Lecture #39: Compression

ɔresentation is largely taken from CS61B lectures by

Compression and Git

- a new object in the repository each time a changed file is committed.
- et crowded as a result.
- e, it *compresses* each object.
- d then (such as when sending or receiving from another it packs objects together into a single file: a "packfile."
- sticking the files together, uses a technique called *ession.*

Delta Compression

- ere will be many versions of a file in a Git repository: nd previous edits of it, each in different commits.
- keep track explicitly of which file came from where, hard in general:
  - file is split into two, or two are spliced together?
- ss that files with same name and (roughly) same size in are probably versions of the same file.
- happens, store one of them as a pointer to the other, changes.

Delta Compression (II)

o versions

V1	V2
ː fully open to my awful	My eyes are fully open to my awful situation.
t once to Roderick and oration. I shall tell him	I shall go at once to Roderick and make him an oration.
ed my forgotten moral	I shall tell him I've recovered my forgotten moral senses, and don't give twopence halfpenny for any consequences.

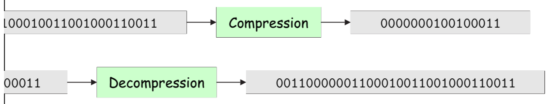
V1	V2
lines from V2]	My eyes are fully open to my awful situation. I shall go at once to Roderick and make him an oration. I shall tell him I've recovered my forgotten moral senses, and don't give twopence halfpenny for any consequences.

Two Unix Compression Programs

```
:37.pic.in      # The GNU version of ZIP
:t37.pic.in     # Another compression program
.pic*
           Size
          (bytes)
:s61b cs61b 31065 Apr 27 23:36 lect37.pic.in
:s61b cs61b 10026 Apr 27 23:36 lect37.pic.in.bz2 # Roughly 1/3 size
:s61b cs61b 10270 Apr 27 23:36 lect37.pic.in.gz
:37.pdf
.pdf*
:s61b cs61b 124665 Mar 30 13:46 lect37.pdf
:s61b cs61b 101125 Mar 30 13:46 lect37.pdf.gz    # Roughly 81% size
:t37.pic.in.gz > lect37.pic.in.ungzip # Uncompress
pic.in lect37.pic.in.ungzip
    # No difference from original (lossless)
37.pic.in.gz > lect37.pic.in.gz.gz
.pic*gz
:s61b cs61b  10270 Apr 27 23:36 lect37.pic.in.gz
:s61b cs61b  10293 Apr 28 00:16 lect37.pic.in.gz.gz
```

Compression and Decompression

- *on algorithm* converts a stream of symbols into another, am.
- *lossless* if the algorithm is *invertible* (no information
- mbol is the bit:



- nply replaced the 8-bit ASCII bit sequences for digits xample, the single character '0' is encoded as 0x30=0b00110000) *inary-coded decimal*).
- bit sequences *codewords*, which we associate with the ne original, uncompressed text.
- r than 50% compression with English text.

Prefix Free Codes

ıeeds pauses between codewords to prevent ambiguities.

▬●●●●▬▬●●●●

ATH, BABE, or BATH.

is that Morse code allows many codewords to be *prefixes* s, so that it's difficult to know when you have come to ıe.

s to devise *prefix-free codes*, in which no codeword is nother.

ays knows when a codeword ends.

Prefix-Free Examples

ding A

1
01
001
0001
00001
000001

Encoding B

space	111
E	010
T	1000
A	1010
O	1011
I	1100
. . .	

, "I ATE" is unambiguously

00100101 in Encoding A, or

101000010 in Encoding B.

tructures might you use to. . .
: HashMap or array Decode?

Shannon-Fano Coding

quency	Encoding
0.35	
0.17	
0.17	
0.16	
0.15	

☞ 👍 ✓ ⚓ ✎

ıncies of all characters in text to be compressed.

ːd characters into two groups of roughly equal frequency.

group with leading 0, right group with leading 1.

all groups are of size 1.

Example: Morse Code

ıple to transmit.

three symbols:
, and pause.
tween codewords.

A ●▬
B ▬●●●
C ▬●▬●
D ▬●●
E ●
F ●●▬●
G ▬▬●
H ●●●●
I ●●
J ●▬▬▬
K ▬●▬
L ●▬●●
M ▬▬
N ▬●
O ▬▬▬
P ●▬▬●
Q ▬▬●▬
R ●▬●
S ●●●
T ▬

U ●●▬
V ●●●▬
W ●▬▬
X ▬●●▬
Y ▬●▬▬
Z ▬▬●●

0 ▬▬▬▬▬
1 ●▬▬▬▬
2 ●●▬▬▬
3 ●●●▬▬
4 ●●●●▬
5 ●●●●●
6 ▬●●●●
7 ▬▬●●●
8 ▬▬▬●●
9 ▬▬▬▬●

Prefix-Free Examples

ding A

1
01
001
0001
00001
000001

Encoding B

space	111
E	010
T	1000
A	1010
O	1011
I	1100
. . .	

