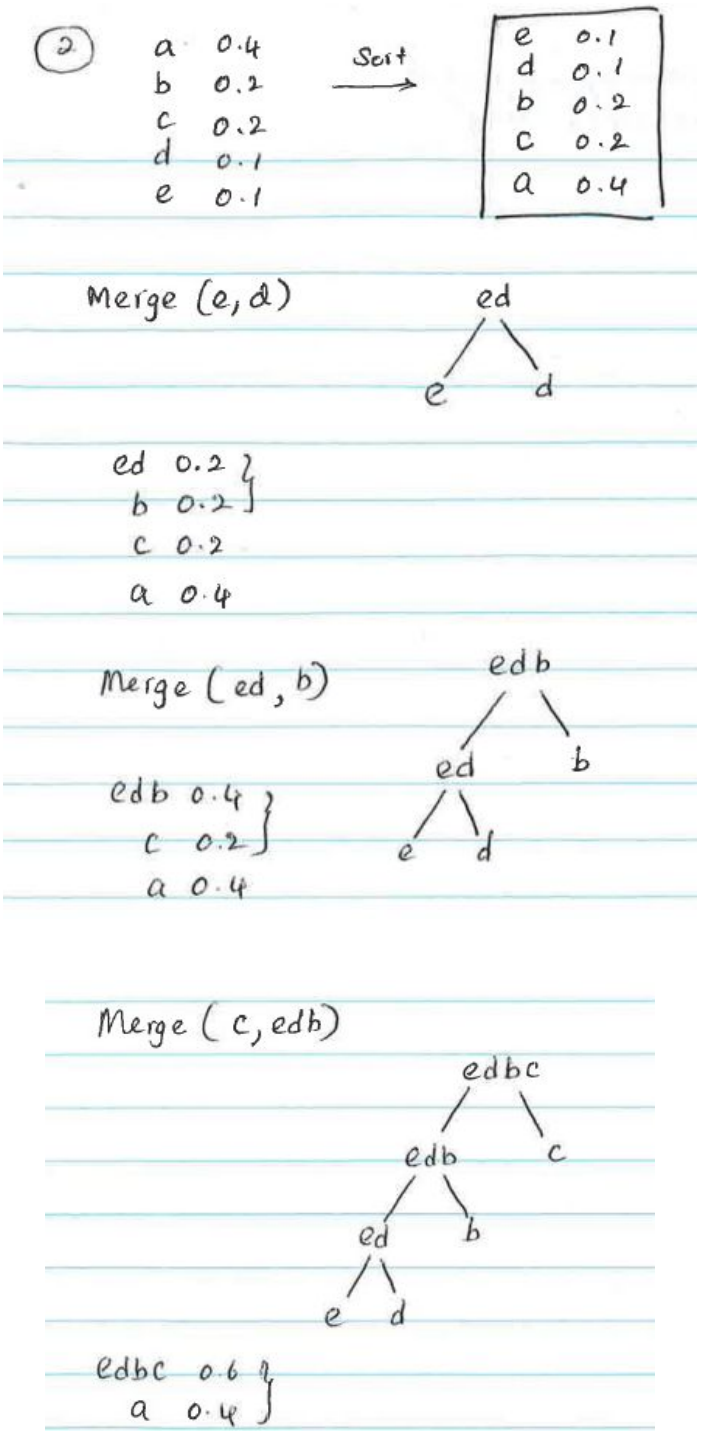
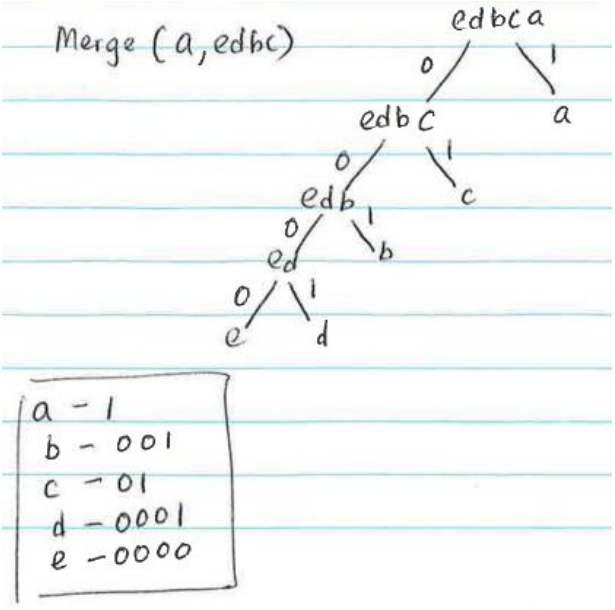
FIT3155: Week 10 Tutorial - Answer Sheet

(Scribe: Dinithi Sumanaweera)

Question 1

Design a Huffman code for a set of characters $\{a, b, c, d, e\}$ with their respective probabilities of $0.4, 0.2, 0.2, 0.1, 0.1$.





