# Part 1

Assume a random variable X has M symbols with the following distribution:

$$f_X\left(x\right)=\frac{1}{6};\text{for } x=\ 1,2,3,4,5,\ \text{and } 6.$$

**Mapping each symbol to a fixed-length code with the minimum number of bits needed to represent** $\boldsymbol{M}$

**symbols:**

```
fprintf("Minimum number of bits needed to map each symbol of f(x) using a fixed-length coc
```

```
Minimum number of bits needed to map each symbol of f(x) using a fixed-length code = 3 bi
```

```
FixedCode_X = ['100' '101' '110' '111' '010' '001'];
dispTable = ["100" "101" "110" "111" "010" "001"];
array2table(dispTable, "VariableNames",{'1' '2' '3' '4' '5' '6'}, "RowNames",{'Codeword'})
```

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 Codeword | "100" | "101" | "110" | "111" | "010" | "001" |

And of course, since each codeword has the same length, it would make sense that the average length of the codewords is *3.*

**Find a Huffman code and calculate its average code length.**

```
X_symbols = (1:6); % Symbols vector
X_prob = [1/6 1/6 1/6 1/6 1/6 1/6]; % Symbol probability vector
[codeword_dict_x,average_length_x] = huffmandict(X_symbols,X_prob);
part1HuffmanTable = cell2table(codeword_dict_x);
part1HuffmanTable.Properties.VariableNames = {'Symbol', 'Codeword'};

disp(part1HuffmanTable)
```

```
    Symbol     Codeword
    ------    ---------

      1       {[  1 1]}
      2       {[  1 0]}
      3       {[0 0 1]}
      4       {[0 0 0]}
      5       {[0 1 1]}
      6       {[0 1 0]}
```

```
%Generate a binary Huffman code, displaying the average code length and the cell array co

%codeword_dict_x = cellfun(@num2str,codeword_dict_x(:,2),'UniformOutput',false);
%disp("Codeword of f(x)");
%huffTableX = cell2table(codeword_dict_x);
%disp((huffTableX));

%[dict,avglen] = huffmandict(X_symbols,X_prob)
%dict = cellfun(@num2str,dict(:,2),'UniformOutput',false)
%disp(dict)

H_X = 0;
 for i=1:length(X_prob)
 % Calculating the Entropy
 H_X = H_X + (-1*X_prob(i)*log2(X_prob(i)));
 end

 %fprintf("Entropy of huffman code of f(x)=%.3f bits",H_X);
 fprintf("Source entropy = %.3f bits",H_X);
```

```
Source entropy = 2.585 bits
```

```
 fprintf("Average Length of huffman code = %.3f bits/symbol ",average_length_x)
```

```
Average Length of huffman code = 2.667 bits/symbol
```

**Compare the average code lengths in parts 1) and 2) above to each other and to the Entropy of the random variable.**

Average Length of Huffman is less than the code of a fixed-length.

The average length of Huffman is between the entropy and entropy + 1

**Comment on your observations:**

Both codes are optimum codes as they are included in the bounds of the optomal code length (i.e. H <= L < H + 1) but Huffman is better as it has average code length less than the other but due to equiprobable Huffman didn't make that much difference than using normal code.

**4) Using the attached .mat file of sample data from the random variable:**

- **Verify the probability mass function of the random variable.**

```
load(strcat(fileparts(matlab.desktop.editor.getActiveFilename),"\ELCN446_Project2_Spring2(
fprintf("The X variable is loaded and it has a length %d ",length(X));
```

```
The X variable is loaded and it has a length 960
```

```
X_uniqueChars = unique(X); % String text has all characters, some are repeatedlenChar=leng
disp(X_uniqueChars);
```

```
     1     2     3     4     5     6
```

```
%PMF%
f=zeros(1,length(X_uniqueChars));
for i=1:length(X_uniqueChars)
% Count the number of occurence of unique characters
f(i)=length(strfind(X,X_uniqueChars(i)));
end
p=zeros(1,length(X_uniqueChars));
PMF_X = [length(length(X_uniqueChars))]; % declare the Probability Mass Function with size
for i=1:length(X_uniqueChars)
% Probabilities for each unique character in the file
p(i)=f(i)/length(X);
PMF_X(i) = p(i); %PMF
end
disp("PMF of X")
```

```
PMF of X
```

```
array2table(PMF_X, "RowNames",{'PMF'}, "VariableNames",{'1' '2' '3' '4' '5' '6'})
```

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 PMF | 0.1667 | 0.1667 | 0.1667 | 0.1667 | 0.1667 | 0.1667 |