, "I ATE" is unambiguously

00100101 in Encoding A, or

101000010 in Encoding B.

tructures might you use to. . .
ode?

Prefix-Free Examples

ding A

1
01
001
0001
00001
000001

Encoding B

space	111
E	010
T	1000
A	1010
O	1011
I	1100
. . .	

, "I ATE" is unambiguously

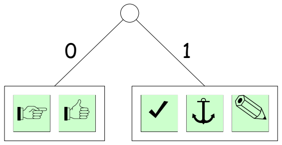
00100101 in Encoding A, or

101000010 in Encoding B.

tructures might you use to. . .
: HashMap or array Decode? Ans: Trie

Shannon-Fano Coding

equency	Encoding
0.35	0. . .
0.17	0. . .
0.17	1. . .
0.16	1. . .
0.15	1. . .



ncies of all characters in text to be compressed.

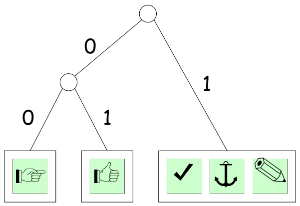
d characters into two groups of roughly equal frequency.

group with leading 0, right group with leading 1.

all groups are of size 1.

Shannon-Fano Coding

equency	Encoding
0.35	00
0.17	01
0.17	1. . .
0.16	1. . .
0.15	1. . .



ncies of all characters in text to be compressed.

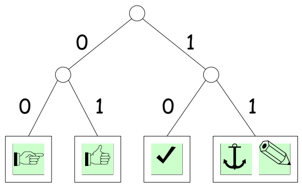
d characters into two groups of roughly equal frequency.

group with leading 0, right group with leading 1.

all groups are of size 1.

Shannon-Fano Coding

equency	Encoding
0.35	00
0.17	01
0.17	10
0.16	11. . .
0.15	11. . .



ncies of all characters in text to be compressed.

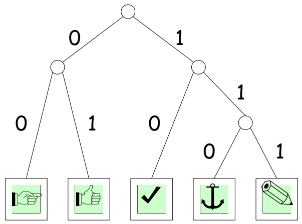
d characters into two groups of roughly equal frequency.

group with leading 0, right group with leading 1.

all groups are of size 1.

Shannon-Fano Coding

equency	Encoding
0.35	00
0.17	01
0.17	10
0.16	110
0.15	111



ncies of all characters in text to be compressed.

d characters into two groups of roughly equal frequency.

group with leading 0, right group with leading 1.

all groups are of size 1.

Can We Do Better?

encoding of symbols to codewords that are bitstrings
r a particular text if it encodes the text in the fewest

o coding is good, but not optimal.

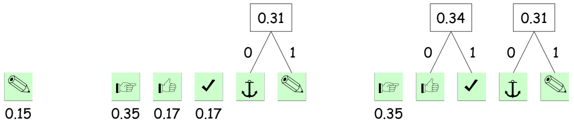
solution was found by an MIT graduate student, David
a class taught by Fano. The students were given the
king the final or solving this problem (i.e., finding the
a proof of optimality).

called *Huffman coding*.

Fano assigned a problem he hadn't been able to solve.
o that occasionally.

article.

Huffman Coding



bol in a node labeled with the symbol's relative frequency

ollowing until there is just one node:

he two nodes with smallest frequencies as children of a
node whose frequency is the sum of those of the two
ng combined.

dge to the left child be labeled '0' and to the right be
.

tree shows the encoding for each symbol: concatenate
els on the path from the root to the symbol.

Comparison

mbol	Frequency	Shannon-Fano	Huffman
☞	0.35	00	0
👍	0.17	01	100
✓	0.17	10	101
⚓	0.16	110	110
✎	0.15	111	111

Shannon-Fano coding takes a weighted average of 2.31
. while Huffman coding takes 2.3.

Example of LZW encoding

trivial mapping of codewords to single symbols.

ting a codeword that matches the longest possible prefix,
maining input, add a new codeword Y that maps to the
followed by the next input symbol.

llowing text as an example:

`cdabcdeabcdefabcdefgabcdefgh"`

(B), the encoding of B. Our codewords will consist of
des (0x00–0x7f).

