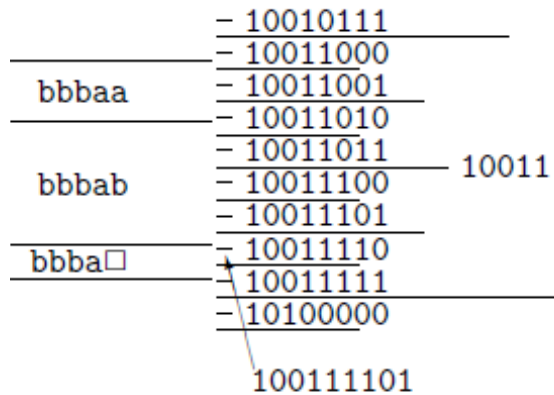
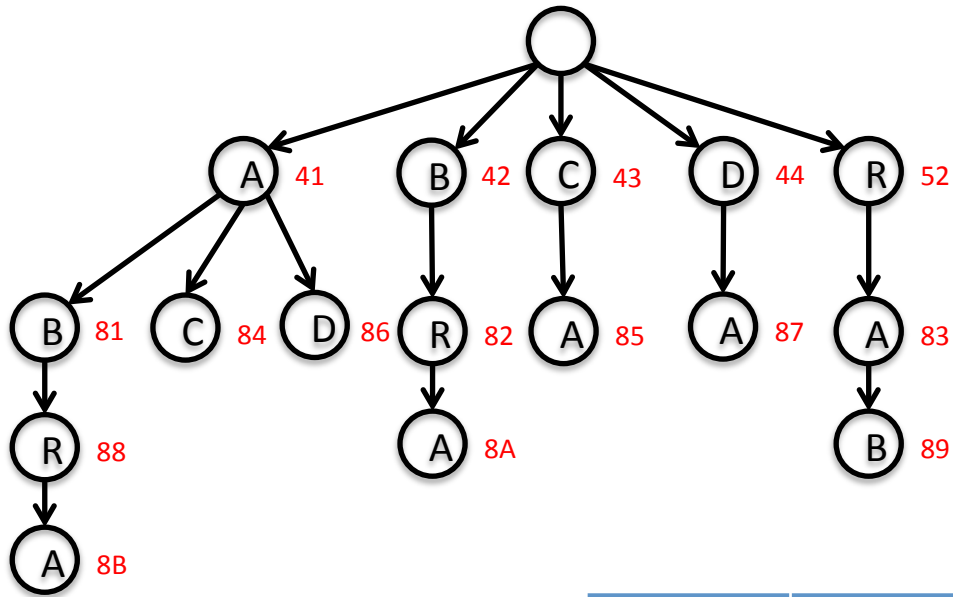


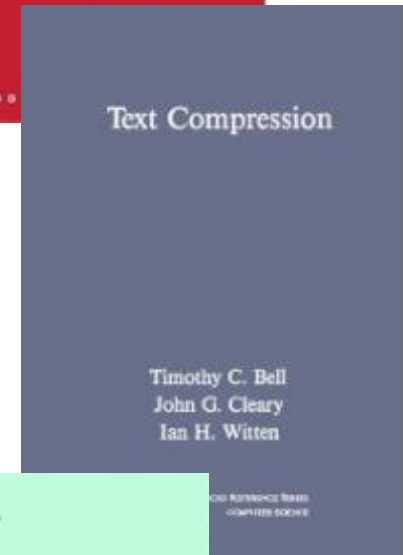
Lossless compression II



Symbol	Probability	Range
a	0.2	[0.0, 0.2)
e	0.3	[0.2, 0.5)
i	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
!	0.1	[0.9, 1.0)

Overview

- Flavors of compression
 - Static vs. Dynamic vs. Adaptive
- Lempel-Ziv-Welch (LZW)
 - Fixed-length codeword
 - Variable-length pattern
- Statistical coding
 - Arithmetic coding
 - Prediction by Partial Match (PPM)



