Data Compression

Some of these lecture slides have been adapted from:

- *Algorithms in C*, Robert Sedgewick.
- *Introduction to Data Compression*, Guy Blelloch.

Data Compression

Compression reduces the size of a file:
- **To save TIME when transmitting it.**
- **To save SPACE when storing it.**

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

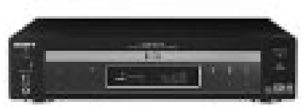
Who needs compression?
- **Moore's law: # transistors on a chip doubles every 18 months.**
- **Parkinson's law: data expands to fill space available.**
- **Text, Images, Sound, Video, . . .**

Applications of Data Compression

Generic file compression.
- **Files: GZIP, BZIP, BOA.**
- **Archivers: PKZIP.**
- **File systems: NTFS.**

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

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



Encoding and Decoding

Message. Data M we want to compress.
Encode. Generate a "compressed" representation C(M) that hopefully uses fewer bits.
Decode. Reconstruct original message or some approximation M'.

M → Encoder → C(M) → Decoder → M'

Compression ratio. bits in C(M) / bits in M.
Lossless. M = M'.
- **Text, C source code, executables.**
- **50-75% or lower.**

Lossy. M ~ M'.
- **Images, sound, video.**
- **10% or lower.**

Simple Ideas

Ancient ideas.

- Human language.
- Morse code.
- Braille.

Fixed length coding.

- 2-bit genetic code for DNA.

Genetic encoding

char	dec	binary
A	0	00
C	1	01
G	2	10
T	3	11

Simple Ideas

Ancient ideas.

- Human language.
- Morse code.
- Braille.

Fixed length coding.

- 2-bit genetic code for DNA.
- 7-bit ASCII code for text.

ASCII coding

char	dec	binary
NUL	0	0000000
…	…	…
>	62	0111110
?	63	0111111
@	64	1000000
A	65	1000001
B	66	1000010
C	67	1000011
…	…	…
~	126	1111110
DEL	127	1111111

a	b	r	a	c	a	d	a	b	r	a
1100001	1100010	1110010	1100001	1100011	1100001	1110100	1100001	1100010	1110010	1100001

$7 \times 11 = 77$ bits

Simple Ideas

Ancient ideas.

- Human language.
- Morse code.
- Braille.

Fixed length coding.

- 2-bit genetic code for DNA.
- 7-bit ASCII code for text.
- 3-bit "abracadabra" code.

abracadabra coding

char	dec	binary
a	0	000
b	1	001
c	2	010
d	3	011
r	4	100

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

$3 \times 11 = 33$ bits

Run-Length Encoding

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

Raster of letter ’q’, lying on its side

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

Run-Length Encoding

Natural encoding: $51 \times 19 + 6 = 975$ bits.
Run-length encoding: $63 \times 6 = 378$ bits.

Raster of letter 'q', lying on its side	RLE
000000000000000000000000000011111111111111000000000	28 14 9
000000000000000000000000001111111111111111110000000	26 18 7
000000000000000000000001111111111111111111111110000	23 24 4
000000000000000000000011111111111111111111111111000	22 26 3
000000000000000000001111111111111111111111111111110	20 30 1
000000000000000000011111110000000000000000001111111	19 7 18 7
000000000000000000011111000000000000000000000011111	19 5 22 5
000000000000000000011100000000000000000000000000111	19 3 26 3
000000000000000000011100000000000000000000000000111	19 3 26 3
000000000000000000011100000000000000000000000000111	19 3 26 3
000000000000000000011100000000000000000000000000111	19 3 26 3
000000000000000000001111000000000000000000000001110	20 4 23 3 1
000000000000000000000011100000000000000000000111000	22 3 20 3 3
011111111111111111111111111111111111111111111111111	1 50
011111111111111111111111111111111111111111111111111	1 50
011111111111111111111111111111111111111111111111111	1 50
011111111111111111111111111111111111111111111111111	1 50
011111111111111111111111111111111111111111111111111	1 50
011000000000000000000000000000000000000000000000011	1 2 46 2

Run-Length Encoding

Run-length encoding (RLE).

- Exploit long runs of repeated characters.
- Replace run by count followed by repeated character, but don't bother if run is less than 3.
  - AAAABBBAABBBBBCCCCCCCCDABCBAAABBBBCCCD
  - 4A3BAA5B8CDABCB3A4B3CD
- Annoyance: how to represent counts.
- Runs in binary file alternate between 0 and 1, so output count only.
- "File inflation" if runs are short.