**Generate the source coded codewords for both the fixed-length code and the Huffman code.**

```
%Generate the source coded code words using the fixed length code:
%disp("Using the fixed Code:");
%disp("Encode the Codewords:")
Transmit_X_fixed =0;

for i=1:length(X)
    temp=0;
    for j=1:length(X_uniqueChars)
```

```
        if X(i)== X_uniqueChars(j)
            if i==1
                Transmit_X_fixed= FixedCode_X(j+temp*2:j+2+temp*2);
               % disp(Transmit_X_fixed)
            else
                Transmit_X_fixed = append(Transmit_X_fixed,FixedCode_X(j+temp*2:j+2+temp*2))
                % disp(Transmit_X_fixed)
            end
            %disp("test")
        end
        temp = temp + 1;
       end
 end


 disp("The bits that are going to be transmitted using the Fixed code:");
```

```
The bits that are going to be transmitted using the Fixed code:
```

```
 disp(Transmit_X_fixed);
```

```
001100101010100010100010010100100100001001001010001100100101110010110110001010110111111010
```

```
 num_of_bits_X_fixed = length(Transmit_X_fixed);
 fprintf("The Total number of bits that are going to be transmitted using fixed code = %d"
```

```
The Total number of bits that are going to be transmitted using fixed code = 2880
```

```
%Decode the data. Verify that the decoded symbols match the original symbols.

disp("Decode the Codewords back to symbols using the fixed code:")
```

```
Decode the Codewords back to symbols using the fixed code:
```

```
Recieve_X_fixed=[];
i=1;
indexX=1;
  while i<=length(Transmit_X_fixed)
      temp = 0;
     for j=1:length(X_uniqueChars)

          if indexX > length(X)
              break;
```

```
        end

        if Transmit_X_fixed(i:(i+2)) == FixedCode_X(j+temp*2:j+2+temp*2)

            Recieve_X_fixed(indexX) = X_uniqueChars(j) ;
            i = i+3;
            indexX=indexX+1;


        end
        temp = temp + 1;
    end
 end
disp(Recieve_X_fixed);
```

```
  Columns 1 through 24

     6     1     2     5     1     5     1     5     5     1     1     1     6     6

  Columns 25 through 48

     6     5     3     4     3     2     1     4     3     4     3     5     4     2

  Columns 49 through 72

     2     6     4     5     3     1     2     6     2     1     5     4     4     3

  Columns 73 through 96

     1     3     5     1     5     4     3     4     3     5     4     3     2     5

  Columns 97 through 120

     5     4     5     1     2     6     3     4     6     6     1     5     1     1

  Columns 121 through 144

     3     1     5     6     6     3     2     3     2     5     1     4     5     5

  Columns 145 through 168

     4     3     1     4     3     3     5     5     3     5     3     6     6     3

  Columns 169 through 192

     4     5     2     5     1     6     1     5     1     6     4     3     1     6
```

```
  Columns 193 through 216

     5     4     4     6     4     4     3     4     3     5     2     6     4     4

  Columns 217 through 240

     4     5     2     3     1     5     6     1     6     3     5     3     4     4

  Columns 241 through 264

     4     1     6     1     5     1     5     1     2     3     1     1     1     1

  Columns 265 through 288

     2     3     3     6     1     4     5     3     1     2     6     4     5     1

  Columns 289 through 312

     4     6     5     6     3     1     4     6     3     4     3     6     3     1

  Columns 313 through 336

     5     2     6     3     2     1     2     2     5     1     4     2     6     3

  Columns 337 through 360

     4     4     2     2     6     2     1     6     1     1     6     6     1     3

  Columns 361 through 384

     2     4     2     5     6     1     6     3     4     5     4     5     4     5

  Columns 385 through 408

     6     3     3     3     6     6     2     1     4     4     4     3     4     5

  Columns 409 through 432

     3     4     2     6     4     6     5     2     1     5     6     5     6     1

  Columns 433 through 456

     2     4     2     5     6     5     1     5     5     2     3     4     5     5

  Columns 457 through 480
```

```
   3     4     4     2     3     5     2     6     1     2     6     4     3     1

Columns 481 through 504

   1     5     2     5     4     3     5     6     3     1     1     2     3     4

Columns 505 through 528

   2     5     2     4     4     4     6     3     3     2     4     6     3     1

Columns 529 through 552

   5     2     6     6     2     6     4     2     6     2     3     4     6     2

Columns 553 through 576

   5     4     5     6     2     3     6     4     1     5     6     5     5     1

Columns 577 through 600

   3     2     4     6     2     4     6     4     2     5     6     3     6     1

Columns 601 through 624

   2     4     1     3     6     1     5     5     1     1     2     4     6     3

Columns 625 through 648

   5     2     2     6     5     1     6     4     6     4     2     6     3     5

Columns 649 through 672

   4     1     6     1     1     3     1     3     4     6     5     4     1     6

Columns 673 through 696

   4     1     6     3     3     5     5     1     2     5     1     2     4     2

Columns 697 through 720

   6     4     3     2     2     1     5     2     6     2     6     3     5     2

Columns 721 through 744

   2     4     5     3     2     1     1     3     6     3     2     1     3     4
```

```
  Columns 745 through 768

     4     4     4     6     5     2     3     2     4     2     3     6     2     3

  Columns 769 through 792

     3     2     1     6     6     4     5     6     6     6     4     4     1     6

  Columns 793 through 816

     3     5     6     3     2     5     5     3     2     5     5     4     5     1

  Columns 817 through 840

     4     6     5     5     6     6     3     3     1     6     5     6     1     6

  Columns 841 through 864

     4     1     1     4     3     4     2     5     4     4     2     5     2     3

  Columns 865 through 888

     1     5     1     4     3     2     1     2     4     5     1     3     1     1

  Columns 889 through 912

     6     1     3     2     3     6     3     6     4     2     4     3     5     4

  Columns 913 through 936

     3     3     1     6     5     6     6     3     5     6     5     2     2     1

  Columns 937 through 960

     2     3     6     2     2     5     6     4     1     4     5     3     1     2
```

```
if isequal(X,Recieve_X_fixed)
    disp("No losses were found when using the Fixed Code since the recieved bits of X are
end
```

```
No losses were found when using the Fixed Code since the recieved bits of X are the same
```

```
%Generate the source coded code words using the Huffman code:
disp("Using the Huffman code:");
```

```
Using the Huffman code:
```

```
[dict_sentX,avglen_sentX] = huffmandict(X_uniqueChars,PMF_X);   %X_unique: Symbols
disp("Code Word of X that is Loaded from the .mat file:");
```

```
Code Word of X that is Loaded from the .mat file:
```

```
dict_sentX = cellfun(@num2str,dict_sentX(:,2),'UniformOutput',false);
disp(dict_sentX);
```

```
    {'1  1'   }
    {'1  0'   }
    {'0  0  1'}
    {'0  0  0'}
    {'0  1  1'}
    {'0  1  0'}
```

```


%disp("Test")
%disp((append(regexprep(cell2mat(dict_sentX(1)), '\s', '') , regexprep(cell2mat(dict_sent
%disp(regexprep(cell2mat(dict_sentX(1)), '\s',''))
%disp(length(regexprep(cell2mat(dict_sentX(1)), '\s','')))
%zxc= regexprep(cell2mat(dict_sentX(1)), '\s','');
%disp(zxc)
%disp("test")
%disp(transpose(cell2mat(dict_sentX(1))))
%disp(regexprep(cell2mat(dict_sentX(3)), '\s', '') + regexprep(cell2mat(dict_sentX(2)), '\

disp("Encode the Codewords:")
```

```
Encode the Codewords:
```

```
Transmit_X =0;
 for i=1:length(X)
    for j=1:length(X_uniqueChars)
     if X(i)== X_uniqueChars(j)
         if i==1
             Transmit_X= regexprep(cell2mat(dict_sentX(j)), '\s', '');
             %disp(Transmit_X)
```

```
        else
            Transmit_X = append(Transmit_X,regexprep(cell2mat(dict_sentX(j)), '\s', ''));
        end
        %disp("test")
    end
   end
end
disp("The bits that are going to be transmitted using the huffman code:");
```

```
The bits that are going to be transmitted using the huffman code:
```

```
disp(Transmit_X);
```

```
0101110011110111101101111111101001001001101011110001011001001010011001000001101100000100
```

```
num_of_bits_X = length(Transmit_X);
fprintf("The Total number of bits that are going to be transmitted = %d", num_of_bits_X);
```

```
The Total number of bits that are going to be transmitted = 2560
```

```
%Decode the data. Verify that the decoded symbols match the original symbols.
%sig = huffmandeco(double(Transmit_X),dict_sentX);

disp("Decode the Codewords back to symbols:")
```

```
Decode the Codewords back to symbols:
```

```
Recieve_X=[];
i=1;
indexX=1;
 while i<=length(Transmit_X)
     for j=3:length(dict_sentX)
     for k = 1:2
          if indexX > length(X)
              break;
          end

          if Transmit_X(i:(i+1)) == regexprep(cell2mat(dict_sentX(k)), '\s', '')

              Recieve_X(indexX) = X_uniqueChars(k) ;
              %disp(Recieve_X)  %for the test
              i = i+2;
              indexX=indexX+1;
```

```
        elseif  Transmit_X(i:(i+2)) == regexprep(cell2mat(dict_sentX(j)), '\s', '')

            Recieve_X(indexX) = X_uniqueChars(j) ;
            i = i+3;
            indexX=indexX+1;

    end
    end
   end
end
disp(Recieve_X);
```

```
  Columns 1 through 24

     6     1     2     5     1     5     1     5     5     1     1     1     6     6

  Columns 25 through 48

     6     5     3     4     3     2     1     4     3     4     3     5     4     2

  Columns 49 through 72

     2     6     4     5     3     1     2     6     2     1     5     4     4     3

  Columns 73 through 96

     1     3     5     1     5     4     3     4     3     5     4     3     2     5

  Columns 97 through 120

     5     4     5     1     2     6     3     4     6     6     1     5     1     1

  Columns 121 through 144

     3     1     5     6     6     3     2     3     2     5     1     4     5     5

  Columns 145 through 168

     4     3     1     4     3     3     5     5     3     5     3     6     6     3

  Columns 169 through 192

     4     5     2     5     1     6     1     5     1     6     4     3     1     6

  Columns 193 through 216
```

```
    5     4     4     6     4     4     3     4     3     5     2     6     4     4

  Columns 217 through 240

    4     5     2     3     1     5     6     1     6     3     5     3     4     4

  Columns 241 through 264

    4     1     6     1     5     1     5     1     2     3     1     1     1     1

  Columns 265 through 288

    2     3     3     6     1     4     5     3     1     2     6     4     5     1

  Columns 289 through 312

    4     6     5     6     3     1     4     6     3     4     3     6     3     1

  Columns 313 through 336

    5     2     6     3     2     1     2     2     5     1     4     2     6     3

  Columns 337 through 360

    4     4     2     2     6     2     1     6     1     1     6     6     1     3

  Columns 361 through 384

    2     4     2     5     6     1     6     3     4     5     4     5     4     5

  Columns 385 through 408

    6     3     3     3     6     6     2     1     4     4     4     3     4     5

  Columns 409 through 432

    3     4     2     6     4     6     5     2     1     5     6     5     6     1

  Columns 433 through 456

    2     4     2     5     6     5     1     5     5     2     3     4     5     5

  Columns 457 through 480

    3     4     4     2     3     5     2     6     1     2     6     4     3     1
```

```
Columns 481 through 504

   1     5     2     5     4     3     5     6     3     1     1     2     3     4

Columns 505 through 528

   2     5     2     4     4     4     6     3     3     2     4     6     3     1

Columns 529 through 552

   5     2     6     6     2     6     4     2     6     2     3     4     6     2

Columns 553 through 576

   5     4     5     6     2     3     6     4     1     5     6     5     5     1

Columns 577 through 600

   3     2     4     6     2     4     6     4     2     5     6     3     6     1

Columns 601 through 624

   2     4     1     3     6     1     5     5     1     1     2     4     6     3

Columns 625 through 648

   5     2     2     6     5     1     6     4     6     4     2     6     3     5

Columns 649 through 672

   4     1     6     1     1     3     1     3     4     6     5     4     1     6

Columns 673 through 696

   4     1     6     3     3     5     5     1     2     5     1     2     4     2

Columns 697 through 720

   6     4     3     2     2     1     5     2     6     2     6     3     5     2

Columns 721 through 744

   2     4     5     3     2     1     1     3     6     3     2     1     3     4

Columns 745 through 768
```

```
     4     4     4     6     5     2     3     2     4     2     3     6     2     3

   Columns 769 through 792

     3     2     1     6     6     4     5     6     6     6     4     4     1     6

   Columns 793 through 816

     3     5     6     3     2     5     5     3     2     5     5     4     5     1

   Columns 817 through 840

     4     6     5     5     6     6     3     3     1     6     5     6     1     6

   Columns 841 through 864

     4     1     1     4     3     4     2     5     4     4     2     5     2     3

   Columns 865 through 888

     1     5     1     4     3     2     1     2     4     5     1     3     1     1

   Columns 889 through 912

     6     1     3     2     3     6     3     6     4     2     4     3     5     4

   Columns 913 through 936

     3     3     1     6     5     6     6     3     5     6     5     2     2     1

   Columns 937 through 960

     2     3     6     2     2     5     6     4     1     4     5     3     1     2
```

```
if isequal(X,Recieve_X)
    disp("No losses were found when using the Huffman Code since the recieved bits of X ar
end
```

```
No losses were found when using the Huffman Code since the recieved bits of X are the sam
```

