Data Compression

- fixed-length codes
- variable-length codes
- an application
- adaptive codes

References:
Algorithms 2nd edition, Chapter 22
`http://www.cs.princeton.edu/algs4/65compression`

Data compression

Compression reduces the size of a file:

- To save space when storing it.
- To save time when transmitting it.
- Most files have lots of redundancy.

Who needs compression?

- Moore's law: # transistors on a chip doubles every 18-24 months.
- Parkinson's law: data expands to fill space available.
- Text, images, sound, video, …

> *" All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value. "* — *Carl Sagan*

Basic concepts ancient (1950s), best technology recently developed.

Applications

Generic file compression.

- Files: GZIP, BZIP, BOA.
- Archivers: PKZIP.
- File systems: NTFS.

Multimedia.

- Images: GIF, JPEG.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.

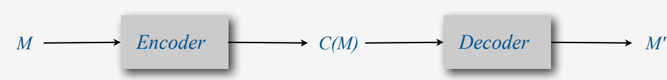
Communication.

- ITU-T T4 Group 3 Fax.
- V.42bis modem.

Databases. Google.

Encoding and decoding

Message. Binary data M we want to compress.
Encode. Generate a "compressed" representation C(M). ← uses fewer bits (you hope)
Decode. Reconstruct original message or some approximation M'.



Compression ratio. Bits in C(M) / bits in M.

Lossless. M = M', 50-75% or lower. ← this lecture
Ex. Natural language, source code, executables.

Lossy. $M \approx M'$, 10% or lower.
Ex. Images, sound, video.

Food for thought

Data compression has been omnipresent since antiquity:
- Number systems.
- Natural languages.
- Mathematical notation.

has played a central role in communications technology,
- Braille.
- Morse code.
- Telephone system.

and is part of modern life.
- MP3.
- MPEG.

Q. What role will it play in the future?

What data can be compressed?

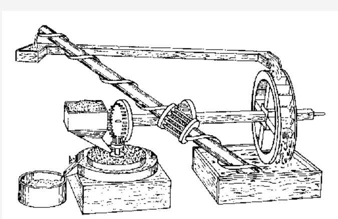
US Patent 5,533,051 on "Methods for Data Compression", which is capable of compression all files.

Slashdot reports of the Zero Space Tuner™ and BinaryAccelerator™.

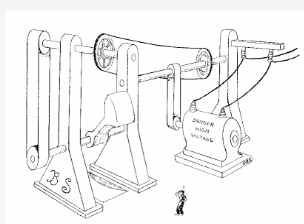
> *" ZeoSync has announced a breakthrough in data compression that allows for 100:1 lossless compression of random data. If this is true, our bandwidth problems just got a lot smaller.… "*

Perpetual motion machines

Universal data compression is the analog of perpetual motion.



Closed-cycle mill by Robert Fludd, 1618



Gravity engine by Bob Schadewald

Reference: Museum of Unworkable Devices by Donald E. Simanek
http://www.lhup.edu/~dsimanek/museum/unwork.htm

What data can be compressed?

Proposition. Impossible to losslessly compress all files.

Pf 1.
- consider all 1,000 bit messages.
- 21000 possible messages.
- only $2^{999} + 2^{998} + \ldots + 1$ can be encoded with ≤ 999 bits.
- only 1 in 2^{499} can be encoded with ≤ 500 bits!

Pf 2 (by contradiction).
- given a file M, compress it to get a smaller file M_1.
- compress that file to get a still smaller file M_2.
- continue until reaching file size 0.
- implication: all files can be compressed with 0 bits!

Rdenudcany in Enlgsih Inagugae

Q. How much redundancy is in the English language?

> *" ... randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to denmtrasote. In a pubiltacion of New Scnieitst you could ramdinose all the letetrs, keipeng the first two and last two the same, and reibadailty would hadrly be aftcfeed. My ansaylis did not come to much beucase the thoery at the time was for shape and senqeuce retigcionon. Saberi's work sugsegts we may have some pofrweul palrlael prsooscers at work. The resaon for this is suerly that idnetiyfing coentnt by paarllel prseocsing speeds up regnicoiton. We only need the first and last two leteters to spot chganes in meniang."* — *Graham Rawlinson*

A. Quite a bit.

- fixed-length codes
- variable-length codes
- an application
- adaptive codes

Fixed-length coding

- Use same number of bits for each symbol.
- k-bit code supports 2^k different symbols.