LZW Step 1

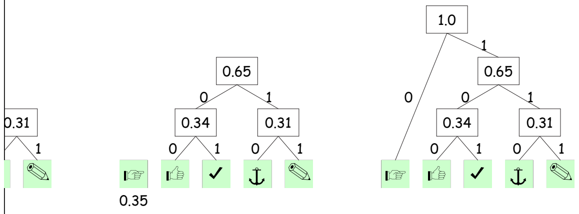
labcdeabcdefabcdefgabcdefgh

match in the table is ‘a’, so output 0x61,

to the table with a new code.

C(B) = 0x61

Huffman Coding



bol in a node labeled with the symbol’s relative frequency

ollowing until there is just one node:

he two nodes with smallest frequencies as children of a
node whose frequency is the sum of those of the two
ng combined.

dge to the left child be labeled ‘0’ and to the right be
.

g tree shows the encoding for each symbol: concatenate
els on the path from the root to the symbol.

LZW Coding

ave used systems with one codeword per symbol.

r compression, must encoded *multiple* symbols per codeword.

w us to code strings such as

```
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ababababababababababababababababab
eabcdefabcdefgabcdefghabcdefghiabcd
```

t can be than less than 43 × weighted average symbol

ling, we create new codewords as we go along, each
g to substrings of the text:

n a trivial mapping of codewords to single symbols.

putting a codeword that matches the longest possible
of the remaining input, add a new codeword Y that maps
ostring X followed by the next input symbol.

LZW Step 0: Initial state

labcdeabcdefabcdefgabcdefgh

C(B) =

LZW Step 3

labcdeabcdefabcdefgabcdefgh

match in the table for remaining input is ‘b’, so output

to the table with a new code.

C(B) = 0x616162

LZW Step 5

labcdeabcdefabcdefgabcdefgh

match in the table for remaining input is now ‘c’, so

to the table with a new code.

C(B) = 0x6161628163

LZW Step 6

labcdeabcdefabcdefgabcdefgh

match in the table for remaining input is now ‘abc’, so

d to the table with a new code.

C(B) = 0x616162816383

LZW Step 2

labcdeabcdefabcdefgabcdefgh

match in the table for remaining input is still ‘a’, so

to the table with a new code.

C(B) = 0x6161

LZW Step 4

labcdeabcdefabcdefgabcdefgh

match in the table for remaining input is now ‘ab’, so
(half as many bits as ‘ab’).

to the table with a new code.

C(B) = 0x61616281

LZW Step 6

abcdeabcdefabcdefgabcdefgh

match in the table for remaining input is now ???, so

to the table with a new code.

C(B) = 0x6161628163??

LZW Step 7

abcdeabcdefabcdefgabcdefgh

match in the table for remaining input is now 'd', so

to the table with a new code.

C(B) = 0x61616281638364

- What's next?
- What is the complete encoding? (When reviewing, try to figure it out before looking at the next slide.)

Decompression

h different input creates a different table, it would need to provide the generated table in order to decode

, though, we don't!

t, starting with the same initial table we did before, x00–0x7f already assigned, we're given

$$C(B) = \mathrm{0x616162816383}$$

ind B.

t starts with aab. What's next?

LZW Decompression, Step 1

281638364

in the table, so add it to B.

previous codeword yet, so don't add anything to the

B = a

LZW Step 7

abcdeabcdefabcdefgabcdefgh

match in the table for remaining input is now 'd', so

to the table with a new code.

C(B) = 0x61616281638364

LZW Final State

abcdeabcdefabcdefgabcdefgh (200 bits)

Code	String
0x87	abcde
0x88	ea
0x89	abcdef
0x8a	fa
0x8b	abcdefg
0x8c	ga
0x8d	abcdefgh

$$C(B) = \mathrm{0x61616281638364856587668967 8b68}$$

(120 bits)

How might you represent this table to allow easily est prefix at each step?

---

constructing the Coding Table (I)

econstruct the table as we process each codeword in

an "the symbols encoded by codeword X," and let Y_k ter k of string Y.

leword, X, in $C(B)$, add $S(X)$ to our result.

decoded two consecutive codewords, X_1 and X_2, *add* a d that maps to $S(X_1) + S(X_2)_0$

apitulate a step in the compression operation that created first place.

from left to right, the table will (almost) always already napping we need for the next codeword.

LZW Decompression, Step 3

281638364

in the table, so add it to B.

codewords—S(0x61)='a' and S(0x62)='b'—so add 'ab' to a new codeword.

B = aab

LZW Decompression, Step 2

281638364

in the table, so add it to B.

codewords—S(0x61)='a' twice—so add 'aa' to the table eword

B = aa

LZW Decompression, Step 5

81638364

in the table, so add it to B.

codewords—S(0x81)='ab' and S(0x63)='c'—so add 'abc' as a new codeword.

B = aababc

LZW Decompression, Step 4

81638364

' in the table, so add it to B.

codewords—S(0x62)='b' and S(0x81)='ab'—so add 'ba' to a new codeword.