Compression models

- **Static models**
 - **Predefined map for all text**, e.g. ASCII, Morse Code
 - Not optimal: different texts = different statistics
- **Dynamic model**
 - Generate **model based on text**
 - Requires **initial pass** before compression can start
 - Must **transmit the model** (e.g. Huffman coding)
- **Adaptive model**
 - More accurate modeling = better compression
 - Decoding must **start from beginning** (e.g. LZW)

LZW

- Lempel-Ziv-Welch compression (LZW)
 - Basis for many popular compression formats
 - e.g. PNG, 7zip, gzip, jar, PDF, compress, pkzip, GIF, TIFF
- Algorithm basics
 - Maintain a table of **fixed-length codewords** for **variable-length patterns** in the input
 - **Built progressively** by compressor and expander
 - Table does not need to be transmitted!
 - Table is of fixed size
 - Entries for all single characters in alphabet
 - Entries for longer substrings encountered

LZW compression example

- Details:
 - 7-bit ASCII characters, 8-bit codeword
 - For real use, 8-bit → ~12-bit codeword
 - Alphabet: 128 characters + 128 longer strings
 - Codeword: 2-digit hex value
 - 00-79 = single characters, e.g. 41 = A, 52 = R
 - 80 = end of file
 - 81-FF = longer strings, e.g. 81 = AB, 88 = ABR
 - Employs lookahead character to add codewords

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
matches	A	B	R	A	C	A	D										
codeword	41	42	52	41	43	41	44										



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87

Mappings found during compression

- **Compressing**

- Find longest string s in table that is prefix of unscanned input
- Write codeword of string s
- Scan one character c ahead
- Associate next free codeword with $s + c$

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
matches	A	B	R	A	C	A	D	AB									
codeword	41	42	52	41	43	41	44	81									



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88

Mappings found during compression

- **Compressing**
 - Find longest string s in table that is prefix of unscanned input
 - Write codeword of string s
 - Scan one character c ahead
 - Associate next free codeword with $s + c$

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
matches	A	B	R	A	C	A	D	AB		RA							
codeword	41	42	52	41	43	41	44	81		83							



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89

Mappings found during compression

- **Compressing**

- Find longest string s in table that is prefix of unscanned input
- Write codeword of string s
- Scan one character c ahead
- Associate next free codeword with $s + c$

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
matches	A	B	R	A	C	A	D	AB		RA		BR					
codeword	41	42	52	41	43	41	44	81		83		82					



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A

Mappings found during compression

- **Compressing**
 - Find longest string s in table that is prefix of unscanned input
 - Write codeword of string s
 - Scan one character c ahead
 - Associate next free codeword with $s + c$

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
matches	A	B	R	A	C	A	D	AB		RA		BR		ABR			
codeword	41	42	52	41	43	41	44	81		83		82		88			



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

Mappings found during compression

- **Compressing**

- Find longest string s in table that is prefix of unscanned input
- Write codeword of string s
- Scan one character c ahead
- Associate next free codeword with $s + c$

input	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
matches	A	B	R	A	C	A	D	AB		RA		BR		ABR			A
codeword	41	42	52	41	43	41	44	81		83		82		88			41

