ADS LAB 5 LAB REPORT

Elsayed, Ahmed

MEMBER B TEAM 15

Problem 12:

1. Since one has 16 equally probably symbols, the probability of occurrence of each is P(x) = 0.0625 and consequently the Shannon's entropy theorem to such probable functions is to be calculated in the following method:

$$H(x) = -\sum_{i=1}^{n} P(x_i) * log_2 P(x_i)$$
$$H(x) = -(P(x) * log_2 P(x) * n)$$
$$H(x) = -(0.0625 * log_2 0.0625 * 16)$$
$$H(x) = 4$$

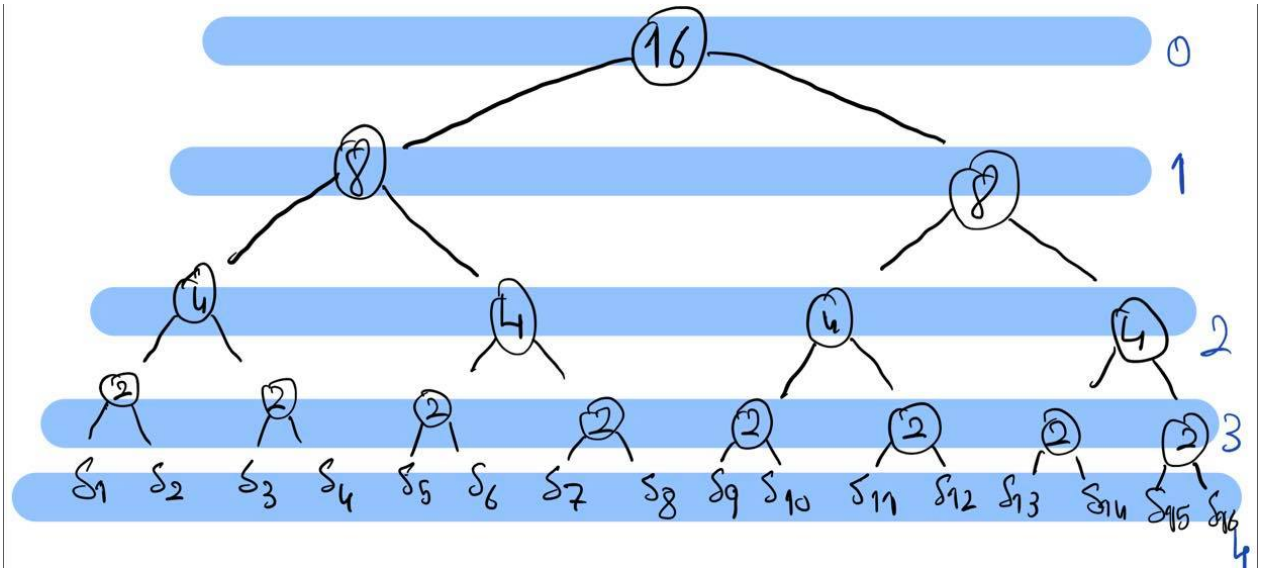
Now it's important to have a good understanding of such result. The previous result means that for such a provided set of symbols, the average number of bits needed to encode a single symbol of them is 4 + 1 which is the average node depth of the tree and also represents the average number of operations (traversals) needed in order to reach an arbitrary symbol in the tree. Now the question is how does this reflect on the coding tree? Well it's quite simple. Since all the symbols are equally probable, the coding tree is generated in a way that all the symbols are on the same depth.