Question 2

In the exercise above, you would have noticed that when performing greedy merging of two nodes/subtrees (of lowest probabilities in the current iteration), we encounter multiple choices (ways) to achieve this, and our strategy was to pick any two among the multiple choices rather *arbitrarily*.

A useful **variation** of Huffman coding is to *always* merge **two shortest (in height) subtrees** whenever a multiple choice exists. This gives what is called the *minimum variance* Huffman coding.

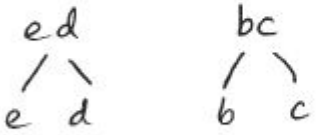
Design a minimum variance Huffman code on the set of characters and probabilities provided above. Compare (i.e., eyeball) the resultant code words between the original method and the this variant.

e 0.1 } ed - 0.2
d 0.1 }
b 0.2
c 0.2
a 0.4

Merge (e,d)

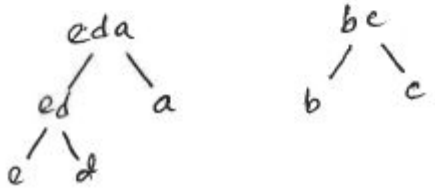
ed - 0.2
b - 0.2 } bc - 0.4
c - 0.2 }
a - 0.4

Merge(b,c)



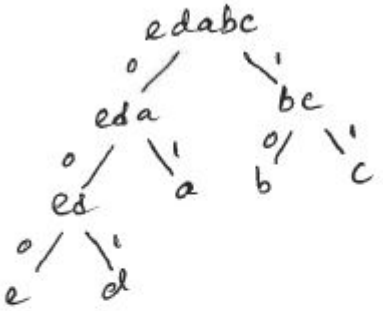
ed - 0.2
bc - 0.4 } eda - 0.6
a - 0.4

Merge (ed , a)



bc - 0.4
eda - 0.6

Merge (bc, eda)



a	=	01
b	=	10
c	=	11
d	=	001
e	=	000

Question 3

Does Huffman encoding always yield a prefix-free code words for the characters it is encoding? If yes, why? If no, why not?

Yes. Since the huffman tree is a binary tree with characters being at the leaves rather than at intermediary nodes, each path from root to a character can be uniquely represented by a binary code. Therefore no such binary code becomes a prefix of another.

Question 4

Encode the following sequence into ⟨offset, length, character⟩ triples using LZ77 algorithm:

barrayar_bar_by_barrayar_bay

Assume the search window size and lookahead buffer sizes are both 15.

LZ77

Search Window size = Lookahead buffer = 15

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28	⟨offset, length, char⟩
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(-,0,b)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(0,0,a)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(0,0,r)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(1,1,a)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(0,0,y)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(5,2,-)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(9,3,-)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(4,1,y)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(7,4,r)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(3,1,y)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(12,4,a)
b a r r a y a r - b a r - b y - b a r r a y a r - b a y	(6,1,$)

Question 5

Decode the encoded triples from above, to recover back the original text

Follow the same decoding procedure as explained next for Question 7

Question 6

A text is encoded using the LZ77 algorithm, which yields the following sequence of triples:

$\langle 0,0,r\rangle\ \langle 0,0,a\rangle\ \langle 0,0,t\rangle\ \langle 2,8,\_\rangle\ \langle 3,1,\_\rangle\ \langle 0,0,r\rangle\ \langle 6,4,t\rangle\ \langle 9,5,t\rangle$

