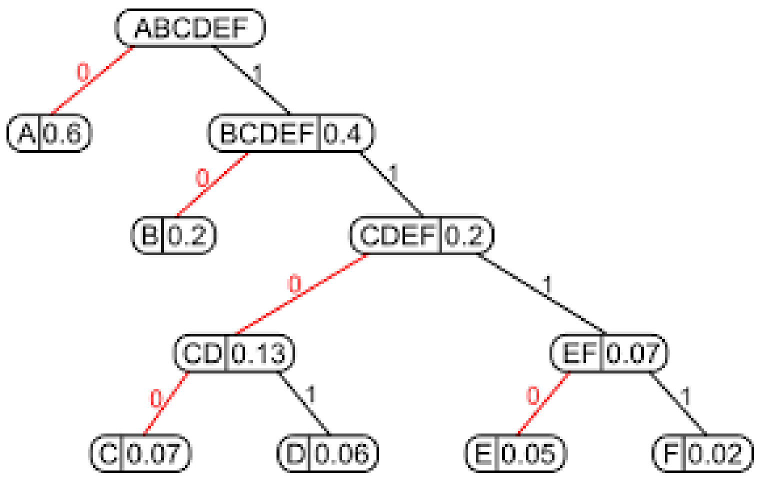
HUFFMAN ENCODING FOR TEXT FILES

AIM:

To encode and decode a text file containing of alphanumeric characters and special symbols, using Huffman encoding technique with the help of Matlab software.

SCHEMATIC:

An example schematic of how this encoding works



PRINCIPLE OF OPERATION:

The basic principle of this encoding technique is Huffman tree technique. Huffman encoding technique is a data compression technique that assigns variable-length codes to characters based on their frequency of occurrence in a given data set. The principle of this encoding technique is to assign shorter codes to more frequent characters and longer codes to less frequent characters. The code assigned for each character will not be a prefix for other character’s code.

ALGORITHM:

Initializing:

1. Create a dictionary for the given basic English characters data set, along with their frequency of occurrence.
2. Scan through the text file, and identify the characters and its frequency of occurrence, and store it in the dictionary.

Finding the probability:

3. Find the probability for each frequency of occurrence for the founded characters in the text file using the formula, and store it as an array.

   Probability = Frequency_of_1_character / total_frequency

Creating the Huffman tree:

4. Construct a Huffman tree by creating a node for each probability, and sort it in descending order.

Adding of nodes of lowest probability:

5. Add two nodes which are of lowest probabilities, and store it in a new node, and store those two nodes to the new node’s right and left side. Sort the nodes again in the descending order. Repeat this procedure until the root of the tree (i.e. last node) is reached.

Assigning codes to each node:

6. Assign 0’s to the left node and 1’s to the right node using the backtracking technique.

7. Append the previous code to the branch node.

Encoding process:

8. Assign the code which has been stored, for each character in the text file, and store these codes in an array. This completes the process of Huffman encoding for text file.

Decoding process:

9. For the purpose of decoding, compare each bit with the code values stored, and find the corresponding character, and store these results in an array or a separate file.

MATLAB CODE:

```
clear all
close all
% Open the text file
fileID = fopen('information.txt','r');
tline = fgetl(fileID);
line = '';

firstLine = true;
while ischar(tline)
    if firstLine
        line = tline;  % No ^ for the first line
        firstLine = false;
    else
        line = [line, '^', tline];  % Add ^ between lines
    end
    tline = fgetl(fileID);
end
len = length(line);
% Set the symbols
character_array =
["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V"
,"W","X","Y","Z",...

"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v",
"w","x","y","z","."...
```

```
                     ","
","0","1","2","3","4","5","6","7","8","9","!","[","]","{","}","(",")","^","/","-
","+","*","\"];
frequency = zeros(1,length(character_array));
probabilty = [];

% Take only unique characters in the file
no_repeat_string = unique(line)

%Creates a dictionary and find the frequency of symbols occuring in the
%file
d = dictionary(character_array, frequency);
for a = 1:length(line)
    d(line(a)) = lookup(d, line(a)) + 1;
end
for b = 1:length(no_repeat_string)
    probabilty(b) = lookup(d, no_repeat_string(b)) / sum(values(d));
end
probabilty
sum(probabilty)




% Function to convert a string to a cell
function cells = str2cell(str)
    str = char(str);
    cells = {};
    for i = 1:length(str)
        cells{end+1} = str(i);
    end
end


%  Function  to  create  the  huffman_dictionary
function dict = huffman_dict(symbols, probabilities)
    % Create initial nodes
    nodes = cell(length(symbols), 1);
    for i = 1:length(symbols)
        nodes{i} = struct('symbol', symbols{i}, 'prob', probabilities(i), 'left', [],
'right', []);
    end

    % Build Huffman tree
    while length(nodes) > 1
        % Sort nodes based on probability
        [~, idx] = sort(cellfun(@(n) n.prob, nodes));
        nodes = nodes(idx);

        % Combine two nodes with smallest probabilities
        left = nodes{1};
        right = nodes{2};
        new_node = struct('symbol', [], ...
                          'prob', left.prob + right.prob, ...
                          'left', left, ...
                          'right', right);

        % Replace first two nodes with their combination
        nodes = [{new_node}; nodes(3:end)];
    end

    % Recursively assign codes
    dict = containers.Map();  % Create empty dictionary
    assign_codes(nodes{1}, '', dict);  % Start from root
end

% Function to assign codes to the tree created by huffman_dict func
function assign_codes(node, code, dict)
    if isempty(node.left) && isempty(node.right)
        dict(node.symbol) = code;
        return;
    end
    if ~isempty(node.left)
        assign_codes(node.left, [code '0'], dict);
    end
    if ~isempty(node.right)
        assign_codes(node.right, [code '1'], dict);
    end
end

character_cell = str2cell(no_repeat_string)
dict = huffman_dict(character_cell,probabilty);

keysList = keys(dict);
keysList
values = values(dict)

% Function to encode the given file with the huffman encoding
function encodestream = encode(symbols,dict)
    encodestream = [];
    for i = 1:length(symbols)
        encodestream = [encodestream,dict(symbols(i))];
    end
end
disp(length(line));
encodestream = encode(line,dict)

% Function to check for match in encoded bits and the symbols
function index = indices(values,encoded)
    index = 0;
    for i = 1:length(values)
        if isequal(values(i),encoded)
            index = i;
        end
    end
end

% Function to decode the huffman encoded bits using the given huffman
% dictionary
function decodestream = decode(encoded,dict)
    bits = {};
    decodestream = [];
    key = keys(dict);
    value = values(dict);
    for i = 1:length(encoded)
        if isempty(bits)
            bits = "";
            temp = string(encoded(i));
            bits = {strcat(bits,temp)};
        else
            bits = string(bits(1));
            temp = string(encoded(i));
            bits = {strcat(bits,temp)};
        end
        if indices(value,bits) ~= 0
            index = indices(value,bits);
            decodestream = [decodestream,key(index)];
            bits = {};
        end
    end
end


decodestream = decode(encodestream,dict);
charArray=[decodestream{:}];
modifiedText = strrep(charArray, '^', sprintf('.\n'))
Lavg = 0;
encoded_values = values;

for i = 1:length(keysList)
    Lavg = Lavg + probabilty(i)*length(encoded_values{i});
end
Lavg
entropy = -1.*(probabilty*(log10(probabilty)./log10(2))')
Efficiency = (entropy/Lavg)*100
compression_ratio = (7*len)/length(encodestream)
```

