Lab4 Report

學號：b08502141　　姓名：石旻翰　　系級：電機四

1 Problem1

1.1 Calculate the entropy of X

$$H(x) = -\sum_{i \in X} p_i log_2(p_i) = 2.53 \qquad [1.1]$$

1.2 Construct the Huffman tree and the Huffman dictionary for X.

Huffman_dict is shown in Table 1, and Huffman_tree is shown in Fig 1

Node	p	Left	Right	Bits
a	0.2			00
b	0.05			10000
c	0.005			100010
d	0.2			01
e	0.3			11
f	0.05			1001
g	0.045			100011
h	0.15			101
cg	0.05	c	g	10001
bf	0.1	b	f	1000
cgbf	0.15	cg	bf	100
cgbfh	0.3	cgbf	h	10
ad	0.4	a	d	0
cgbfhe	0.6	cgbfh	e	1
adcgbfhe	1	ad	cgbfhe	

Table 1: Huffman Dictionary

1.3 Verify whether the codewords constructed by your Huffman tree satisfy the Kraft inequality or not.

By the following equation, we can verify that my Huffman tree satisfies Kraft inequality.

$$\sum_{j=1}^{M} 2^{-l(a_j)} = 3 \cdot 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2 \cdot 2^{-6} = 1 \leq 1 \qquad [1.2]$$

1.4 Find the average codewords length L for the dictionaries. Do they satisfy the source-coding theorem?

$$L = \sum_{j=1}^{8} p(a_j) l(a_j) = 2.6 \qquad [1.3]$$

Thus, by $H(X) \leq 2.6 \leq H(X) + 1$, we can verify that it satisfies the source-coding theorem.

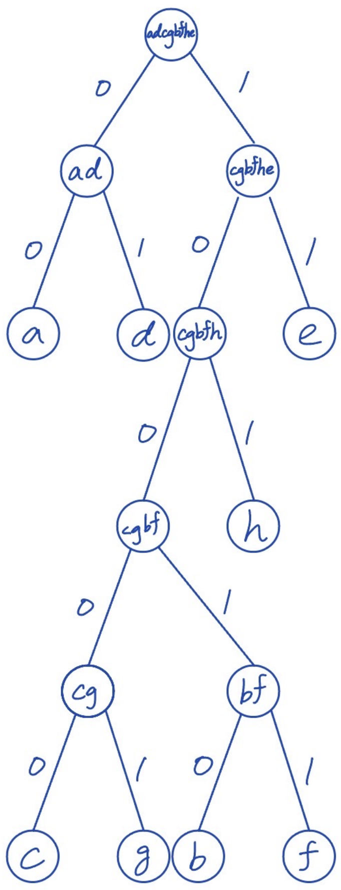


Figure 1: Huffman tree

1.5 Encode the sequence of symbols {g, a, c, a, b}.

$$\{g, a, c, a, b\} \xrightarrow{Enc.} 0000110000001000010 \qquad [1.4]$$

1.6 Decode the bitstream

$$0000110000001000010 \xrightarrow{Dec.} gacab \qquad [1.5]$$

1.7 Let T_ε^n denote the typical set of X with ε = 0.1 and n = 10. Find 10 members in the set T_ε^n.

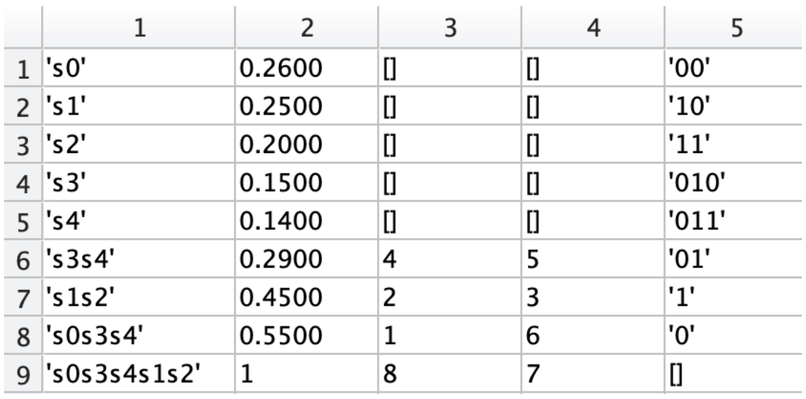
Following is the ten elements of T_ε^n when $\varepsilon = 0.1, n = 10$. This is resulted from random sampling in matlab, and by random sampling 1000 times, the probability of samples in T_ε^n is about 26%.