2. This time, one must count in the different probability of the 2 different symbols and then count in the probability of the other 14 equally probable symbols.

$$H(x) = -\sum_{i=1}^{n} P(x_i) * log_2 P(x_i)$$
$$H(x) = -(P(x) * log_2 P(x) * 2 + P(x) * log_2 P(x) * 14)$$
$$H(x) = -(0.22 * log_2 0.22 * 2 + 0.04 * log_2 0.04 * 14)$$
$$H(x) = 3.56$$

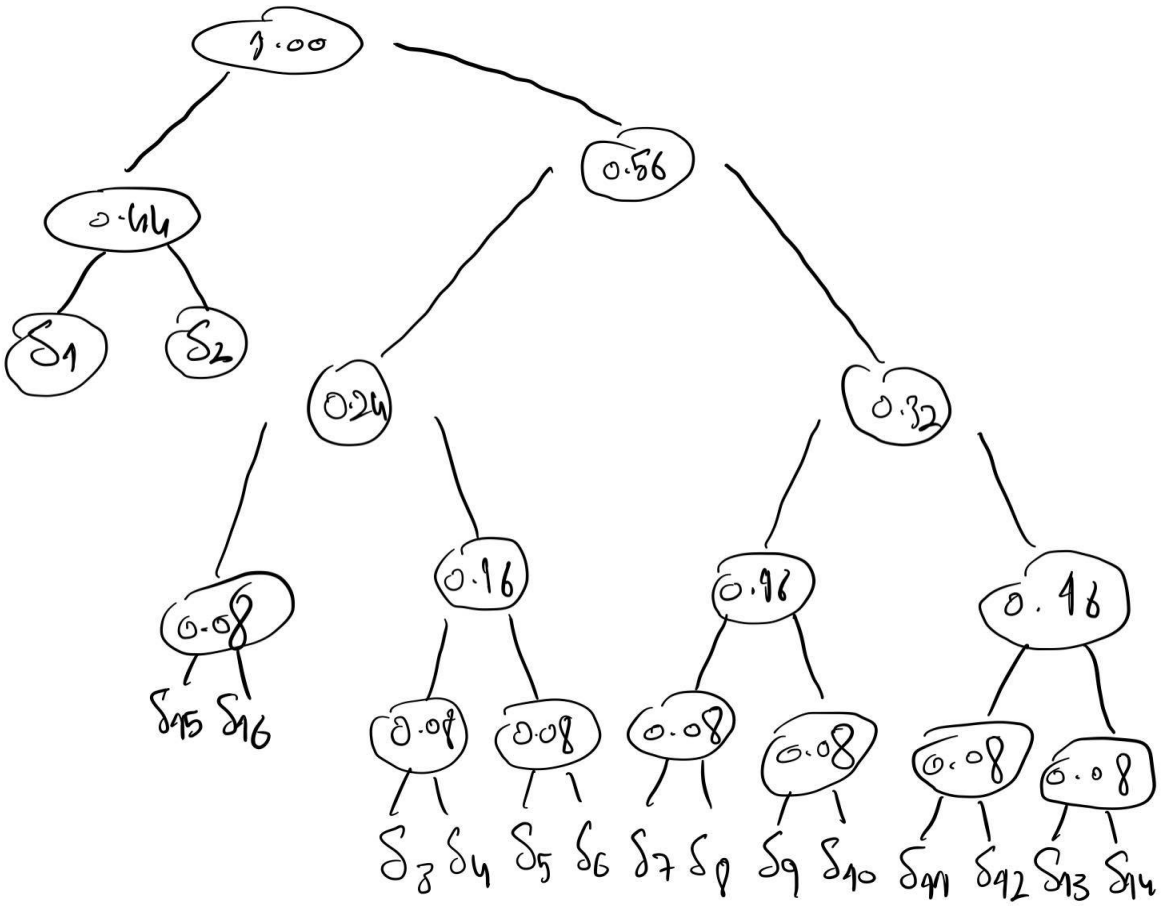
The result of the entropy in the information compression in this case is 3.56 + 1. This means the average of the number of bits needed to encode such set of symbols is 4.56 bits and this can never be possible since bits are a whole (undividable) so the set will be encoded into either 4 or 5 bits for each symbol.

3. The following schematic shows the proposed construction for a 16- equally probable symbol encoding tree according to the Huffman greedy algorithm:



Through the previous picture, one can see that each symbol takes 4 bits (4 traversals) to be encoded.

