encoding
compression
encryption

- ASCII utf-8 utf-16
- zip mpeg jpeg
- AES RSA diffie-hellman
Expressing characters ...

ASCII and Unicode, conventions of how characters are expressed in bits.

**ASCII (7 bits) - 128 characters**

00 - 7F
Expressing characters ...

ASCII and Unicode, conventions of how characters are expressed in bits.

**ASCII (7 bits) - 128 characters**
00 - 7F
Expressing characters ...

ASCII and Unicode, conventions of how characters are expressed in bits.

ASCII (7 bits) - 128 characters
00 - 7F

Unicode designed to encode any language more than 109,000 characters
e.g. Chinese, 20,902 ideogram characters

Room for expansion: 1,114,112 code points in the range 0_{hex} to 10FFFF_{hex}
various encodings UTF-8 UTF-16
Basic Multilingual Plane
0000 - FFFF

<table>
<thead>
<tr>
<th>BMP</th>
<th>SMP</th>
<th>SIP</th>
<th>Plane 1: Supplementary Multilingual Plane</th>
<th>Plane 2: Supplementary Ideographic Plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000–0FFF</td>
<td>9000–8FFF</td>
<td>10000–10FFF</td>
<td>11000–11FFF</td>
<td>12000–12FFF</td>
</tr>
<tr>
<td>1000–1FFF</td>
<td>9000–8FFF</td>
<td>11000–11FFF</td>
<td>12000–12FFF</td>
<td>13000–13FFF</td>
</tr>
<tr>
<td>2000–2FFF</td>
<td>A000–AFFF</td>
<td>12000–12FFF</td>
<td>13000–13FFF</td>
<td>1B000–1BFFF</td>
</tr>
<tr>
<td>3000–3FFF</td>
<td>B000–BFFF</td>
<td>13000–13FFF</td>
<td>1B000–1BFFF</td>
<td>20000–20FFF</td>
</tr>
<tr>
<td>4000–4FFF</td>
<td>C000–CFFF</td>
<td>13000–13FFF</td>
<td>1B000–1BFFF</td>
<td>20000–20FFF</td>
</tr>
<tr>
<td>5000–5FFF</td>
<td>D000–DFFF</td>
<td>13000–13FFF</td>
<td>1B000–1BFFF</td>
<td>20000–20FFF</td>
</tr>
<tr>
<td>6000–6FFF</td>
<td>E000–EFFF</td>
<td>1D000–1DFFF</td>
<td>1F000–1FFF</td>
<td>27000–27FFF</td>
</tr>
<tr>
<td>7000–7FFF</td>
<td>F000–FFF</td>
<td>16000–16FFF</td>
<td>1F000–1FFF</td>
<td>27000–27FFF</td>
</tr>
</tbody>
</table>

All code points in the BMP are accessed as a single code unit in UTF-16 encoding and can be encoded in one, two or three bytes in UTF-8. Code points in Planes 1 through 16 (supplementary planes, or, informally, astral planes) are accessed as surrogate pairs in UTF-16 and encoded in four bytes in UTF-8.
<table>
<thead>
<tr>
<th>Code point</th>
<th>Binary code point</th>
<th>UTF-8 bytes</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0000 to U+007F</td>
<td>0xxxxxx</td>
<td>0xxxxxx</td>
<td>'§' U+0024 = 00100100 → 00100100 → 0x24</td>
</tr>
<tr>
<td>U+0080 to U+07FF</td>
<td>00000yyy yyyyy</td>
<td>110yyyyy 10xxxxx</td>
<td>'€' U+00A2 = 00000000 10100010 → 11000010 10100010 → 0xC2 0xA2</td>
</tr>
<tr>
<td>U+0800 to U+FFFF</td>
<td>zzzzyyy yyyyy</td>
<td>1110zzzz 10yyyyy 10xxxxx</td>
<td>'€' U+20AC = 01010000 10101100 → 11100010 1000010 10101100 → 0xE2 0x82 0xAC</td>
</tr>
<tr>
<td>U+010000 to U+10FFFF</td>
<td>000wwwzz zzzzyyy yyyyy</td>
<td>11110www 10zzzzzz 10yyyyy 10xxxxx</td>
<td>'☐' U+24B62 = 0000010 01001011 01100010 → 11110000 10100100 10101101 10100010 → 0xF0 0xA4 0xAD 0xA2</td>
</tr>
</tbody>
</table>