**Compare the total number of bits to be transmitted in each case.**

The Total number of bits to be transmitted using the fixed-length code is greater than using the Huffman code since it’s 2880 in fixed-length code while it is 2560 in Huffman code so it is clear that the Huffman code is more efficient than the fixed-length code. (i.e. send less number of bits)

**Decode the codewords back to symbols and check for any losses compared to the original data.**

No losses were found since the generated vector (The Recieved vector) is equal to the original vector.

# Part 2

**Map each symbol to a fixed-length code with the minimum number of bits needed to represent *M***

**symbols:**

For F(y):

```
disp("F(y) :");
```

```
F(y) :
```

```
fprintf("Minimum number of bits needed to map each symbol of f(y) using a fixed-length coc
```

```
Minimum number of bits needed to map each symbol of f(y) using a fixed-length code = 3 bi
```

```
fprintf("Average code length for a fixed-length code = %.2f",ceil(log2(6)));
```

```
Average code length for a fixed-length code = 3.00
```

```
FixedCode_Y = ['100' '101' '110' '111' '010' '001'];
```

So f(Y=1) will have the code: 100

f(Y=2) will have the code: 101

f(Y=3) will have the code: 110

f(Y=4) will have the code: 111

f(Y=5) will have the code: 010

f(Y=6) will have the code: 001

Note: the code may be different every time

**2) Find a Huffman code and calculate its average code length.**

```
Y_symbols = (1:6); % Symbols vector
Y_prob = [(0.5)^1 (0.5)^2 (0.5)^3 (0.5)^4 (0.5)^5 (0.5)^5]; % Symbol probability vector
[codeword_dict_y,average_length_y] = huffmandict(Y_symbols,Y_prob);
disp("Size of the representation of the code with the minimum number of bits for each Y")
```

```
Size of the representation of the code with the minimum number of bits for each Y
```

```
disp(['   symbol','  ','Size of codeword'])
```

```
   symbol  Size of codeword
```

```
disp(codeword_dict_y);
```

```
    {[1]}    {[        0]}
    {[2]}    {[      1 0]}
    {[3]}    {[    1 1 0]}
    {[4]}    {[  1 1 1 0]}
    {[5]}    {[1 1 1 1 1]}
    {[6]}    {[1 1 1 1 0]}
```

```
%Generate a binary Huffman code, displaying the average code length and the cell array co

codeword_dict_y = cellfun(@num2str,codeword_dict_y(:,2),'UniformOutput',false);
disp("Codeword of f(y)");
```

```
Codeword of f(y)
```

```
huffTableY = cell2table(codeword_dict_y);
disp((huffTableY));
```

```
     codeword_dict_y
    _________________

    {'0'            }
    {'1  0'         }
    {'1  1  0'      }
    {'1  1  1  0'   }
    {'1  1  1  1  1'}
    {'1  1  1  1  0'}
```

```
%[dict,avglen] = huffmandict(X_symbols,X_prob)
%dict = cellfun(@num2str,dict(:,2),'UniformOutput',false)
%disp(dict)

H_Y = 0;
 for i=1:length(Y_prob)
 % Calculating the Entropy
 H_Y = H_Y + (-1*Y_prob(i)*log2(Y_prob(i)));
 end

 fprintf("Source Entropy=%.3f bits",H_Y);
```

```
Entropy of huffman code of f(y)=1.938 bits
```

```
 fprintf("Average Length of huffman code of f(y)=%.3f ",average_length_y);
```

```
Average Length of huffman code of f(y)=1.938
```

**3) Compare the average code lengths in parts 1) and 2) above to each other and to the Entropy of the**

**random variable.**

Average Length of Huffman is less than the code of a fixed-length.

The average length of Huffman code is almost equal to the entropy so it's most efficient coding compared to use than the fixed-length code.

**Comment on your observations:**

The fixed length code isn't an optimal code while Huffman is an optimal code and Huffman is better as it has average code length less than the other.

**4) Using the attached .mat file of sample data from the random variable:**

- **Verify the probability mass function of the random variable.**

```
fprintf("The file .mat is already loaded from part I");
```

```
The file .mat is already loaded from part I
```

```
%load("ELCN446_Project2_Spring2022.mat");       %load the attached .mat file    (You don
fprintf("The Y variable is loaded and it has a length %d ",length(Y));
```

```
The Y variable is loaded and it has a length 960
```

```
Y_uniqueChars = unique(Y); % String text has all characters, some are repeatedlenChar=leng
disp(Y_uniqueChars);
```

```
     1     2     3     4     5     6
```

```
%PMF%
f=zeros(1,length(Y_uniqueChars));
for i=1:length(Y_uniqueChars)
% Count the number of occurence of unique characters
f(i)=length(strfind(Y,Y_uniqueChars(i)));
end
p=zeros(1,length(Y_uniqueChars));
PMF_Y = [length(length(Y_uniqueChars))]; % declare the Probability Mass Function with size
for i=1:length(Y_uniqueChars)
% Probabilities for each unique character in the file
p(i)=f(i)/length(Y);
PMF_Y(i) = p(i); %PMF
end
disp("PMF of Y")
```

```
PMF of Y
```

```
disp(transpose(PMF_Y));
```

```
    0.5000
    0.2500
    0.1250
    0.0625
    0.0312
    0.0312
```

```

