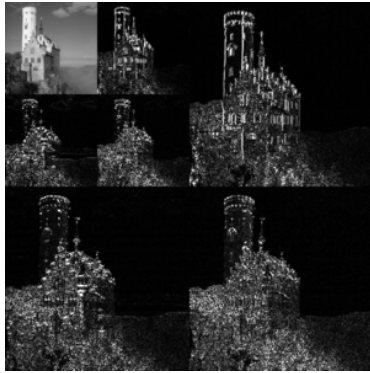


## Lesson 24: Use Smart Encoding!

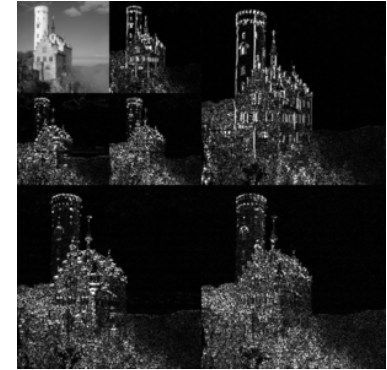


*File compression, Huffman codes, wavelets*

## File Compression

- File compression comes in two varieties
  - ★ Exact compression (e.g. zip used on text files)
  - ★ Lossy compression (e.g. jpeg used on pictures—jpeg can also be loss-less or exact)■
- Good exact compression (also known as entropy encodings) can give a compression ratio around 25%■
- Lossy compression can give a compression ratio from 10-1%■
- Important for saving space, but lossy compression can also be used for noise reduction■
- Even used for plagiarism detection!■

1. **Huffman codes**
2. Wavelets



## Entropy Encoding

- Exact encodings use the principle of using short words for frequently occurring sequences (symbols) and longer words for sequences that occur less often■
- Claude Shannon showed that for an alphabet of  $n$  symbols where the probability of symbol  $i$  occurring is  $p_i$  no code exists which can transmit information in less than

$$-\sum_{i=1}^n p_i \log_2(p_i) \text{ bits}$$

asymptotically this compression can be achieved■

- Different encoding schemes differ in the way they identify symbols of the alphabet■—this is rather specialist and we won't go into this■

## Huffman Coding

- Given a sequence of symbols and their probabilities of occurrence, Huffman code provides a way of coding up the information
- It is an example of a **greedy** strategy that happens to be optimal
- Like many greedy strategies it is easily implemented using a priority queue
- It is used in the UNIX compress program and in the exact part of JPEG
- The idea is to assign short codes to commonly used symbols

## Symbol Frequency

- We start from an alphabet describing the original document
  - This might be the set of characters
  - For an image it might be the set of pixel values
  - It might be pairs of pixel values
- We compute the number of occurrences of each symbol

Symbol	# Occurrences
a	145
b	67
:	:

## Encoding

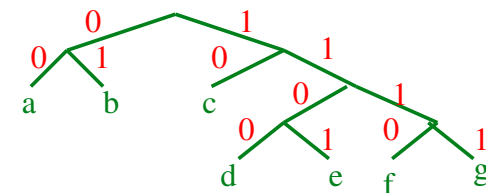
- We want to assign a code to each symbol
- To save space we want to assign short codes to frequently used symbols
- There is a problem: decoding
- If we assigned a code

$e \rightarrow 0$	$a \rightarrow 1$	$r \rightarrow 01$
$o \rightarrow 10$	$i \rightarrow 11$	$t \rightarrow 000$

etc. we could compress a document very efficiently but we could never decode it uniquely

## Huffman Trees

- Once again tree come to the rescue!
- We assign each symbol to a leaf of a binary tree
- We use the position of the branch as an encoding of the symbol



a	→	00
b	→	01
c	→	10
d	→	1100
e	→	1101
f	→	1110
g	→	1111

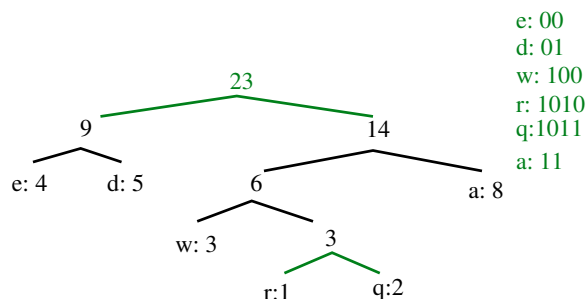
01001111111111011100  
 LR L L R R R R R R R R R R L R R R L L  
 b a g g e d

- The decoding is unique

## Generating the Huffman Tree

- We are left with the problem of constructing the Huffman tree such that frequently occurring letters have short codes
- A greedy approach is to iteratively build a tree by
  - combine the two most infrequent symbols into a subtree
  - Add their scores and treat them as a single symbol