The following schematic shows the proposed construction for a 16-symbol set with 2-equally probable symbols and 14 with lower probability than the previous 2 but equally probable and that is done according to the Huffman greedy algorithm:



Through the last picture, one can know for a fact that the previous set can never be encoded into 3.56 bits like it's mentioned bits are undividable so it's clear now that the AVERAGE of such bits' encoding is 4.56 but none of the symbols actually lie on a 4.56 depth of the tree because there isn't such depth.

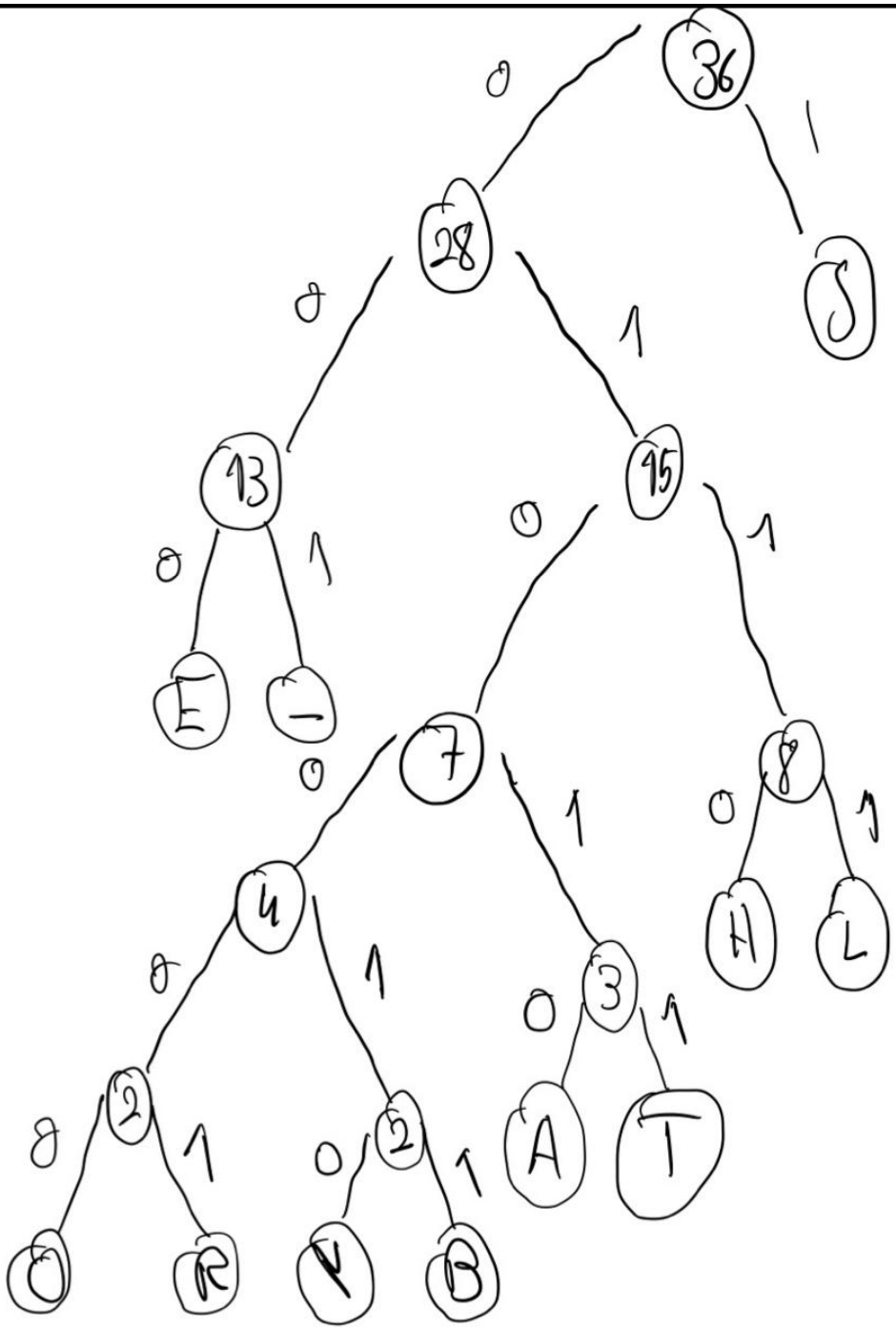
At the 3rd depth of the tree, it's visible that there is an empty right node, this has occurred in this case due to having the non-optimal case of the coding sequence of the Huffman Trie[1]

[1] More information on that can be found through the following link, retrieved on 31/12/20.

