Data Compression

Compression reduces the size of a file:

* To save space when storing it
* To save time when transmitting it
* Most files have a lot of redundancy

Who needs compression?

* Moore’s Law: Number or transistors on a chip doubles every 18-24 months
* Parkinsion’s Law: data expands to fill the space available
* Text, images, sound, video, etc.

Basic concepts date back even to the 1950s, but there is new tech

Generic file compression

* Files: GZIP, BZIP
* Archivers: PKZIP
* File systems: NTFS, HFS+

Multimedia

* Images: GIF, JPEG
* Sound: MP3
* Video: MPEG, DivX

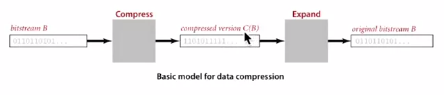
Communication

* V.42bis modem
* Skype

Databases: MANGA companies

Lossless compression and expansion

Message: binary data B we want to compress  
Compress: generates a “compressed” representation C(B) <- uses fewer bits   
Expand: Reconstructs original bitstream B



Compression ratio: Bits is C(B) / bits in B

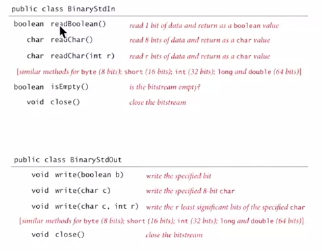
Ex: 50-75% or better compression ratio for natural language

Use of data compression over time:

* Number systems
* Natural languages
* Mathematical notation
* Communications tech:
  + Braille
  + Morse code
* Modern life
  + MP3
  + MPEG

Reading and writing binary data

Algorithms II course binary IO class APIs



C:\Users\Zach\AppData\Local\Microsoft\Windows\INetCache\Content.Word\dcex.png

Proposition: No algorithm can compress every bitstream

Proof 1: [by contradiction]

* Suppose you have a universal data compression algorithm U   
  that can compress every bitstream
* Given bitstream Bo, compress it to get smaller bitstring B1
* Compress B1 to get smaller bitstring B2
* Continue until reaching the size 0
* Implication: all bitstrings can be compressed to 0 bits!

Proof 2: [by counting]

* Suppose your algorithm that can compress all 1,000-bit strings
* 21000 possible bitstrings with 1,000 bits
* Only 1 + 2 + 4 + … + 2998 + 2999 can be encoded with <= 999 bits
* Similarly, only 1 in 2499 bitstrings can be encoded with 2<= bits

Undecidability: very difficult to find the best possible compression algorithm  
for different applications. The only true way is finding the program that   
created the file in the first place.

Much redundancy, especially in the English language, makes some compression easier

Run-Length Coding

* Simple type of redundancy in a bitstream: long runs of repeated bits

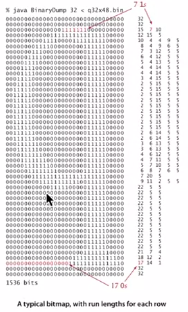
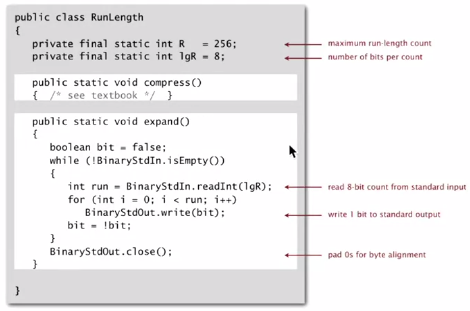


* Representation: 4-bit counts to represent alternating runs of 0s and 1s:  
  15 0s, then 7 1s, then 7 0s, then 11 1s

C:\Users\Zach\AppData\Local\Microsoft\Windows\INetCache\Content.Word\rle.png

How many bits to store counts? Use 8 generally, but 4 are used above  
If longer than 255 (max count), intersperse runs of length 0.

Java Implementation



Application: compress a bitmap

Typical b&w scanned image:

* 300 px/ inch
* 8.5 x 11 inches
* 300 x 8.5 x 11 == 8.415 million bits

Observation: bits are mostly white

Typical amount of text on a page

40 lines x 75 chars per line = 3,000 chars

Data compression summary:

Lossless compression:

* Represent fixed-length symbols with variable-length codes [Huffman]
* Represent variable-length symbols with fixed-length codes [LZW]

Lossy compression: [okay when can afford to lose bits, such as with media]

* JPEG, MPEG, MP3
* FFT, wavelets, fractals

Theoretical limits on compression: Shannon entropy



Practical compression: use extra knowledge whenever possible (info about the file)