Applications.

- Black and white graphics.
  - compression ratio improves with resolution!
- JPEG (Joint Photographic Experts Group).

Variable Length Encoding

Variable-length encoding.

- Use DIFFERENT number of bits to encode different character.
  - use fewer bits for more frequent letters (ETAONIS)
  - use more bits for rarer letters (ZQJXKVB)

Char	Freq	Char	Freq
E	12.51	M	2.53
T	9.25	F	2.30
A	8.04	P	2.00
O	7.60	G	1.96
I	7.26	W	1.92
N	7.09	Y	1.73
S	6.54	B	1.54
R	6.12	V	0.99
H	5.49	K	0.67
L	4.14	X	0.19
D	3.99	J	0.16
C	3.06	Q	0.11
U	2.71	Z	0.09

Linotype machine, 1886

Variable Length Encoding

Variable-length encoding.

- Use DIFFERENT number of bits to encode different character.

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

3-bit coding: $3 \times 11 = 33$ bits

char	encoding
a	1
b	001
c	0000
d	0001
r	01

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

variable length coding: 23 bits

Variable Length Decoding

But, then how do we decode?

- Variable length codes can be ambiguous.

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

char	encoding
a	1
b	0
c	10
d	01
r	00

- How do we avoid ambiguity?
  - One solution: ensure no encoding is a PREFIX of another.
  - 011 is a prefix of 011100.

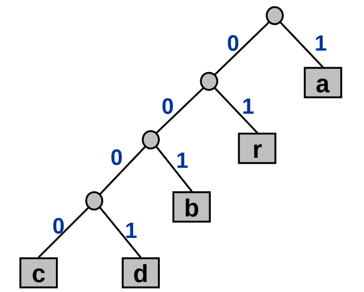
a	b			r		a	c				a	d				a	b			r		a
1	0	0	1	0	1	1	0	0	0	0	1	0	0	0	1	1	0	0	1	0	1	1

char	encoding
a	1
b	001
c	0000
d	0001
r	01

Implementing Prefix-Free Codes

How to represent?

- Use a binary trie.
  - symbols are stored in leaves
  - encoding is path to leaf



Encoding.

- Start at leaf of tree corresponding to symbol s.
- Follow path up to the root.
- Print bits in reverse order.

Decoding.

- Start at root of tree.
- Take right branch if bit is 0; left branch if 1.
- When at a leaf node, print symbol and return to root.

char	encoding
a	1
b	001
c	0000
d	0001
r	01

Huffman Coding

OK, but how do I find a good prefix-free coding scheme?

- Greed.

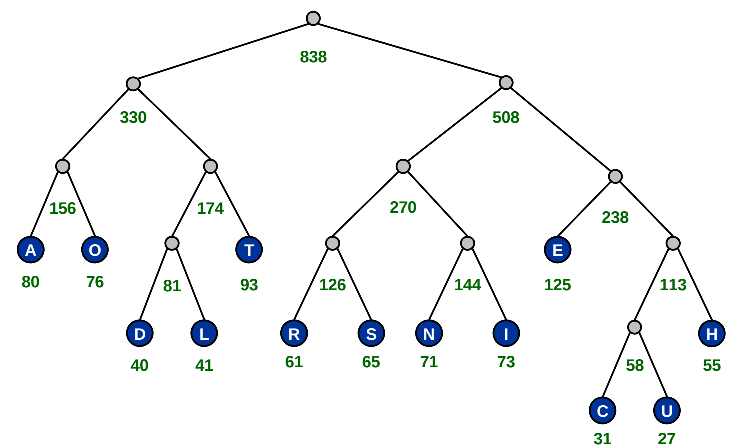
To compute Huffman prefix-free code:

- Count character frequencies p_s for each symbol s in file.
- Start with a forest of trees, each consisting of a single vertex corresponding to each symbol s with weight p_s.
- Repeat:
  - select two trees with min weight p_1 and p_2
  - merge into single tree with weight $p_1 + p_2$

Applications: JPEG, MP3, MPEG, PKZIP.

Huffman Coding Example

Char	Freq	Huff
E	125	110
T	93	000
A	80	000
O	76	011
I	73	1011
N	71	1010
S	65	1001
R	61	1000
H	55	1111
L	41	0101
D	40	0100
C	31	11100
U	27	11101
Total	838	3.62



Huffman Encoding

Theorem (Huffman, 1952). Huffman coding is optimal prefix-free code.