4. s1= "she_sells_sea_shells_by_the_seashore". This string is going to be considered for each letter of it and how much its frequency is:

Letter	Occurrence Frequency
S	8
E	7
_	6
L	4
H	4
A	2
B	1
T	1
O	1
R	1
Y	1

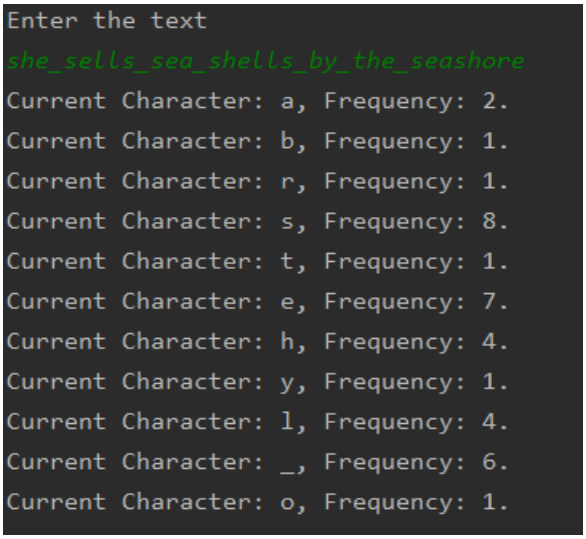
The tree can be constructed in the following way:



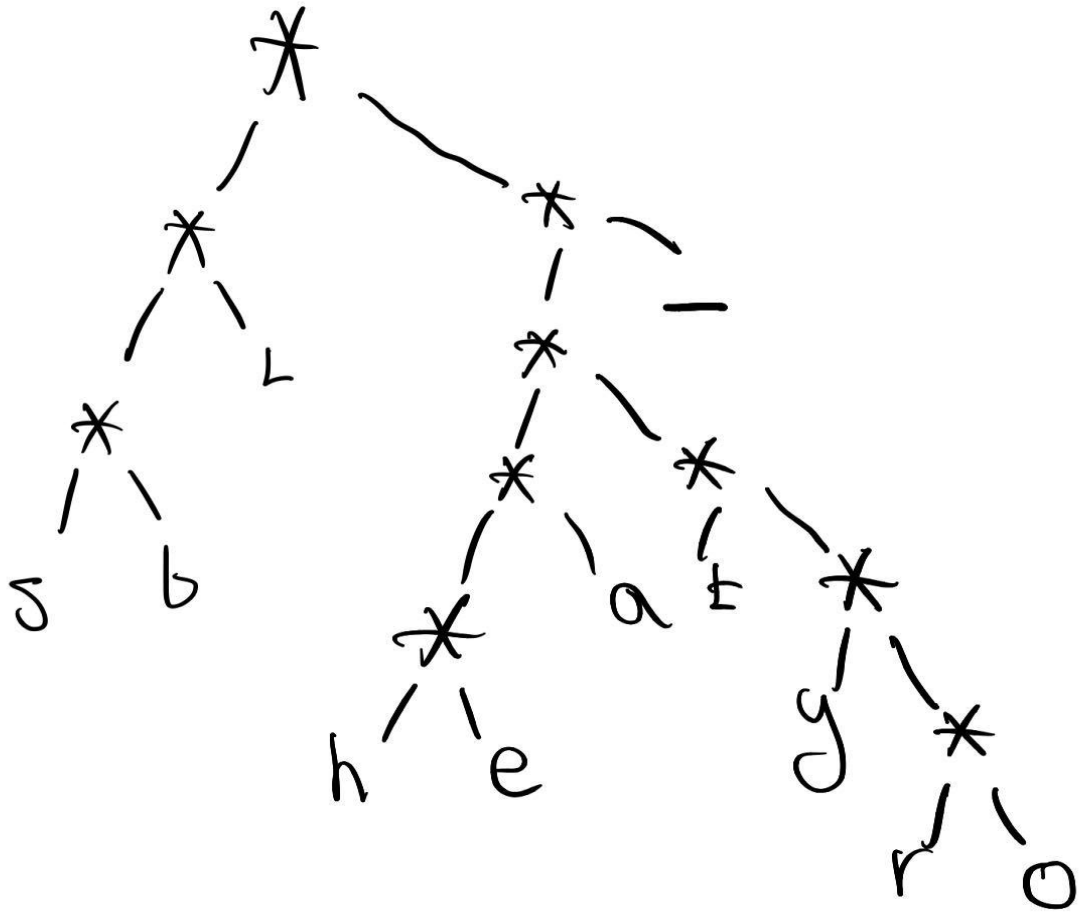
5. The number of encoded bits of S1 can be calculated as follows:
   # of bits = 8*1 + 3*7 + 3*6 + 4*4*2 + 5*2 + 5*1 + 6*1*4= 118 bits