string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

Mappings found during compression

- **Compressing**

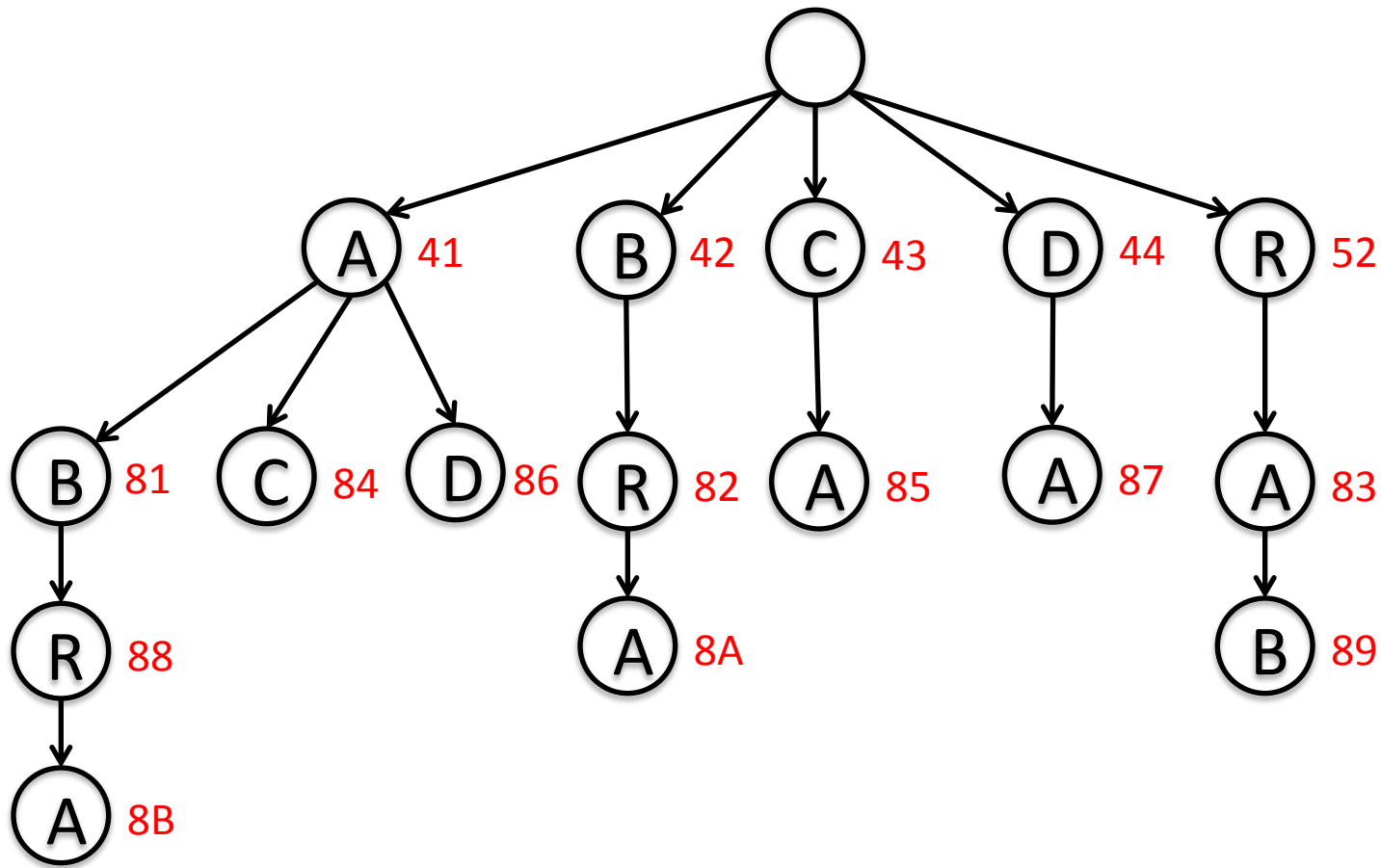
- Find longest string *s* in table that is prefix of unscanned input
- Write codeword of string *s*
- Scan one character *c* ahead
- Associate next free codeword with *s + c*

Input:
 17 ASCII characters * 7 bits = 119 bits
 Output:
 12 codewords * 8 bits = 96 bits
 Compression ratio:
 82%

Compression data structure

- LZW compression uses two table operations:
 - Find **longest-prefix match** from current position
 - **Add entry** with character added to longest match
- Trie data structure, "**retrieval**"
 - Linked tree structure
 - Path in tree defines string

- **Compressing**
 - Find longest string s in table that is prefix of unscanned input
 - Write codeword of string s
 - Scan one character c ahead
 - Associate next free codeword with $s + c$



string	codeword
...	...
A	41
B	42
C	43
...	...

