Lecture 7:
Run-Length, Golomb, and Tunstall Codes

Thinh Nguyen
Oregon State University

Outline

- Run-Length Coding
- Golomb Coding
- Tunstall Coding

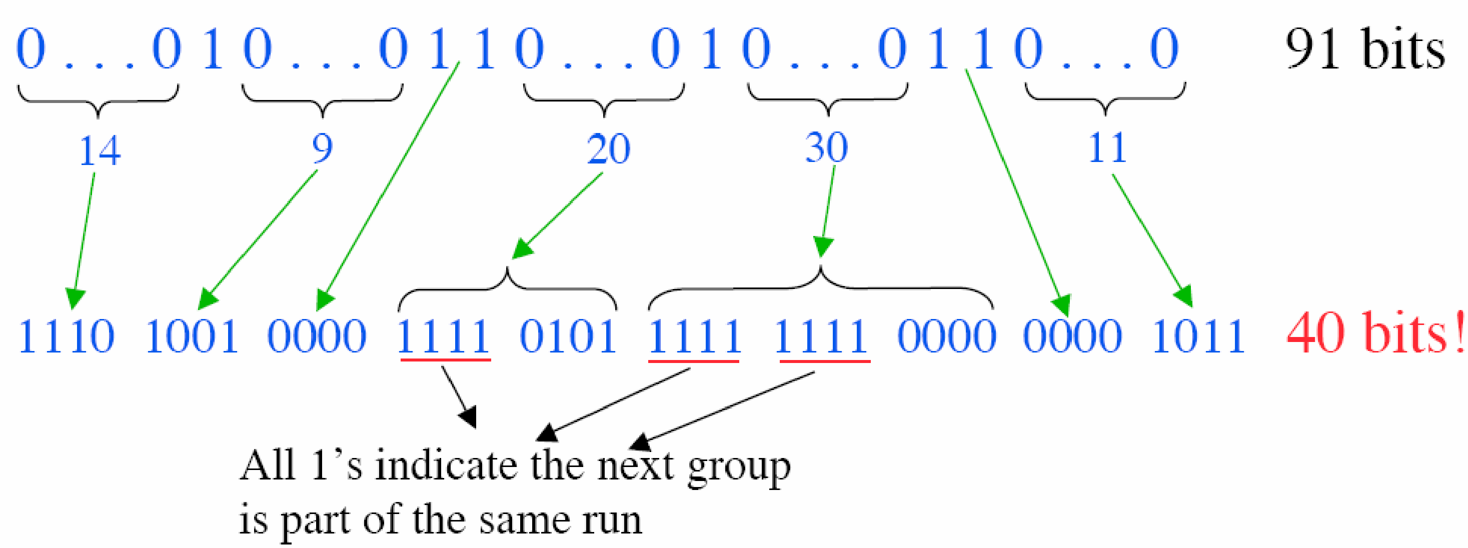
Lossless coding: Run-Length encoding (RLE)

- Redundancy is removed by not transmitting consecutive identical symbols (pixels or character values that are equal).
- The repeated value can be coded once, along with the number of times it repeats.
- Useful for coding black and white images e.g. fax.

Binary RLE

- Code the run length of 0's using k bits. Transmit the code.
- Do not transmit runs of 1's.
- Two consecutive 1's are implicitly separately by a zero-length run of zero.

Example: suppose we use k = 4 bits to encode the run length (maximum run length of 15) for following bit patterns



RLE Performance

- Worst case behavior: transition occurs on each bit. Since we use k bits to represent the transition, we waste k-1 bits.
- Best case behavior: no transition and use k bits to represent run length then the compression ratio is $(2^k-1)/k$.

Can you improve RLE coding?

- Why use fixed length coding for the length of a run?

Golomb Coding

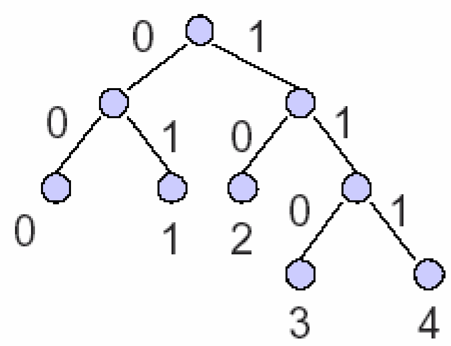
- How to code a potential infinite number of symbols?
  - Code the number of consecutive heads in a sequence of coin tosses.
  - 110, 1111110, 11111110, ...

Golomb Coding

- Let $n = qm + r$ where $0 \leq r < m$.
  - Divide m into n to get the quotient q and remainder r.
- Code for n has two parts:

1. q is coded in unary.
2. r is coded as a fixed prefix code.

Example: $m = 5$



code for r

Example

- $n = qm + r$ is represented by:

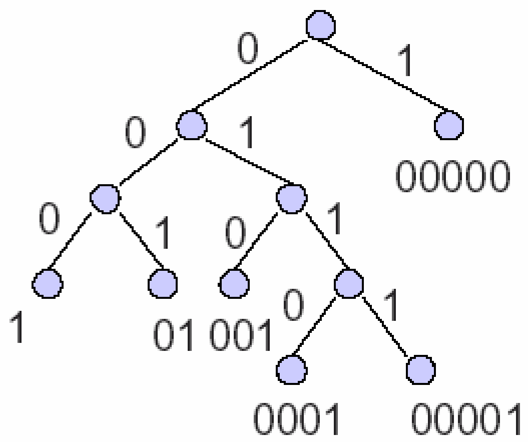
$$\overbrace{11\cdots1}^{q}0\hat{r}$$

  – where $\hat{r}$ is the fixed prefix code for r

- Example (m=5):

2	6	9	10	27
010	1001	10111	11000	11111010

Another Way of Looking at Golomb Code (m=5)



input	output
00000	1
00001	0111
0001	0110
001	010
01	001
1	000

Run-Length Example, m = 5

00000010000000001000000000010001001.....
1
00000010000000001000000000010001001.....
001
00000010000000001000000000010001001.....
1
00000010000000001000000000010001001.....
0111

In this example we coded 17 bit in only 9 bits.