   The number of Zeros in the encoded string will be 69, The number of Ones in the encoded string will be 49.

6. For more information on the implementation, please check out the attached source code. The following screenshot shows that the frequency test was successful:



7. The pre-order traversal the tree will go as follows:

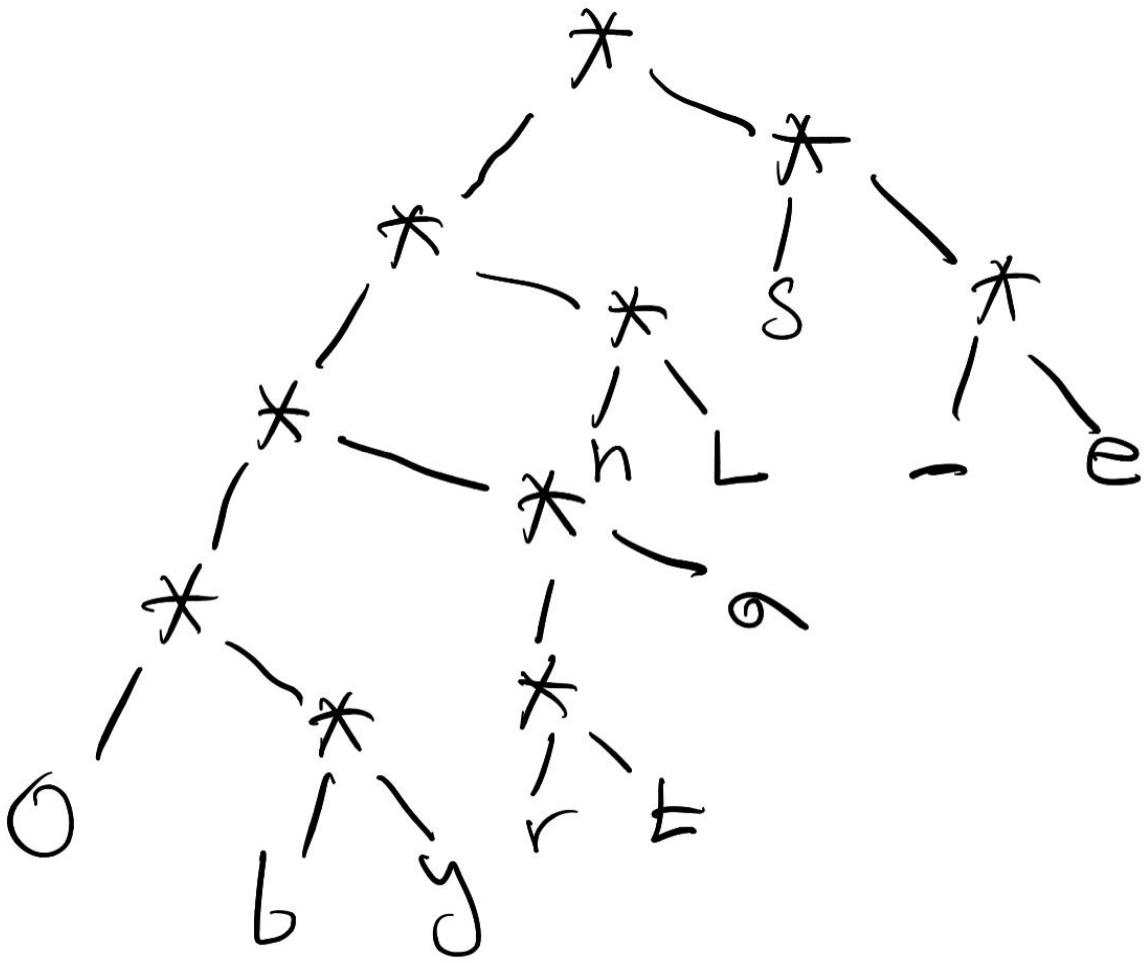


Report

Problem 14

1. For more information on the implementation of the method, please check out the method `treeEncoding(Node node)` in the attached source code.
   The two following screenshots show the pre-order traversal of the resulted encoded Huffman trie and the tree visualization of such traversal as a test. Both are done for the following string:
   “she_sells_sea_shells_by_the_seashore”.

```
Current Character: a, Frequency: 2.
Current Character: b, Frequency: 1.
Current Character: r, Frequency: 1.
Current Character: s, Frequency: 8.
Current Character: t, Frequency: 1.
Current Character: e, Frequency: 7.
Current Character: h, Frequency: 4.
Current Character: y, Frequency: 1.
Current Character: l, Frequency: 4.
Current Character: _, Frequency: 6.
Current Character: o, Frequency: 1.
****o*by**rta*hl*s*_e
```

