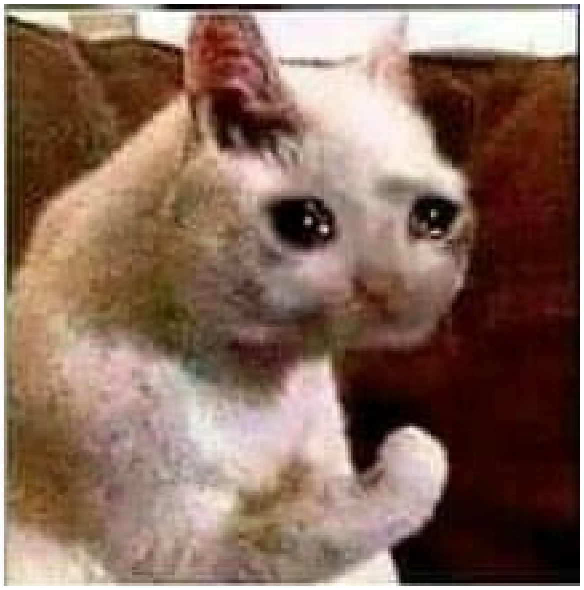
Table of Contents

```
clc;
clear all;
close all;
```

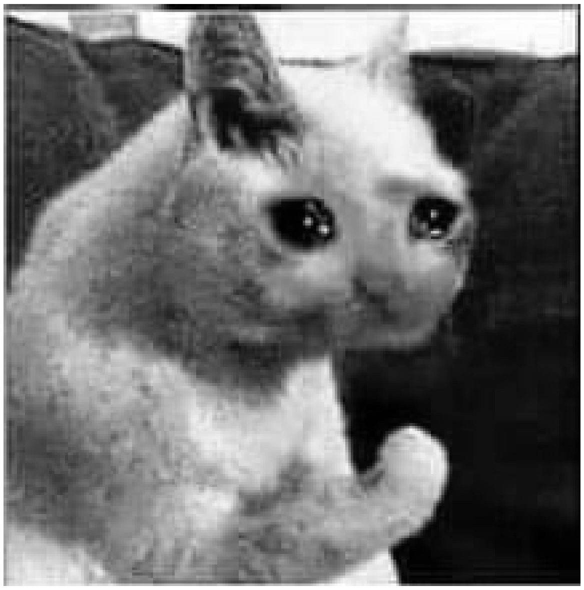
Import Image:

```
RGB = imread('finallydidit.png'); % for any random image
imshow(RGB)
title('RGB image');
RGB = rgb2gray(RGB);
figure
imshow(RGB)
%RGB = imread('test.png'); % this image has equal probs
%imshow(RGB)
title('input Gray image');
```

RGB image



input Gray image



Calculate probabilities of each unique symbol:

```
sym=unique(RGB);
sizergb(1:2)=size(RGB);
for s=1:length(sym)
    i=find(RGB==sym(s));
    prob_dist(s)=length(i)/(sizergb(1).*sizergb(2));
end
```

Testing Example:

```
%sym =[1 2 3 4 5];
%prob_dist =[0.2 0.2 0.2 0.2 0.2];
%sym =1:length(prob_dist);
```

Sum all probrobilities to check that (Total probs = 1):

```
total = sum(prob_dist);
display(total)
```

```

total =

    1.0000

```

initialize probs:

```
for i = 1:length(sym)
    sorted_sym{i} = sym(i);
end
init_sym = sorted_sym;
init_prob = prob_dist;
```

sorting and combing probs:

