

File Compression

Most methods exploit either:

- Repeated patterns in files, e.g.,
1010101010101010 (grey colour?)
the dog and the cat (repeated “the”).
Similar successive frames in video.
- Frequency information: e.g.,
“e” occurs frequently, so better have short code to store it.

5

Compression Ratio

How much smaller will compressed file be?

Depends on how much repetition in input file; generally most useful for graphics files, produced using some drawing tool, where large areas of uniform “colour”.

But note that for binary files, must take into account size of integers vs size of bits:

```
1111000011111000    4453
      2 bytes         4 bytes (at least!)
```

7

Run-length Encoding

Simplest file compression method simply looks for “runs” of repeated characters, e.g.,

```
aaaaaaabbbbbbbccccc
```

and replaces with count + relevant character:

```
7a8b4c
```

For binary files don’t even need to specify the character; assume files always start with a zero.

```
111111110000111111
coded as 0846
```

Easy to code; simple loop reads in single character and increments counter until different character read in.

6

Variable Length Encoding (Huffman encoding)

Based on principle that:

- Some characters are more common than others.
- So use special short codes for them.
- Normally 1 byte required for each character (using ASCII codes). Variable length encoding finds codes for common characters less than 1 byte, but codes for rare characters more than 1 byte.
- In binary, ASCII code for ‘e’ is 01100101. But as it’s so common can we just use a single bit? or 2 bits e.g., 1 or 01.
- How can we find an optimal set of codes given information on how frequently different characters occur.

8

Variable Length Encoding

Rather than 8 bits of each char, now chars will have variable length, e.g.:

z: 16 bits
e: 2 bits

But how can we tell where one character ends and another begins?

1011001000

Is that:

10 followed by 11001000
1011 followed by 001000

or what?

Huffman Encoding

Huffman encoding is method of constructing optimal code tree given frequency information on characters.

Build tree from bottom up.

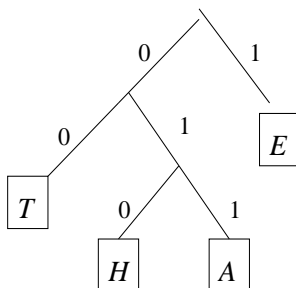
- Start by creating leaf nodes corresponding to all the characters. Score of node is number of occurrences of that character.
- Repeat:
 - Combine two nodes with lowest score, creating parent node with these two as its children.
 - Score of this node is sum of scores of children.

Until all nodes combined and have single root.

Variable length encoding

Avoid this problem by ensuring that you never have two codes such that one code starts with another code (e.g., 1010 and 10).

Can find suitable set of codes from a binary tree:



Codes read off by taking 0/1 value, depending on whether left or right hand branch, as descend tree.

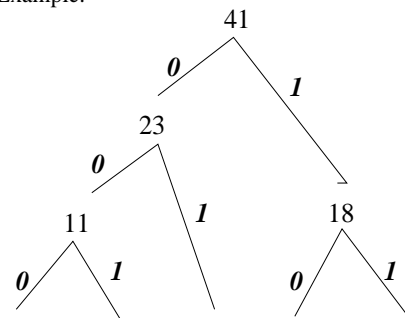
T=00 (ie, left, left) E=1 H=010 A=011

Decoding easy: Descend left/right branch according to whether next bit is 0 or 1; when at leaf branch output character and return to root.

e.g., 000101 ?

Huffman Encoding

Example:



A:5 B:6 E:12 C:8 D:10

Rare characters A and B get shorter codes than common characters.

- Result of all this is coding scheme where common characters have short codes.
- Good for text files (20-30% reduction).

Substitutional Compression

- Finds repeated *sequences*, not just runs.
e.g., *the* cat and *the* dog.
- Replaces later sequences with a reference to earlier one. Either:
 - Stores character sequences in a dictionary. Replaces sequence with a dictionary index.
 - Keeps track of (say) last N bytes in string, and replaces repeated sequence with reference back to last occurrence.
- Method used widely for text files; gzip, zip etc use it. May get 50% compression.

13

Lossy Compression

- All methods do far are *lossless* - can restore exact copy of original.
- For images/video *lossy* methods are OK, where some fine detail lost:
 - JPEG: Lossy compression for images.
 - MPEG: Video compression.
- Lossy compression obviously useless for text

15

Substitutional Compression

Dictionary method uses trick to decide what to put in dictionary, avoiding storing every possible sequence:

Go through an example text file (say, d[]), character by character:

- at character i, find largest k such that sequence d[i].d[i+k-1] is already in the dictionary. (k=0 if d[i] not in it).
- Add this character sequence to the dictionary

```
d = ``the_thin_chin``
```

```
dictionary items added:  
t h e _ t h h i i n _ c c h i n  
(_ = space)
```

Result is that long common strings are added; long rare ones aren't.

14

JPEG

Roughly:

- Transform image to obtain *spatial frequencies* (similar to fourier transform used).
- High spatial frequencies = fine detail. So throw them away.
- Now if restore image by doing reverse transform, fine detail lost.. but may be invisible to human eye.
- Can vary degree of *lossiness* depending on quality/compression criteria.
- Can reduce size by factor of 5 without perceptible loss in quality.

16

MPEG: Video Compression

Roughly:

- Uses JPEG compression for frame.
- Looks at *differences* between frames, rather than recording every one (successive frames will be very similar).

17

Summary

Have looked at:

- Run-length
- Variable length
- Substitutional
- Lossy (image/video)

Use information on:

- Repetition (e.g., run-length, substitutional, MPEG)
- Frequency (Variable length)

Methods are often combined.

18