CIE 425: Information Theory and Coding Wed., Dec. 02, 2020

Project Part 1 Report

By: Alaa Alajmy (201700095),
Hassan Youssef (201601850),
Moemen Gaafar (201700873)

In 1951, 69 years before the spring semester of 2020, Professor Robert Fano gave his Information Theory students the option between a term paper and a final exam. One student, David Huffman did not want to study for the exam and chose to work for long hours on what he did not know was an open-ended question his professor himself was seeking to solve. Just a little before he gave up, David succeeded and created an optimal symbol coding algorithm that surpassed his professor's. [1]

In this project, we implement the Huffman Algorithm on MATLAB and test it on two sample files.

Contents:

I. Introduction

Huffman coding is an optimal, variable-length, prefix, symbol coding algorithm used for lossless data compression. Generally, the Huffman algorithm aims to give less common symbols codes with more bits, while reserving much shorter codes for the more common ones. The Huffman algorithm recursively finds and adds the two smallest elements in an array of probabilities to create a tree with a root node holding a total probability of 1, then assigns a zero or a one to each child of the tree's parent nodes.

II. Methods

The frame of work holds 5 main functions:

a. **GetProbabilities:**
Input: an array of the used 33 characters and the input text.
Output: an array of each character's probability.
This function calculates probabilities by looping through the input text's characters and counting each character's occurrences, then dividing by the total number of characters in the text.

b. **GetEntropy:**
Input: the array of probabilities returned from GetProbabilities.
Output: the entropy of our symbols for the input text.
This function applies the entropy formula below to calculate the minimum possible number of bits that could be used to encode each character of our used text.

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

c. **HuffmanCreator:**
Input: an array of the used characters and the array of their probabilities.
Output: an array of Huffman codes for each character.
This function applies the Huffman algorithm to create a variable-length, prefix code for each character, then print a table of each character and its corresponding code. In its processes, this function uses a smaller function 'mixup' that takes an array *x* and an array of indices *a* and sorts the elements in *x* by the order by indices dictated in *a*.

d. **HuffmanEncoder:**
   Input: an array of the used characters, the array of codes returned from HuffmanCreator, and the input text to be encoded.
   Output: No output is returned but a file of the encoded text is created.
   This function loops through the input text's characters and prints each character's code in the output file.
e. **HuffmanDecoder:**
   Input: an array of the used characters, the array of codes returned from HuffmanCreator, and the text to be decoded.
   Output: the decoded text.
   This function loops through the encoded text's characters and compares the incoming stream of characters with the created Huffman codes to print the decoded characters in the output file.

III. Results

Our implementation successfully encoded and decoded the test file provided with the project statement and a second test file of our own. Used on the provided test file, our implementation returned:

- Symbols' entropy = 4.257011
- Fixed-length coding number of bits/symbols = 6.000000
- Fixed-length coding efficiency = 0.709502
- Huffman coding number of bits/symbols = 4.276353
- Huffman coding efficiency = 0.995477

IV. References

[1] "Discovery of Huffman Codes," *Discovery of Huffman Codes | Mathematical Association of America*. [Online]. Available: https://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes. [Accessed: 02-Dec-2020].

V. Code

Our main script and our created functions are attached in the following pages.

Contents

```
clear; clc;

%Read the text file
fileID = fopen('Test_text_file.txt','r');
input = fscanf(fileID, '%c');

%Calculate probability of each character
%Order: a-z ().,/-
characters = ['a':'z', ' ', '(', ')', '.', ',', '/', '-'];
probs = GetProbabilities(input, characters);

%Calculate the entropy
entropy = GetEntropy(probs);
fprintf("Entropy = %f\n\n", entropy);

%Calculate number of bits/symbol for fixed length coding
numBits_fixedLength = ceil(log2(length(characters)));

%Calculate efficiency of fixed length coding
efficiency_fixedLength = entropy / numBits_fixedLength;

fprintf("For fixed length coding:\n");
fprintf("-Number of bits/symbols = %f\n-Efficiency = %f\n\n", numBits_fixedLength, efficiency
_fixedLength);
```

```
Entropy = 4.257011

For fixed length coding:
-Number of bits/symbols = 6.000000
-Efficiency = 0.709502
```

Huffman Encoding

```
%Create Huffman code for each character
fprintf("Initiating Huffman encoding...\n\n");
codes = HuffmanCreator(characters, probs);
```

```
%Encode test file
HuffmanEncoder(characters, codes, input);
```

```
Initiating Huffman encoding...

Symbol Code
a      1100
b      000101
c      00011
d      0000
e      001
f      010100
g      010110
h      111111
i      0111
j      11101111110
k      11101111111
l      11110
m      111110
n      1001
o      1000
p      11100
q      11101111100
r      0110
s      0100
t      1101
u      111010
v      1110110
w      0101011
x      11101110
y      0101010
z      010111100
       101
(      010111101
)      111011110
.      01011111
,      0101110
/      11101111101
-      000100
```