Ex. 7-bit ASCII code.

char	decimal	code
NUL	0	0000000
...	...	
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
...	...	
DEL	127	1111111

a	b	r	a	c	a	d	a	b	r	a	!
1100001	1100010	1110010	1100001	1100011	1100001	1100100	1100001	1100010	1110010	1100001	1111111

12 symbols × 7 bits per symbol = 84 bits in code

Fixed-length coding

- Use same number of bits for each symbol.
- k-bit code supports 2^k different symbols.

Ex. 3-bit custom code.

char	code
a	000
b	001
c	010
d	011
r	100
!	111

a	b	r	a	c	a	d	a	b	r	a	!
000	001	100	000	010	000	011	000	001	100	000	111

12 symbols × 3 bits per symbol = 36 bits in code

Fixed-length coding: general scheme

- Count number of different symbols.
- ~ lg M bits suffice to support M different symbols.

Ex. Genomic sequences.

- 4 different nucleotides.
- 2 bits suffice.
- Amazing but true: initial databases in 1990s did not use such a code!

~ 2N bits to encode genome with N nucleotides

a	c	t	a	c	a	g	a	t	g	a	a
00	01	10	00	01	00	11	00	10	11	00	00

2-bit DNA code

char	code
a	00
c	01
t	10
g	11

Important detail. Decoder needs to know the code!

Variable-length coding

Use different number of bits to encode different symbols.

Ex. Morse code.

Letters		Numbers	
A	•—	1	•————
B	—•••	2	••———
C	—•—•	3	•••——
D	—••	4	••••—
E	•	5	•••••
F	••—•	6	—••••
G	——•	7	——•••
H	••••	8	———••
I	••	9	————•
J	•———	0	—————
K	—•—		
L	•—••		
M	——		
N	—•		
O	———		
P	•——•		
Q	——•—		
R	•—•		
S	•••		
T	—		
U	••—		
V	•••—		
W	•——		
X	—••—		
Y	—•——		
Z	——••		

Issue. Ambiguity.

• • • _ _ _ • • •

SOS ?
IAMIE ?
EEWNI ?
V7O ?

Remark. Separate words with medium gap.

Variable-length coding

Q. How do we avoid ambiguity?
A. Ensure that no codeword is a prefix of another.

char	code	
S	•••	← prefix of V
E	•	← prefix of I, S
I	••	← prefix of S
V	•••_	

Ex 1. Fixed-length code.
Ex 2. Append special stop symbol to each codeword.

Ex 3. Custom prefix-free code.

char	code
a	0
b	111
c	1010
d	100
r	110
!	1101

a	b	r	a	c	a	d	a	b	r	a	!
0	1 1 1	1 1 0	0	1 0 1 0	0	1 0 0	0	1 1 1	1 1 0	0	1 0 1 1

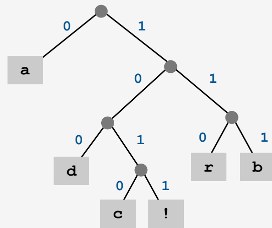
28 bits to encode message (vs. 36 bits for fixed-length 3-bit code)

Prefix-free code: encoding and decoding

Q. How to represent a prefix-free code?

A. Binary trie.

- Symbols are stored in leaves.
- Codeword is path to leaf.



Encoding.

- Method 1: start at leaf; follow path up to the root, and print bits in reverse order.
- Method 2: create ST of symbol-codeword pairs.

Decoding.

