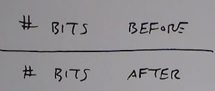
**LOSSLESS IMAGE CODING**

Used in:

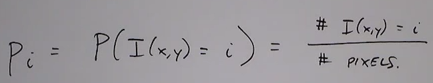
* Medical diagnosis
* Legal reasons
* Archival
* Satellite imagery
* High Quality Analysis
* And mostly file compression

To measure the performance of our compression we can use the **compression ratio**:

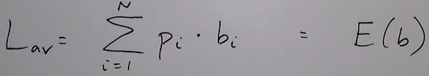


For real images, lossless compression ratios can be in the range of 2 – 10

Consider the image histogram to be a probability mass function:



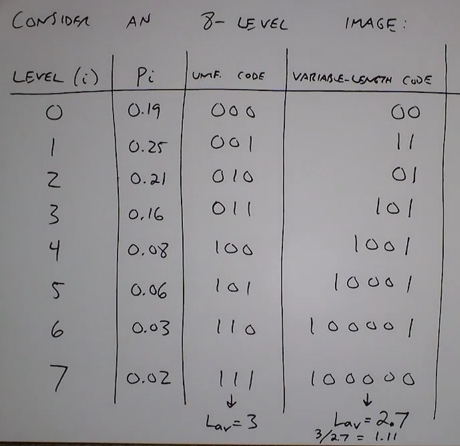
The average bits per pixel:



For uniform coding without compression:  i.e. 8 bits.

The key is to code more frequently occurring pixels with fewer bits and less frequently occurring pixels with more bits.

Coding of pixels is done using Huffman Coding

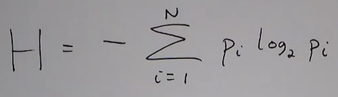


Compression ratio = 1.11

Now two questions pop:

1. What is the best code?
2. What are the fundamental limits of how to code? What is the minimum?

Entropy decides the best code



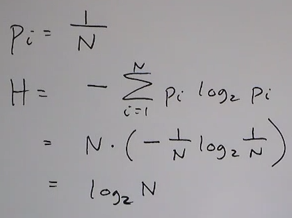
* Why log(2) -> because we are dealing with bits (0 & 1)
* Why negative sign -> log of number less than zero is negative

Entropy measures how uncertain a random variable is.

- worst case: Uniform pmf

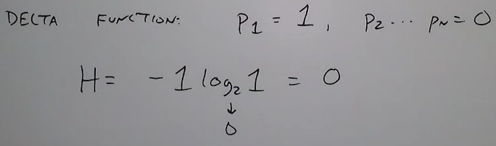
- best case: delta function all pixels in one bin

1. So for a uniform distribution (worst case):



For image having 8 levels it would be 3

1. For delta function (best case):



The entropy is the lowest number of average bits per symbol that can be used to code a distribution

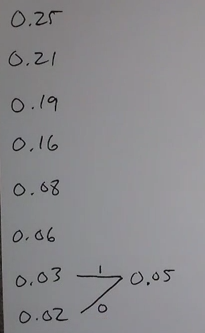
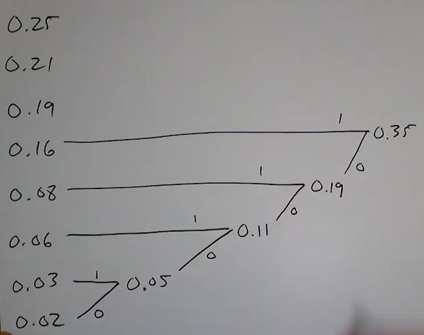
This answers the first question of what the best code is. Now the question of how to design a code that approaches the best code.

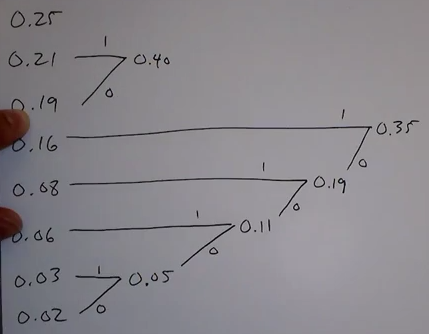
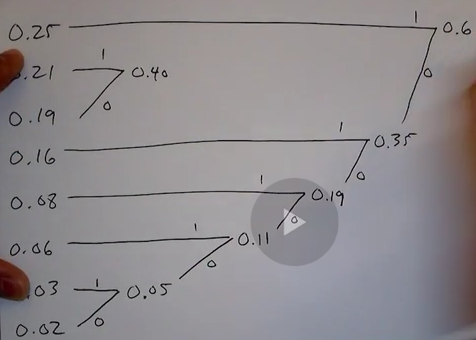
**Huffman Coding**