Huffman Decoding

```
%Decode test file
fileID_Encoded = fopen('encodedText.txt','r');
inputEncoded = fscanf(fileID_Encoded, '%c');
fclose(fileID_Encoded);
fprintf("\nInitiating Huffman decoding...\n\n");
decoded = HuffmanDecoder(characters, codes, inputEncoded);

%Check if the input and the decoded texts are identical
if isequal(input, decoded)
    fprintf("Success!\n\n");
else
    fprintf("I don't think you are passing this course...\n\n");
```

```
end

%Write the decoded text to a separate file
fileID_Decoded = fopen('decodedText.txt', 'w');
fprintf(fileID_Decoded, decoded);

%Calculate efficiency of Huffman coding
numBits_Huffman = 0;
for i = 1:length(codes)
    numBits_Huffman = numBits_Huffman + probs(i) * length(codes{i});
end
efficiency_Huffman = entropy / numBits_Huffman;

fprintf("For Huffman coding:\n");
fprintf("-Number of bits/symbols = %f\n-Efficiency = %f\n\n", numBits_Huffman, efficiency_Huf
fman);
```

```
Initiating Huffman decoding...

Success!

For Huffman coding:
-Number of bits/symbols = 4.276353
-Efficiency = 0.995477
```

Entropy Function

```
function entropy = GetEntropy(probs)
% This function calculates the entropy using the provided probabilities.

entropy = 0;
for p = probs
    if p ~= 0
      entropy = entropy - p * log2(p);
    end
end
end
```

Probabilities Function

```
function probs = GetProbabilities(input, characters)
% This function returns the probabilities of all the input characters in
% the given input text.

charCount = zeros(1,length(characters));
for i = 1:length(characters)
    for c = input
        if c == characters(i)
            charCount(i) = charCount(i) + 1;
        end
    end
end
probs = charCount ./ length(input);
```

```
end
```

The Mixup Function

```
function y = mixup(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.

n = length(x);
y = cell(1, n);
for i = 1:n
    y(i) = x(a(i));
end
end
```

Huffman Code Creator

```
function codes = HuffmanCreator(symsArray, probsArray)
% This function creates the Huffman code for each character and displays
% the encoding table.

n = length(probsArray);
% This cell will be used to store the codes for each character
codes = cell(1, n);
% This cell will be used to keep track of the encoded characters
temp = cell(1, n);

for i = 1:n
    temp{i} = i;
end

% This cell will be used to keep track of the probabilities of each
% character or group of characters
temp2 = [probsArray; 1:n];

for i = 1:n-1
    % Order the characters according to their probabilities
    temp2 = (sortrows(temp2.', 1)).';
    temp = mixup(temp, temp2(2, :));

    % For the character (or group of characters) with the least probability
    % add 0 to its/their code
    for j = 1:length(temp{1})
        codes{temp{1}(j)} = strcat('0', codes{temp{1}(j)});
    end

    % For the character (or group of characters) with the second least
    % probability add 1 to its/their code
    for j = 1:length(temp{2})
        codes{temp{2}(j)} = strcat('1', codes{temp{2}(j)});
    end

    % Save the characters with the least two probabibilities as one element
    temp{1} = [temp{1} temp{2}];
    temp2(1, 1) = temp2(1,1) + temp2(1, 2);
    % Replace the second character with 0 and probability of 2
    % so that it is excluded in the next loop
    temp{2} = 0;
    temp2(1, 2) = 2;

    % Reorder the indices of the characters
    temp2(2, :) = 1:n;
end

% Print the encoding table
fprintf('Symbol Code\n');
for i = 1:n
    fprintf('%c      %s\n', symsArray(i), codes{i});
end

end
```

Huffman Encoder

```
function HuffmanEncoder(symsArray, codes, input)
% This function encodes the characters in the input text using the input
%code and saves the encoded text in a file in the same directory.

fileID = fopen('encodedText.txt', 'w');
for i = input
    for j = 1:length(symsArray)
        if i == symsArray(j)
            fprintf(fileID, codes{j});
        end
    end
end
fclose(fileID);
end
```

Huffman Decoder

```
function decoded = HuffmanDecoder(characters, codes, inputEncoded)
% This function decodes the characters in the input text using the input
%code and returns the decoded text.

decoded = [];
currentCode = [];
for i = inputEncoded
   currentCode = [currentCode i];
   for j = 1:length(codes)
       if isequal(currentCode, codes{j})
           decoded = [decoded characters(j)];
           currentCode = [];
           break;
       end
   end
end
end
```