- No variable length code uses fewer bits.

Corollary. Greed is good.

Implementation.

- Two passes.
  - tabulate symbol frequencies and build trie
  - encode file by traversing trie
- Use priority queue for delete min and insert.
- O(M + N log N).

> M = file size
> N = # distinct symbols

Difficulties.

- Have to transmit code (trie).
- Not optimal!

What Data Can be Compressed?

US Patent 5,533,051 on "Methods for Data Compression."

- Capable of compressing 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. . . ."

Impossible claims for lossless compression.

- Consider all 1000 bit messages.
- 2^{1000} possible messages.
- Only $2^{999} + 2^{998} + \ldots + 1$ can be encoded with ≤ 999 bits.
- Only 1 in 2^{499} can even be encoded with ≤ 500 bits!

A Difficult File To Compress

One million pseudo-random characters (a – p)

```
fclkkacifobjofmkgdcoiicnfmcpcjfccabckjamolnihkbgobcjbngjiceeelpfgcjiihppenefllhglfemdemgahlb
piggmllmnefnhjelmgjncjcidlhkglhceninidmmgnobkeglpnadanfbecoonbiehglmpnhkkamdffpacjmgojmcaabp
cjcecplfbgamlidceklhfkkmioljdnoaagiheiapaimlcnlljniggpeanbmojgkccogpmkmoifioeikefjidbadgdcep
nhdpfjaeeapdjeofklpdeghidbgcaiemajllhnndigeihbebifemacfadnknhlbgincpmimdogimgeeomgeljfjgklkd
gnhafohonpjbmlkapddhmepdnckeajebmeknmeejnmenbmnnfefdbhpmigbbjknjmobimamjjaaaffhlhiggaljbaijn
ebidpaeigdgoghcihodnlhahllhhoojdfacnhadhgkfahmeaebccacgeojgikcoapknlomfignanedmajinlompjoaif
iaejbcjcdibpkofcbmjiobbpdhfilfajkhfmppcngdneeinpnfafaeladbhhifechinknpdnplamackphekokigpddmm
jnbngklhibohdfeaggmclllmdhafkldmimdbplggbbejkcmhlkjocjjlcngckfpfakmnpiaanffdjdlleiniilaenbni
kgfnjfcophbgkhdgmfpoehfmkbpiaignphogbkelphobonmfghpdgmkfedkfkchceeldkcofaldinljjcgafimaanelm
fkokcjekefkbmegcgjifjcpjppnabldjoaafpbdafifgcoibbcmoffbbgigmngefpkmbhbghlbdjngenldhgnfbdlcmj
dmoflhcogfjoldfjpaokepndejmnbiealkaofifekdjkgedgdlgbioacflfjlafbcaemgpjlagbdgilhcfdcamhfmppf
gohjphlmhegjechgdpkkljpndphfcnnganmbmnggpphnckbieknjhilafkegboilajdppcodpeoddldjfcpialoalfeo
mjbphkmhnpdmcpgkgeaohfdmcnegmibjkajcdcpjcpgjminhhakihfgiiachfepffnilcooiciepoapmdjniimfbolch
kibkbmhbkgconimkdchahcnhapfdkiapikencegcjapkjkfljgdlmgncpbakhjidapbldcgeekkjaoihbnbigmhboeng
pmedliofgioofdcphelapijcegejgcldcfodikalehbccpbbcfakkblmoobdmdgdkafbbkjnidoikfakjclbchambcpa
epfeinmenmpoodadoecbgbmfkkeabilaoeoggghoekamaibhjibefmoppbhfbhffapjnodlofeihmjahmeipejlfhloe
fgmjhjnlomapjakhhjpncomippeanbikkhekpcfgbgkmklipfbiikdkdcbolofhelipbkbjmjfoempccneaebklibmca
ddlmjdcajpmhhaeedbbfpjafcndianlfcjmmbfncpdcccodeldhmnbdjmeajmboclkggojghlohlbhgjkhkmclohkgja
mfmcchkchmiadjgjhjehflcbklfifackbecgjoggpbkhlcmfhipflhmnmifpjmcoldbeghpcekhgmnahijpabnomnokl
djcpppbcpgcjofngmbdcpeeeiiiclmbbmfjkhlanckidhmbeanmlabncnccpbhoafajjicnfeenppoekmlddholnbdja
pbfcajblbooiaepfmmeoafedflmdcbaodgeahimcgpcammjljoebpfmghogfckgmomecdipmodbcempidfnlcggpgbff
oncajpncomalgoiikeolmigliikjkolgolfkdgiijjiooiokdihjbbofiooibakadjnedlodeeiijkliicnioimablfd
pjiafcfineecbafaamheiipegegibioocmlmhjekfikfeffmddhoakllnifdhckmbonbchfhhclecjamjildonjjdpif
ngbojianpljahpkindkdoanlldcbmlmhjfomifhmncikoljjhebidjdphpdepibfgdonjljfgifimniipogockpidamn
kcpipglafmlmoacjibognbplejnikdoefccdpfkomkimffgjgielocdemnblimfmbkfbhkelkpfoheokfofochbmifle
ecbglmnfbnfncjmefnihdcoeiefllemnohlfdcmbdfebdmbeebbalggfbajdamplphdgiimehglpikbipnkkecekhilc
hhhfaeafbbfdmcjojfhppongl kfdmhjpcieofcnjgkpibcbiblfpnjlejkcppbhopohdghljlcokhdoahfmlglbdklia
jbmnkkfcoklhlelhjhoiginaimgcabcfebmjdnbfhohkjphnklcbhcjpgbadakoecbkjcaebbanhnfhpnfkfbfpohmnk
ligpgfkjadomdjjnhlnfailfpcmnololdjekeolhdkebiffebajjpclghllmemegncknmkkeoogilijmmkomllbkkabe
lmodcohdhppdakbelmlejdnmbfmcjdebefnjihnejmnogeeafldabjcgfoaehldcmkbnbafpciefhlopicifadbppgmf
ngecjhefnkbjmliodhelhicnfoongngemddepchkokdjafegnpgledakmbcpcmkckhbffeihpkajginfhdolfnlgnade
famlfocdibhfkiaofeegppcjilndepleihkpkkgkphbnkggjiaolnolbjpobjdcehglelckbhjilafccfipgebpc....
```