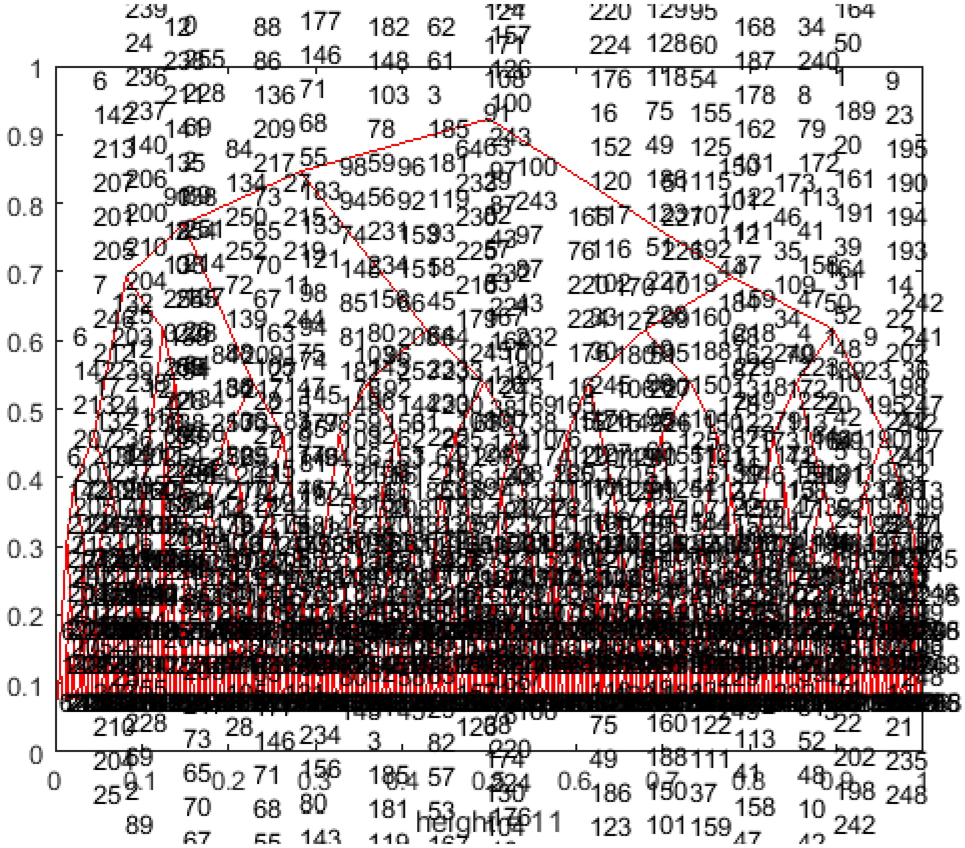
```
sorted_prob = prob_dist;
count = 1;
while (length(sorted_prob) > 1)
    % Sort probs
    [sorted_prob,indeces] = sort(sorted_prob,'ascend');
    % Sort symbol based on indeces
    sorted_sym = sorted_sym(indeces);
    % Create new symbol
    new_node = strcat(sorted_sym(2),sorted_sym(1));
    new_prob = sum(sorted_prob(1:2));
    % Dequeue used symbols from "old" queue
    sorted_sym =  sorted_sym(3:length(sorted_sym));
    sorted_prob = sorted_prob(3:length(sorted_prob));
    % Add new symbol back to "old" queue
    sorted_sym = [sorted_sym, new_node];
    sorted_prob = [sorted_prob, new_prob];
    % Add new symbol to "new" queue
    newq_sym(count) = new_node;
    newq_prob(count) = new_prob;
    count = count + 1;
end
```

Sort all tree elements and applying Huffman Tree :

```
tree = [newq_sym,init_sym];
tree_prob = [newq_prob, init_prob];
[sorted_tree_prob,indeces] = sort(tree_prob,'descend');
sorted_tree = tree(indeces);
parent(1) = 0;
for i = 2:length(sorted_tree)
    % Extract symbol
    me = sorted_tree{i};
    % Find parent's symbol (search until shortest match is found)
    count = 1;
    parent_maybe = sorted_tree{i-count};
    different = strfind(parent_maybe,me);
    while (isempty(different))
        count = count + 1;
        parent_maybe = sorted_tree{i-count};
        different = strfind(parent_maybe,me);
    end
    parent(i) = i - count;
end
```

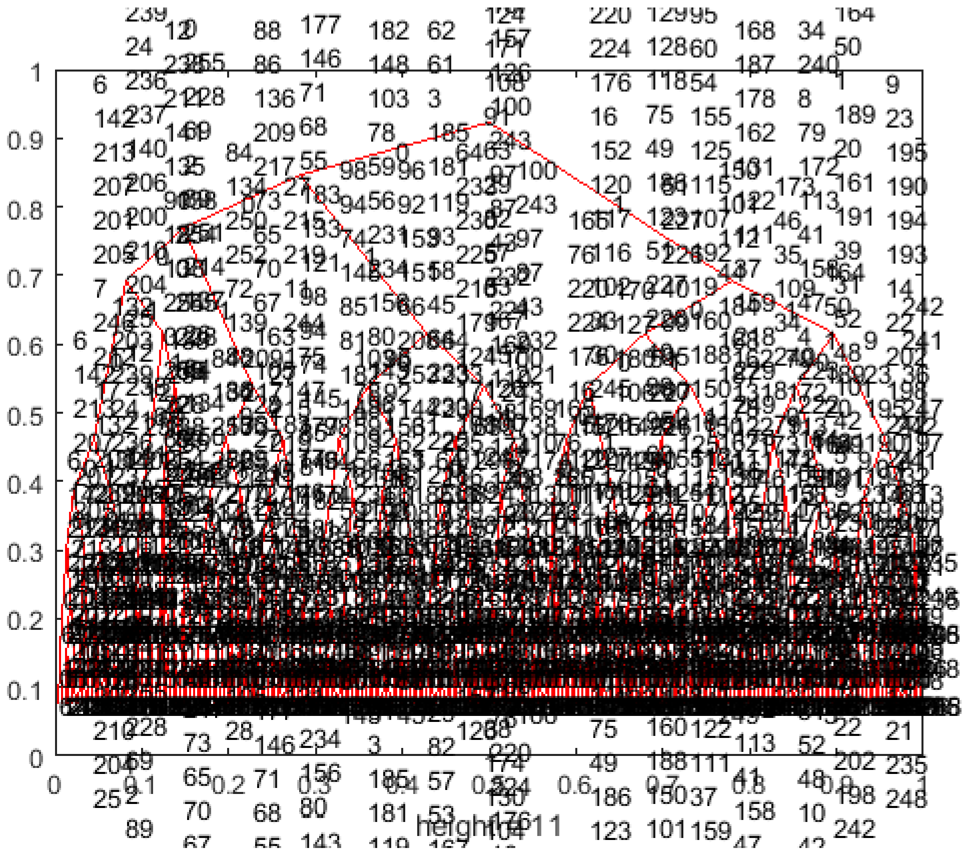
plot Huffman Tree:

```
figure
treeplot(parent);
%title(strcat('Huffman Coding Tree - "',sym,'"'));
[xs,ys,h,s] = treelayout(parent);
text(xs,ys,sorted_tree);
```