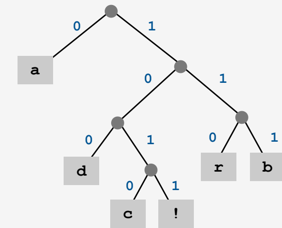
- Start at root of tree.
- Go left if bit is 0; go right if 1.
- If leaf node, print symbol and return to root.

char	code
a	0
b	111
c	1010
d	100
r	110
!	1011

Representing the code

Q. How to transmit the trie?

A. Send preorder traversal of trie.



* used as sentinel for internal node

```
*a**d*c!*rb
12
0111110010100100011111001011
```

preorder traversal

chars to decode

the message bits (pack 8 to the byte)

char	code
a	0
b	111
c	1010
d	100
r	110
!	1011

Note. If message is long, overhead of transmitting trie is small.

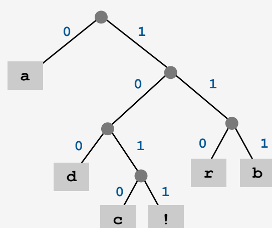
Prefix-free decoding: Java implementation

```
public class PrefixFreeDecoder
{
   private Node root = new Node();
   private class Node
   {
      private char ch;
      private Node left, right;
      public Node()
      {
         ch = StdIn.readChar();
         if (ch == '*')
         {
            left  = new Node();
            right = new Node();
         }
      }

      public boolean isInternal()
      {
         return left != null && right != null;
      }

   public void decode()
   {  /* See next slide. */  }

}
```

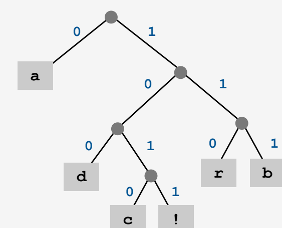


build binary trie from preorder traversal
`*a**d*c!*rb`

Prefix-free decoding: Java implementation (cont)

```
public void decode()
{
   int N = StdIn.readInt();
   for (int i = 0; i < N; i++)
   {
      Node x = root;
      while (x.isInternal())
      {
         char bit = StdIn.readChar();
         if      (bit == '0') x = x.left;
         else if (bit == '1') x = x.right;
      }
      StdOut.print(x.ch);
   }
}
```

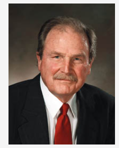
use bits, not chars in actual applications



decode message **abracadabra!** *from bits*
12
0111110010100100011111001011

Huffman coding

Q. What is the best variable-length code for a given message?
A. Huffman code.



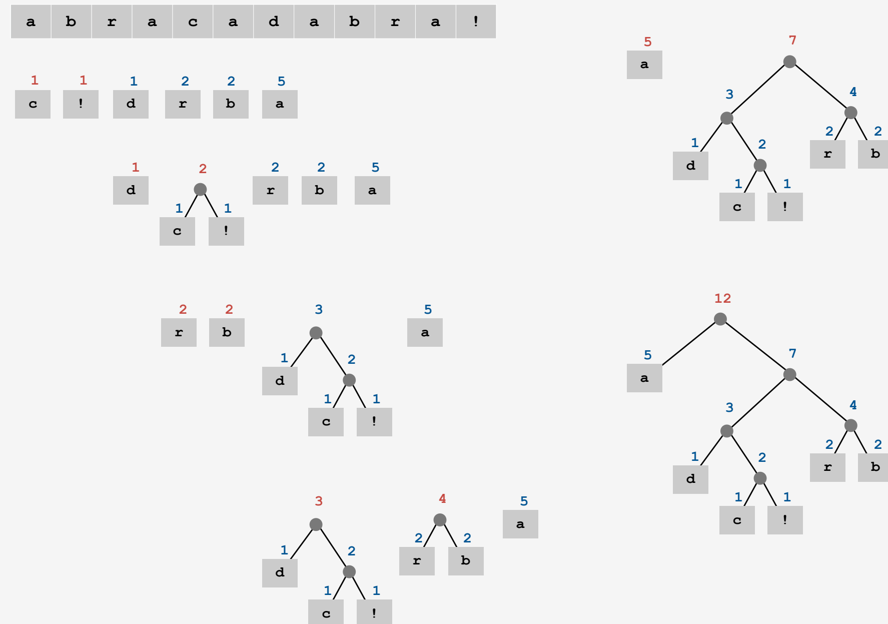
David Huffman

To compute Huffman code:

- Count frequency p_s for each symbol s in message.
- Start with one node corresponding to each symbol s (with weight p_s).
- Repeat until single trie formed:
  - select two tries with min weight p_1 and p_2
  - merge into single trie with weight $p_1 + p_2$

Applications. JPEG, MP3, MPEG, PKZIP, GZIP, …

Huffman coding example



Huffman trie construction code

```
int[] freq = new int[R];
for (int i = 0; i < input.length(); i++)
   freq[input.charAt(i)]++;

MinPQ<Node> pq = new MinPQ<Node>();
for (int r = 0; r < R; r++)
   if (freq[r] > 0)
      pq.insert(new Node((char) r, freq[r], null, null));

while (pq.size() > 1)
{
   Node x = pq.delMin();
   Node y = pq.delMin();
   Node parent = new Node('*', x.freq + y.freq, x, y);
   pq.insert(parent);
}
root = pq.delMin();
```

tabulate frequencies

initialize PQ

merge tries

two subtrees

internal node marker

total frequency

Huffman encoding summary

Proposition. [Huffman 1950s] Huffman coding is an optimal prefix-free code.

no prefix-free code uses fewer bits

Implementation.

- Pass 1: tabulate symbol frequencies and build trie.
- Pass 2: encode file by traversing trie or lookup table.

Running time. Use binary heap ⇒ $O(N + R \log R)$.

input chars

distinct symbols

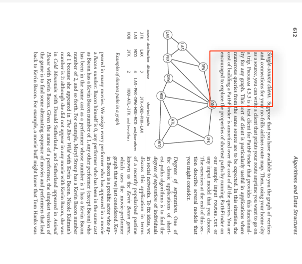
Q. Can we do better? [stay tuned]

- fixed-length codes
- variable-length codes
- **an application**
- adaptive codes

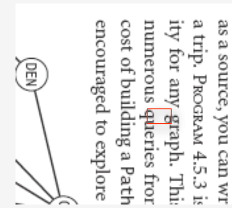
An application: compress a bitmap

Typical black-and-white-scanned image.

- 300 pixels/inch.
- 8.5-by-11 inches.
- 300*8.5*300*11 = 8.415 million bits.



Observation. Bits are mostly white.



Typical amount of text on a page.
40 lines * 75 chars per line = 3000 chars.

Natural encoding of a bitmap

Natural encoding. $(19 \times 51) + 6 = 975$ bits.

one bit per pixel — to encode number of characters per line

```
000000000000000000000000000011111111111111000000000
000000000000000000000000001111111111111111110000000
000000000000000000000001111111111111111111111110000
000000000000000000000011111111111111111111111111000
000000000000000000001111111111111111111111111111110
000000000000000000011111110000000000000000001111111
000000000000000000011111000000000000000000000011111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000001111000000000000000000000001110
000000000000000000000011100000000000000000000111000
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011000000000000000000000000000000000000000000000011
```

19-by-51 raster of letter 'q' lying on its side

Run-length encoding of a bitmap

Run-length encoding. $(63 \times 6) + 6 = 384$ bits.

63 6-bit run lengths

```
000000000000000000000000000011111111111111000000000
000000000000000000000000001111111111111111110000000
000000000000000000000001111111111111111111111110000
000000000000000000000011111111111111111111111111000
000000000000000000001111111111111111111111111111110
000000000000000000011111110000000000000000001111111
000000000000000000011111000000000000000000000011111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000001111000000000000000000000001110
000000000000000000000011100000000000000000000111000
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011000000000000000000000000000000000000000000000011
```

19-by-51 raster of letter 'q' lying on its side

```
51
28 14 9
26 18 7
23 24 4
22 26 3
20 30 1
19 7 18 7
19 5 22 5
19 3 26 3
19 3 26 3
19 3 26 3
19 3 26 3
20 4 23 3 1
22 3 20 3 3
1 50
1 50
1 50
1 50
1 50
1 2 46 2
```

run-length encoding

Run-length encoding

Goal. Exploit long runs of repeated characters.
Bitmaps Runs alternate between 0 and 1; just output run lengths.