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85

string	codeword
AD	86
DA	87
ABR	88
RAB	89
BRA	8A

```

private static final int R = 256;           // number of input chars
private static final int L = 4096;         // number of codewords = 2^W
private static final int W = 12;          // codeword width

public static void compress()
{
    String input = BinaryStdIn.readString(); // read input as string
    TST<Integer> st = new TST<Integer>();     // trie data structure

    for (int i = 0; i < R; i++)              // codewords for single chars
        st.put("" + (char) i, i);
    int code = R+1;                          // R is codeword for EOF

    while (input.length() > 0)
    {
        String s = st.longestPrefixOf(input);
        BinaryStdOut.write(st.get(s), W);    // write W-bit codeword for s
        int t = s.length();
        if (t < input.length() && code < L)
            st.put(input.substring(0, t + 1), code++); // add new codeword
        input = input.substring(t);
    }
    BinaryStdOut.write(R, W);                // write last codeword
    BinaryStdOut.close();
}

```


input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R										



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83

Mappings found during compression

• Expansion

- Write the current string **val**
- Read codeword **x** from input
- Set **s** to string for codeword **x**
- Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
- Set **val=s**

```

valold    = A
x          = 43
s          = C
c          = C
nextcode = 84
valnew   = C

```


input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C								



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85

Mappings found during compression

- Expansion
 - Write the current string **val**
 - Read codeword **x** from input
 - Set **s** to string for codeword **x**
 - Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
 - Set **val=s**

```

valold    = A
x          = 44
s          = D
c          = D
nextcode = 86
valnew   = D

```

input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A							



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86

Mappings found during compression

- Expansion
 - Write the current string **val**
 - Read codeword **x** from input
 - Set **s** to string for codeword **x**
 - Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
 - Set **val=s**

```

valold    = D
x          = 81
s          = AB
c          = A
nextcode  = 87
valnew    = AB

```

input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D						



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87

Mappings found during compression

- Expansion
 - Write the current string **val**
 - Read codeword **x** from input
 - Set **s** to string for codeword **x**
 - Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
 - Set **val=s**

```

valold    = AB
x          = 83
s          = RA
c          = R
nextcode  = 88
valnew    = RA

```

input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D	AB					



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88

Mappings found during compression

- Expansion
 - Write the current string **val**
 - Read codeword **x** from input
 - Set **s** to string for codeword **x**
 - Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
 - Set **val=s**

```

valold = RA
x       = 82
s       = BR
c       = B
nextcode = 89
valnew = BR

```

input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D	AB	RA				



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89

Mappings found during compression

• Expansion

- Write the current string **val**
- Read codeword **x** from input
- Set **s** to string for codeword **x**
- Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
- Set **val=s**

```

valold    = BR
x          = 88
s          = ABR
c          = A
nextcode  = 8A
valnew    = RA

```

input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D	AB	RA	BR			



string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A

Mappings found during compression

- Expansion
 - Write the current string **val**
 - Read codeword **x** from input
 - Set **s** to string for codeword **x**
 - Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
 - Set **val=s**

val _{old}	=	ABR
x	=	41
s	=	A
c	=	A
next _{code}	=	8B
val _{new}	=	RA



input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D	AB	RA	BR	ABR		

string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

Mappings found during compression

- Expansion
 - Write the current string **val**
 - Read codeword **x** from input
 - Set **s** to string for codeword **x**
 - Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
 - Set **val=s**

```

valold    = A
x          = 80
s          =
c          =
nextcode =
valnew   =

```



input	41	42	52	41	43	41	44	81	83	82	88	41	80
output	A	B	R	A	C	A	D	AB	RA	BR	ABR	A	

string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

Initial set of character codewords

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

Mappings found during compression

• Expansion

- Write the current string **val**
- Read codeword **x** from input
- Set **s** to string for codeword **x**
- Set next unassigned codeword to **val+c** where **c** is 1st char in **s**
- Set **val=s**

```

valold    =
x          =
s          =
c          =
nextcode =
valnew   =

```


Expansion data structure

- For a W -bit codeword, look up string value
 - An array of size 2^W

string	codeword
...	...
A	41
B	42
C	43
D	44
...	...
R	52
...	...

string	codeword
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

- Expansion
 - Write the current string val
 - Read codeword x from input
 - Set s to string for codeword x
 - Set next unassigned codeword to $val+c$ where c is 1st char in s
 - Set $val=s$