TEST CASE:

This is the test case which is saved in a text file format.

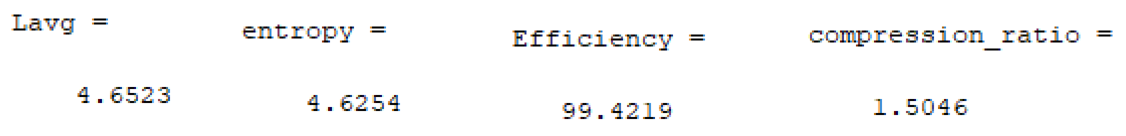
INFORMATION THEORY

Information theory is the mathematical study of the quantification, storage, and communication of information. The field was established and formalized by Claude Shannon in the 1940s,[1] though early contributions were made in the 1920s through the works of Harry Nyquist and Ralph Hartley. It is at the intersection of electronic engineering, mathematics, statistics, computer science, neurobiology, physics, and electrical engineering.[2][3]

A key measure in information theory is entropy. Entropy quantifies the amount of uncertainty involved in the value of a random variable or the outcome of a random process. For example, identifying the outcome of a fair coin flip (which has two equally likely outcomes) provides less information (lower entropy, less uncertainty) than identifying the outcome from a roll of a die (which has six equally likely outcomes). Some other important measures in information theory are mutual information, channel capacity, error exponents, and relative entropy. Important sub-fields of information theory include source coding, algorithmic complexity theory, algorithmic information theory and information-theoretic security.

Applications of fundamental topics of information theory include source coding/data compression (e.g. for ZIP files), and channel coding/error detection and correction (e.g. for DSL). Its impact has been crucial to the success of the Voyager missions to deep space,[4] the invention of the compact disc, the feasibility of mobile phones and the development of the Internet and artificial intelligence.[5][6][3] The theory has also found applications in other areas, including statistical inference,[7] cryptography, neurobiology,[8] perception,[9] signal processing,[2] linguistics, the evolution[10] and function[11] of molecular codes (bioinformatics), thermal physics,[12] molecular dynamics,[13] black holes, quantum computing, information retrieval, intelligence gathering, plagiarism detection,[14] pattern recognition, anomaly detection,[15] the analysis of music,[16][17] art creation,[18] imaging system design,[19] study of outer space,[20] the dimensionality of space,[21] and epistemology.[22].

PERFORMANCE ANALYSIS:



```
Lavg =          entropy =         Efficiency =       compression_ratio =

    4.6523          4.6254           99.4219              1.5046
```

OBSERVATION:

1. The code efficiently compresses text using Huffman encoding.
2. It accurately reconstructs the original text using a decode routine.
3. It calculates:
   - The average code length of the compressed representation.
   - The entropy, representing the ideal average bits per symbol.
   - The efficiency, which tells how close the encoding is to optimal.
   - The compression ratio, showing how much space has been saved.

INFERENCE:

1. The length of code for each symbol depends on the probability of those symbol (i.e. its frequency of occurrence).
2. When this file is to be encoded in ASCII format, it would require 7 bits/symbol. But when encoded using Huffman technique, around 4.5 bits/symbol is required which would give a compression ratio of 1.5
3. The compression ratio here is around 1.5, which denotes the file is reduced this much time.
4. Huffman encoding’s compression is not possible when the data is purely random, and doesn’t have any predictability.

RESULT:

Thus by using the Huffman encoding technique, a text file containing of alphanumeric characters and special symbols has been encoded and decoded.