Q. How to encode run lengths? (!)

`0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0` *20-bit message*

`2 6 2 5 5` *run lengths*

`0 1 0 1 1 0 0 1 0 1 0 1 1 0 1` *run lengths encoded using 3-bit code (15 bits)*

```
001: 1
010: 2
101: 3
100: 4
101: 5
110: 6
111: 7
```

Note. Runs are long in typical applications (such as black-and-white bitmaps).

RLE and Huffman codes in the wild

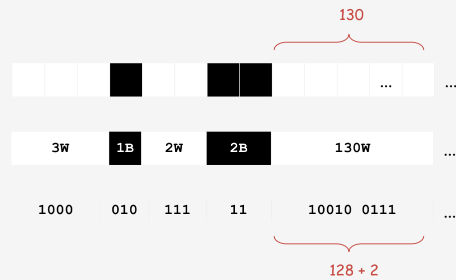
ITU-T T4 Group 3 Fax. [for black-and-white bitmap images]

- Up to 1728 pixels per line.
- Typically mostly white.

Step 1. Use run-length encoding.
Step 2. Encode run lengths using two Huffman codes.

one for white and one for black

based on statistics from huge number of faxes



run	white	black
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
…	…	…
63	00110100	000001100111
64+	11011	0000001111
128+	10010	000011001000
…	…	. . .
1728+	010011011	0000001100101

Black and white bitmap compression: another approach

Fax machine (~1980).

- Slow scanner produces lines in sequential order.
- Compress to save time (reduce number of bits to send).

Electronic documents (~2000).

- High-resolution scanners produce huge files.
- Compress to save space (reduce number of bits to save).

Idea.

- use OCR to get back to ASCII (!)
- use Huffman on ASCII string (!)

Bottom line. Any extra information about file can yield dramatic gains.

- fixed-length codes
- variable-length codes
- an application
- **adaptive codes**

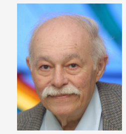
Statistical methods

Static model. Same model for all texts.
- Fast.
- Not optimal: different texts have different statistical properties.
- Ex: ASCII, Morse code.

Dynamic model. Generate model based on text.
- Preliminary pass needed to generate model.
- Must transmit the model.
- Ex: Huffman code.

Adaptive model. Progressively learn and update model as you read text.
- More accurate modeling produces better compression.
- Decoding must start from beginning.
- Ex: LZW.

Lempel-Ziv-Welch encoding




LZW encoding.
- Create ST associating a fixed-length codeword with some previous substring.
- When input matches string in ST, output associated codeword.
- Length of strings in ST grows, hence compression.

To send (encode) message M.
- Find longest string s in ST that is a prefix of unsent part of M.
- Send codeword associated with s.
- Add s · x to ST, where x is next char in M.

Ex. ST: `a`, `aa`, `ab`, `aba`, `abb`, `abaa`, `abaab`, `abaaa`.
- unsent part of M: `abaababbb`...
- s = `abaab`, x = `a`.
- Output integer associated with s; insert `abaaba` into ST.

LZW encoding example

input	code	add to ST
a	97	ab
b	98	br
r	114	ra
a	97	ac
c	99	ca
a	97	ad
d	100	da
a		
b	128	abr
r		
a	130	rac
c		
a	132	cad
d		
a	134	dab
b		
r	129	bra
a	97	

ASCII

key	value
NUL	0
...	...
a	97
b	98
c	99
d	100
e	101
f	102
g	103
...	...
r	114
...	...
DEL	127

ST

key	value
ab	128
br	129
ra	130
ac	131
ca	132
ad	133
da	134
abr	135
rac	136
cad	137
dab	138
bra	139
...	...

To send (encode) M.
- Find longest string s in ST that is a prefix of unsent part of M
- Send integer associated with s.
- Add s · x to ST, where x is next char in M.

LZW encoding example

input	code	add to ST
a	97	ab
b	98	br
r	114	ra
a	97	ac
c	99	ca
a	97	ad
d	100	da
a		
b	128	abr
r		
a	130	rac
c		
a	132	cad
d		
a	134	dab
b		
r	129	bra
a	97	

Message.
- 7-bit ASCII.
- 19 chars.
- 133 bits.

