Huffman coding creates optimal prefix codes. For its implementation, we need the probabilities of occurrence of each symbol that we found in the previous question. Additionally, the entropy of the source must be calculated because it defines the limit of optimal compression, so that we can finally compare the code length with this and see if our coding is efficient.

Επίλυση :

i) **Solution:**

i) We will use the formula Σ = p * $\log_2(1/p)$.
The entropy of the encoding is requested, so I will calculate the entropy of the source. Since the probabilities do not change after encoding, nor does the number of discrete states decrease, it is correct to say that **Source Entropy = Encoding Entropy**.
We write the command:

```
entropia_pigis = sum(pithanotites .* log2(1 ./ pithanotites));
```

and we get **H(x) = 3.7831**.

With quick calculations, I verify the result.

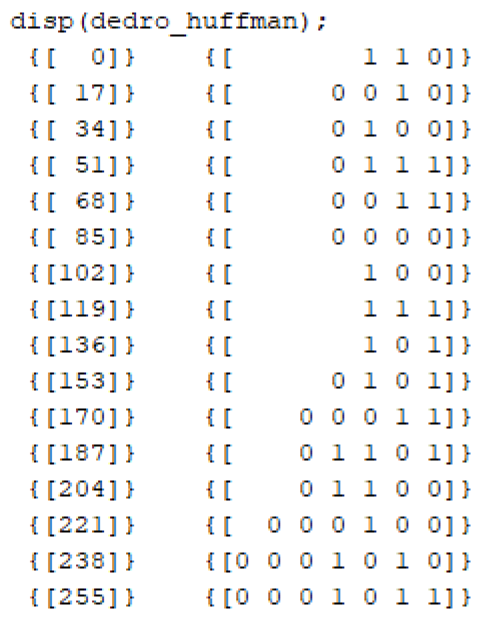
```
0.0962 * log(10.3950103950104) = 0.0962 * 3.378 = 0.3249636
0.0814 * log(12.28501228501229)= 0.0814 * 3.619 = 0.2945866
0.0683 * log(14.64128843338214)= 0.0683 * 3.872 = 0.2644576
0.0625 * log(16)= 0.0625 * 4 = 0.25
0.0768 * log(13.020833333333330)= 0.0768 * 3.703 = 0.2843904
0.0905 * log(11.04972375690608)= 0.0905 * 3.466 = 0.313673
0.1132 * log(8.833922261484099)= 0.1132 * 3.143 = 0.3557876
0.0906 * log(11.03752759381898)= 0.0906 * 3.464 = 0.3138384
0.0965 * log(10.36269430051813)= 0.0965 * 3.373 = 0.3254945
0.0671 * log(14.90312965722802)= 0.0671 * 3.8975 = 0.26152225
0.0389 * log(25.70694087403599)= 0.0389 * 4.684 = 0.1822076
0.0329 * log(30.3951367781155)=  0.0329 * 4.926 = 0.1620654
0.0337 * log(29.67359050445104)= 0.0337 * 4.891 = 0.1648267
0.0259 * log(38.61003861003861)= 0.0259 * 5.271 = 0.1365189
0.0224 * log(44.64285714285714)= 0.0224 * 5.48 = 0.122752
0.0031 * log(322.5806451612903)= 0.0031 * 8.334 = 0.0258354
```

```
0.3249636+0.2945866+0.2644576+0.25+0.2843904+0.313673+0.3557876+0.3138384+0.3254945+0.26152225+0.1822076+0.1620654+0.1648267+0.1365189+0.122752+0.0258354
=
3.78291995
```

ii) To calculate the code length, I will first construct the Huffman tree.

It must be constructed so that codes can be assigned to the source symbols. Then, according to what we found, all the pixels of the image are encoded. The result will be a long string of 0s and 1s. The encoding (huffmanenco) is not necessary for finding the code length, but it will be needed in a subsequent question.
Let's first create the tree:

```
dedro_huffman= huffmandict(simvola_pigis, pithanotites);
```

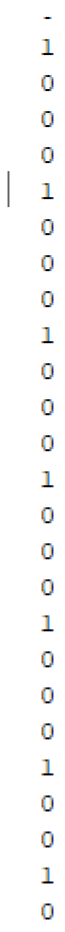


```
disp(dedro_huffman);
  {[  0]}    {[        1 1 0]}
  {[ 17]}    {[      0 0 1 0]}
  {[ 34]}    {[      0 1 0 0]}
  {[ 51]}    {[      0 1 1 1]}
  {[ 68]}    {[      0 0 1 1]}
  {[ 85]}    {[      0 0 0 0]}
  {[102]}    {[        1 0 0]}
  {[119]}    {[        1 1 1]}
  {[136]}    {[        1 0 1]}
  {[153]}    {[      0 1 0 1]}
  {[170]}    {[    0 0 0 1 1]}
  {[187]}    {[    0 1 1 0 1]}
  {[204]}    {[    0 1 1 0 0]}
  {[221]}    {[  0 0 0 1 0 0]}
  {[238]}    {[0 0 0 1 0 1 0]}
  {[255]}    {[0 0 0 1 0 1 1]}
```

And immediately after, the encoding:

```
kodikopoihsh = huffmanenco(I(:), dedro_huffman);
```

As we said, thanks to the list of codes we found above, it assigns codes to all the pixels of the image.