UTF-8: first 128 characters (US-ASCII) need one byte; next 1,920 characters need two bytes to encode.

In UTF-8:
first 128 characters (00–7F US-ASCII) need one byte;
next 1,920 characters (80–7FF) need two bytes to encode;
next (800–FFFF) each need two bytes to encode;
next (10000–10FFFF) each need four bytes.

Good for English and European texts – not so good for others. Cyrillic and Greek alphabet pages in UTF-8 may be double the size, Thai and Devanagari, (Hindi) letters triple the size, compared with an encoding adapted to these character sets.

GB18030 is another encoding form for Unicode, from the Standardization Administration of China. It is the official character set of the People’s Republic of China (PRC). GB abbreviates Guójiā Biāozhǔn (国家标准), which means national standard in Chinese.
Huffman encoding (1952)

- Variable length encoding
  - use shorter codes for common letters

<table>
<thead>
<tr>
<th>letter</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.05541</td>
</tr>
<tr>
<td>B</td>
<td>0.02119</td>
</tr>
<tr>
<td>C</td>
<td>0.02913</td>
</tr>
<tr>
<td>D</td>
<td>0.04792</td>
</tr>
<tr>
<td>E</td>
<td>0.16081</td>
</tr>
<tr>
<td>F</td>
<td>0.01702</td>
</tr>
<tr>
<td>G</td>
<td>0.03906</td>
</tr>
<tr>
<td>H</td>
<td>0.02761</td>
</tr>
<tr>
<td>I</td>
<td>0.08007</td>
</tr>
<tr>
<td>J</td>
<td>0.00269</td>
</tr>
<tr>
<td>K</td>
<td>0.01661</td>
</tr>
<tr>
<td>L</td>
<td>0.03666</td>
</tr>
<tr>
<td>M</td>
<td>0.02425</td>
</tr>
<tr>
<td>N</td>
<td>0.10185</td>
</tr>
<tr>
<td>O</td>
<td>0.02880</td>
</tr>
<tr>
<td>P</td>
<td>0.02005</td>
</tr>
<tr>
<td>Q</td>
<td>0.00089</td>
</tr>
<tr>
<td>R</td>
<td>0.07620</td>
</tr>
<tr>
<td>S</td>
<td>0.06745</td>
</tr>
<tr>
<td>T</td>
<td>0.06785</td>
</tr>
<tr>
<td>U</td>
<td>0.04013</td>
</tr>
<tr>
<td>V</td>
<td>0.01000</td>
</tr>
<tr>
<td>W</td>
<td>0.01266</td>
</tr>
<tr>
<td>X</td>
<td>0.05000</td>
</tr>
<tr>
<td>Y</td>
<td>0.00258</td>
</tr>
<tr>
<td>Z</td>
<td>0.01211</td>
</tr>
</tbody>
</table>

Just as some characters are more frequent in some languages – and so different languages require different encodings to reduce the size of the encoded text – so different characters have different frequencies within a given language.

Can we use shorter codes for more frequent characters? What would such a code look like?
This tree represents a Huffman encoding.

The 26 characters of the alphabet are at the leaves of the tree.

Each node, except the root node, is labelled, either 0 or 1.

Each non-leaf node has two children, one labelled 0, the other labelled 1.