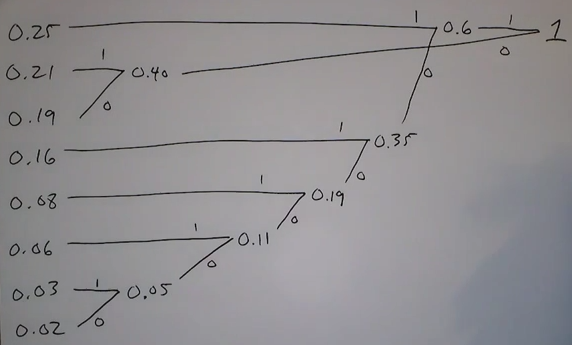
It is the optimal (minimum average number of bits) for independently coding N symbols.

1. Arrange the symbols in decreasing probability
2. Merge the 2 nodes with lowest probability
3. Assign 0/1 to top/bottom
4. Iterate this process till done
5. Read code from root-to-leaf

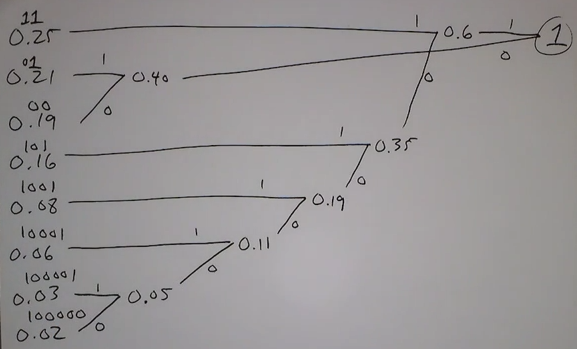
We will consider the same example as above:

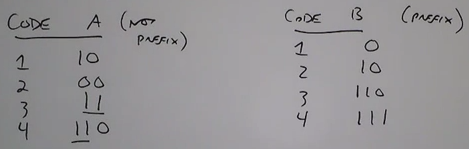


The final code is:



The Huffman code is a prefix code meaning no code word is a prefix of any other codeword.

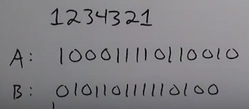
Consider the following two codewords:



* In code B none of the symbol start with any other symbol
* But in code A symbol 4 has same prefix as symbol 3

So while decoding in code A, system does not know whether to consider it as ‘3’ or ‘4’.

Following is the resulting code of A and B for this number:

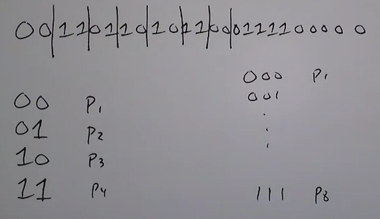


In B everything is unique there you can clearly demarcate when to stop: 

But in code A it will have to wait for symbol to be decoded to assure it has made correct decision. 

Hence a ‘prefix’ codes can be decoded instantaneously.

Here we are coding independently. But we can do better by considering pairs of symbols or more.



Assigning Huffman code to ‘pairs’ or ‘triplets’ of binary symbols. Not to be considered for huge amount of data like very big images. Can be used when the levels are less (2-levels, 4 – levels, 8 – levels) anything beyond this would be time consuming.

Another variation is the Truncated Huffman Coding

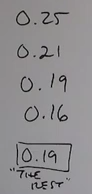
* One disadvantage of Huffman code is that the code of very rarely occurring symbols can be extremely long. (consider a 256 level image, it would be too long)

To avoid extremely long codewords for rarely occurring symbols:

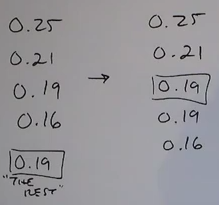
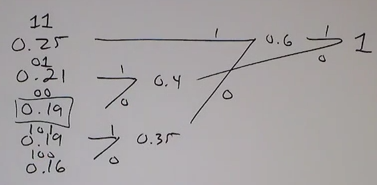
* Huffman code the most probable ‘k’ symbols in the source
* Replace the rest with a prefix (+) a fixed length code.

Now consider the same example used for Huffman coding:

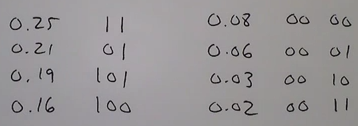
* Combine the last 4 symbols to one (‘the rest’)



* Now arrange them in decreasing order of probability:

Now the final coding (left the top four frequently occurring symbols and right is the least occurring symbols (‘the rest’))



Here the average length is . Not as good as the other techniques but would be better if the length of the levels were much higher.

***Lempel-Ziv Coding:*** (basis for .gif, .tiff, .png, .zip):

pujitha@12345

9952450434