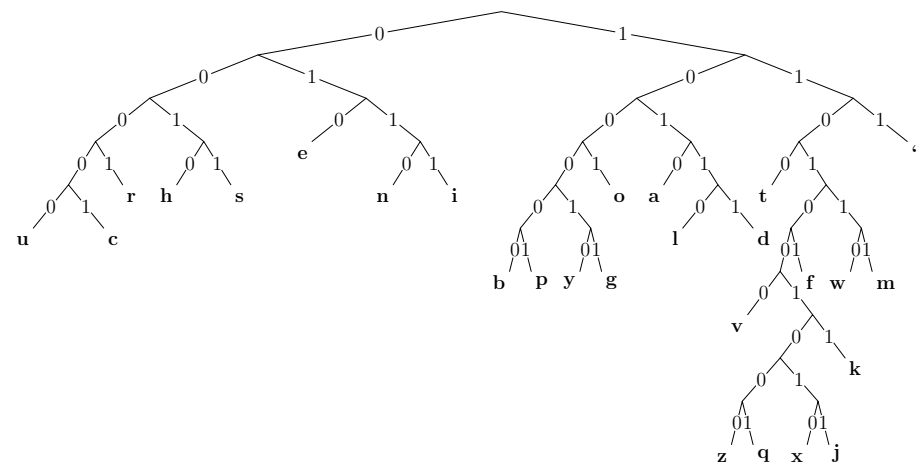
aaaaeedwqqadewwaaddreaad → 11111100000110010111011111...



## Implementing Huffman Encoding

- To implement Huffman encoding you need
  - A class to build Huffman trees by combining subtrees
  - A way to find the least frequently used symbols or symbol combinations
- Priority queues are ideal for this application
- They allow you to find the least frequently used symbols (`removeMin`) and to add new symbols (`add`)
- To decode you follow the Huffman tree

## English Letters



the quick brown fox jumps over the lazy dog

1100001001011111010010010000001110000111010011111100000000110011101100110111101  
0110011101001010111110100101100000110111100001001111100111010000100001111110000  
100101111011010101101001000100010111101111001100011  
344 bits  
211 bits

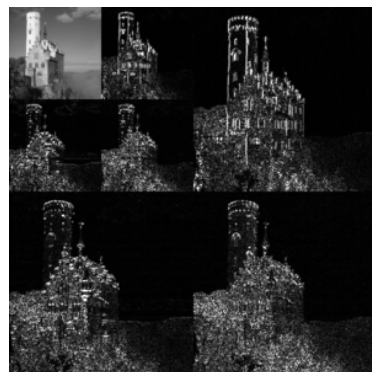
## Greedy Strategy

- Huffman encoding is an example of a **Greedy solution pattern**
- That is we look for local optimality (i.e. we combine the two least frequently used symbols)
- In this case, we obtain global optimality (i.e. the Huffman tree obtained gives an optimal Huffman code)
- There are a number of important problems where greedy algorithms lead to global optimality (we saw this earlier)
- For these algorithms priority queues commonly are used for implementing the algorithm

- Huffman code is optimal given the frequency of symbols
- However, there is considerable art in identifying which 'symbols' to use
- Advanced compression algorithms (LZ78, LZW Lempel-Ziv-Welch) build dictionaries of sequences seen in the files—they tend to be rather specialised
- Some recent algorithms (e.g. Burrows-Wheeler) transform the file in such a way that similar symbols are mapped to adjacent sites—depends on the generating mechanism of the language

## Outline

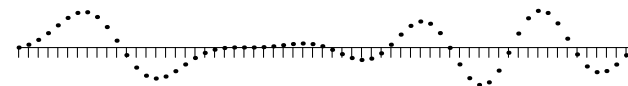
1. Huffman codes
2. Wavelets



- One way of spotting plagiarism is to compare the compressed lengths of two files and the length of the compressed file when the two files are concatenated first
- If the files have the same structure the concatenated version can often be significantly reduced
- Also used in identifying closeness of species in constructing phylogenetic trees

## Signals and Energies

- We consider compressing a signal  $x = (x_0, x_1, \dots, x_{n-1})$



- We can define the “energy” as the squared deviations

$$E = \sum_{i=1}^n x_i^2$$

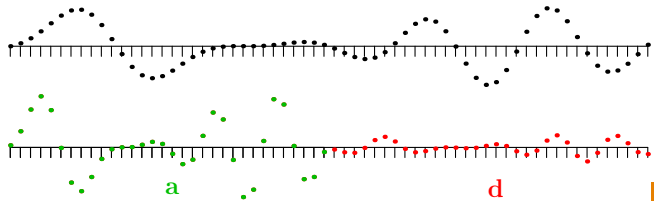
- Our strategy in lossy compression is to transmit as much “energy” in as few bits as possible
- There are different strategies to achieve good compress