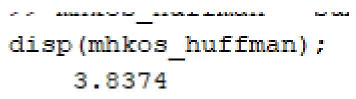
```
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
1
0
```

We observe that the result is a huge string of 0s and 1s, since the image has many pixels, a huge number of bits will be required for its encoding.

Next, let's calculate the code length. This will be done using the formula Σ (p * L).

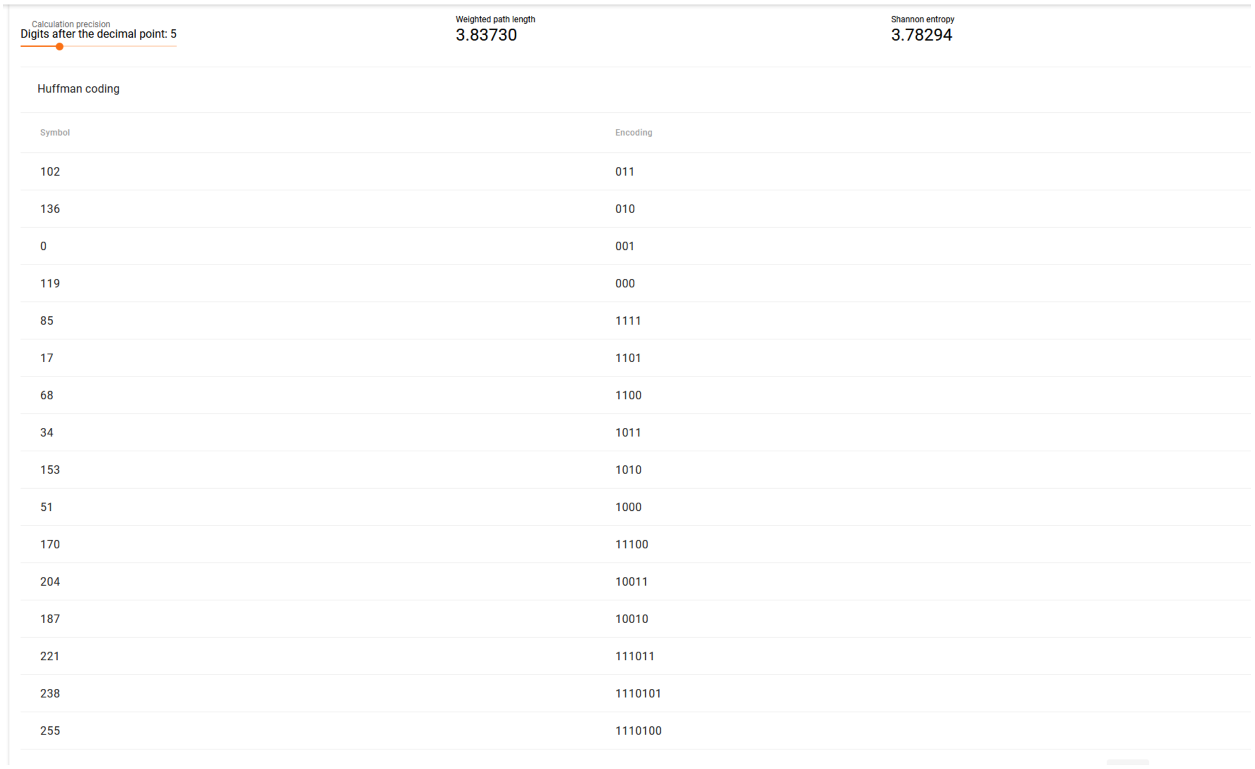
```
mhkos_huffman = sum(pithanotites .* arrayfun(@(x)
length(dedro_huffman{x, 2}), 1:size(dedro_huffman, 1)));
```

With this code, I first created a list containing the Huffman tree. The arrayfun function examines each number in the list to see what length each element has. Then it multiplies the length of the element by its probability of occurrence, and finally, it adds all the results together.
So, the length is:



```
disp(mhkos_huffman);
     3.8374
```

Using a calculator, I confirmed the results.



Calculation precision
Digits after the decimal point: 5

Weighted path length
3.83730

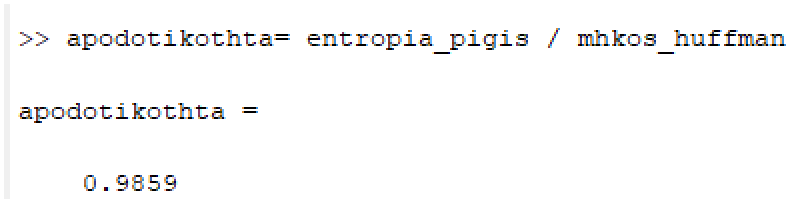
Shannon entropy
3.78294

Huffman coding

Symbol	Encoding
102	011
136	010
0	001
119	000
85	1111
17	1101
68	1100
34	1011
153	1010
51	1000
170	11100
204	10011
187	10010
221	111011
238	1110101
255	1110100

iii)

To find the efficiency I must do the following calculation: Entropy/length of the code



```
>> apodotikothta= entropia_pigis / mhkos_huffman

apodotikothta =

    0.9859
```

We find that it is η=98.59%.