```

- **Generate the source coded codewords for both the fixed-length code and the Huffman code.**

```
%Generate the source coded code words using the fixed length code:
disp("Using the fixed Code:");
```

```
Using the fixed Code:
```

```
disp("Encode the Codewords:")
```

```
Encode the Codewords:
```

```
Transmit_Y_fixed =0;

for i=1:length(Y)
    temp=0;
    for j=1:length(Y_uniqueChars)
     if Y(i)== Y_uniqueChars(j)
         if i==1
             Transmit_Y_fixed= FixedCode_Y(j+temp*2:j+2+temp*2);

         else
             Transmit_Y_fixed = append(Transmit_Y_fixed,FixedCode_Y(j+temp*2:j+2+temp*2))

         end
         %disp("test")
     end
     temp = temp + 1;
    end
 end


 disp("The bits that are going to be transmitted using the Fixed code:");
```

```
The bits that are going to be transmitted using the Fixed code:
```

```
 disp(Transmit_Y_fixed);
```

```
10010010010010010010001011011010011111111110010010010110010110000110111110010010011011011
```

```
num_of_bits_Y_fixed = length(Transmit_Y_fixed);
fprintf("The Total number of bits that are going to be transmitted using fixed code = %d'
```

```
The Total number of bits that are going to be transmitted using fixed code = 2880
```

```
%Decode the data. Verify that the decoded symbols match the original symbols.

disp("Decode the Codewords back to symbols using the fixed code:")
```

```
Decode the Codewords back to symbols using the fixed code:
```

```
Recieve_Y_fixed=[];
i=1;
indexY=1;
  while i<=length(Transmit_Y_fixed)
      temp = 0;
     for j=1:length(Y_uniqueChars)

          if indexY > length(Y)
              break;
          end

          if Transmit_Y_fixed(i:(i+2)) == FixedCode_Y(j+temp*2:j+2+temp*2)

              Recieve_Y_fixed(indexY) = Y_uniqueChars(j) ;
              i = i+3;
              indexY=indexY+1;


          end
          temp = temp + 1;
     end
  end
 disp(Recieve_Y_fixed);
```

```
  Columns 1 through 24

     1     1     1     1     1     1     1     5     3     3     1     4     4     4

  Columns 25 through 48

     1     1     1     3     3     3     1     2     1     1     2     1     1     2
```

```
Columns 49 through 72

  3     2     5     1     2     2     1     3     1     1     6     1     2     1

Columns 73 through 96

  2     1     1     3     1     1     1     1     3     2     2     1     2     2

Columns 97 through 120

  1     3     3     1     3     1     3     1     1     2     1     1     1     2

Columns 121 through 144

  6     2     2     1     1     3     2     1     1     1     1     1     1     2

Columns 145 through 168

  1     4     2     1     2     1     1     2     3     1     6     3     1     2

Columns 169 through 192

  1     1     2     3     1     1     1     1     2     2     3     1     2     2

Columns 193 through 216

  3     2     1     2     4     2     3     2     2     2     1     2     2     3

Columns 217 through 240

  1     1     2     1     1     2     1     1     2     1     4     1     1     1

Columns 241 through 264

  1     1     1     2     1     2     4     2     1     1     1     2     5     3

Columns 265 through 288

  1     1     1     5     5     1     1     2     4     1     2     1     2     1

Columns 289 through 312

  4     4     2     1     1     4     4     1     1     2     2     2     3     2

Columns 313 through 336
```

```
   3     1     1     2     1     2     1     1     3     3     1     6     1     1

Columns 337 through 360

   1     2     1     1     2     1     2     1     1     1     1     4     3     3

Columns 361 through 384

   1     2     1     3     2     1     2     2     1     5     1     3     5     1

Columns 385 through 408

   3     2     1     2     1     1     1     2     3     4     1     6     1     5

Columns 409 through 432

   1     1     2     6     1     3     3     1     1     3     1     2     1     1

Columns 433 through 456

   6     1     2     4     6     4     1     2     1     2     3     1     1     5

Columns 457 through 480

   1     3     1     1     2     3     1     1     1     2     2     4     4     3

Columns 481 through 504

   2     2     1     1     1     2     3     1     2     1     2     2     2     6

Columns 505 through 528

   3     3     1     1     3     2     1     1     2     2     1     1     2     1

Columns 529 through 552

   1     4     1     3     2     2     3     1     1     2     1     1     1     1

Columns 553 through 576

   1     1     3     1     1     1     1     5     1     1     1     1     1     4

Columns 577 through 600

   1     5     1     2     1     3     1     2     1     1     4     1     1     1
```

```
  Columns 601 through 624

     1     1     1     1     2     1     3     1     1     3     1     3     2     1

  Columns 625 through 648

     1     2     2     1     1     1     1     1     6     1     2     1     3     1

  Columns 649 through 672

     1     2     2     5     6     2     2     4     3     1     2     1     3     1

  Columns 673 through 696

     2     2     1     1     2     4     1     2     3     1     6     1     5     2

  Columns 697 through 720

     1     1     1     2     1     2     1     1     1     5     1     6     5     1

  Columns 721 through 744

     1     4     1     6     2     1     2     2     1     3     1     1     6     2

  Columns 745 through 768

     1     1     3     1     4     6     1     2     2     2     1     2     2     3

  Columns 769 through 792

     1     2     2     1     3     1     3     1     1     1     1     1     2     1

  Columns 793 through 816

     2     3     1     1     2     1     4     1     2     1     1     1     1     2

  Columns 817 through 840

     1     2     1     6     2     3     1     3     2     2     1     1     4     1

  Columns 841 through 864

     1     1     4     1     1     2     3     1     2     3     1     4     2     1

  Columns 865 through 888
```

```
     3     1     1     4     1     1     1     2     1     2     1     2     1     1

  Columns 889 through 912

     2     1     1     1     1     2     1     1     2     2     2     3     1     4

  Columns 913 through 936

     4     6     3     2     1     1     1     1     3     1     1     1     1     1

  Columns 937 through 960

     1     3     3     1     6     1     3     2     1     4     1     3     3     1
```

```
if isequal(Y,Recieve_Y_fixed)
    disp("No losses were found when using the Fixed Code since the recieved bits of Y are
end
```

```
No losses were found when using the Fixed Code since the recieved bits of Y are the same
```

```
%Generate the source coded code words using the Huffman code:
disp("Using the Huffman code:");
```

```
Using the Huffman code:
```

```
[dict_sentY,avglen_sentY] = huffmandict(Y_uniqueChars,PMF_Y);   %X_unique: Symbols
disp("Code Word of Y that is Loaded from the .mat file:");
```

```
Code Word of Y that is Loaded from the .mat file:
```

```
dict_sentY = cellfun(@num2str,dict_sentY(:,2),'UniformOutput',false);
disp(dict_sentY);
```

```
    {'0'            }
    {'1  0'         }
    {'1  1  0'      }
    {'1  1  1  0'   }
    {'1  1  1  1  1'}
    {'1  1  1  1  0'}
```

```
%disp("Test")
%disp((append(regexprep(cell2mat(dict_sentX(1)), '\s', '') , regexprep(cell2mat(dict_sentX
%disp(regexprep(cell2mat(dict_sentX(1)), '\s',''))
%disp(length(regexprep(cell2mat(dict_sentX(1)), '\s','')))
%zxc= regexprep(cell2mat(dict_sentX(1)), '\s','');
%disp(zxc)
%disp("test")
%disp(transpose(cell2mat(dict_sentX(1))))
%disp(regexprep(cell2mat(dict_sentX(3)), '\s', '') + regexprep(cell2mat(dict_sentX(2)), '\

disp("Encode the Codewords:")
```

```
Encode the Codewords:
```

```
Transmit_Y =0;
 for i=1:length(Y)
    for j=1:length(Y_uniqueChars)
     if Y(i)== Y_uniqueChars(j)
         if i==1
             Transmit_Y= regexprep(cell2mat(dict_sentY(j)), '\s', '');

         else
             Transmit_Y = append(Transmit_Y,regexprep(cell2mat(dict_sentY(j)), '\s', ''));
         end
         %disp("test")
     end
    end
 end
 disp("The bits that are going to be transmitted using the huffman code:");
```

```
The bits that are going to be transmitted using the huffman code:
```

```
 disp(Transmit_Y);
```

```
000000011111110110011101110111000010010011110101110000110110110010001000100001010011011111
```

```
 num_of_bits_Y = length(Transmit_Y);
 fprintf("The Total number of bits that are going to be transmitted = %d", num_of_bits_Y);
```

```
The Total number of bits that are going to be transmitted = 1860
```

```
%Decode the data. Verify that the decoded symbols match the original symbols.
%sig = huffmandeco(double(Transmit_X),dict_sentX);

disp("Decode the Codewords back to symbols:")
```

```
Decode the Codewords back to symbols:
```

```
Recieve_Y=[];
i=1;
indexY=1;
 while i<=length(Transmit_Y)
     for j=5:length(dict_sentY)

          if indexY > length(Y)
              break;
          end
         if Transmit_Y(i) == regexprep(cell2mat(dict_sentY(1)), '\s', '')
             Recieve_Y(indexY) = Y_uniqueChars(1) ;
              %disp(Recieve_Y)  %for the test
              i = i+1;
              indexY=indexY+1;
              break;

        elseif Transmit_Y(i:(i+1)) == regexprep(cell2mat(dict_sentY(2)), '\s', '')

              Recieve_Y(indexY) = Y_uniqueChars(2) ;
              %disp(Recieve_Y)  %for the test
              i = i+2;
              indexY=indexY+1;
              break;

        elseif Transmit_Y(i:(i+2)) == regexprep(cell2mat(dict_sentY(3)), '\s', '')


              Recieve_Y(indexY) = Y_uniqueChars(3) ;
              %disp(Recieve_Y)  %for the test
              i = i+3;
              indexY=indexY+1;
              break;

        elseif Transmit_Y(i:(i+3)) == regexprep(cell2mat(dict_sentY(4)), '\s', '')
```

```
            Recieve_Y(indexY) = Y_uniqueChars(4) ;
            %disp(Recieve_Y)  %for the test
            i = i+4;
            indexY=indexY+1;
            break;

        elseif  Transmit_Y(i:(i+4)) == regexprep(cell2mat(dict_sentY(j)), '\s', '')

            Recieve_Y(indexY) = Y_uniqueChars(j) ;
           % disp(Recieve_Y)  %for the test
            i = i+5;
            indexY=indexY+1;
            break;

    end
   end
 end
disp(Recieve_Y);
```

```
 Columns 1 through 24

   1     1     1     1     1     1     1     5     3     3     1     4     4     4

 Columns 25 through 48

   1     1     1     3     3     3     1     2     1     1     2     1     1     2

 Columns 49 through 72

   3     2     5     1     2     2     1     3     1     1     6     1     2     1

 Columns 73 through 96

   2     1     1     3     1     1     1     1     3     2     2     1     2     2

 Columns 97 through 120

   1     3     3     1     3     1     3     1     1     2     1     1     1     2

 Columns 121 through 144

   6     2     2     1     1     3     2     1     1     1     1     1     1     2

 Columns 145 through 168
```

```
     1     4     2     1     2     1     1     2     3     1     6     3     1     2

  Columns 169 through 192

     1     1     2     3     1     1     1     1     2     2     3     1     2     2

  Columns 193 through 216

     3     2     1     2     4     2     3     2     2     2     1     2     2     3

  Columns 217 through 240

     1     1     2     1     1     2     1     1     2     1     4     1     1     1

  Columns 241 through 264

     1     1     1     2     1     2     4     2     1     1     1     2     5     3

  Columns 265 through 288

     1     1     1     5     5     1     1     2     4     1     2     1     2     1

  Columns 289 through 312

     4     4     2     1     1     4     4     1     1     2     2     2     3     2

  Columns 313 through 336

     3     1     1     2     1     2     1     1     3     3     1     6     1     1

  Columns 337 through 360

     1     2     1     1     2     1     2     1     1     1     1     4     3     3

  Columns 361 through 384

     1     2     1     3     2     1     2     2     1     5     1     3     5     1

  Columns 385 through 408

     3     2     1     2     1     1     1     2     3     4     1     6     1     5

  Columns 409 through 432

     1     1     2     6     1     3     3     1     1     3     1     2     1     1
```