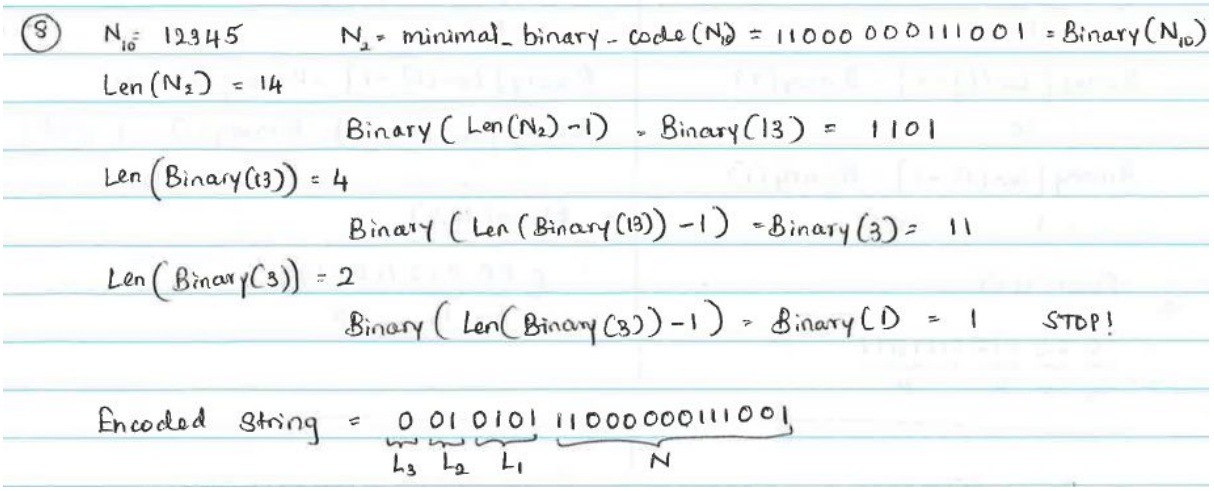
Assume the sizes of the search window and lookahead buffer are both 10.

		Index	12345678901234567890
			ratatatatat_a_rat_at
			Decoded string S so far
Decode(<0,0,r>)	- write new char r		r
Decode(<0,0,a>)	- write new char a		ra
Decode(<0,0,t>)	- write new char t		rat
Decode(<2,8,_>)	- copy S[2] and concat		rata
	- copy S[3] and concat		ratat
	- copy S[4] and concat		ratata
	- copy S[5] and concat		ratatat
	- copy S[6] and concat		ratatata
	- copy S[7] and concat		ratatatat
	- copy S[8] and concat		ratatatata
	- copy S[9] and concat		ratatatatat
	- write new char _		ratatatatat_
Decode(<3,1,_>)	- copy S[10] and concat		ratatatatat_a
	- write new char _		ratatatatat_a_
Decode(<0,0,r>)	- write new char r		ratatatatat_a_r
Decode(<6,4,t>)	- copy S[10] and concat		ratatatatat_a_ra
	- copy S[11] and concat		ratatatatat_a_rat
	- copy S[12] and concat		ratatatatat_a_rat_
	- copy S[13] and concat		ratatatatat_a_rat_a
	- write new char t		ratatatatat_a_rat_at
Decode(<9,5,t>)	- copy S[12] and concat		ratatatatat_a_rat_at_
	- copy S[13] and concat		ratatatatat_a_rat_at_a
	- copy S[14] and concat		ratatatatat_a_rat_at_a_
	- copy S[15] and concat		ratatatatat_a_rat_at_a_r
	- copy S[16] and concat		ratatatatat_a_rat_at_a_ra
	- write new char t		ratatatatat_a_rat_at_a_rat