Assign 0 and 1 to symbols in Huffman Tree:

```
for i = 2:length(sorted_tree)
    % Get my coordinate
    my_x = xs(i);
    my_y = ys(i);
    % Get parent coordinate
    parent_x = xs(parent(i));
    parent_y = ys(parent(i));
    % Calculate weight coordinate (midpoint)
    mid_x = (my_x + parent_x)/2;
    mid_y = (my_y + parent_y)/2;
    % Calculate weight (positive slope = 0, negative = 1)
    slope  = (parent_y - my_y)/(parent_x - my_x);
    if (slope < 0)
        weight(i) = 1; %assign 1
    else
        weight(i) = 0; %assign 0
    end
    text(mid_x,mid_y,num2str(weight(i)));
end
```



Extract all codewords from Huffman Tree:

```
for i = 1:length(sorted_tree)
 code{i} = '';
 index = i;
 p = parent(index);
 while(p ~= 0)
  % Turn weight into code [bits(0,1) to string]
  w = num2str(weight(index));
  % Concatenate code symbol
  code{i} = strcat(w,code{i});
  % Continue towards root
  index = parent(index);
  p = parent(index);
 end
end
allcodeword = [sorted_tree', code'];
```

Extract the codewords for the probs:

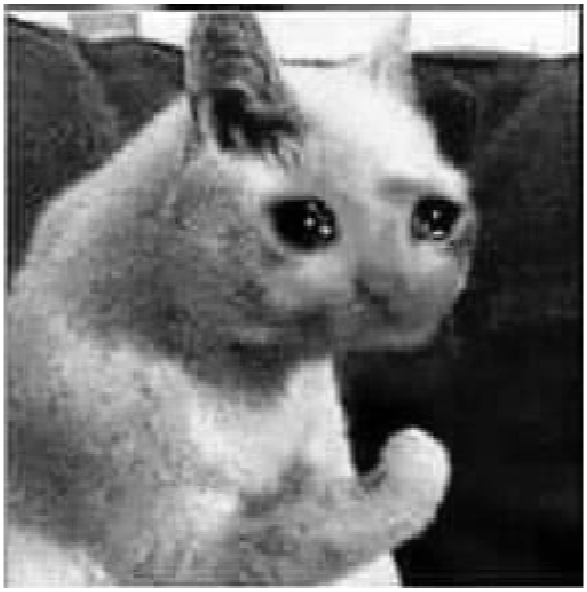
```
for k=1:length(init_sym)
    for kk=1:length(allcodeword)
        if (length(allcodeword{kk}) == length(init_sym{k})...
                && allcodeword{kk}==init_sym{k})
            codeword{k}=allcodeword{kk,2};
        end
    end
end
sym_codeword=[init_sym',codeword'];
```

Mapping the image (convert pixels to bits):

```
for c=1:length(sym)
    for cc=1:sizergb(1)
        for ccc=1:sizergb(2)
        if (sym_codeword{c,1}==RGB(cc,ccc))
            rgb_encoded{cc,ccc}=sym_codeword{c,2};
        end
        end
    end
end
```

Decoding the encoded image (convert bits to pixels)

```
for c=1:length(sym)
    for cc=1:sizergb(1)
        for ccc=1:sizergb(2)
            if (length(sym_codeword{c,2}) ==
 length(rgb_encoded{cc,ccc}))
                if (sym_codeword{c,2} == rgb_encoded{cc,ccc})
                rgb_decoded{cc,ccc}=sym_codeword{c,1};
                end
            end
        end
    end
end
RGB_decoded=cell2mat(rgb_decoded);
imwrite(RGB_decoded,'test.png')
Output_RGB = imread('test.png');
imshow(Output_RGB)
title('Output Gray image');
```



Output Gray image

the sizes of the input image and the output one from source coding:

```
size_of_image=(sizergb(1)*sizergb(1)*8);
display(size_of_image)
encodedword_to_char=strjoin(rgb_encoded);
bits_of_encodedword=strrep(encodedword_to_char,' ','');
size_of_encoded_image=length(bits_of_encodedword);
display(size_of_encoded_image)
```

```

size_of_image =

      380192


size_of_encoded_image =

      372204

```

compression ratio of the coded image:

```
compression_ratio=(size_of_encoded_image)./(size_of_image);
display(compression_ratio)
```

```

compression_ratio =

    0.9790

```

The End

Published with MATLAB® R2016a