Columns 433 through 456

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 1 | 2 | 4 | 6 | 4 | 1 | 2 | 1 | 2 | 3 | 1 | 1 | 5 |

Columns 457 through 480

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 1 | 1 | 2 | 3 | 1 | 1 | 1 | 2 | 2 | 4 | 4 | 3 |

Columns 481 through 504

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | 1 | 1 | 2 | 3 | 1 | 2 | 1 | 2 | 2 | 2 | 6 |

Columns 505 through 528

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 1 | 1 | 3 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 1 |

Columns 529 through 552

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 3 | 2 | 2 | 3 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |

Columns 553 through 576

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 1 | 1 | 1 | 1 | 5 | 1 | 1 | 1 | 1 | 1 | 4 |

Columns 577 through 600

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 1 | 2 | 1 | 3 | 1 | 2 | 1 | 1 | 4 | 1 | 1 | 1 |

Columns 601 through 624

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 3 | 1 | 3 | 2 | 1 |

Columns 625 through 648

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 6 | 1 | 2 | 1 | 3 | 1 |

Columns 649 through 672

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 5 | 6 | 2 | 2 | 4 | 3 | 1 | 2 | 1 | 3 | 1 |

Columns 673 through 696

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | 1 | 2 | 4 | 1 | 2 | 3 | 1 | 6 | 1 | 5 | 2 |

Columns 697 through 720

```
     1     1     1     2     1     2     1     1     1     5     1     6     5     1

  Columns 721 through 744

     1     4     1     6     2     1     2     2     1     3     1     1     6     2

  Columns 745 through 768

     1     1     3     1     4     6     1     2     2     2     1     2     2     3

  Columns 769 through 792

     1     2     2     1     3     1     3     1     1     1     1     1     2     1

  Columns 793 through 816

     2     3     1     1     2     1     4     1     2     1     1     1     1     2

  Columns 817 through 840

     1     2     1     6     2     3     1     3     2     2     1     1     4     1

  Columns 841 through 864

     1     1     4     1     1     2     3     1     2     3     1     4     2     1

  Columns 865 through 888

     3     1     1     4     1     1     1     2     1     2     1     2     1     1

  Columns 889 through 912

     2     1     1     1     1     2     1     1     2     2     2     3     1     4

  Columns 913 through 936

     4     6     3     2     1     1     1     1     3     1     1     1     1     1

  Columns 937 through 960

     1     3     3     1     6     1     3     2     1     4     1     3     3     1
```

```
if isequal(Y,Recieve_Y)
```

```
    disp("No losses were found when using the Huffman Code since the recieved bits of Y ar
end
```

```
No losses were found when using the Huffman Code since the recieved bits of Y are the sam
```

**Compare the total number of bits to be transmitted in each case.**

The Total number of bits to be transmitted using the naive code is greater than using the Huffman code since it's 2880 in naive code while it is 1860 in Huffman code

so it is clear that the Huffman code is more efficient than the naive code. (i.e. send less number of bits)

- **Decode the codewords back to symbols and check for any losses compared to the original data.**

No losses were found since the generated vector (The Recieved) is equal to the original vector

**For F(Z):**

**1) Map each symbol to a fixed-length code with the minimum number of bits needed to represent $M$**

**symbols.**

```
disp("F(z) :");
```

```
F(z) :
```

```
fprintf("Minimum number of bits needed to map each symbol of f(z) using a fixed-length coc
```

```
Minimum number of bits needed to map each symbol of f(z) using a fixed-length code = 3 bi
```

```
fprintf("Average code length for a fixed-length code = %.2f",ceil(log2(6)));
```

```
Average code length for a fixed-length code = 3.00
```

```
FixedCode_Z = ['100' '101' '110' '111' '010' '001'];
```

So f(Z=1) will have the code: 100

f(Z=2) will have the code: 101

f(Z=3) will have the code: 110

f(Z=4) will have the code: 111

f(Z=5) will have the code: 010

f(Z=6) will have the code: 001

Note: the code may be different every time

**2) Find a Huffman code and calculate its average code length.**

```
Z_symbols = (1:6); % Symbols vector
Z_prob = [0.05 0.10 0.30 0.25 0.15 0.15]; % Symbol probability vector
[codeword_dict_z,average_length_z] = huffmandict(Z_symbols,Z_prob);
disp("Size of the representation of the code with the minimum number of bits for each Z")
```

```
Size of the representation of the code with the minimum number of bits for each Z
```

```
disp(['   symbol','  ','Size of codeword'])
```

```
   symbol  Size of codeword
```

```
disp(codeword_dict_z);
```

```
    {[1]}    {[1 1 1]}
    {[2]}    {[1 1 0]}
    {[3]}    {[  0 0]}
    {[4]}    {[  1 0]}
    {[5]}    {[0 1 1]}
    {[6]}    {[0 1 0]}
```

```
%Generate a binary Huffman code, displaying the average code length and the cell array co

codeword_dict_z = cellfun(@num2str,codeword_dict_z(:,2),'UniformOutput',false);
disp("Codeword of f(z)");
```

```
Codeword of f(z)
```

```
huffTableZ = cell2table(codeword_dict_z);
disp((huffTableZ));
```

```
    codeword_dict_z
    _______________

      {'1  1  1'}
      {'1  1  0'}
      {'0  0'   }
      {'1  0'   }
      {'0  1  1'}
      {'0  1  0'}
```

```
%[dict,avglen] = huffmandict(X_symbols,X_prob)
%dict = cellfun(@num2str,dict(:,2),'UniformOutput',false)
%disp(dict)

H_Z = 0;
 for i=1:length(Z_prob)
 % Calculating the Entropy
 H_Z = H_Z + (-1*Z_prob(i)*log2(Z_prob(i)));
 end

 fprintf("Source Entropy = %.3f bits",H_Z);
```

```
Entropy of huffman code of f(z)=2.390 bits
```

```
 fprintf("Average Length of huffman code of f(z)=%.3f ",average_length_z);
```

```
Average Length of huffman code of f(z)=2.450
```

**3) Compare the average code lengths in parts 1) and 2) above to each other and to the Entropy of the**

**random variable.**

Average Length of Huffman is less than the code of a fixed-length so Huffman is more efficient.

The average length for Huffman code is between the entropy and entropy + 1 so it's more optimum.

**Comment on your observations:**

Both codes are optimum codes as they are included in the bounds of the optomal code length (i.e. H <= L < H + 1) but Huffman is better as it has average code length less than the other but due to equiprobable Huffman didn't make that much difference than using normal code.