Given a stream of bits, we can decode it as follows:
We start at the root and use successive bits from the stream to tell us which path to take through the tree, until we reach a leaf node. When we reach a leaf node, we write out the letter at that node and jump back to the root.

To encode a text, for each character, we just find the path from the root to the leaf labelled with that letter, and write out the sequence of bit-labels on that path.

The more-common letters are higher-up in the tree.
Lossless compression

- exploit statistical redundancy
- represent data concisely
- without error

- eg an html file has many occurrences of
  - <p>
- encode these with short sequences

Huffman encoding is an example of lossless compression. We find a way to encode a message using fewer bits, that allows us to recreate the original message exactly.

We can compute an optimal encoding for any text. Unless the text is very short, sending the encoding then the encoded text will be shorter than just sending the original.

The same idea as for Huffman encoding can be used to encode common sequences of characters (eg common words in English, or particular patterns that are common in the file in question). This gives encodings such as zip and gzip used to compress files on the internet. This speeds up the web.
Multimedia files are often very large. They don’t have the same kinds of repeated patterns that we see in text – so compression algorithms designed for text don’t typically do much for music or pictures. A musician never plays the exactly the same note twice (and even if she did, random variations in the recording would introduce perhaps imperceptible differences).
On the other hand, for multimedia files, the details of the encoding may not be so important. We care what the music sounds like, or what a picture looks like. Imperceptible differences don’t matter, and for some applications (eg speech) even perceptible differences don’t matter provided we still get the message.

For example, telephones only transmit part of the speech signal. They are designed for communication. Listening to music down the telephone is an impoverished experience.

Even for music, there are well-researched effects that mean that some changes are imperceptible. For example, a loud sound ‘masks’ softer sounds at nearby frequencies. The ear can’t hear whether they are there or not. So an encoding for music (such as MP3) can drop these softer sounds, imperceptibly.

Tricks such as this allow music to be compressed so it takes up less space on a memory stick and uses less bandwidth when transmitted over the internet.
There are many competing encodings for images.

Some (eg SVG) are descriptions of geometric objects, that can be rendered in many different ways.

Others are representations of the rendered form of a photograph or image.
There are many competing encodings for images.

Some (eg SVG) are descriptions of geometric objects, that can be rendered in many different ways.

Others are representations of the rendered form of a photograph or image.
There are many competing encodings for images.

Some (eg SVG) are descriptions of geometric objects, that can be rendered in many different ways.

Others are representations of the rendered form of a photograph or image.
## Image Compression Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPG or JPEG</td>
<td>Joint Photographic Expert Group</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphics Interchange Format</td>
</tr>
<tr>
<td>TIF or TIFF</td>
<td>Tagged Image File Format</td>
</tr>
<tr>
<td>PNG</td>
<td></td>
</tr>
<tr>
<td>SVG</td>
<td></td>
</tr>
</tbody>
</table>

There are many competing encodings for images.

Some (eg SVG) are descriptions of geometric objects, that can be rendered in many different ways.

Others are representations of the rendered form of a photograph or image.
There are many competing encodings for images.

Some (eg SVG) are descriptions of geometric objects, that can be rendered in many different ways.

Others are representations of the rendered form of a photograph or image.
There are many competing encodings for images. Some (eg SVG) are descriptions of geometric objects, that can be rendered in many different ways. Others are representations of the rendered form of a photograph or image.
PNG vs JPEG

SOMETIMES PNG IS SUPERIOR...

FOR ILLUSTRATIONS, SCREENSHOTS, ANYTHING WITH TEXT...

...WEBCOMICS, GRAPHS, LOGOS...

NOPE! ALWAYS JPEG!!!

JPEG ONLY!!! NO THINKING REQUIRED!!