A tricky case: compression

ABABABA

input	A	B	A	B	A	B	A
matches							
codeword							

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword

*Mappings found during
compression*

A tricky case: compression

ABABABA

input	A	B	A	B	A	B	A
matches	A						
codeword	41						

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81

*Mappings found during
compression*

A tricky case: compression

ABABABA

input	A	B	A	B	A	B	A
matches	A	B					
codeword	41	42					

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81
BA	82

*Mappings found during
compression*

A tricky case: compression

ABABABA

input	A	B	A	B	A	B	A
matches	A	B	AB				
codeword	41	42	81				

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81
BA	82
ABA	83

*Mappings found during
compression*

A tricky case: compression

ABABABA

input	A	B	A	B	A	B	A
matches	A	B	AB		ABA		
codeword	41	42	81		83		

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81
BA	82
ABA	83

*Mappings found during
compression*

A tricky case: expansion

ABABABA

input	41	42	81	83	80
output					

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword

*Mappings found during
compression*

A tricky case: expansion

ABABABA

input	41	42	81	83	80
output	A				

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81

*Mappings found during
compression*

A tricky case: expansion

ABABABA

input	41	42	81	83	80
output	A	B			

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81
BA	82

*Mappings found during
compression*

A tricky case: expansion

ABABABA

input	41	42	81	83	80
output	A	B	AB		

We need to know the first character of 83 in order to enter 83 into the table!

string	codeword
...	...
A	41
B	42
C	43
...	...

Initial set of codewords

string	codeword
AB	81
BA	82

Mappings found during compression

A tricky case: expansion

ABABABA

input	41	42	81	83	80
output	A	B	AB	ABA	

string	codeword
...	...
A	41
B	42
C	43
...	...

*Initial set of
codewords*

string	codeword
AB	81
BA	82
ABA	83

*Mappings found during
compression*

LZW decisions

- How big of a symbol table?
 - How long is message?
 - Whole message similar model?
 - Many variations...
- What to do when symbol table fills up?
 - Throw away and start over (e.g. GIF)
 - Throw away when not effective (e.g. Unix compress)
 - Many variations...
- Put longer substrings in symbol table?
 - Many variations...

LZW variants

- Lempel-Ziv variants

- LZ77, published Lempel & Ziv, 1977, not patented
 - PNG
- LZ78, published Lempel & Ziv in 1978, patented
- LZW, Welch extension to LZ78, patented (expired 2003)
 - GIF, TIFF, Pkzip
- Deflate = LZ77 variant + Huffman coding
 - 7zip, gzip, jar, PDF



Some experiments

- **Compressing Moby Dick**

- Using Java programs for LZW and Huffman
 - Different dictionary sizes for LZW or resetting when full
- Compared to zip and gzip

```
1,191,463  mobydick.txt
485,561    mobydick.gz
485,790    mobydick.zip

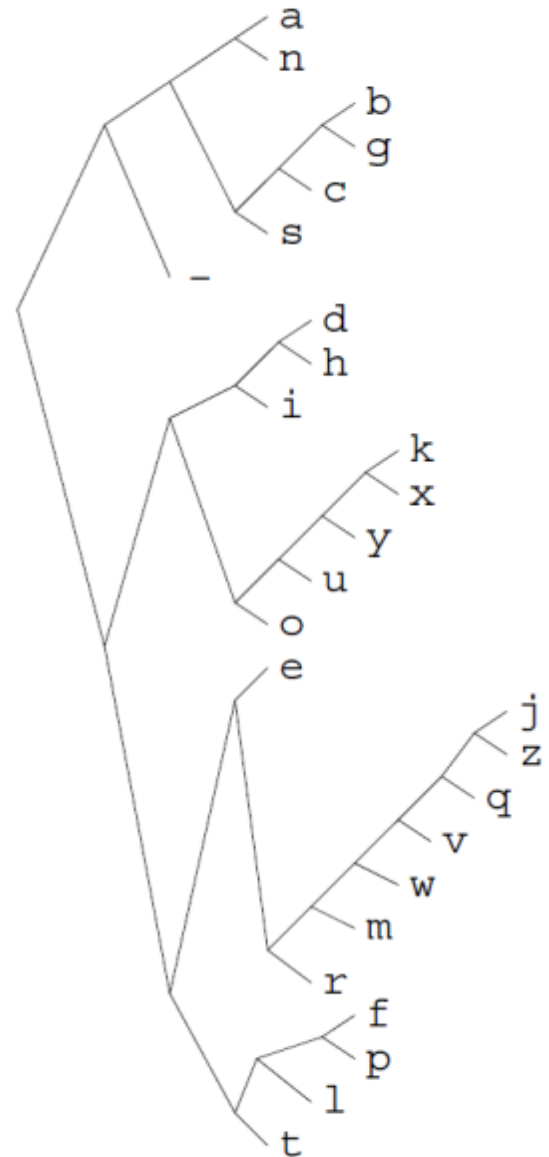
667,651    mobydick.huff
597,060    mobydick.lzw
814,578    mobydick.huff.lzw
592,830    mobydick.lzw.huff
682,400    mobydick.lzw.reset

541,261    mobydick.lzw14
521,700    mobydick.lzw15
503,050    mobydick.lzw16
501,193    mobydick.lzw16.huff
521,700    mobydick.lzw17
514,393    mobydick.lzw18
571,548    mobydick.lzw20
```