A Difficult File To Compress

170 characters

```c
#include <stdio.h>
#include <stdlib.h>
#define N 1000000
int main(void) {
   int i;
   for (i = 0; i < N; i++)
      putchar('a' + rand() % 16);
   return 0;
}
```

> Easy to store 2 8-bit characters per byte for 50% compression.

Compression Achieved

```
% gcc rand.c
% a.out > temp.txt
% compress -c < temp.txt > temp.Z
% gzip -c < temp.txt > temp.gz
% bzip2 -c < temp.txt > temp.bz2
```

Resulting Files

```
% ls -lr temp*
    170 rand.c
1000000 temp.txt
 576861 temp.Z
 570872 temp.gz
 499329 temp.bz2
```

Information Theory

Intrinsic difficulty of compression.

- Short program generates large data file.
- Optimal compression algorithm has to discover program!
- Undecidable problem.

So how do we know if our algorithm is doing well?

- Want lower bound on # bits required by ANY compression scheme.

Language Model

How compression algorithms work?

- Exploit bias on input messages.
- Word "Princeton" occurs more frequently than "Yale."
- White patches occur in typical images.

Compression is all about probability.

- Formulate probabilistic model to predict symbols.
  - simple: character counts, repeated strings
  - complex: models of a human face
- Use model to encode message.
- Use same model to decode message.

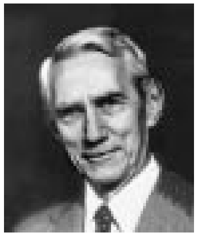
Example. Order 0 Markov model.

- Each symbol s generated randomly with fixed probability p(s), independently.

Entropy

Information content of symbol. $I(s) = \log_2 \frac{1}{p(s)}$

- Intuition: numbers of bits you should use to encode the symbol.
- Lower probability $\Rightarrow$ higher information content.



Claude Shannon
(1916 - 2001)

Entropy. (Shannon 1948) $H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$

- Weighted average of information content over all symbols.
- Interface between coding and model.

Entropy Examples

Entropy examples over two-symbol alphabet {a, b}.

	p(a)	p(b)	H(S)
Model 1	1/2	1/2	1
Model 2	0.900	0.100	0.469
Model 3	0.990	0.010	0.0808
Model 4	1	0	0

Entropy examples over five-symbol alphabet {a, b, c, d, e}.

	p(a)	p(b)	p(c)	p(d)	p(e)	H(S)
Model 5	1/5	1/5	1/5	1/5	1/5	2.322

Entropy and Compression

Shannon's theorem (1948). Avg # bits per symbol ≥ entropy.