$$T_\varepsilon^n = \{\text{ghaheadehh, eaghefedea, eadaagedha, ddffaedeee, dgeehdfede,}\\ \text{bdgaaaeeed, dhdahhdadd, adfaheeebe, ahhfaddeae, aeaaeaggea}\}$$

2 Problem2

For more details, you can run the code called lab4.m

2.1 Print the Hufman dictionary dict associated with the symbols and the probabilities

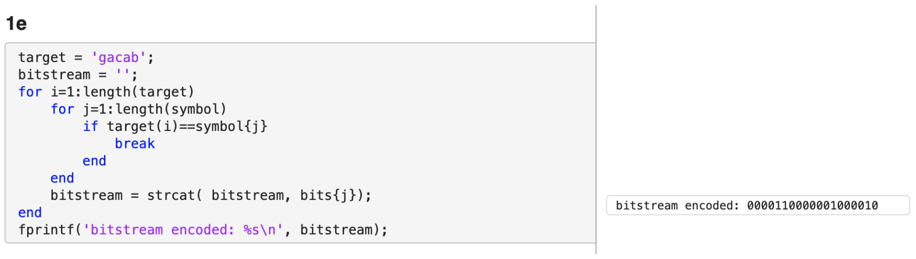


	1	2	3	4	5
1	's0'	0.2600	[]	[]	'00'
2	's1'	0.2500	[]	[]	'10'
3	's2'	0.2000	[]	[]	'11'
4	's3'	0.1500	[]	[]	'010'
5	's4'	0.1400	[]	[]	'011'
6	's3s4'	0.2900	4	5	'01'
7	's1s2'	0.4500	2	3	'1'
8	's0s3s4'	0.5500	1	6	'0'
9	's0s3s4s1s2'	1	8	7	[]

Figure 2: Huffman dictionary

2.2 Implement a Huffman encoding function and verify your results with Problem 1.5.

From Fig.3 and Fig. 4, we can see the result is the same as that in Problem 1.5.

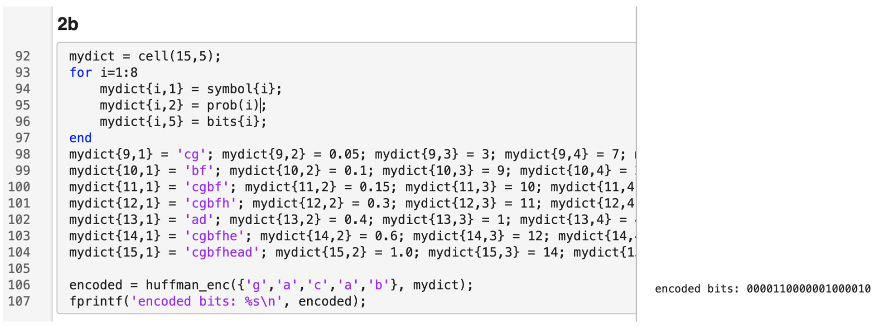


1e

```
target = 'gacab';
bitstream = '';
for i=1:length(target)
    for j=1:length(symbol)
        if target(i)==symbol{j}
            break
        end
    end
    bitstream = strcat( bitstream, bits{j});
end
fprintf('bitstream encoded: %s\n', bitstream);
```

```
bitstream encoded: 0000110000001000010
```