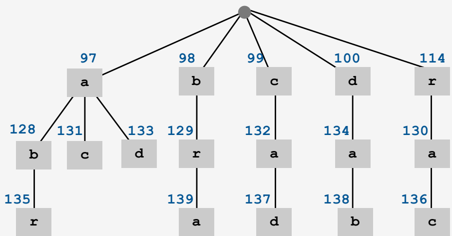
Encoding.
- 7-bit codewords.
- 14 codewords.
- 112 bits.

Key point. Don't need to send ST (!)

LZW encode ST implementation

Q. How to do longest prefix match?
A. Use a trie for the ST.

Encode.
- Lookup string suffix in trie.
- Output ST index at bottom.
- Add new node to bottom of trie.



ASCII

key	value
NUL	0
...	...
a	97
b	98
c	99
d	100
e	101
f	102
g	103
...	...
r	114
...	...
DEL	127

ST

key	value
ab	128
br	129
ra	130
ac	131
ca	132
ad	133
da	134
abr	135
rac	136
cad	137
dab	138
bra	139
...	...

Note. All prefixes of strings are also in ST.

Lempel-Ziv-Welch decoding

LZW decoding.
- Create ST and associate an integer with each useful string.
- When input matches string in ST, output associated integer.
- Length of strings in ST grows, hence compression.
- Decode by rebuilding ST from code.

To decode received message to M.
- Let s be ST entry associated with received integer.
- Add s to M.
- Add p · x to ST, where x is first char in s, p is previous value of s.

Ex. ST: **a, aa, ab, aba, abb, abaa, abaab, abaaa,**
- unsent part of M: **abaababbb**...
- s = **abaab**, x = **a**.
- Output integer associated with s; insert **abaaba** into ST.

LZW decoding example

codeword	output	add to ST
97	a	
98	b	ab
114	r	br
97	a	ra
99	c	ac
97	a	ca
100	d	ad
128	a	
	b	da
130	r	
	a	abr
132	c	
	a	rac
134	d	
	a	cad
129	b	
	r	dab
97	a	bra
255	STOP	

role of keys and values switched

key	value
0	
...	...
97	a
98	b
99	c
100	d
...	
114	r
...	...
127	

key	value
128	ab
129	br
130	ra
131	ac
132	ca
133	ad
134	da
135	abr
136	rac
137	cad
138	dab
139	bra
...	
255	

Use an array to implement ST

To decode received message to M.
- Let s be ST entry associated with received integer
- Add s to M.
- Add p · x to ST, where x is first char in s, p is previous value of s.

LZW implementation details

How big to make ST?
- How long is message?
- Whole message similar model?
- [many variations have been developed]

What to do when ST fills up?
- Throw away and start over. GIF
- Throw away when not effective. Unix compress
- [many other variations]

Why not put longer substrings in ST?
- [many variations have been developed]

LZW in the real world

Lempel-Ziv and friends.

- LZ77.
- LZ78.
- LZW.
- Deflate = LZ77 variant + Huffman.

LZ77 not patented ⇒ widely used in open source
LZW patent #4,558,302 expired in US on June 20, 2003
some versions copyrighted

PNG: LZ77.
Winzip, gzip, jar: deflate.
Unix compress: LZW.
Pkzip: LZW + Shannon-Fano.
GIF, TIFF, V.42bis modem: LZW.
Google: zlib which is based on deflate.

never expands a file

Lossless compression ratio benchmarks

Calgary corpus. Standard data compression benchmark.

year	scheme	bits / char
1967	ASCII	7.00
1950	Huffman	4.70
1977	LZ77	3.94
1984	LZMW	3.32
1987	LZH	3.30
1987	move-to-fromt	3.24
1987	LZB	3.18
1987	gzip	2.71
1988	PPMC	2.48
1994	SAKDC	2.47
1994	PPM	2.34
1995	Burrows-Wheeler	2.29
1997	BOA	1.99
1999	RK	1.89

← next assignment

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. [not covered in this course]

- JPEG, MPEG, MP3.
- FFT, wavelets, fractals, SVD, …

Limits on compression. Shannon entropy.

Theoretical limits closely match what we can achieve in practice.

Practical compression. Use extra knowledge whenever possible.