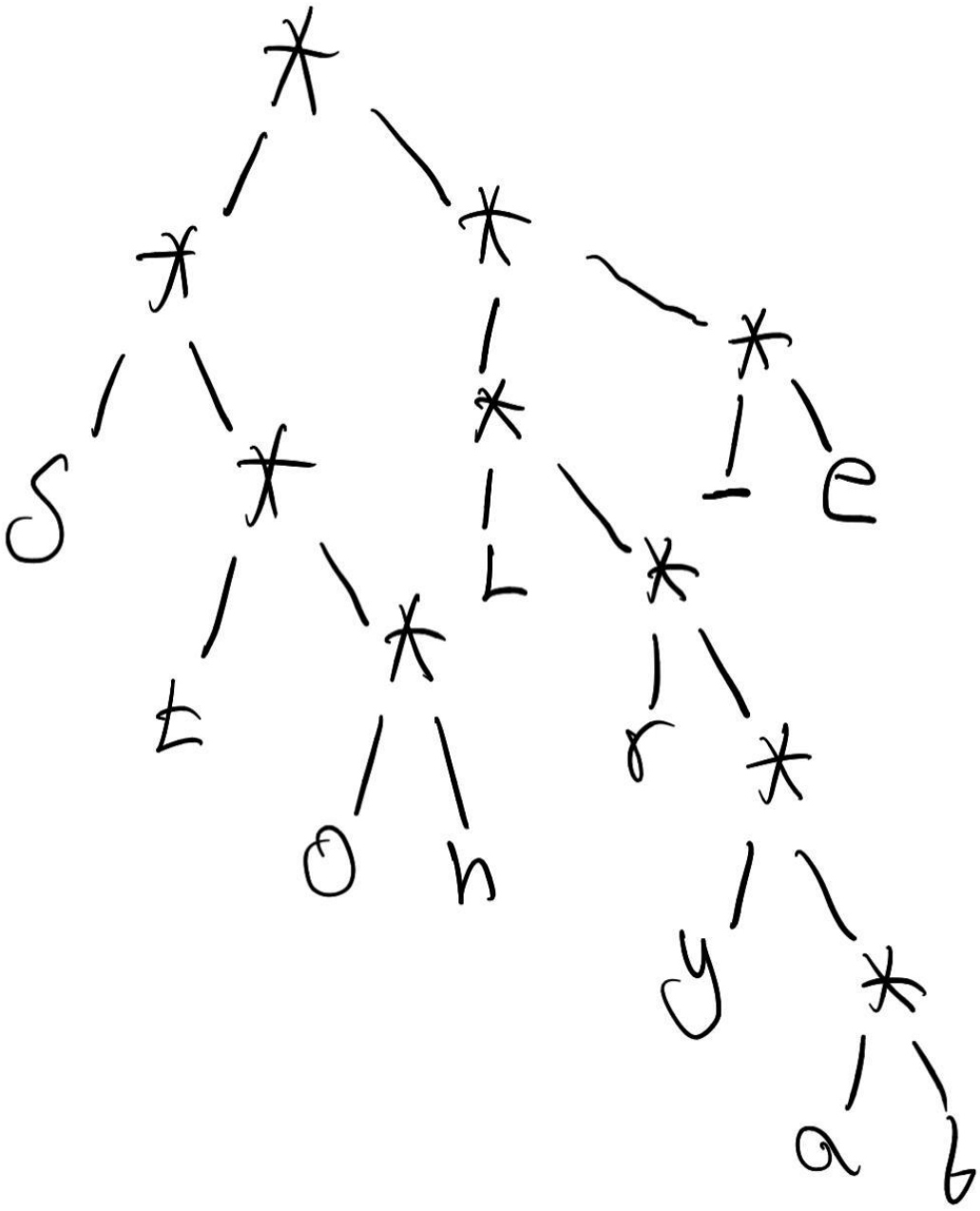


Through the previous tree visualization, it can be clearer that the lower frequency nodes are at the very end of the tree (which is reasonable since we are using Huffman greedy algorithm). This shows that the resulted string which is **"****o*by**rta*hl*s*_e"** is a reasonable tree result for the pre-order traversal of the Huffman tree.

2. In comparison with preparation task 7, one can see that in this tree representation the frequency of the occurring characters is well considered, specially in representing the encoded sequence of each character.
3. For more information on the implementation, please check out the attached source code to this PDF. The following screenshot shows the frequencies of occurrence of the characters of the following string: **"selly_sells_her_shorts_by_the_seattle_store"**:

```
Current Character: a, Frequency: 1.
Current Character: r, Frequency: 3.
Current Character: b, Frequency: 1.
Current Character: s, Frequency: 7.
Current Character: t, Frequency: 5.
Current Character: e, Frequency: 7.
Current Character: h, Frequency: 3.
Current Character: y, Frequency: 2.
Current Character: l, Frequency: 5.
Current Character: _, Frequency: 7.
Current Character: o, Frequency: 2.
