Information theory

ECE 452s – Fall 2022
Project part 1

Abdelrahman Osama Abdelghaffar Abdelhadi 1804660
Alaa Emad Eldin Mahmoud Mohamed Shalaby 1806000
Abdelrahman Tarek Mahmoud 1809095

MATLAB code
First
Function to get the symbols probability
```
%this function takes the text file in variable (file)
%and the charachters that we will calculte its probability in variable
(char)
function probs = Probabilities(file, char)
charFreq = zeros(1,length(char));  %array that stores the frequency of
each character
%we will make for loop to loop at each character we are searching for
for i = 1:length(char)
    %this for loop will loop on the file to count tha number of
    %repeatitions of the chracter in it
    for c = file
        if c == char(i)
            charFreq(i) = charFreq(i) + 1;
        end
    end
end
probs = charFreq ./ length(file);  %get probability of each character
in the whole file
end
```

Function to get the symbols information
```
%this function calculates the information of each character
function info = Info(probs)

for i = 1:length(probs)
    if probs(i) ~= 0
        info(i) = - log2(probs(i));
    end
end
end
```

Function to get the entropy
```
%this function takes the calculated probabilities and get the entropy
function entropy = Entropy(probs)
entropy = 0; %initial value
for i = probs
    if i ~= 0 %to avoid calculating log(0) that gives Math Error
      entropy = entropy - i * log2(i);
    end
end
end
```

Second

Function to generate Huffman dictionary

```
function codes = HuffmanCreator(characters, probsArray)
n = length(probsArray);
codes = cell(1, n); %cell of matrices each matrix will carry 1 row of
the bits of the code
temp = cell(1, n);
for i = 1:n
    temp{i} = i; %cell of arrays only numbered to be used as index for
the cell codes
end
temp2 = [probsArray; 1:n]; %making a matrix of 2 rows to number the
probabilities with indices from 1 to n
for i = 1:n-1
    temp2 = (sortrows(temp2.', 1)).'; %getting transpose of the matrix
and sorting it according to the first column (probabilities)
    temp = sortTempAsTemp2(temp, temp2(2, :)); %sorting temp by the
same sort of second row of temp2 (indices after sorting)
    for j = 1:length(temp{1})
        codes{temp{1}(j)} = strcat('0', codes{temp{1}(j)}); %adding 0
bit to the least probability symbol
    end
    for j = 1:length(temp{2})
        codes{temp{2}(j)} = strcat('1', codes{temp{2}(j)}); %adding 1
bit to the second least probability symbol
    end
    temp2(1, 1) = temp2(1,1) + temp2(1, 2); %add the least two
probabilities and store it in first element of the array
    temp{1} = [temp{1} temp{2}]; %make the first array representing
the first two old arrays as we did in the array temp2 because temp is
connected to temp2
    temp2(1, 2) = 100; %put very high probability for the element we
don't need anymore to send it to the end of the matrix because we need
yo sum the first two elements of the least probabilities next time
    temp{2} = 0; %cancel the corresponding array to the removed
probability
    temp2(2, :) = 1:n; %renumbering the array of probabilities to re-
sort it again according to the new probabilities
end
fprintf('Huffman table\n');
for i = 1:n
    fprintf('%c      %s\n', characters(i), codes{i});  %print
characters and the strings that have the codes
end
end
```

Function created to be used in dictionary formation

```
function y = sortTempAsTemp2(x, a)
%This function takes an array x and an array of indices a and sorts
the elements in x by the order by indices dictated in a.
%we sort temp according to temp2
n = length(x);
y = cell(1, n);
for i = 1:n
    y(i) = x(a(i));
end
end
```

Function to encode the text file

```
function Encoding(characters, codes, input)
fileID = fopen('encodedText.txt', 'w+'); %open file to write the
encoded text in it
for i = input
    for j = 1:length(characters) %to search for all characters
        if i == characters(j) %if the input file contains this
character we execute the code
            fprintf(fileID, codes{j}); %print the code of the
character in the file of encoded text
        end
    end
end
fclose(fileID); %closing the file
end
```

Function to decode the encoded text file

```
function decoded = Decoding(characters, codes)
fileID_Encoded = fopen('encodedText.txt','r'); %read the text file
encodedText = fscanf(fileID_Encoded, '%c'); %reads and converts data
from a text file into array in column order
fileID_Decoded = fopen('decodedText.txt', 'w+'); %open file to write
the encoded text in it
decoded = [];
currentCode = [];
for i = encodedText %takes every charachter of the encoded text
   currentCode = [currentCode i]; %if the code of the characters
didn't equal the current code we take the next bit in the file and
search for the new current code in our codes
   for j = 1:length(codes) %counter j is used to search for current
code in our character codes
       if isequal(currentCode, codes{j})
           decoded = [decoded characters(j)]; %we get the character if
the current code is similarto its code
           currentCode = []; %making current code empty to serch for
the code of next character
           fprintf(fileID_Decoded, characters(j)); %print the
character in decoded text
           break;
       end
   end
end
fclose(fileID_Decoded); %closing the file
end
```

Third

Function to calculate the efficiency

```
function Efficiency(codes, probs, entropy)
%to calculate the efficiency we need to calculate the average code length
avgLength = 0;
for i = 1:length(probs)
    avgLength = avgLength + probs(i)*length(codes{i}); %calculate average code length
end
efficiency = entropy / avgLength; %calculate efficiency
fprintf('The average code word length = %f\nThe efficiency = %f\n', avgLength, efficiency);
end
```