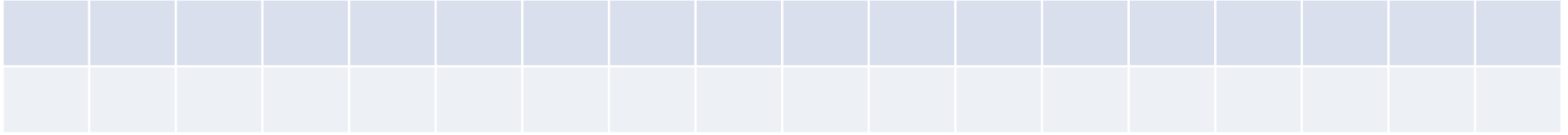
Enter statistical coding...

- **Natural language quite predictable**
 - ~ 1 bit of entropy per symbol
 - Huffman coding still requires 1-bit min per symbol
 - We're forced to use an integral number of bits
 - Dictionary-based (LZW and friends)
 - Memorizes sequences, but no notion of frequency
- **Statistical coding**
 - Use long, **specific context for prediction**
 - $P(A \mid \text{The_United_State_of_}) = ?$
 - **Blend knowledge** using contexts of different lengths
 - Adaptive: model **updates and change** as text seen
 - Often after every letter!

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.0575	4.1	4	0000
b	0.0128	6.3	6	001000
c	0.0263	5.2	5	00101
d	0.0285	5.1	5	10000
e	0.0913	3.5	4	1100
f	0.0173	5.9	6	111000
g	0.0133	6.2	6	001001
h	0.0313	5.0	5	10001
i	0.0599	4.1	4	1001
j	0.0006	10.7	10	1101000000
k	0.0084	6.9	7	1010000
l	0.0335	4.9	5	11101
m	0.0235	5.4	6	110101
n	0.0596	4.1	4	0001
o	0.0689	3.9	4	1011
p	0.0192	5.7	6	111001
q	0.0008	10.3	9	110100001
r	0.0508	4.3	5	11011
s	0.0567	4.1	4	0011
t	0.0706	3.8	4	1111
u	0.0334	4.9	5	10101
v	0.0069	7.2	8	11010001
w	0.0119	6.4	7	1101001
x	0.0073	7.1	7	1010001
y	0.0164	5.9	6	101001
z	0.0007	10.4	10	1101000001
-	0.1928	2.4	2	01



Guess the phrase



A simple unigram model of English

'	0.0071063
-	0.0000001
.	0.0000384
</s>	0.0368291
<sp>	0.1653810
a	0.0595248
b	0.0112864
c	0.0174441
d	0.0282733
e	0.0890307
f	0.0127512
g	0.0213974
h	0.0403836
i	0.0586443
j	0.0018080
k	0.0117826

l	0.0343399
m	0.0247835
n	0.0490316
o	0.0762119
p	0.0134453
q	0.0003078
r	0.0408972
s	0.0433802
t	0.0680194
u	0.0273347
v	0.0083669
w	0.0210079
x	0.0010829
y	0.0295698
z	0.0005395