ratatatatat_a_rat_at_a_rat

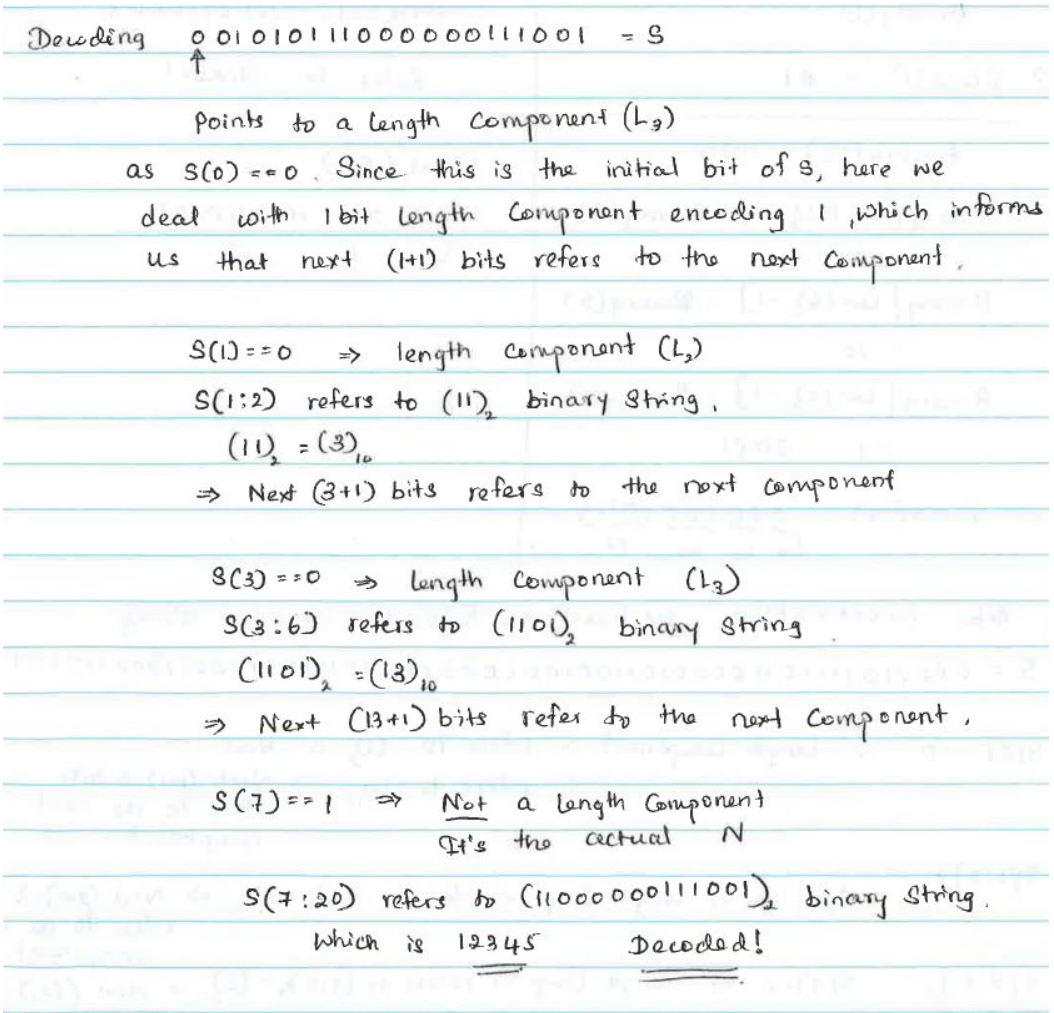
Question 7

Encode N = 12345 using the Elias variable-length encoding of integers



Question 8

Decode the above encoding to recover back N = 12345



Question 9

Encode, using the Elias variable-length encoding of integers, following sequence of integers:
123,100,1,23,561

Binary (123)
$= (123)_2 = 1111011$

Binary (100)
$= (100)_2 = 1100100$

Binary $[\text{Len}(123)_2 - 1]$ = Binary (7 - 1)
= 110
Binary $[\text{Len}(6)_2 - 1]$ = Binary (2)
= 10
Binary $[\text{len}(2)_2 - 1]$ = Binary (1)
= 1 STOP!

⇒ Elias (123)
= 0 00 010 1111011
L_3 L_2 L_1 N

Binary $[\text{len}(100)_2 - 1]$ = Binary (6)
= 110
Binary $[\text{Len}(6)_2 - 1]$ = Binary (2) = 10
Binary $[\text{len}(2)_2 - 1]$ = Binary (1) = 1 STOP!

⇒ Elias (100)
= 0 00 010 1100100
L_3 L_2 L_1 N

Binary (1) = 1

⇒ Elias (1) = 0

Binary (23) = 10111
Binary $[\text{Len}(23)_2 - 1]$ = Binary (4)
= 100
Binary $[\text{Len}(4)_2 - 1]$ = Binary (2)
= 10
Binary $[\text{Len}(2)_2 - 1]$ = Binary (1)
= 1 STOP!

⇒ Elias (23) = 0 00 000 10111
L_3 L_2 L_1 N

Binary (561) = 1000110001

Refer to Slides!

⇒ Elias (561)
= 0 01 0001 1000110001
L_3 L_2 L_1 N

Question 10

Concatenate the variable length binary encodings of the above sequence, and decode this bit string to recover back the full sequence of integers

S = 000010111101100001011001001000000101110010001000110001

S[0] == 0 ⇒ Length Component ⇒ refers to $(1)_2$ ⇒ Next
⇒ refers to $(1)_{10}$ ⇒ Next (1+1)=2 bits refers to the next component

S[1:2] S[1] == 0 ⇒ Length Comp. ⇒ refers to $(10)_2 = (2)_{10}$ ⇒ Next (2+1)=3 bits refers to the next component

S[3:5] S[3] == 0 ⇒ Length Comp. ⇒ refers to $(110)_2 = (6)_{10}$ ⇒ Next (6+1)=7 bits

S[6:12] S[6] == 1 ⇒ Int Comp ⇒ $(1111011)_2 = 123$ Decoded!

Contd...