**s*t*oh**l*r*y*ab*_e
```

4. The generated tree for the previously mentioned string can be interpreted through a string as the following: **“**s*t*oh**l*r*y*ab*_e”,** the tree is constructed in the following matter using pre-order traversal as follows:



5. The encoded symbols for both Strings can be deduced as follows

Character	Encoded Symbol
S	10
E	111
_	110
L	011
H	010

A	0011
B	00010
T	00101
O	0000
R	00100
Y	00011

S1 = "***she_sells_sea_shells_by_the_seashore***" can be represented as an encoded string in the following way:
"10010111110101110110111011010111001111010010111011011100001000011110001010101111101011100111001000000100111"

# of bits = 111 bits

Character	Encoded Symbol
S	00
E	111
_	110
L	100
H	0111
A	101110
B	101111
T	010
O	0110
R	1010
Y	10110

S2 = "***selly_sells_her_shorts_by_the_seattle_store***" can be represented as an encoded string in the following way:
"00111100100101101100011110010000110011111110101100001110110101001000110101111101101100100111111110001111011100100101001111100001001101010111"

# of bits = 140 bits

References:

1. ADS Lecture 8 Sedgwick Coding
2. ADS Lecture 8 Notes, Author: Prof. Wolfgang Renz
3. Shannon's Information and Data Compression Entropy
4. Minimum Priority Queue Implementation