From text to a real number

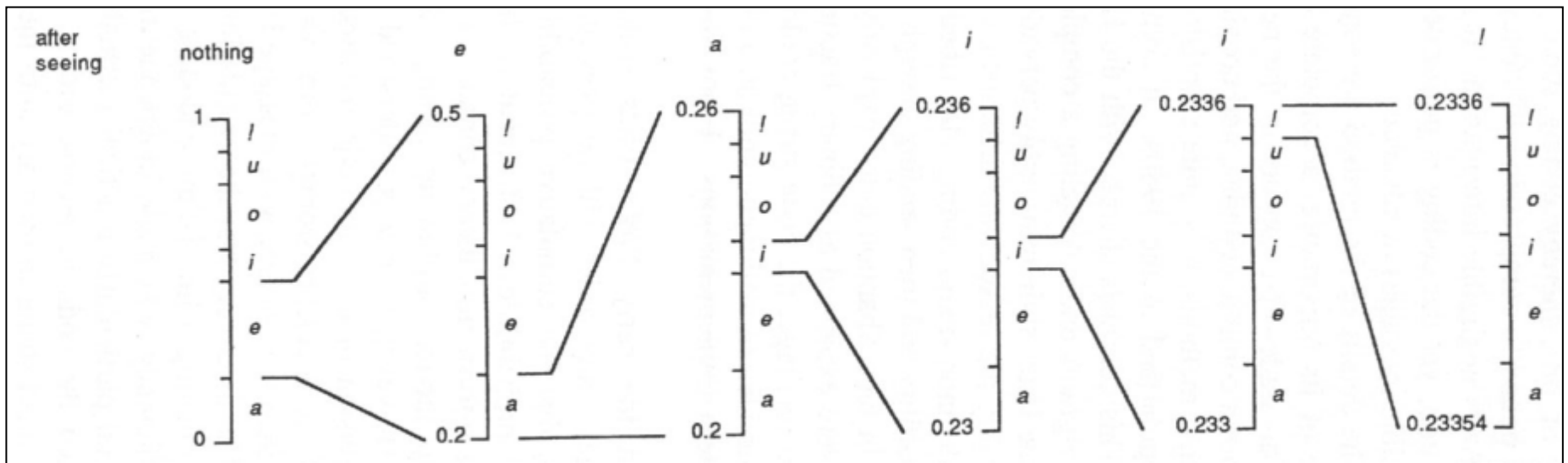
- Arithmetic coding

- Message represented by real interval in $[0, 1)$
- More precise interval = specifies more bits
 - e.g. $[0.28272722, 0.28272724)$ = "it was the best of times"
 - Or any number in that interval, 0.28272723

- Example

- Alphabet = {aeiou!}
- Transmission = eaii!

Symbol	Probability	Range
a	0.2	$[0.0, 0.2)$
e	0.3	$[0.2, 0.5)$
i	0.1	$[0.5, 0.6)$
o	0.2	$[0.6, 0.8)$
u	0.1	$[0.8, 0.9)$
!	0.1	$[0.9, 1.0)$



Bell, Cleary, Witten. Text Compression.

Transmission = **eaii!**

After seeing	Range
	[0.0, 1.0]
e	[0.2, 0.5)
a	[0.2, 0.26)
i	[0.23, 0.236)
i	[0.233, 0.2336)
!	[0.23354, 0.2336)

Symbol	Probability	Range
a	0.2	[0.0, 0.2)
e	0.3	[0.2, 0.5)
i	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
!	0.1	[0.9, 1.0)

Note: Encoder/decoder need to agree on symbol to terminate message, here we use !