B = aabab

LZW Decompression, Step 6

81638364

c' in the table, so add it to B.

codewords—S(0x63)='c' and S(0x83)='abc'—so add 'ca' as a new codeword.

B = aababcabc

LZW Decompression, Step 6

81638364

? in the table, so add it to B.

codewords—S(???)=??? and S(???)=???—so add ??? to a new codeword.

B = aababc???

LZW Decompression, Step 7

816383 64

in the table, so add it to B.

codewords—S(0x83)=’abc’ and S(0x64)=’d’—so add ‘abcd’ as a new codeword.

B = aababcabcd

---

constructing the Coding Table (II)

s slide, I said “. . . the table will (almost) always already napping we need. . . ”

ly, there are cases where it doesn’t.

string B=’`cdcdcdc`’ as an example.

code it, we end up with

C(B) = 0x63648082

causes trouble. . .

Tricky Decompression, Step 1

082

in the table, so add it to B.

previous codeword yet, so don’t add anything to the

B = c

Tricky Decompression, Step 2

082

in the table, so add it to B.

codewords—S(0x63)=’c’ and S(0x64)=’d’—so add ‘cd’ to a new codeword

B = cd

Tricky Decompression, Step 3

82

d’ in the table, so add it to B.

codewords—S(0x64)=’d’ and S(0x80)=’cd’—so add ‘dc’ to a new codeword

B = cdcd

Tricky Decompression, Step 4

82

) is not yet in the table. What now?

B = cdcd???

hat we could look *ahead* while coding, but can only look decoding.

re out what 0x82 is going to be by looking back.

Decompression, Step 4 (Second Try)

82

o be figured out).

ecoded S(0x80)="cd" and now have S(0x82)=Z, so will
o the table as S(0x82).

with S(0x80) and therefore Z_0 must be 'c'!

) = S(0x80)+Z_0 = 'cdc'.

B = cdcdcdc

LZW Algorithm

d for its inventors: Lempel, Ziv, and Welch.

used at one time, but because of patent issues became
ular (especially among open-source folks).

expired in 2003 and 2004.

the .gif files, some PDF files, the BSD Unix `compress`
sewhere.

merous other (and better) algorithms (such as those in
p2).

ation here is considerably simplified.

fixed-length (8-bit) codewords, but the full algorithm
variable-length codewords using (!) Huffman coding
ing the compression).

lgorithm clears the table from time to time to get rid
sed codewords.

Some Thoughts

a compressed text doesn't result in much compression.

be impossible to keep compressing a text?

at takes no input and produces an output can be thought
dings of that output.

he following question: Given a bitstream, what is the
shortest program that can produce it?

ific bitstream, there is a specific answer!

p concept, known as Kolmogorov Complexity.

Some Thoughts

a compressed text doesn't result in much compression.

be impossible to keep compressing a text?

u'd be able to compress any number of different messages

at takes no input and produces an output can be thought
dings of that output.

he following question: Given a bitstream, what is the
shortest program that can produce it?

ific bitstream, there is a specific answer!

p concept, known as Kolmogorov Complexity.

More Thoughts

weird that one can compress much at all.

00-character ASCII text (8000 bits), and suppose we
mpress it by 50%.

00 distinct messages in 8000 bits, but only 2^{4000} possible
4000 bits.

atter what one's scheme, one can encode only 2^{-4000} of
8000-bit messsages by 50%! Yet we do it all the time.

texts have a great deal of redundancy (aka low *information*

igh entropy—such as random bits, previously compressed
rypted texts—are nearly incompressible.

Git

uses a different scheme from LZW for compression: a
of LZ77 and Huffman coding.

of like delta compression, but within the same text.

xt such as

`ippi, two Mississippi`

ng like

`ippi, two <11,7>`

`1,7>` is intended to mean "the next 11 characters come
t that ends 7 characters before this point."

symbols to the alphabet to represent these (length,
lusions.

Huffman encode the result.

Wrapping Up

pression saves space (and bandwidth) by exploiting redundancy

d Shannon-Fano coding represent individual symbols of h shorter codewords.

nilar codes represents multiple symbols with shorter

heir codewords to the text being compressed.

ession both uses redundancy and exploits the fact that umers of compressed data (like humans) can't really use nation that could be encoded.

Lossy Compression

plications, like compressing video and audio streams, it ecessary to be able to reproduce the exact stream.

refore get more compression by throwing away some

e is a limit to what human senses respond to.

we don't hear high frequencies, or see tiny color variations.

formats like JPEG, MP3, or MP4 use *lossy compression* uct output that is (hopefully) imperceptibly different ginal at large savings in size and bandwidth.

more of this in EE120 and other courses.