**4) Using the attached .mat file of sample data from the random variable:**

- **Verify the probability mass function of the random variable.**

```
fprintf("The file .mat is already loaded from part I");
```

```
The file .mat is already loaded from part I
```

```
%load("ELCN446_Project2_Spring2022.mat");       %load the attached .mat file    (You don
fprintf("The Z variable is loaded and it has a length %d ",length(Z));
```

```
The Z variable is loaded and it has a length 960
```

```
Z_uniqueChars = unique(Z); % String text has all characters, some are repeatedlenChar=leng
disp(Z_uniqueChars);
```

```
     1     2     3     4     5     6
```

```
%PMF%
f=zeros(1,length(Z_uniqueChars));
for i=1:length(Z_uniqueChars)
% Count the number of occurence of unique characters
f(i)=length(strfind(Z,Z_uniqueChars(i)));
end
p=zeros(1,length(Z_uniqueChars));
PMF_Z = [length(length(Z_uniqueChars))]; % declare the Probability Mass Function with size
for i=1:length(Z_uniqueChars)
% Probabilities for each unique character in the file
p(i)=f(i)/length(Z);
PMF_Z(i) = p(i); %PMF
end
```

```
disp("PMF of Z")
```

```
PMF of Z
```

```
disp(transpose(PMF_Z));
```

```
    0.0500
    0.1000
    0.3000
    0.2500
    0.1500
    0.1500
```

- **Generate the source coded codewords for both the fixed-length code and the Huffman code.**

```
%Generate the source coded code words using the fixed length code:
disp("Using the fixed Code:");
```

```
Using the fixed Code:
```

```
disp("Encode the Codewords:")
```

```
Encode the Codewords:
```

```
Transmit_Z_fixed =0;

for i=1:length(Z)
    temp=0;
    for j=1:length(Z_uniqueChars)
     if Z(i)== Z_uniqueChars(j)
         if i==1
             Transmit_Z_fixed= FixedCode_Z(j+temp*2:j+2+temp*2);
            % disp(Transmit_X_fixed)
         else
             Transmit_Z_fixed = append(Transmit_Z_fixed,FixedCode_Z(j+temp*2:j+2+temp*2))
             % disp(Transmit_X_fixed)
         end
         %disp("test")
     end
     temp = temp + 1;
    end
 end
```

```
disp("The bits that are going to be transmitted using the Fixed code:");
```

The bits that are going to be transmitted using the Fixed code:

```
disp(Transmit_Z_fixed);
```

1111101111111111111100100111000111110010111100111011000111111001011001010101000111010110100

```
num_of_bits_Z_fixed = length(Transmit_Z_fixed);
fprintf("The Total number of bits that are going to be transmitted using fixed code = %d'
```

The Total number of bits that are going to be transmitted using fixed code = 2880

```
%Decode the data. Verify that the decoded symbols match the original symbols.

disp("Decode the Codewords back to symbols using the fixed code:")
```

Decode the Codewords back to symbols using the fixed code:

```
Recieve_Z_fixed=[];
i=1;
indexZ=1;
  while i<=length(Transmit_Z_fixed)
      temp = 0;
     for j=1:length(Z_uniqueChars)

          if indexZ > length(Z)
              break;
          end

          if Transmit_Z_fixed(i:(i+2)) == FixedCode_Z(j+temp*2:j+2+temp*2)

              Recieve_Z_fixed(indexZ) = Z_uniqueChars(j) ;
              i = i+3;
              indexZ=indexZ+1;


          end
          temp = temp + 1;
     end
  end
```

```
disp(Recieve_Z_fixed);
```

```
  Columns 1 through 24

     4     3     4     4     4     4     6     6     3     6     4     1     2     4

  Columns 25 through 48

     5     6     3     2     2     6     6     4     4     6     5     5     2     3

  Columns 49 through 72

     3     3     3     1     3     4     5     4     5     3     2     6     6     4

  Columns 73 through 96

     4     4     6     1     3     1     5     5     3     5     3     4     3     4

  Columns 97 through 120

     5     6     1     1     4     3     1     5     3     3     5     3     3     4

  Columns 121 through 144

     2     4     4     4     4     1     3     5     3     4     4     4     5     4

  Columns 145 through 168

     4     5     4     6     3     3     6     4     4     5     6     4     6     4

  Columns 169 through 192

     2     4     2     4     5     5     3     4     3     5     6     5     4     3

  Columns 193 through 216

     4     5     2     3     5     3     3     3     5     3     3     3     5     3

  Columns 217 through 240

     3     3     4     2     3     5     6     4     2     6     5     3     4     3

  Columns 241 through 264

     4     1     4     3     6     3     6     3     3     3     3     6     1     5
```

