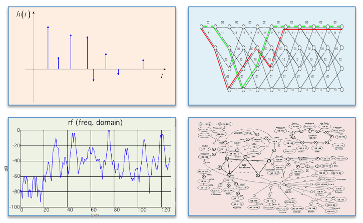


INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2012 Lecture #2

- Huffman codes wrap-up: Properties & limitations
- Adaptive variable-length codes: LZW

$$\begin{tikzcd}
\text{bits}_{IN} \arrow[r] & \boxed{\text{Source Coding}} \arrow[r] & \left\{\text{Transmit/Receive (or Store/Retrieve)}\right\} \arrow[r] & \boxed{\text{Source Decoding}} \arrow[r] & \text{bits}_{OUT}
\end{tikzcd}$$

Example from Last Lecture

choice$_i$	*p_i*	*log$_2$(1/p_i)*	*p_i * log$_2$(1/p_i)*	*Huffman encoding*	*Expected length*
"A"	1/3	1.58 bits	0.528 bits	10	0.667 bits
"B"	1/2	1 bit	0.5 bits	0	0.5 bits
"C"	1/12	3.58 bits	0.299 bits	110	0.25 bits
"D"	1/12	3.58 bits	0.299 bits	111	0.25 bits
			1.626 bits		1.667 bits

Entropy is 1.626 bits/symbol, expected length of Huffman encoding is 1.667 bits/symbol.

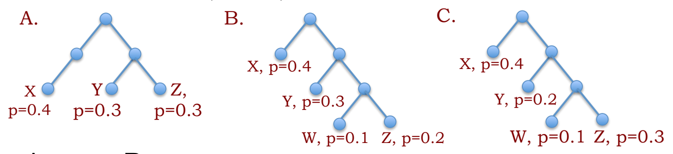
How do we do better?

16 Pairs: 1.646 bits/sym
64 Triples: 1.637 bits/sym
256 Quads: 1.633 bits/sym

Halftime Quiz

- Write your **name** and **recitation time** on a sheet of paper.
- Write the question number and your answer on it.
- Turn the paper in at the end of lecture to me.

1. Which of these (A, B, C) is a valid Huffman code tree?



Answer: B

2. What is the *expected length* of the code in tree C above?

Answer: 0.4*1 + 0.2*2 + 0.1*3 + 0.3*3 = 2.0 bits

How to Send Shakespeare's Sonnets?!

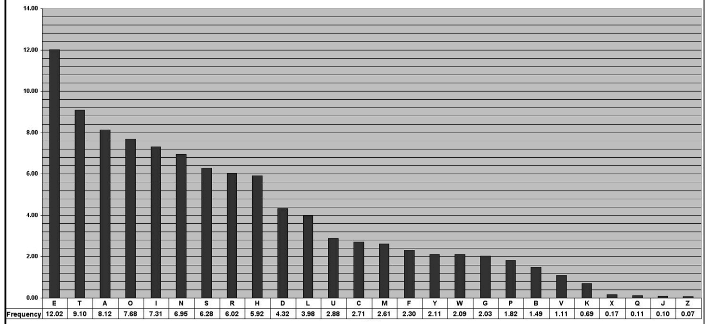
I.

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thy self thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And, tender churl, mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

XVIII.

Shall I compare thee to a summer's day?
Thou art more lovely and more temperate:
Rough winds do shake the darling buds of May,
And summer's lease hath all too short a date:
Sometime too hot the eye of heaven shines,
And often is his gold complexion dimmed,
And every fair from fair sometime declines,
By chance, or nature's changing course untrimmed:
But thy eternal summer shall not fade,
Nor lose possession of that fair thou ow'st,
Nor shall death brag thou wander'st in his shade,
When in eternal lines to time thou grow'st,
So long as men can breathe, or eyes can see,
So long lives this, and this gives life to thee.

What is the Entropy of English?



