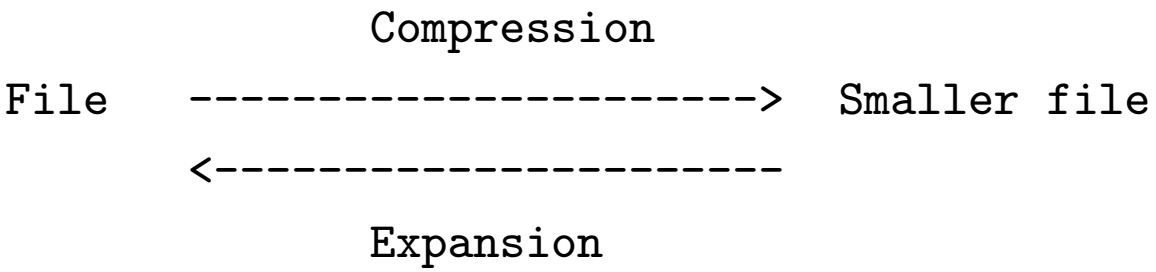
Data Structures and Algorithms
Compression Methods

Goodrich & Tamassia Section 12.4

- Introduction
- Run-length encoding
- Variable length (Huffman) encoding
- Substitutional Compression
- Lossy compression methods: JPEG, MPEG, MP3, ...

File Compression



e.g., zip/unzip; compress/uncompress

Motivation:

- Reduce file space needed (e.g., for video clips)
- Often built into disk controllers/Database Management Systems
- Reduce time to copy files over network (e.g., WWW)

DVDs would only hold seconds of video if compression methods were NOT used.

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.

Run-length Encoding

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

aaaaaaabbbbbbbbbbcccc

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.

11111111000111111

coded as 0846

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

Compression Ratio

How much smaller will the compressed file be?

Comp. ratio = Size of orig. / Size of compressed

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 the number used (e.g. `int` is 32 bits in Java):

```
1111000011111000    4453
    16 bits         32 bits
```

Compression ratio = 16/32 = 0.5 = bad!

Typical compression ratios from production compression software like `zip` on text files are around 3.

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.

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:

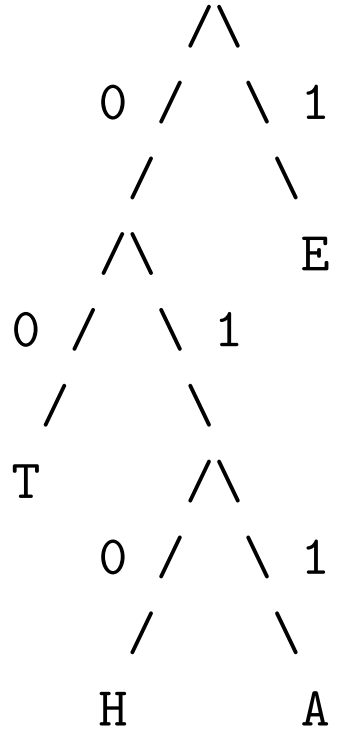
```
10 followed by 11001000
1011 followed by 001000
```

or what?

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

Exercise: What do the following strings correspond to?

- 01001100
- 0001001100
- 001011

Exercise: Use the tree above to encode the following strings

- TAT
- HATE
- HEAT
- HEATH

Huffman Encoding

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

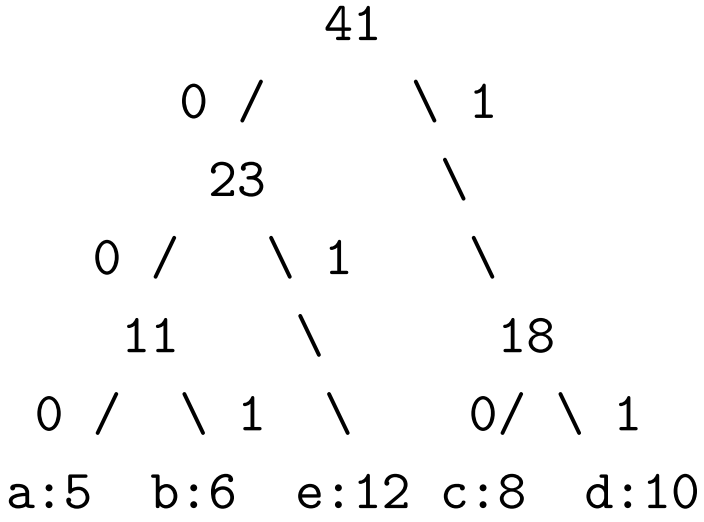
Build tree from bottom up.

- Start by creating leaf nodes corresponding to all the characters. Score of node is number of occurences 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.

Huffman Encoding

Example:



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

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

Exercise: Produce a Huffman encoding for characters with the following frequencies:

P:3 Q:2 R:8 S:10 T:12

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 occurence.
- Method used widely for text files; gzip, zip etc use it. May get 50% compression.

LZW Compression

Lempel, Ziv and Welch suggested a clever method to decide what to put in dictionary, avoiding storing every possible sequence:

Initial dictionary contains all individual characters, e.g. ASCII, UNICODE, or

0 1 2 3
b e t w

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

- at each character find the longest prefix of d that is already in the dictionary
- output the code for the prefix
- Add the prefix plus the next character to the dictionary

LZW Example

`d = ‘‘wewetweb’’`

`Code produced:`

Result is that long common strings are added; long rare ones are not.

Question: Do we need to send the dictionary along with the encoded file?

Lossy Compression

- All methods so 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

JPEG

JPEG is an ISO standard named after the Joint Photographic Experts Group. There are several standards, e.g. baseline JPEG and lossless JPEG. Lots more info available on WWW.

Roughly:

- Transform image to obtain *spatial frequencies* (similar to fourier transform).
- 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.

MPEG: Video Compression

MPEG produced by an ISO working party: the Moving Picture Expert Group. Several standards:

- MPEG-1: poor video quality
- MPEG-2: higher quality DVDROM/TV quality
- No MPEG-3!
- MPEG-4: interactivity, mobile devices
- *watch this space!*

Roughly:

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

Audio Compression

The widely-used MP3 audio standard is the audio layer (Layer III) of MPEG-1.

It uses both

- lossy techniques, e.g. minimal audition threshold, masking effect
- lossless techniques, e.g. Huffman encoding

New formats will appear, e.g. audio-only MPEG-4 (.m4a) is an emerging standard

Lots more info on web, e.g. in wikipedia, or `http://www.mp3-tech.org/`

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, as in most production compression software