Figure 3: Original Huffman Encoding



2b

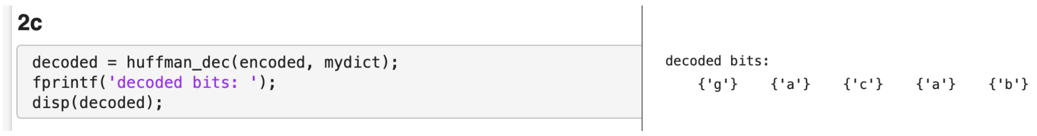
```
mydict = cell(15,5);
for i=1:8
    mydict{i,1} = symbol{i};
    mydict{i,2} = prob(i);
    mydict{i,5} = bits{i};
end
mydict{9,1} = 'cg'; mydict{9,2} = 0.05; mydict{9,3} = 3; mydict{9,4} = 7;
mydict{10,1} = 'bf'; mydict{10,2} = 0.1; mydict{10,3} = 9; mydict{10,4} =
mydict{11,1} = 'cgbf'; mydict{11,2} = 0.15; mydict{11,3} = 10; mydict{11,4
mydict{12,1} = 'cgbfh'; mydict{12,2} = 0.3; mydict{12,3} = 11; mydict{12,4
mydict{13,1} = 'ad'; mydict{13,2} = 0.4; mydict{13,3} = 1; mydict{13,4} =
mydict{14,1} = 'cgbfhe'; mydict{14,2} = 0.6; mydict{14,3} = 12; mydict{14,
mydict{15,1} = 'cgbfhead'; mydict{15,2} = 1.0; mydict{15,3} = 14; mydict{1

encoded = huffman_enc({'g','a','c','a','b'}, mydict);
fprintf('encoded bits: %s\n', encoded);
```

```
encoded bits: 0000110000001000010
```

Figure 4: My Huffman Encoding

2.3 Implement a Huffman decoder function and verify your results with Problem 1.6.

From Fig.5, we can see the result is the same as that in Problem 1.6.



2c

```
decoded = huffman_dec(encoded, mydict);
fprintf('decoded bits: ');
disp(decoded);
```

```
decoded bits:
    {'g'}    {'a'}    {'c'}    {'a'}    {'b'}
```

Figure 5: Huffman Decoding

3 Problem3

3.1 Generate a sequence of n = 10 symbols according to the probability.

1. Symbol sequence: ahhedahaad (randomly generated)
2. Binary data: 10001001011110001101011 (encode)
3. Binary data length: 23

3.2 Plot the histogram of $L_n^{(1)}, L_n^{(2)}, ..., L_n^{(R)}$ and indicate the mean (denoted by $L_n^{(R)}$) in the title

In Fig. 6, we can see that $L_{10}^{200} = 25.815$, which is very close to $n\bar{L} = 26$.