I LOVE JPEG!!! JPEG! JPEG!
RGB - 24 bits

Grayscale - 8 bits
JPEG always uses **lossy** JPG compression, but the degree of compression can be chosen – for higher quality and larger files, or lower quality and smaller files.
JPEG always uses **lossy** JPG compression, but the degree of compression can be chosen – for higher quality and larger files, or lower quality and smaller files.
GIF

Indexed colour - 1 to 8 bits (2 to 256 colours)
GIF

Indexed colour - 1 to 8 bits (2 to 256 colours)

GIF uses **lossless** compression, effective on indexed colour. GIF files contain no dpi information for printing purposes.
GIF

Indexed colour - 1 to 8 bits (2 to 256 colours)

GIF uses **lossless** compression, effective on indexed colour. GIF files contain no dpi information for printing purposes.
Indexed colour - 1 to 8 bits (2 to 256 colours)

GIF uses **lossless** compression, effective on indexed colour. GIF files contain no dpi information for printing purposes.
**TIF**

**RGB** - 24 or 48 bits

**Grayscale** - 8 or 16 bits

**Indexed colour** - 1 to 8 bits
TIF

RGB - 24 or 48 bits

Grayscale - 8 or 16 bits

Indexed colour - 1 to 8 bits

For TIF files, most programs allow either no compression or LZW compression (lossless, but is less effective for 24 bit color images).
TIF

**RGB** - 24 or 48 bits

**Grayscale** - 8 or 16 bits

**Indexed colour** - 1 to 8 bits

For TIF files, most programs allow either no compression or LZW compression (lossless, but is less effective for 24 bit color images).
PNG

**RGB** - 24 or 48 bits

**Grayscale** - 8 or 16 bits

**Indexed colour** - 1 to 8 bits
PNG

RGB - 24 or 48 bits

Grayscale - 8 or 16 bits

Indexed colour - 1 to 8 bits

PNG uses ZIP compression which is lossless.
PNG

**RGB** - 24 or 48 bits

**Grayscale** - 8 or 16 bits

Indexed colour - 1 to 8 bits

PNG uses ZIP compression which is lossless.
PNG

**RGB** - 24 or 48 bits

**Grayscale** - 8 or 16 bits

Indexed colour - 1 to 8 bits

PNG uses ZIP compression which is lossless.

PNG was created to improve upon and replace GIF as an image-file format not requiring a patent license.
Lossy Compression

• In a lossy compression scheme, some of the original information is lost.

• It is impossible to produce an exact replica of the original signal when the audio or video is played.

• Lossy compression schemes add artefacts, small imperfections created by the loss of the actual data.
Keys are used to encrypt (lock) and decrypt (unlock) whatever data is being encrypted/decrypted.

Symmetric-key algorithms use a single shared key; keeping data secret requires keeping this key secret.

Public-key algorithms use a public key and a private key. The public key is made available to anyone (often by means of a digital certificate). A sender encrypts data with the public key; only the holder of the private key can decrypt this data.
public key

lock (public key)

unlock (private key)

**Step 1:** Give your public key to the sender

**Step 2:** Sender uses your public key to encrypt the plaintext

**Step 3:** Sender gives the ciphertext to you

**Step 4:** Use your private key (and passphrase) to decrypt the ciphertext

Saturday, 3 December 2011
making a shared secret

Alice makes up a secret: x
Bob makes up a secret: y

Alice sends Bob \( A = g^x \)
Bob sends Alice \( B = g^y \)

Bob calculates \( A^y = g^{xy} \)
Alice calculates \( B^x = g^{xy} \)

Diffie–Hellman key exchange method allows two strangers (with no prior knowledge of each other) to jointly establish a shared secret key over an insecure communications channel.

Two or more parties use a public exchange to agree on a shared secret they can use as a key without revealing the key to any eavesdropper.

The first publicly known key agreement protocol was this Diffie–Hellman exponential key exchange.

Anonymous key exchange, like Diffie–Hellman, does not provide authentication of the parties, and is thus vulnerable to Man-in-the-middle attacks.

In practice the computation uses modular arithmetic to keep the sizes of numbers involved manageable.