## Wavelets

- With wavelets we try to re-represent the signal so as to squeeze as much energy as possible into fewer bits
- The easiest way to do this is with Haar wavelets

$$a_i = \frac{x_{2i} + x_{2i+1}}{\sqrt{2}} \quad d_i = \frac{x_{2i} - x_{2i+1}}{\sqrt{2}}$$

- Define new signal  $(a_0, a_1, a_2, \dots, a_{n/2-1}, d_0, d_1, \dots, d_{n/2-1})$



## Inverse Transform

- The wavelet transform can be easily reversed

$$a_i = \frac{x_{2i} + x_{2i+1}}{\sqrt{2}} \quad d_i = \frac{x_{2i} - x_{2i+1}}{\sqrt{2}}$$

$$x_{2i} = \frac{a_i + d_i}{\sqrt{2}} \quad x_{2i+1} = \frac{a_i - d_i}{\sqrt{2}}$$

- Can compute transform using vectors (wavelets)

$$a_i = V_i \cdot x \quad d_i = W_i \cdot x$$

- These vectors are orthogonal to each other ( $V_i \cdot V_j = 0$ ,  $V_i \cdot W_j = 0$ , etc.)

## Carrier and Difference Signals

- The terms  $a_i = (x_{2i} + x_{2i+1})/\sqrt{2}$  takes the “average” of the signal, but compresses it in half the space
- The terms  $d_i = (x_{2i} - x_{2i+1})/\sqrt{2}$  takes the difference and is small if the signal does not change much
- The energy is conserved since

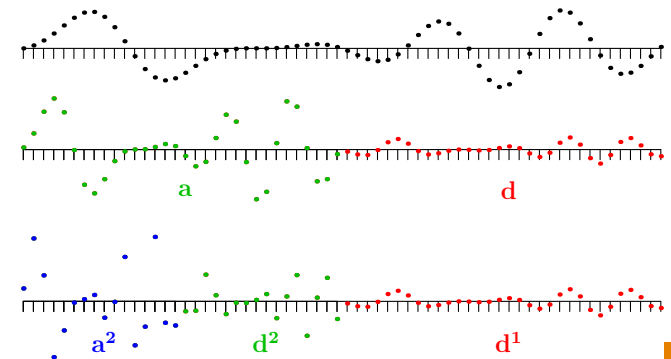
$$a_i^2 + d_i^2 = \left( \frac{x_{2i} + x_{2i+1}}{\sqrt{2}} \right)^2 + \left( \frac{x_{2i} - x_{2i+1}}{\sqrt{2}} \right)^2$$

$$= \frac{x_{2i}^2 + 2x_{2i}x_{2i+1} + x_{2i+1}^2 + x_{2i}^2 - 2x_{2i}x_{2i+1} + x_{2i+1}^2}{2} = x_{2i}^2 + x_{2i+1}^2$$

- Attempt to push all the energy into the carrier signal,  $a_i$

## And So On. . .

- We can repeat the process again to concentrate the energy further
- We apply the Haar transform just to the carry part  
 $a = (a_0, a_1, \dots, a_{n/2-1})$



## Daubechies Wavelets

- Ingrid Daubechies suggested a host of wavelets which do better than Haar for smooth signals
- The simplest is Daub4 defined by

$$a_i = c_0 x_{2i} + c_1 x_{2i+1} + c_2 x_{2i+2} + c_3 x_{2i+3}$$

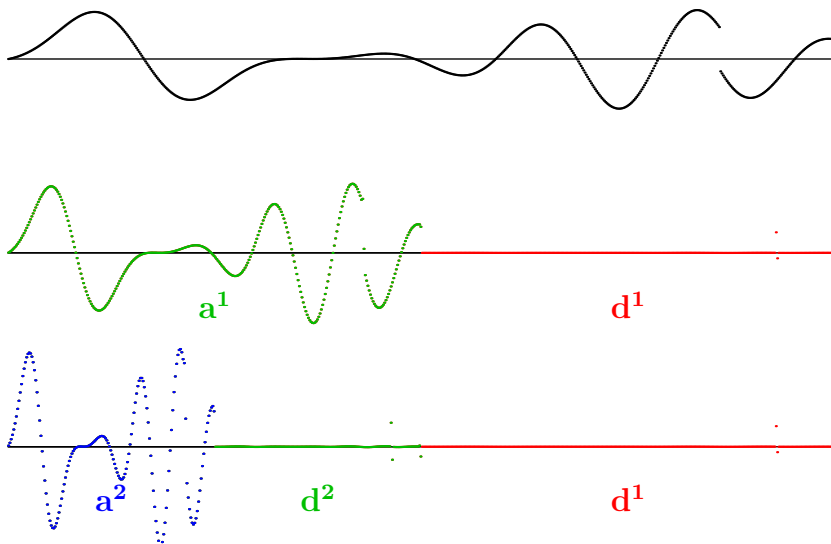
$$d_i = c_3 x_{2i} - c_2 x_{2i+1} + c_1 x_{2i+2} - c_0 x_{2i+3}$$