Context modeling

- Prediction by Partial Match (PPM)
 - $P(\text{next symbol})$ depends on previous symbol(s)
 - Blending for dealing with 0-frequency problem
 - Easy to build as adaptive
 - Learn from the text as you go
 - No trie to send as in Huffman coding
 - 1984, but still competitive for text compression
 - Various variants, PPM-A/B/C/D/Z/*
 - Implemented in 7-zip, open source packages, PPM for XML, PPM for executables, ...

PPM

- An alphabet A with q symbols
- Order = How many previous symbols to use

...

2 = condition on 2 previous symbols, $P(x_n \mid x_{n-1}, x_{n-2})$

1 = condition on 1 previous symbol, $P(x_n \mid x_{n-1})$

0 = condition on current symbol, $P(x_n)$

-1 = uniform over the alphabet, $P(x_n) = 1/q$

- For a given order:
 - Probability of next symbol based on counting occurrences given prior context

$$p_o(\phi) = \frac{c_o(\phi)}{C_o}$$

TABLE 6-1 CALCULATION OF BLENDED PROBABILITIES (SIMPLE SCHEME AND ESCAPE METHOD A) FOR THE MESSAGE "cacbcaabca"

o	Context	Counts C_o	Predictions — $p_o(\phi)$ $c_o(\phi)$										Weight w_o	Escape e_o
			a		b		c		d		e			
4	abca:	0	—	0	—	0	—	0	—	0	—	0	0	—
3	bca:	1	1	1	0	0	0	0	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$
2	ca:	2	$\frac{1}{2}$	1	0	0	$\frac{1}{2}$	1	0	0	0	0	$\frac{1}{3}$	$\frac{1}{3}$
1	a:	3	$\frac{1}{3}$	1	$\frac{1}{3}$	1	$\frac{1}{3}$	1	0	0	0	0	$\frac{1}{8}$	$\frac{1}{4}$
0	:	10	$\frac{4}{10}$	4	$\frac{2}{10}$	2	$\frac{4}{10}$	4	0	0	0	0	$\frac{5}{132}$	$\frac{1}{11}$
-1	—	—	$\frac{1}{5}$	—	$\frac{1}{5}$	—	$\frac{1}{5}$	—	$\frac{1}{5}$	—	$\frac{1}{5}$	—	$\frac{1}{264}$	0
Blended probabilities			$\frac{956}{1320}$		$\frac{66}{1320}$		$\frac{296}{1320}$		$\frac{1}{1320}$		$\frac{1}{1320}$			

Bell, Cleary, Witten. Text Compression.

$$p(\phi) = \sum_{o=-1}^m w_o p_o(\phi).$$

Escape probabilities

- Need weights to blend probabilities
 - Compute based on "escape" probability
 - Allocate some mass in each model order for when a lower-order model should make prediction instead
 - Method A:
 - Add one to the count of characters seen in a context
 - $e_o = 1 / (C_o + 1)$
 - Method D:
 - u = number of unique characters seen
 - $e_o = (u / 2) / C_o$

Lossless compression benchmarks

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-front	3.24
1987	LZB	3.18
1987	gzip	2.71
1988	PPMC	2.48
1994	PPM	2.34
1995	Burrows-Wheeler	2.29
1997	BOA	1.99
1999	RK	1.89

Data compression using Calgary corpus

My benchmark

```
1,191,463  mobydick.txt
485,561   mobydick.gz
485,790   mobydick.zip
371,394   mobydick.bz2

667,651   mobydick.huff
597,060   mobydick.lzw
814,578   mobydick.huff.lzw
592,830   mobydick.lzw.huff
682,400   mobydick.lzw.reset

541,261   mobydick.lzw14
521,700   mobydick.lzw15
503,050   mobydick.lzw16
501,193   mobydick.lzw16.huff
521,700   mobydick.lzw17
514,393   mobydick.lzw18
571,548   mobydick.lzw20

325,378   mobydick.ppm
```

Summary

- Dictionary-based compression
 - LZW and variants
 - Memorizes sequences in the data
- Statistical coding
 - Language model produces probabilities
 - Probability sequence defines point on $[0, 1)$
 - Use arithmetic coding to convert to bits
 - Prediction by Partial Match (PPM)