Table of Contents

..... 1
STEP 1: Read grayscale image (BEST FIX) 1
STEP 2: Flatten image into 1D array 1
STEP 3: Calculate symbol frequencies 2
STEP 4: Build Huffman Tree 2
STEP 5: Generate Huffman Codes 2
STEP 6: Huffman Encode 2
STEP 7: Huffman Decode 3
STEP 8: Display Images 3
STEP 9: Compression Statistics 4
---------- 5
---------- 5


```
% --------------------------------
% Huffman Coding for Grayscale Image
% (Toolbox-Free, MATLAB Online Safe)
% --------------------------------
% Author: BT23ECE100

clc;
clear;
close all;
```

STEP 1: Read grayscale image (BEST FIX)

```
[file, path] = uigetfile({'*.jpg;*.png;*.bmp;*.tif', ...
                          'Image Files (*.jpg, *.png, *.bmp, *.tif)'}, ...
                          'Select a grayscale image');

if isequal(file,0)
    error('Image selection cancelled. Please select an image file.');
end

img = imread(fullfile(path, file));

if size(img,3) == 3
    img = rgb2gray(img);
end

img = uint8(img);
```

STEP 2: Flatten image into 1D array

```
pixels = img(:);
```

STEP 3: Calculate symbol frequencies

```
symbols = unique(pixels);
freq = zeros(length(symbols),1);

for i = 1:length(symbols)
    freq(i) = sum(pixels == symbols(i));
end

prob = freq / sum(freq);
```

STEP 4: Build Huffman Tree

```
nodes = struct('symbol', {}, 'prob', {}, 'left', {}, 'right', {});

for i = 1:length(symbols)
    nodes(i).symbol = symbols(i);
    nodes(i).prob   = prob(i);
    nodes(i).left   = [];
    nodes(i).right  = [];
end

while length(nodes) > 1
    % Sort nodes by probability (ascending)
    [~, idx] = sort([nodes.prob]);
    nodes = nodes(idx);

    % Merge two least probable nodes
    newNode.symbol = [];
    newNode.prob   = nodes(1).prob + nodes(2).prob;
    newNode.left   = nodes(1);
    newNode.right  = nodes(2);

    nodes(1:2) = [];
    nodes(end+1) = newNode;
end

huffmanTree = nodes;
```

STEP 5: Generate Huffman Codes

```
codes = containers.Map('KeyType','double','ValueType','char');
codes = generateCodes(huffmanTree, '', codes);
```

STEP 6: Huffman Encode

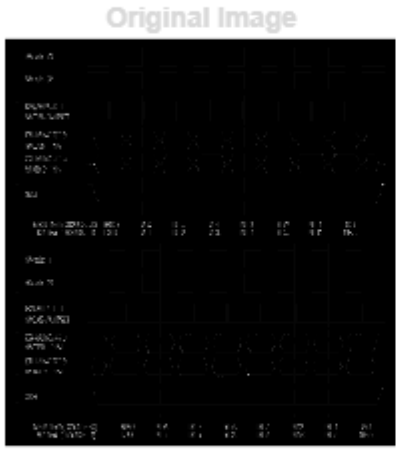
```
encoded_bits = '';
for i = 1:length(pixels)
    encoded_bits = [encoded_bits codes(double(pixels(i)))]; %#ok<AGROW>
end
```

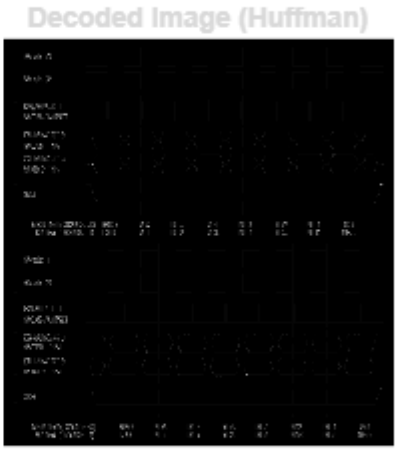
STEP 7: Huffman Decode

```
decoded_pixels = zeros(size(pixels), 'uint8');
currentNode = huffmanTree;
idx = 1;

for i = 1:length(encoded_bits)
    if encoded_bits(i) == '0'
        currentNode = currentNode.left;
    else
        currentNode = currentNode.right;
    end

    % Leaf node reached
    if isempty(currentNode.left) && isempty(currentNode.right)
        decoded_pixels(idx) = uint8(currentNode.symbol);
        idx = idx + 1;
        currentNode = huffmanTree;
    end
end

decoded_img = reshape(decoded_pixels, size(img));
```

STEP 8: Display Images

```
figure('Name','Huffman Image Compression','Color','w');

subplot(1,2,1);
imshow(img);
title('Original Image');
axis off;

subplot(1,2,2);
imshow(decoded_img);
title('Decoded Image (Huffman)');
axis off;
```

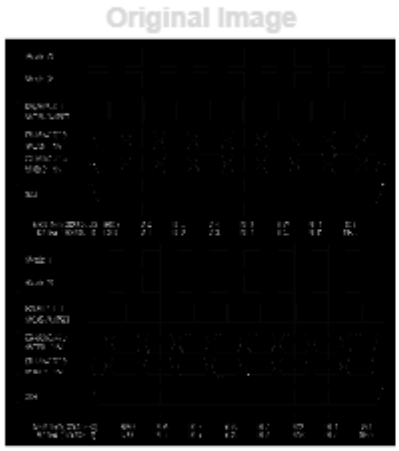


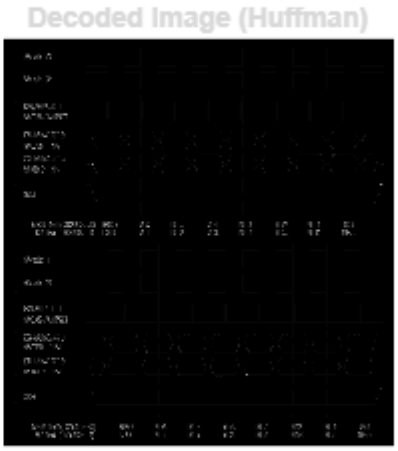


STEP 9: Compression Statistics

```
original_size   = numel(pixels) * 8;      % bits
compressed_size = length(encoded_bits);   % bits
compression_ratio = original_size / compressed_size;

fprintf('\n--- Compression Statistics ---\n');
fprintf('Original size (bits):   %d\n', original_size);
fprintf('Compressed size (bits): %d\n', compressed_size);
fprintf('Compression ratio:      %.2f\n', compression_ratio);
fprintf('------------------------------\n');
```

```
--- Compression Statistics ---
Original size (bits):   3834768
Compressed size (bits): 685037
Compression ratio:      5.60
------------------------------
```





Local Function (MUST be at end of file)

```
function codes = generateCodes(node, code, codes)

    if isempty(node.left) && isempty(node.right)
        codes(node.symbol) = code;
        return;
    end

    codes = generateCodes(node.left,  [code '0'], codes);
    codes = generateCodes(node.right, [code '1'], codes);
end
```

Published with MATLAB® R2025b