$$c_0 = \frac{1 + \sqrt{3}}{4\sqrt{2}} \quad c_1 = \frac{3 + \sqrt{3}}{4\sqrt{2}} \quad c_2 = \frac{3 - \sqrt{3}}{4\sqrt{2}} \quad c_3 = \frac{1 - \sqrt{3}}{4\sqrt{2}}$$

- Again conserves energy

$$\sum_{i=1}^{n/2} a_i^2 + b_i^2 = \sum_{i=1}^n x_i^2$$

## Daub4



## Properties of Daub4

- Similar to the Haar transform

$$c_0 + c_1 + c_2 + c_3 = \sqrt{2}, \quad c_3 - c_2 + c_1 - c_0 = 0$$

so the carrier signal ( $a_i$ ) is approximately  $\sqrt{2}$  times the original and the difference part ( $d_i$ ) is equal to 0 for a flat signal,  $x$

- However in addition

$$0c_3 - 1c_2 + 2c_1 - 3c_0 = 0$$

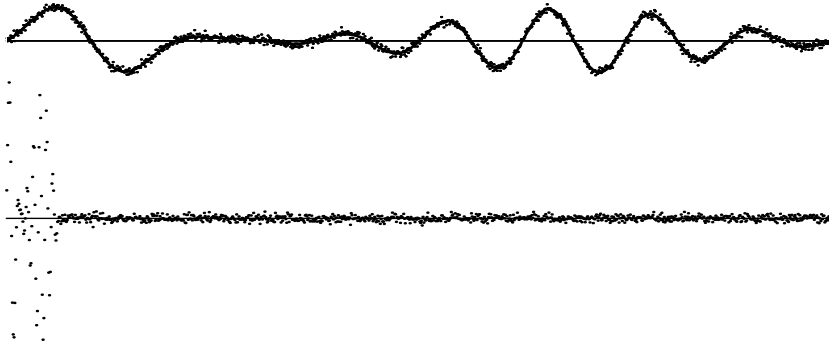
so the difference part ( $d_i$ ) is equal to 0 for any linear signal,  $x$

## Signal Compression

- To compress the signal we can set all components of the transformed signal whose magnitude lies below a threshold to 0
- We transmit the non-zero magnitude together with a binary mask showing the position of the non-zero magnitude
- We can reduce the accuracy (number of decimal places) of the non-zero magnitudes (quantisation)—this is repaired on inverting transform
- We can compress the binary mask using Huffman encoding or other scheme

## Noise Reduction

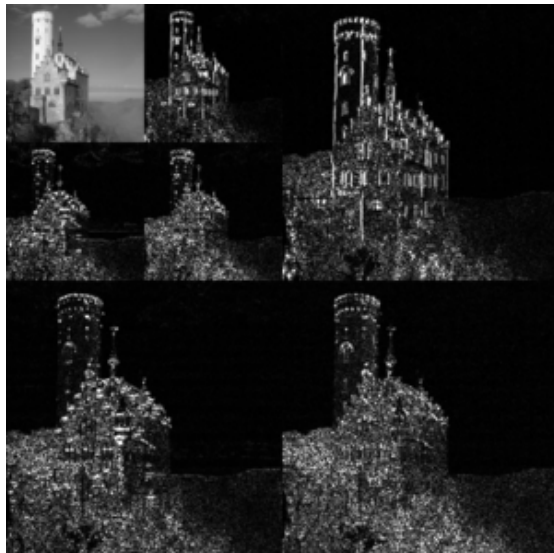
- Can also be used in noise reduction



## Other Wavelets

- Can use high-order wavelets which captures more energy in the carrier signal, e.g. Daub10 or Daub20
- Many other wavelets capture other properties (e.g. Coiflets capture properties of a continuous signal sampled at discrete points)
- Efficiency of wavelets depend on how well the capture underlying properties of signals
- Can also construct 2-d wavelets for image compression (jpeg-2000)

## 2-D Wavelets



## Summary

- File compression is an important task in its own right
- Files may either be compressed losslessly or lossily
- Lossy compression is typically much more efficient (e.g. an order of magnitude smaller)
- Huffman encoding often lies at the lowest level in many compression algorithms
- Wavelets illustrate a strategy of changing the representation to concentrate the energy of a signal