- ANY compression scheme must use at least H(S) bits per symbol on average.
- Cornerstone result of information theory.
- Assumes data source S is order 0 Markov model (but basic ideas extend to more general models).

Huffman's coding theorem (1952). Huffman code is optimal.

- H(S) ≤ avg # bits per symbol ≤ H(S) + 1.
- Assumes data source S is order 0 Markov model.
- Wastes up to 1 bit per symbol.
  - if H(S) is very small (close to 0), this matters
  - can do better with "arithmetic coding"

Entropy of the English Language

How much information is in each character of English language?

How can we measure it? Shannon's experiment (1951).

- Asked humans to predict next character given previous text.
- The number of guesses required for right answer:

# of guesses	1	2	3	4	5	≥ 6
Probability	0.79	0.08	0.03	0.02	0.02	0.05

- Shannon's estimate = 0.6 - 1.3.

Lossless Compression Ratio for Calgary Corpus

Year	Scheme	Bits / char
----	ASCII	7.00
1950	Huffman	4.70
1977	LZ77	3.94
1984	LZMW	3.32
1987	LZH	3.30
1987	Move-to-front	3.24
1987	LZB	3.18
1987	Gzip	2.71
1988	PPMC	2.48
1988	SAKDC	2.47
1994	PPM	2.34
1995	Burrows-Wheeler	2.29
1997	BOA	1.99
1999	RK	1.89

Entropy	Bits / char
Char by char	4.5
8 chars at a time	2.4
Asymptotic	1.3

Statistical Methods

Estimate symbol frequencies and assign codewords based on this.

Static model. Same distribution for all texts.

- Fast.
- Not optimal since different texts exhibit different distributions.
- Ex: ASCII, Morse code, Huffman "Etaoin Shrdlu" code.

Dynamic model. Generate model based on text.

- Preliminary pass needed to compute frequencies.
- Must transmit encoding table (the model).
- Ex: Huffman code using frequencies from text.

Adaptive models. Progressively learn distribution as you read text.

- More accurate models produce better compression.
- Decoding must start from beginning.
- Ex: LZW.

LZW Algorithm

Lempel-Ziv-Welch (variant of LZ78).
- Adaptively maintain dictionary of useful strings.
- If input matches word in dictionary, output index instead of string.

Algorithm.
- Find longest word W in dictionary that is a prefix of string starting at current index.
- Output index for W followed by x = next symbol.
- Add Wx to dictionary.

Example.
- Dictionary: a, aa, ab, aba, abb, abaa, abaab.
- String starting at current index: . . . abaababbb. . .
- W = abaab, x = a.
- Output index for W, insert abaaba into dictionary.

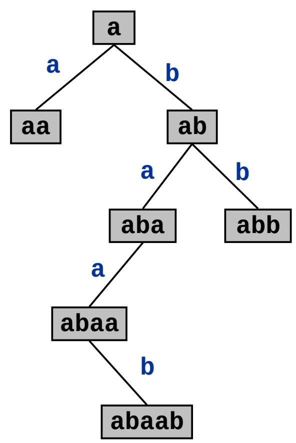
LZW Example

Input	Send
SEND	256
i	105
t	116
t	116
y	121
_	32
b	98
i	
t	258
t	
y	260
_	
b	262
i	
t	258
_	
b	
i	266
n	110
STOP	257

Dictionary			
Index	Word	Index	Word
0		258	it
...		259	tt
32	_	260	ty
...		261	y_
97	a	262	_b
98	b	263	bi
...		264	itt
122	z	265	ty_
...		266	_bi
256	SEND	267	it_
257	STOP	268	_bin

LZW Implementation

Implementation.
- Use binary trie for dictionary.
  - by construction, all prefixes of every dictionary word also in dictionary
- Create dictionary on-the-fly.
- Encode.
  - lookup string suffix in trie
  - output dictionary index at bottom
  - add new node to bottom of trie
- Decode.
  - build trie (faster)
  - build dictionary (less space)



What to do when dictionary gets too large?
- Throw away and start over. GIF
- Throw away when not effective. Unix compress
- Throw away least recently used item.

Summary

Lossless compression.
- Simple approaches.
  - RLE
- Represent fixed length symbols with variable length codes.
  - Huffman
- Represent variable length symbols with fixed length codes.
  - LZW

Lossy compression.
- Not covered in COS 226.
- Signal processing, wavelets, fractals, ...

Limits on compression.
- Entropy.