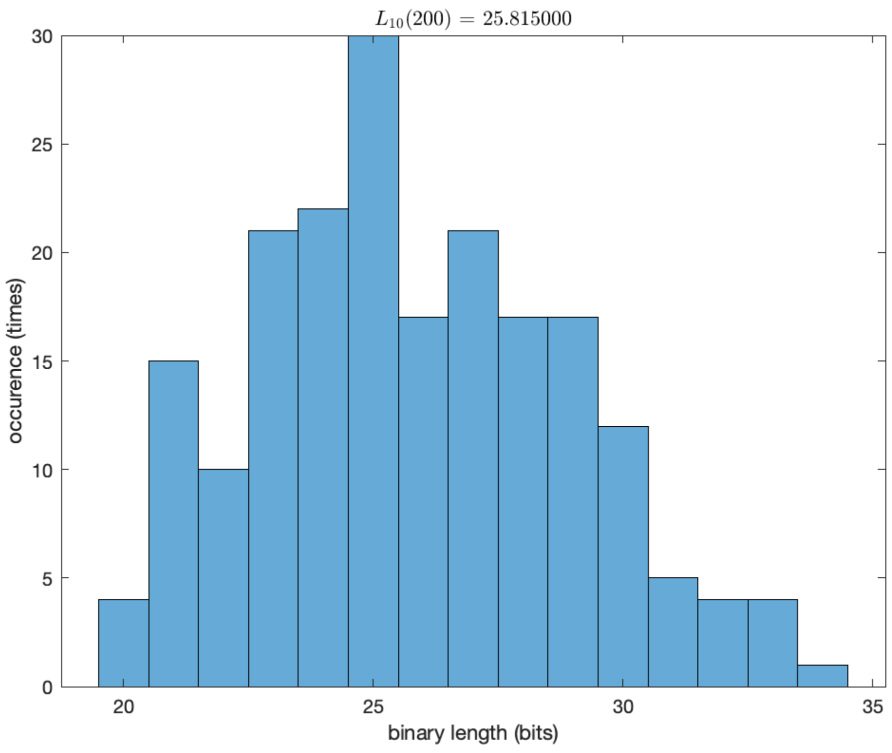


Figure 6: Histogram of $L_n^{(1)}, L_n^{(2)}, ..., L_n^{(R)}$

3.3 Plot $\bar{L}(R)$ versus R with different n

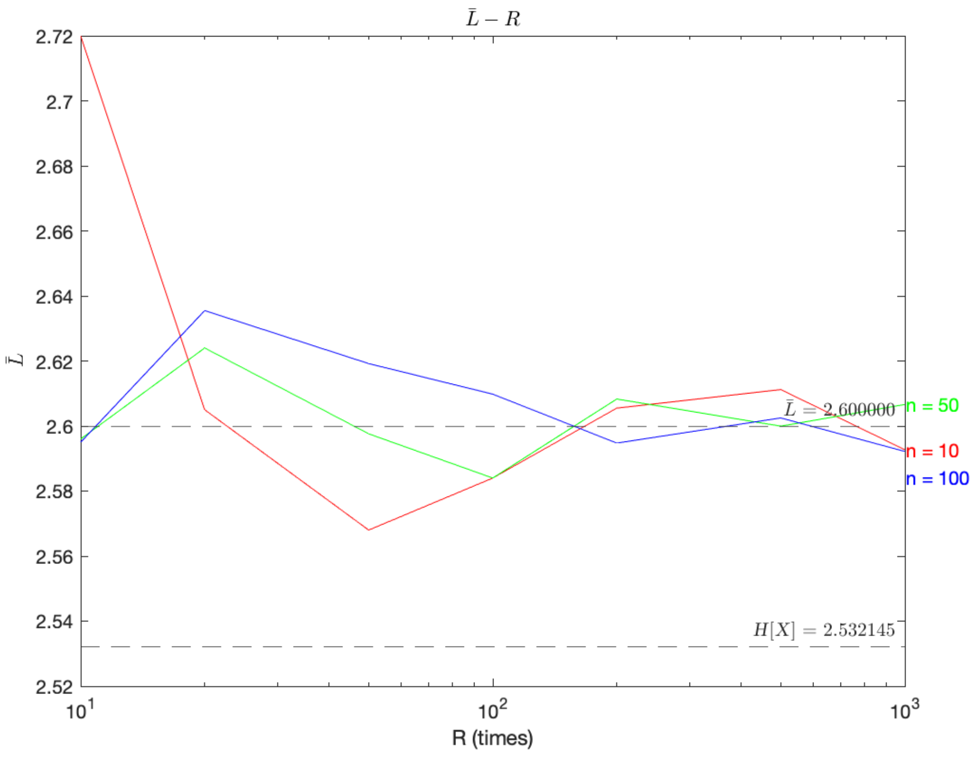


Figure 7: $\bar{L}(R)$ versus R with different n

3.4 Comment

From Fig.7, we can see:
1. In each case, when R is large enough, the average would be close to the expected value($\bar{L}$) which is larger than H(X) by a little bit.
2. In each case, the value of $\bar{L}(R)$ would not lower than the value of H(X).