Fourth

Main function, calculating compression ratio and checking if the input file and decoded file are the same

```
fileID = fopen('trial.txt','r'); %read the text file
input = fscanf(fileID, '%c'); %reads and converts data from a text file into array in column order

characters = ['a':'z','A':'Z','0':'9',' ','[',']','(',')','.',',','/','-','+','!','@','#','$','%','^','&','*'];

%Calculate the probabilities
probs = Probabilities(input, characters)
fprintf("__________________________________________________________\n\n");

%Calculate the information
info = Info(probs)
fprintf("__________________________________________________________\n\n");

%Calculate the entropy
entropy = Entropy(probs)
fprintf("__________________________________________________________\n\n");

%Generate the Huffman coding table
codes = HuffmanCreator(characters, probs);
fprintf("__________________________________________________________\n\n");
```

```
%generate text file contains the Encoded text
Encoding(characters, codes, input);

%take the encoded input to decode it
Decoded = Decoding(characters, codes);

%take the codes and their probabilities to get average code length
then calculate efficiency using entropy
Efficiency(codes, probs, entropy);
fprintf("__________________________________________________________\n\
n");

%getting size of input text file
fileID1 = fopen('trial.txt');
fseek(fileID1, 0, 'eof');
filesize1 = ftell(fileID1);
fclose(fileID1);
%getting size of input text file
fileID2 = fopen('encodedText.txt');
fseek(fileID1, 0, 'eof');
filesize2 = ftell(fileID2);
fclose(fileID2);
%printing the compression ratio
fprintf('Compression ratio = size of encoded text / size of input text
= %f/%f = %f', filesize2, filesize1, filesize2/filesize1 );
fprintf("\n__________________________________________________________\
n\n");


%Checking if decoded text file = the input text file
fileID3 = fopen('trial.txt','r');
inputText = fscanf(fileID3, '%c');
fileID4 = fopen('trial.txt','r');
decodedText = fscanf(fileID4, '%c');
if(decodedText == inputText);
    fprintf('Both files are the same\n');
end
```

Output of the code

We used the uploaded file on LMS as trial.txt

```
>> main

probs =

  Columns 1 through 15

    0.0528    0.0081    0.0290    0.0268    0.0965    0.0148    0.0129    0.0255    0.0560    0.0006
0.0031    0.0247    0.0205    0.0537    0.0559

  Columns 16 through 30

    0.0169    0.0008    0.0552    0.0480    0.0626    0.0163    0.0060    0.0064    0.0015    0.0118
0.0003    0.0073    0.0007    0.0038    0.0022

  Columns 31 through 45

    0.0025    0.0009    0.0005    0.0007    0.0027    0.0000    0.0000    0.0075    0.0037    0.0029
0.0024    0.0052    0.0000    0.0017    0.0038

  Columns 46 through 60

    0.0044    0.0022    0.0002    0.0017    0.0000    0.0003         0    0.0004    0.0018    0.0015    0.0012
0.0009    0.0007    0.0002    0.0002

  Columns 61 through 75

    0.0002    0.0003    0.1827    0.0000    0.0000    0.0018    0.0018    0.0104    0.0086    0.0003
0.0008    0.0001         0         0         0

  Columns 76 through 80

         0    0.0000         0         0         0
```

```
info =

  Columns 1 through 15

    4.2424    6.9484    5.1076    5.2221    3.3733    6.0794    6.2753    5.2928    4.1583   10.7100
```

8.3466   5.3400   5.6076   4.2181   4.1602

  Columns 16 through 30

    5.8908   10.2726   4.1789   4.3798   3.9980   5.9374   7.3731   7.2781   9.4123   6.4016
11.9026   7.1002   10.5670   8.0423   8.8437

  Columns 31 through 45

    8.6511   10.1250   10.8687   10.4369   8.5070   14.7100   14.7100   7.0637   8.0661   8.4246
8.6950   7.5945   16.2949   9.2396   8.0328

  Columns 46 through 60

    7.8233   8.8111   12.1250   9.2075   14.7100   11.9026   0   11.2075   9.1250   9.4001
9.7251   10.1864   10.4123   12.0470   12.2075

  Columns 61 through 75

   12.2075   11.9026   2.4523   14.2949   14.2949   9.0855   9.0855   6.5841   6.8561   11.6511
10.2288   13.7100   0   0   0

  Columns 76 through 77

         0   14.7100

---

entropy =

    4.4161

Huffman table
a   0100
b   1100001
c   10100
d   01010
e   000
f   101010
g   010110
h   00111
i   1001

j	11010001001
k	110111101
l	00110
m	110101
n	0110
o	1000
p	110011
q	0101110000
r	0111
s	0010
t	1011
u	110001
v	11011011
w	11011111
x	1101101001
y	1101110
z	110000001100
A	0101111
B	11011010000
C	11000001
D	110010010
E	110100110
F	1100000010
G	11000000111
H	11011010001
I	110100111
J	1101111001101011
K	110111100110000
L	1010111
M	10101101
N	110110101
O	110100011
P	11010010
Q	11011110011010101
R	010111001
S	11001000
T	11010000
U	110010011
V	010111011110
W	010111010
X	110111100110001
Y	110000001101
Z	11011110011010100000000
0	01011101110

```
1      101011000
2      1101111000
3      1101000101
4      0101110110
5      11011110010
6      010111011111
7      1101111001110
8      1101111001111
9      110100010000
       111
[      110111100110110
]      110111100110111
(      101011001
)      110000000
.      1101100
,      1100101
/      110100010001
-      0101110001
+      11011110011001
!      11011110011010100000001
@      1101111001101010100000001
#      110111100110101010000001
$      11011110011010100001
%      110111100110100
^      110111100110101000001
&      1101111001101010001
*      110111100110101001
```

The average code word length = 4.423191
The efficiency = 0.998405

Compression ratio = size of encoded text / size of input text = 355629.000000/80401.000000 = 4.423191

Both files are the same