	E	T	A	O	I	N	S	R	H	D	L	U	C	M	F	Y	W	G	P	B	V	K	X	Q	J	Z
Frequency	12.02	9.10	8.12	7.68	7.31	6.95	6.28	6.02	5.92	4.32	3.98	2.88	2.71	2.61	2.30	2.11	2.09	2.03	1.82	1.49	1.11	0.69	0.17	0.11	0.10	0.07

Assuming IID letter distribution,
entropy works out to 4.177 bits per letter

http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html

In fact, English text has lots of context

- What's the next letter in the snippet

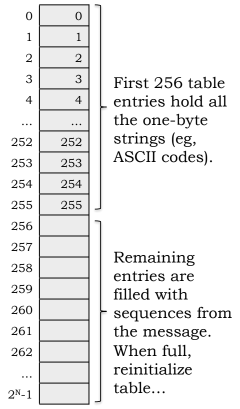
Nothing can be said to be certain, except death and ta<u>?</u>

- But X has a very low occurrence probability (0.0017) in English words
  - Letters are not independently distributed!
- Shannon and others have found that the entropy of English text is a lot lower than 4.177
  - Shannon estimated 0.6-1.3 bits/letter using human expts
  - More recent estimates: 1-1.5 bits/letter

Can we do adaptive variable-length encoding?

LZW: An Adaptive Variable-length Code

- Algorithm first developed by Ziv and Lempel (LZ88, LZ78), later improved by Welch.
- As message is processed, encoder builds a "string table" that maps symbol sequences to an N-bit fixed-length code. Table size = 2^N
- Transmit table indices, usually shorter than the corresponding string → compression!
- Note: String table can be reconstructed by the decoder using information in the encoded stream – the table, while central to the encoding and decoding process, *is never transmitted*!


0 0
1 1
2 2
3 3
4 4
... ...
252 252
253 253
254 254
255 255
256
257
258
259
260
261
262
...
2^N-1
First 256 table entries hold all the one-byte strings (eg, ASCII codes).
Remaining entries are filled with sequences from the message. When full, reinitialize table...

LZW Encoding

```
STRING = get input symbol
WHILE there are still input symbols DO
   SYMBOL = get input symbol
   IF STRING + SYMBOL is in the STRINGTABLE THEN
       STRING = STRING + SYMBOL
   ELSE
       output the code for STRING
       add STRING + SYMBOL to STRINGTABLE
       STRING = SYMBOL
   END
END

output the code for STRING
```

1. Accumulate message bytes in S as long as S appears in table.
2. When S+b isn't in table: send code for S, add S+b to table.
3. Reinitialize S with b, back to step 1.

Example: Encode "abbbabbbab…"

256	ab
257	bb
258	bba
259	abb
260	bbab
261	
262	

1. Read a; string = a
2. Read b; ab not in table
   output 97, add ab to table, string = b
3. Read b; bb not in table
   output 98, add bb to table, string = b
4. Read b; bb in table, string = bb
5. Read a; bba not in table
   output 257, add bba to table, string = a
6. Read b, ab in table, string = ab
7. Read b, abb not in table
   output 256, add abb to table, string = b
8. Read b, bb in table, string = bb
9. Read a, bba in table, string = bba
10. Read b, bbab not in table
    output 258, add bbab to table, string = b

Encoder Notes

- The encoder algorithm is greedy – it's designed to find the longest possible match in the string table before it makes a transmission.
- The string table is filled with sequences actually found in the message stream. No encodings are wasted on sequences not actually found in the input data.
- Note that in this example the amount of compression increases as the encoding progresses, i.e., more input bytes are consumed between transmissions.
- Eventually the table will fill and then be reinitialized, recycling the N-bit codes for new sequences. So the encoder will eventually adapt to changes in the probabilities of the symbols or symbol sequences.
- Remarkably, LZW actually achieves entropy for long messages!

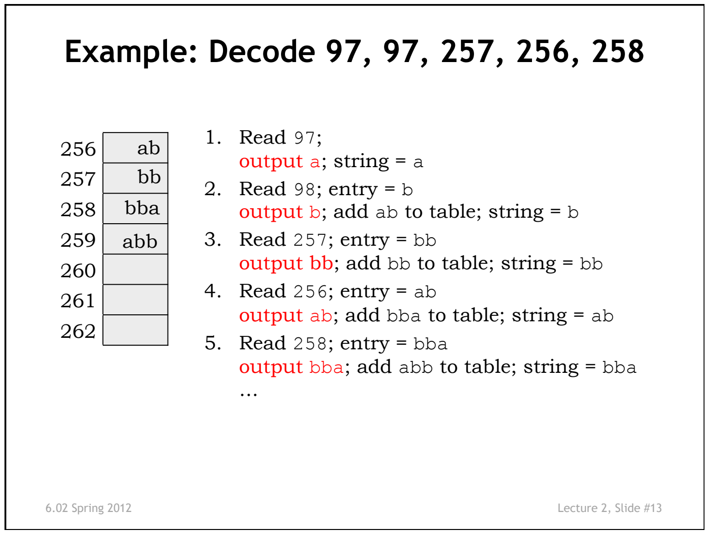
LZW Decoding, 1st Attempt (somewhat wrong!)

```
Read CODE
STRING = TABLE[CODE] // translation table @ decoder

WHILE there are still codes to receive DO
    Read CODE
    ENTRY = TABLE[CODE]

    output ENTRY
    add STRING+ENTRY[0] to the translation table
    STRING = ENTRY
END
```

Easy: use table lookup to convert code to message string
Less easy: build table that's identical to that in encoder



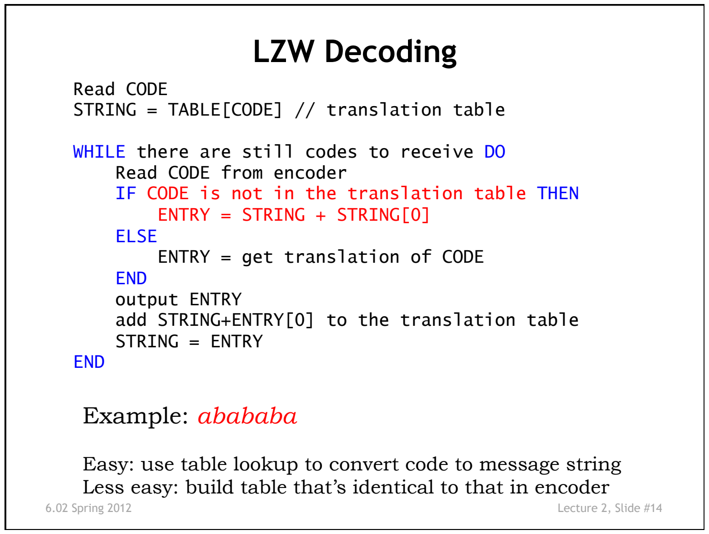
Example: Decode 97, 97, 257, 256, 258

256	ab
257	bb
258	bba
259	abb
260	
261	
262	

1. Read `97`;
   output `a`; string = `a`
2. Read `98`; entry = `b`
   output `b`; add `ab` to table; string = `b`
3. Read `257`; entry = `bb`
   output bb; add `bb` to table; string = `bb`
4. Read `256`; entry = `ab`
   output `ab`; add `bba` to table; string = `ab`
5. Read `258`; entry = `bba`
   output `bba`; add `abb` to table; string = `bba`

   …

6.02 Spring 2012 Lecture 2, Slide #13



LZW Decoding

```
Read CODE
STRING = TABLE[CODE] // translation table

WHILE there are still codes to receive DO
    Read CODE from encoder
    IF CODE is not in the translation table THEN
        ENTRY = STRING + STRING[0]
    ELSE
        ENTRY = get translation of CODE
    END
    output ENTRY
    add STRING+ENTRY[0] to the translation table
    STRING = ENTRY
END
```

Example: *abababa*

Easy: use table lookup to convert code to message string
Less easy: build table that's identical to that in encoder

6.02 Spring 2012 Lecture 2, Slide #14