Columns 265 through 288

3 5 1 4 4 2 3 5 2 3 5 4 6 3

Columns 289 through 312

3 4 6 4 3 3 5 4 5 4 6 4 6 6

Columns 313 through 336

5 5 4 4 6 3 4 3 4 5 5 3 2 4

Columns 337 through 360

3 5 4 3 5 3 4 5 3 2 3 1 6 3

Columns 361 through 384

2 4 1 3 3 6 3 5 4 4 4 3 5 4

Columns 385 through 408

6 4 3 6 6 6 5 4 3 4 3 1 3 4

Columns 409 through 432

6 5 5 4 2 4 4 2 5 4 2 2 3 6

Columns 433 through 456

4 3 3 4 4 5 4 4 2 3 3 3 1 5

Columns 457 through 480

3 4 4 6 6 4 3 6 6 5 2 3 4 4

Columns 481 through 504

5 4 3 2 2 3 3 6 1 3 6 3 2 6

Columns 505 through 528

6 2 6 3 2 6 3 4 4 2 3 3 4 3

Columns 529 through 552

```
     6     6     4     3     5     5     4     3     5     4     6     3     4     2

  Columns 553 through 576

     3     5     6     2     4     3     5     4     3     3     4     4     4     5

  Columns 577 through 600

     3     3     4     4     3     6     3     4     6     4     6     1     3     4

  Columns 601 through 624

     5     6     3     5     3     6     5     6     5     3     4     3     3     6

  Columns 625 through 648

     4     3     3     2     4     6     3     3     2     5     2     3     2     4

  Columns 649 through 672

     4     6     3     3     4     3     5     4     2     5     5     3     3     6

  Columns 673 through 696

     2     6     3     6     6     5     6     3     3     5     3     3     6     3

  Columns 697 through 720

     5     5     1     4     1     2     3     6     4     1     1     4     3     2

  Columns 721 through 744

     2     1     4     1     2     3     5     2     6     6     2     3     3     6

  Columns 745 through 768

     3     3     5     4     3     3     4     6     3     3     3     3     1     4

  Columns 769 through 792

     6     3     4     6     3     4     1     6     3     6     4     6     4     3

  Columns 793 through 816

     3     5     5     3     3     2     2     4     3     3     6     3     3     4
```

```
  Columns 817 through 840

     6     3     4     2     3     3     6     3     5     4     4     6     3     5

  Columns 841 through 864

     1     5     6     6     4     3     3     4     3     4     6     2     3     2

  Columns 865 through 888

     2     3     4     4     2     4     5     4     3     5     3     6     2     3

  Columns 889 through 912

     3     4     5     4     4     2     4     2     2     3     5     3     4     3

  Columns 913 through 936

     3     3     4     3     5     4     3     4     6     3     3     4     4     4

  Columns 937 through 960

     2     3     3     3     6     6     3     1     3     4     2     4     5     4
```

```
if isequal(Z,Recieve_Z_fixed)
    disp("No losses were found when using the Fixed Code since the recieved bits of Z are
end
```

```
No losses were found when using the Fixed Code since the recieved bits of Z are the same
```

```
%Generate the source coded code words using the Huffman code:
disp("Using the Huffman code:");
```

```
Using the Huffman code:
```

```
[dict_sentZ,avglen_sentZ] = huffmandict(Z_uniqueChars,PMF_Z);   %X_unique: Symbols
disp("Code Word of Z that is Loaded from the .mat file:");
```

```
Code Word of Z that is Loaded from the .mat file:
```

```
dict_sentZ = cellfun(@num2str,dict_sentZ(:,2),'UniformOutput',false);
disp(dict_sentZ);
```

```
{'1  1  1'}
{'1  1  0'}
{'0  0'   }
{'1  0'   }
{'0  1  1'}
{'0  1  0'}
```

```


%disp("Test")
%disp((append(regexprep(cell2mat(dict_sentX(1)), '\s', '') , regexprep(cell2mat(dict_sentX
%disp(regexprep(cell2mat(dict_sentX(1)), '\s',''))
%disp(length(regexprep(cell2mat(dict_sentX(1)), '\s','')))
%zxc= regexprep(cell2mat(dict_sentX(1)), '\s','');
%disp(zxc)
%disp("test")
%disp(transpose(cell2mat(dict_sentX(1))))
%disp(regexprep(cell2mat(dict_sentX(3)), '\s', '') + regexprep(cell2mat(dict_sentX(2)), '\

disp("Encode the Codewords:")
```

```
Encode the Codewords:
```

```
Transmit_Z =0;
 for i=1:length(Z)
    for j=1:length(Z_uniqueChars)
     if Z(i)== Z_uniqueChars(j)
         if i==1
             Transmit_Z= regexprep(cell2mat(dict_sentZ(j)), '\s', '');
             %disp(Transmit_X)
         else
             Transmit_Z = append(Transmit_Z,regexprep(cell2mat(dict_sentZ(j)), '\s', ''));
         end
         %disp("test")
     end
    end
 end
 disp("The bits that are going to be transmitted using the huffman code:");
```

```
The bits that are going to be transmitted using the huffman code:
```

```
disp(Transmit_Z);
```

```
1000101010100100100001010111110100100000010100001100011110011010001101100100101010010011O
```

```
num_of_bits_Z = length(Transmit_Z);
fprintf("The Total number of bits that are going to be transmitted = %d", num_of_bits_Z)
```

```
The Total number of bits that are going to be transmitted = 2352
```

```
%Decode the data. Verify that the decoded symbols match the original symbols.
%sig = huffmandeco(double(Transmit_X),dict_sentX);

disp("Decode the Codewords back to symbols:")
```

```
Decode the Codewords back to symbols:
```

```
Recieve_Z=[];
i=1;
indexZ=1;
 while i<=length(Transmit_Z)
     for j=5:length(dict_sentZ)
     for l=3:4
     for k = 1:2
          if indexZ > length(Z)
              break;
          end

          if Transmit_Z(i:(i+1)) == regexprep(cell2mat(dict_sentZ(l)), '\s', '')

              Recieve_Z(indexZ) = Z_uniqueChars(l) ;
              %disp(Recieve_X)  %for the test
              i = i+2;
              indexZ=indexZ+1;

          elseif  Transmit_Z(i:(i+2)) == regexprep(cell2mat(dict_sentZ(j)), '\s', '')

              Recieve_Z(indexZ) = Z_uniqueChars(j) ;
              i = i+3;
              indexZ=indexZ+1;

          elseif  Transmit_Z(i:(i+2)) == regexprep(cell2mat(dict_sentZ(k)), '\s', '')
```

```
            Recieve_Z(indexZ) = Z_uniqueChars(k) ;
            i = i+3;
            indexZ=indexZ+1;

    end
    end
   end
   end
end
disp(Recieve_Z);
```

```
 Columns 1 through 24

    4     3     4     4     4     4     6     6     3     6     4     1     2     4

 Columns 25 through 48

    5     6     3     2     2     6     6     4     4     6     5     5     2     3

 Columns 49 through 72

    3     3     3     1     3     4     5     4     5     3     2     6     6     4

 Columns 73 through 96

    4     4     6     1     3     1     5     5     3     5     3     4     3     4

 Columns 97 through 120

    5     6     1     1     4     3     1     5     3     3     5     3     3     4

 Columns 121 through 144

    2     4     4     4     4     1     3     5     3     4     4     4     5     4

 Columns 145 through 168

    4     5     4     6     3     3     6     4     4     5     6     4     6     4

 Columns 169 through 192

    2     4     2     4     5     5     3     4     3     5     6     5     4     3

 Columns 193 through 216

    4     5     2     3     5     3     3     3     5     3     3     3     5     3
```

```
Columns 217 through 240

   3     3     4     2     3     5     6     4     2     6     5     3     4     3

Columns 241 through 264

   4     1     4     3     6     3     6     3     3     3     3     6     1     5

Columns 265 through 288

   3     5     1     4     4     2     3     5     2     3     5     4     6     3

Columns 289 through 312

   3     4     6     4     3     3     5     4     5     4     6     4     6     6

Columns 313 through 336

   5     5     4     4     6     3     4     3     4     5     5     3     2     4

Columns 337 through 360

   3     5     4     3     5     3     4     5     3     2     3     1     6     3

Columns 361 through 384

   2     4     1     3     3     6     3     5     4     4     4     3     5     4

Columns 385 through 408

   6     4     3     6     6     6     5     4     3     4     3     1     3     4

Columns 409 through 432

   6     5     5     4     2     4     4     2     5     4     2     2     3     6

Columns 433 through 456

   4     3     3     4     4     5     4     4     2     3     3     3     1     5

Columns 457 through 480

   3     4     4     6     6     4     3     6     6     5     2     3     4     4

Columns 481 through 504
```

```
     5     4     3     2     2     3     3     6     1     3     6     3     2     6

  Columns 505 through 528

     6     2     6     3     2     6     3     4     4     2     3     3     4     3

  Columns 529 through 552

     6     6     4     3     5     5     4     3     5     4     6     3     4     2

  Columns 553 through 576

     3     5     6     2     4     3     5     4     3     3     4     4     4     5

  Columns 577 through 600

     3     3     4     4     3     6     3     4     6     4     6     1     3     4

  Columns 601 through 624

     5     6     3     5     3     6     5     6     5     3     4     3     3     6

  Columns 625 through 648

     4     3     3     2     4     6     3     3     2     5     2     3     2     4

  Columns 649 through 672

     4     6     3     3     4     3     5     4     2     5     5     3     3     6

  Columns 673 through 696

     2     6     3     6     6     5     6     3     3     5     3     3     6     3

  Columns 697 through 720

     5     5     1     4     1     2     3     6     4     1     1     4     3     2

  Columns 721 through 744

     2     1     4     1     2     3     5     2     6     6     2     3     3     6

  Columns 745 through 768

     3     3     5     4     3     3     4     6     3     3     3     3     1     4
```

```
  Columns 769 through 792

     6     3     4     6     3     4     1     6     3     6     4     6     4     3

  Columns 793 through 816

     3     5     5     3     3     2     2     4     3     3     6     3     3     4

  Columns 817 through 840

     6     3     4     2     3     3     6     3     5     4     4     6     3     5

  Columns 841 through 864

     1     5     6     6     4     3     3     4     3     4     6     2     3     2

  Columns 865 through 888

     2     3     4     4     2     4     5     4     3     5     3     6     2     3

  Columns 889 through 912

     3     4     5     4     4     2     4     2     2     3     5     3     4     3

  Columns 913 through 936

     3     3     4     3     5     4     3     4     6     3     3     4     4     4

  Columns 937 through 960

     2     3     3     3     6     6     3     1     3     4     2     4     5     4
```

```
if isequal(Z,Recieve_Z)
    disp("No losses were found when using the Huffman Code since the recieved bits of Z a
end
```

```
No losses were found when using the Huffman Code since the recieved bits of Z are the sam
```

**Compare the total number of bits to be transmitted in each case.**

The Total number of bits to be transmitted using the naive code is greater than using the Huffman code since it's 2880 in naive code while it is 2352 in Huffman code so it is clear that the Huffman code is more efficient than the naive code. (i.e. send less number of bits)

- **Decode the codewords back to symbols and check for any losses compared to the original data.**

No losses were found since the generated vector (The recieved) is equal to the original vector.

### Comment on your observations of Part II compared to Part I

Huffman is always better than the fixed code as it generates an optimum code and efficient code compared to the fixed.

In addition, Huffman generates least number of codewords to the highest probability so decreases the average code length

## Part 3

In this part, we are required to compress a text file using *Huffman* codes.

