Experiment No.:10

Title: Implementation of Huffman codes using suitable software.

Roll No.: ________ *Batch:* _________
Date of Performance: _______________
Date of Assessment: _________________

Particulars	**Marks**
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding (05)	
Total (20)	
Signature of Staff Member	

Experiment No.: 10

Title: Implementation of Huffman codes using suitable software.

Aim: To simulate and analyze the performance of a source coding technique, specifically Huffman coding, by compressing a data source and evaluating the compression efficiency and reconstructed data integrity.

Apparatus / Software Required:

1. MATLAB software (with Communications Toolbox recommended, although basic MATLAB functions suffice)
2. Computer with MATLAB installed

Theory:

Source coding is a process of encoding data from a source to reduce redundancy and represent the information efficiently, minimizing the number of bits required for transmission or storage. Huffman coding is a widely used lossless source coding technique that assigns variable-length codes to symbols based on their probability of occurrence. Symbols with higher probabilities get shorter codes, while less frequent symbols get longer codes, leading to overall compression. The coding process involves building a Huffman tree based on symbol probabilities and generating corresponding codeword's. Decoding recovers the original data from the encoded bit stream without loss.

1. Average code word length $\bar{L}$, is 5

$\bar{L}$ =∑ **nj p (xj)**

j =1

= 3*0.1+3*0.2+2*0.3+2*0.2+2*0.2

= 0.3+0.6+0.6+0.4+0.4

= 2.30

Average code word length $\bar{L}$ = 2.30 bits/symbol

2. Entropy is, 5

H(X) = ∑ **p (xj) log**2 **(1 / p (xj))**

j=1

= -0.1 log2 0.1 - 0.2 log2 0.2 - 0.3 log2 0.3 - 0.2 log2 0.2 - 0.2 log2 0.2

= 0.3321 + 0.4643 + 0.5210 + 0.4643 + 0.4643

Entropy H (X) = 2.2464 bits/symbol

3. Efficiency is given by,

$\eta = (H(X)/\bar{L})*100$

= (2.2464/2.30)*100

Efficiency is = 97.66%

Procedure / Outline:

1. Define an input symbol set and their corresponding probabilities/frequencies.
2. Use Huffman coding algorithm to generate the source codebook (codeword's for each symbol).
3. Encode a data sequence using the generated Huffman code.
4. Calculate the average codeword length and compare it with the source entropy to assess coding efficiency.
5. Decode the encoded bitstream back to the original data and verify the correctness.
6. Optionally, analyze compression ratio by comparing original data size with encoded data size.

Observation:

The average codeword length is close to the source entropy but never less, indicating optimal compression. The decoded data matches the original data perfectly, confirming lossless compression. Compression ratio demonstrates the efficiency gained by using Huffman coding compared to fixed-length coding.

MATLAB Code Example:

```
clc;
clear all;
close all;

% --- Inputs ---
m = input('Enter number of symbols = ');
p = zeros(1,m);
for i = 1:m
   p(i) = input(sprintf('Probability of symbol %d = ', i));
end
p = p / sum(p);              % normalize
```

```
symbols = 1:m;

% --- Initialize nodes and codes ---
nodesProb = p(:)';           % row vector of probabilities
nodesSymbols = cell(1,m);    % each cell holds the list of symbols in that node
for i = 1:m
   nodesSymbols{i} = i;
end
codes = cell(1,m);           % codes{symbol} holds its code (string)
for i = 1:m, codes{i} = ''; end

% --- Build Huffman codes (iterative merge) ---
while numel(nodesProb) > 1
   [~, idx] = sort(nodesProb);      % sort ascending
   i1 = idx(1); i2 = idx(2);        % two smallest
   % prepend bits: choose '0' for first, '1' for second (arbitrary)
   for s = nodesSymbols{i1}
      codes{s} = ['0' codes{s}];
   end
   for s = nodesSymbols{i2}
      codes{s} = ['1' codes{s}];
   end
   % merge nodes: update i1, remove i2
   nodesSymbols{i1} = [nodesSymbols{i1} nodesSymbols{i2}];
   nodesProb(i1) = nodesProb(i1) + nodesProb(i2);
   nodesSymbols(i2) = [];
   nodesProb(i2) = [];
end

% --- Display codes ---
disp('Huffman Codes:')
for i = 1:m
   fprintf('Symbol %d -> %s\n', i, codes{i});
end

% --- Metrics: average length, entropy, efficiency ---
lenVec = cellfun(@length, codes);
L = sum(p .* lenVec);                % avg code length (bits/symbol)
H = sum(p .* log2(1./p));            % entropy
fprintf('\nAvg length = %.4f bits/symbol\n', L);
fprintf('Entropy = %.4f bits/symbol\n', H);
fprintf('Efficiency = %.2f%%\n\n', (H/L)*100);

% --- Generate random message according to p (no randsrc) ---
N = m;                               % test message length (you can change)
```

```
cdf = cumsum(p);
r = rand(N,1);
msg = arrayfun(@(x) find(cdf>=x,1), r)'; % row vector of sampled symbols

% --- Encode ---
enc = '';
for i = 1:N
   enc = [enc codes{msg(i)}];
end

% --- Decode (simple prefix-match) ---
dec = [];
buf = '';
for k = 1:length(enc)
   buf = [buf enc(k)];
   matched = false;
   for s = 1:m
      if strcmp(buf, codes{s})
         dec = [dec s];
         buf = '';
         matched = true;
         break;
      end
   end
   % (if no match, continue accumulating bits)
end

fprintf('Encoding/Decoding OK? %d\n', isequal(msg, dec));
```

Output:

```
Enter number of symbols = 5

Probability of symbol 1 = 0.1

Probability of symbol 2 = 0.2

Probability of symbol 3 = 0.3

Probability of symbol 4 = 0.2

Probability of symbol 5 = 0.2

Huffman Codes:

Symbol 1 -> 110

Symbol 2 -> 111

Symbol 3 -> 10

Symbol 4 -> 00
Symbol 5 -> 01

Avg length = 2.3000 bits/symbol
Entropy = 2.2464 bits/symbol
Efficiency = 97.67%

Encoding/Decoding OK? 1
```

Conclusion: