Name: Vivek More **Batch:** M13

Roll no: 35141

```
%====================================================================
% Experiment No. 9: To Implementation of Shannon Fano codes by using suitable software.
%====================================================================

clc;
clear all;
close all;
%====================================================================
% --- Recursive Function for Shannon-Fano Algorithm (Final Version) ---
%====================================================================
function codes = shannon_fano_encoder(symbols, probs)
    n = length(probs);
    codes = cell(n, 1); % Initialize cell array for codes

    if n > 1
        % Find the split point that makes the two probability sums as close as possible
        cumulative_probs = cumsum(probs);
        total_prob = sum(probs);
        min_diff = inf;
        split_idx = 0;
        for i = 1:(n-1)
            diff = abs(2*cumulative_probs(i) - total_prob);
            if diff < min_diff
                min_diff = diff;
                split_idx = i;
```

```
        end
    end

    codes1 = shannon_fano_encoder(symbols(1:split_idx), probs(1:split_idx));
    codes2 = shannon_fano_encoder(symbols(split_idx+1:n), probs(split_idx+1:n));
    for i = 1:length(codes1)
        codes{i} = ['0', codes1{i}];
    end
    for i = 1:length(codes2)
        codes{split_idx + i} = ['1', codes2{i}];
    end
  end
end

% Define the symbols and their corresponding probabilities from the problem statement
symbols = {1, 2, 3, 4, 5, 6};
probabilities = [0.3, 0.25, 0.2, 0.12, 0.08, 0.05];

% --- 1. Source Entropy Calculation (Same for Both Methods) ---
% H = -sum(p_i * log2(p_i))
entropy = -sum(probabilities .* log2(probabilities));



%==========================================================================
% PART I: Shannon-Fano Coding Implementation
%==========================================================================

% The Shannon-Fano algorithm is implemented using a recursive function.
% First, sort the probabilities and symbols in descending order.
```

```
[sorted_probs, sort_order] = sort(probabilities, 'descend');
sorted_symbols = symbols(sort_order);

shannon_fano_codes_sorted = shannon_fano_encoder(sorted_symbols, sorted_probs);
shannon_fano_codes = cell(size(symbols));
shannon_fano_codes(sort_order) = shannon_fano_codes_sorted;

% --- Calculations for Shannon-Fano ---
sf_lengths = cellfun(@length, shannon_fano_codes); % Length of each codeword
sf_avg_length = sum(probabilities .* sf_lengths);  % Average length (L)
sf_efficiency = entropy / sf_avg_length;         % Efficiency (eta)
sf_redundancy = 1 - sf_efficiency;             % Redundancy (gamma)

%=========================================================================
% PART II: Huffman Coding Implementation (Corrected)
%=========================================================================

% The Huffman algorithm is implemented using the built-in Octave function.
numeric_symbols = 1:length(symbols);
huffman_dict = huffmandict(numeric_symbols, probabilities);
huffman_codes = huffman_dict;

% --- Calculations for Huffman ---
huff_lengths = cellfun(@length, huffman_codes);
huff_avg_length = sum(probabilities .* huff_lengths);
huff_efficiency = entropy / huff_avg_length;
huff_redundancy = 1 - huff_efficiency;
clc;
```

```
%==========================================================================
% --- Display Final Results ---
%==========================================================================

% --- Shannon-Fano Results Table ---
fprintf('=====================================================\n');
fprintf('         PART I: Shannon-Fano Code Results\n');
fprintf('=====================================================\n');
disp('Generated Codes:');
for i = 1:length(symbols)
   fprintf('  Symbol x%d (p=%.2f): %s\n', symbols{i}, probabilities(i),
shannon_fano_codes{i});
end
disp('-----------------------------------------------------');
fprintf('1. Source Entropy (H)       : %.4f bits/symbol\n', entropy);
fprintf('2. Average Code Length (L)  : %.4f bits/symbol\n', sf_avg_length);
fprintf('3. Code Efficiency (eta)    : %.4f or %.2f%%\n', sf_efficiency, sf_efficiency * 100);
fprintf('4. Redundancy (gamma)       : %.4f or %.2f%%\n', sf_redundancy, sf_redundancy *
100);
fprintf('=====================================================\n\n');



% --- Huffman Results Table ---
fprintf('=====================================================\n');
fprintf('          PART II: Huffman Code Results\n');
fprintf('=====================================================\n');
disp('Generated Codes:');
for i = 1:length(symbols)
   % Convert the numeric vector code (e.g., [1 0 1]) to a text string ('101') for printing.
   code_as_string = strrep(num2str(huffman_codes{i}), ' ', '');
```

```
    fprintf('  Symbol %s (p=%.2f): %s\n', symbols{i}, probabilities(i), code_as_string);
end
disp('--------------------------------------------------');
fprintf('1. Source Entropy (H)       : %.4f bits/symbol\n', entropy);
fprintf('2. Average Code Length (L)  : %.4f bits/symbol\n', huff_avg_length);
fprintf('3. Code Efficiency (eta)    : %.4f or %.2f%%\n', huff_efficiency, huff_efficiency * 100);
fprintf('4. Redundancy (gamma)       : %.4f or %.2f%%\n', huff_redundancy, huff_redundancy * 100);
fprintf('==================================================\n');
```

OUTPUT:

```
==================================================
         PART I: Shannon-Fano Code Results
==================================================
Generated Codes:
  Symbol x1 (p=0.30): 00
  Symbol x2 (p=0.25): 01
  Symbol x3 (p=0.20): 10
  Symbol x4 (p=0.12): 110
  Symbol x5 (p=0.08): 1110
  Symbol x6 (p=0.05): 1111
--------------------------------------------------
1. Source Entropy (H)       : 2.3601 bits/symbol
2. Average Code Length (L)  : 2.3800 bits/symbol
3. Code Efficiency (eta)    : 0.9917 or 99.17%
4. Redundancy (gamma)       : 0.0083 or 0.83%
==================================================

==================================================
          PART II: Huffman Code Results
==================================================
Generated Codes:
  Symbol   (p=0.30): 00
  Symbol   (p=0.25): 01
  Symbol   (p=0.20): 11
  Symbol   (p=0.12): 101
  Symbol   (p=0.08): 1000
  Symbol   (p=0.05): 1001
--------------------------------------------------
1. Source Entropy (H)       : 2.3601 bits/symbol
2. Average Code Length (L)  : 2.3800 bits/symbol
3. Code Efficiency (eta)    : 0.9917 or 99.17%
4. Redundancy (gamma)       : 0.0083 or 0.83%
==================================================
>>
```