First, We need to acquire the text. You can select a text file by pressing the browse button:

```
textFromFile = false;
[file , path] = uigetfile ('*.txt');

if(~file)
    disp('No file was selected.');
else
    disp(strcat("Selected file: ", file));
    originalText = fileread (strcat(path,file));
    textFromFile = true;
end
```

```
No file was selected.
```

Selected text:

```
disp(originalText);
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque congue gravida leo,
```

Our file should consist of these characters only:

```
charList = char(strcat(char(65:90),char(97:122), char(49:57), ",. "))
```

```
charList = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789,. '
```

We now extract the character data from the file and find its *Probability Mass Function*, after removing all invalid characters:

```
originalText(~ismember(originalText, charList)) = '';
sortedText = sort(originalText);
[labels, charPMF] = PMF(sortedText);
pmfTable = table(transpose(labels), transpose(charPMF), 'VariableNames', {'Character', 'Pr
disp(pmfTable)
```

```
    Character    Probability
    _________    ___________

                     0.14225
        ,           0.014619
        .           0.017712
        A         0.00056227
        C          0.0014057
        D           0.001968
        F          0.0014057
        I          0.0011245
        L         0.00028114
        M          0.0022491
        N          0.0025302
        P          0.0036548
        Q         0.00084341
        S         0.00084341
        V         0.00084341
        a           0.072533
        b           0.006185
        c           0.037391
        d           0.019961
```

```
    e          0.093899
    f         0.0075907
    g          0.010964
    h          0.006185
    i          0.078156
    j        0.00056227
    l          0.050323
    m          0.039921
    n          0.048074
    o          0.036267
    p          0.018555
    q         0.0092775
    r          0.050042
    s          0.062693
    t          0.072814
    u          0.070284
    v            0.0149
    x         0.0011245
```

### Huffman Encoding

We can now use the built in function to generate a Huffman code for our file:

```
% The huffmandict function can't deal with the symbols as chars. We need to
% cast them into doubles.


symbols = double(labels);
dict = huffmandict(symbols, charPMF);
huffTable = cell2table(dict);
huffTable.Properties.VariableNames = {'Character', 'Codeword'};
huffTable.Character = char(huffTable.Character);

disp(huffTable)
```

```
    Character            Codeword
    _________    _________________________

                 {[                 0 1 1]}
        ,        {[           1 0 0 1 1 1]}
        .        {[           1 0 0 1 0 0]}
        A        {[1 1 1 1 0 1 1 1 0 0 0]}
        C        {[    1 1 1 1 0 0 0 1 1]}
        D        {[    1 0 0 1 0 1 0 0 1]}
```

```
F        {[    1 1 1 1 0 0 0 1 0]}
I        {[  1 1 1 1 0 0 0 0 0 0]}
L        {[1 1 1 1 0 1 1 1 0 0 1]}
M        {[    1 0 0 1 0 1 0 0 0]}
N        {[      1 1 1 1 0 1 1 0]}
P        {[      1 0 0 1 0 1 0 1]}
Q        {[  1 1 1 1 0 0 0 0 1 1]}
S        {[  1 1 1 1 0 0 0 0 1 0]}
V        {[  1 1 1 1 0 0 0 0 0 1]}
a        {[              0 1 0 0]}
b        {[        1 1 1 1 0 1 0]}
c        {[            0 0 1 0 0]}
d        {[          0 0 0 0 1 0]}
e        {[                1 1 0]}
f        {[        1 0 0 1 0 1 1]}
g        {[          1 1 1 1 1 0]}
h        {[        1 1 1 1 0 0 1]}
i        {[              0 0 0 1]}
j        {[  1 1 1 1 0 1 1 1 0 1]}
l        {[              1 0 1 0]}
m        {[            0 0 0 0 0]}
n        {[              1 1 1 0]}
o        {[            0 0 1 0 1]}
p        {[          0 0 0 0 1 1]}
q        {[          1 1 1 1 1 1]}
r        {[              1 0 1 1]}
s        {[              1 0 0 0]}
t        {[              0 0 1 1]}
u        {[              0 1 0 1]}
v        {[          1 0 0 1 1 0]}
x        {[    1 1 1 1 0 1 1 1 1]}
```

Now, we are ready to encode our text file:

```
encoded = huffmanenco(originalText, dict);
huffman = num2str(encoded);
huffman = strrep(huffman,' ','');
ascii = dec2bin(originalText);
ascii = reshape(ascii',1,[]);

codeTable = cell2table({ascii; huffman});
codeTable.Properties.VariableNames = {'Encoded File'};
```

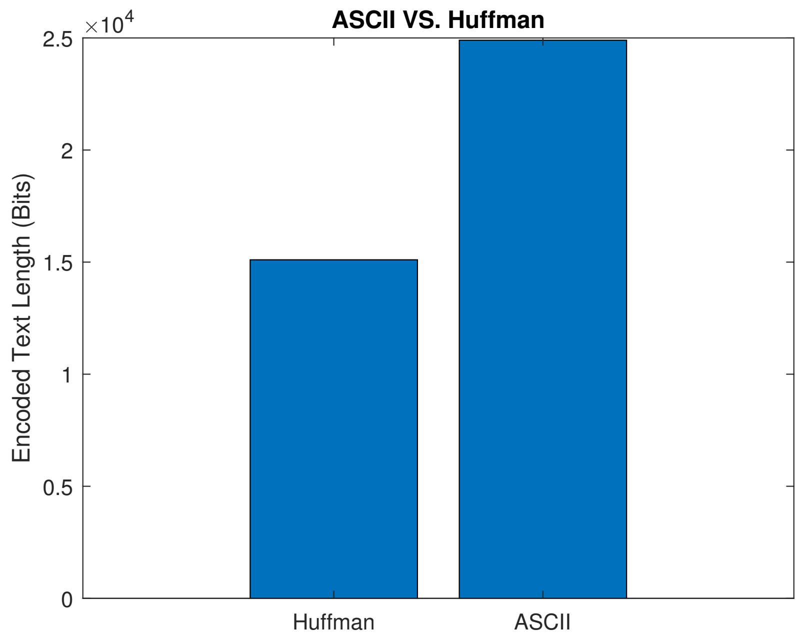
```
codeTable.Properties.RowNames = {'ASCII', 'Huffman'};
disp(codeTable)
```

```
                 ---------------------------------------------------------------------------

    ASCII        {'1001100110111111100101100101110110101000001101001111000011100111110101111
    Huffman      {'111101110010010110111100000001100010000111000010100000011000010001011010
```

We know find the compression ratio:

```
huffLen = length(huffman);
asciiLen = length(ascii);
lenArr = [huffLen asciiLen];
gra = bar(lenArr);
xticklabels({'Huffman', 'ASCII'});
title('ASCII VS. Huffman');
ylabel('Encoded Text Length (Bits)');
```




We can now calculate the compression ratio by applying the formula:

$$\frac{\text{Length of Huffman code}}{\text{Length of ASCII code}} \times 100$$

```
% compRatio = ((asciiLen - huffLen)/asciiLen) * 100;
% fprintf(" = %f %%", compRatio)
compRatio = (huffLen / asciiLen) * 100;
fprintf(" = %f %%", compRatio)
```

```
= 60.657055 %
```

## Decoding

We now decode both the ASCII and the Huffman files to check for errors. We expect none due to the one-to-one mapping in both ASCII and Huffman:

```
decodedHuff = char(huffmandeco(encoded, dict));
decodedASCII = zeros(1, asciiLen);

for i = 1:7:asciiLen-6
    % We need to decode each 7 bits into a single character

    letterBits = extractBetween(ascii, i, i+6);

    letter = char(bin2dec(letterBits));
    if letter == ' '
        decodedASCII = strcat(decodedASCII," ");
    else
        decodedASCII = strcat(decodedASCII, letter);
    end

end
```

```
Warning: Input should be a string, character array, or cell array of character arrays.
```

```
decodeTable = cell2table({decodedASCII; decodedHuff});
decodeTable.Properties.VariableNames = {'Decoded File'};
decodeTable.Properties.RowNames = {'ASCII', 'Huffman'};
disp(decodeTable);
```

```
              ------------------------------------------------------------------------

ASCII         "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque con
Huffman       "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque con
```

```
% And now, we compare the decoded text with the original text:

if(strcmp(decodedHuff, originalText))
    disp("Huffman codes are error free as expected.")
end
```

```
Huffman codes are error free as expected.
```

```
if(strcmp(decodedASCII, originalText))
    disp("ASCII codes are error free as expected.")
end
```

```
ASCII codes are error free as expected.
```

We now output the Huffman decoded file for further inspection:

```
fid = fopen(strcat(path,"huffman.txt"),'w+');
```

```
Error using string/strcat (line 37)
Inputs must be character vectors, cell arrays of character vectors, or string arrays.
```

```
fprintf(fid, decodedHuff);
fclose(fid);
```

## Comments

- Huffman coding is a lossless coding method, which by definition means it's error free.
- ASCII codes are made for computers to represent text, not for compression. We have 64 characters in our character list, which means we need only $\log_2(64) = 6$ Bits to represent each charachter, not 7 Bits.

## Functions

```
function  [uniqueChars, pmfCharArr] = PMF(text)
            % Returns the probability of each charachter in an array

            textLen = length(text);
            uniqueChars = unique(text);
            uniques = length(uniqueChars);
            pmfCharArr = zeros(1, uniques);
```

```
            for idx = 1:uniques
                pmfCharArr(idx) = sum(text == uniqueChars(idx)) / textLen ;
            end
end
```