4 Bonus

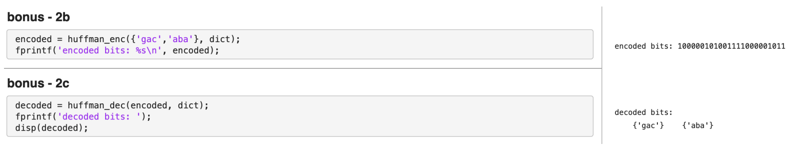
將所有symbol均改爲三個一組，可以視爲是一組長度爲三倍長的新的symbol，因此我沿用了之前implement的function，但此時symbol數量變爲8^3個。

4.1 Print the Hufman dictionary dict associated with the symbols and the probabilities

For the length of dict is quite long, for more details, you can run the code called lab4.m, and the corresponding block is bonus2-a.

4.2 Encode and Decode

Because the length of message must be the multiple of 3, I change the message from ’gacab’ into ’gacaba’. From Fig.8, we can see the encode binary sequence is 100000101001111000001011, whose length is 24, and the decode function gets the correct result as well.



bonus - 2b

```
encoded = huffman_enc({'gac','aba'}, dict);
fprintf('encoded bits: %s\n', encoded);
```

```
encoded bits: 100000101001111000001011
```

bonus - 2c

```
decoded = huffman_dec(encoded, dict);
fprintf('decoded bits: ');
disp(decoded);
```

```
decoded bits:
    {'gac'}    {'aba'}
```

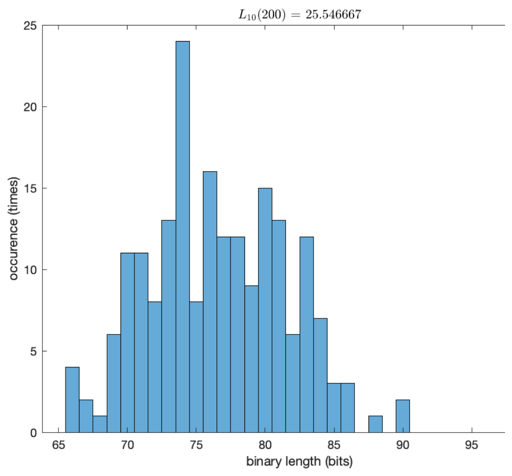
Figure 8: Result of encode and decode.

4.3 Generate a sequence of n = 10 symbols according to the probability.

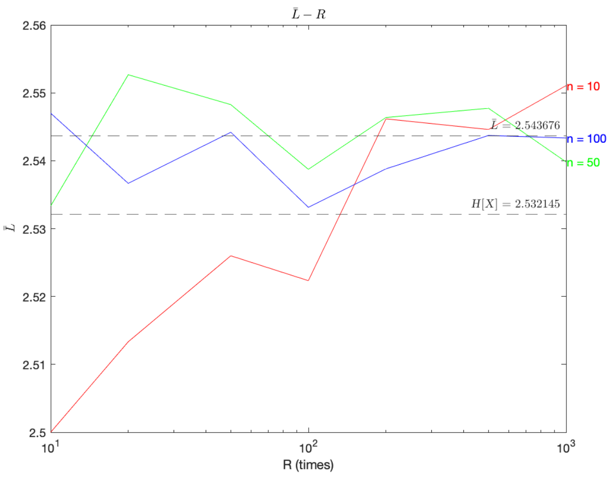
Here, the actual n should be 30, and I randomly create a sequence, whose length is 30, and the result is:
1. Symbol sequence: headaeeahdhhabeaaddadhhhdhaged(randomly generated)
2. Binary data: 10111100011001111011111101000100010000000011100111001001010110011100111111
3. Binary data length: 74

4.4 Comment and plots

For we set $X = X_1X_2X_3$, the length of dict is largerso it can encode more efficiently, which leads to lower , and it is more closed to H(X). In addition, when n = 50, R = 100, the difference is larger, and for each case the value is more close to the expected value, which can be verified in the case of n = 100, R $\leq$ 500.



(a) Histogram



(b) Line Chart

Figure 9: Plots