Choosing m

- Suppose that 0 has the probability p and 1 has probability 1-p.
- The probability of 0^n1 is $p^n(1-p)$. The Golomb code of order

$$m = \left\lceil -1 \Big/ \log_2 p \right\rceil$$

  is optimal.
- Example: p = 127/128.

$$m = \left\lceil -1 \Big/ \log_2 (127/128) \right\rceil = 89$$

Compression of Golomb Code

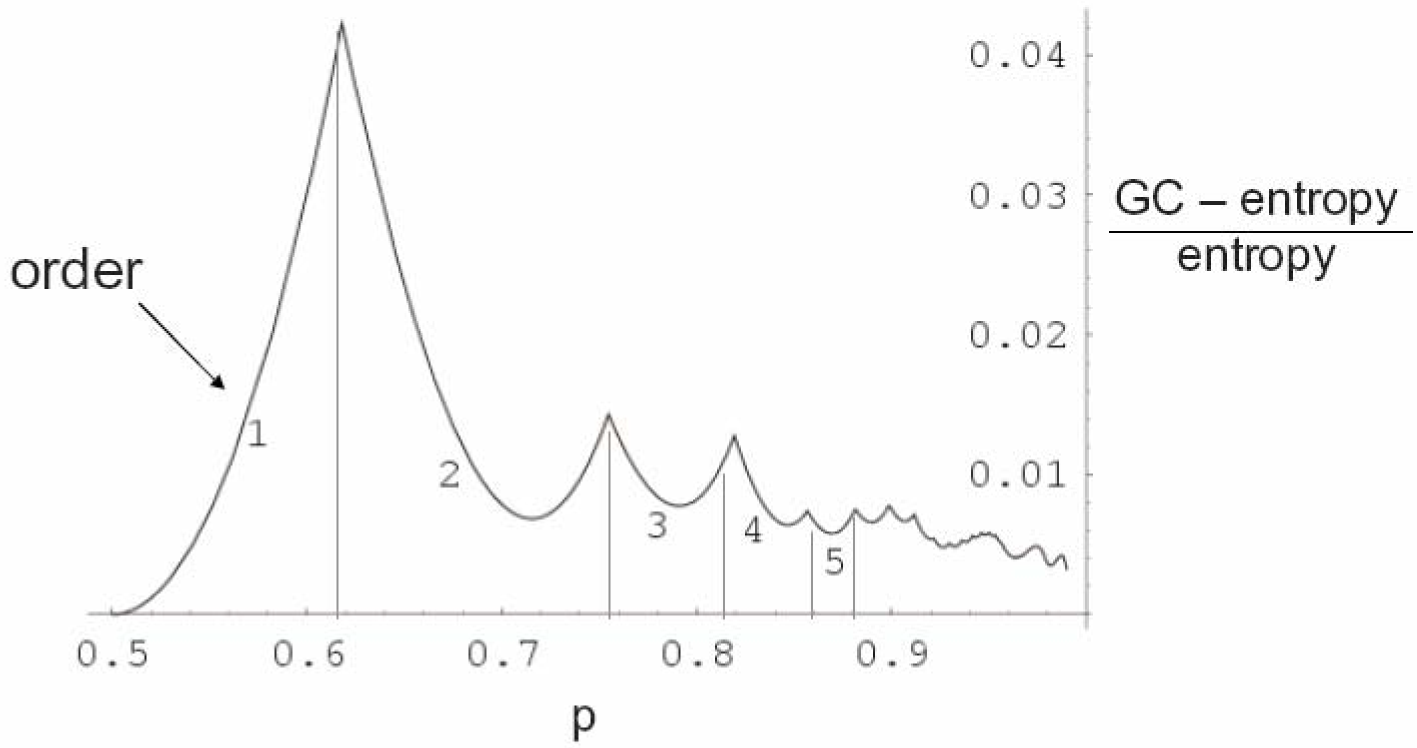
$$\text{Average Bit Rate} = \frac{\text{Average output code length}}{\text{Average input code length}}$$

- m = 4 as an example. With p as the probability of 0.

$$ABR = \frac{p^4 + 3(1-p^4)}{4p^4 + 4p^3(1\text{-}p) + 3p^2(1\text{-}p) + 2p(1-p) + (1-p)}$$

ouput	1	011	010	001	000
input	0000	0001	001	01	1
probability	p^4	$p^3(1\text{-}p)$	$p^2(1\text{-}p)$	$p(1\text{-}p)$	$1\text{-}p$

GC Performance



Notes on GC

- Useful for binary compression when one symbol is much more likely than another.
  - binary images
  - fax documents
  - bit planes for wavelet image compression
- Need a parameter (the order)
  - training
  - adaptively learn the right parameter
- Variable-to-variable length code
- Last symbol needs to be a 1
  - coder always adds a 1
  - decoder always removes a 1

Tunstall Code

- Variable-to-fixed length code
- Example

input	output
a	000
b	001
ca	010
cb	011
cca	100
ccb	101
ccc	110

a	b	cca	cb	ccc	...
000	001	110	011	110	...

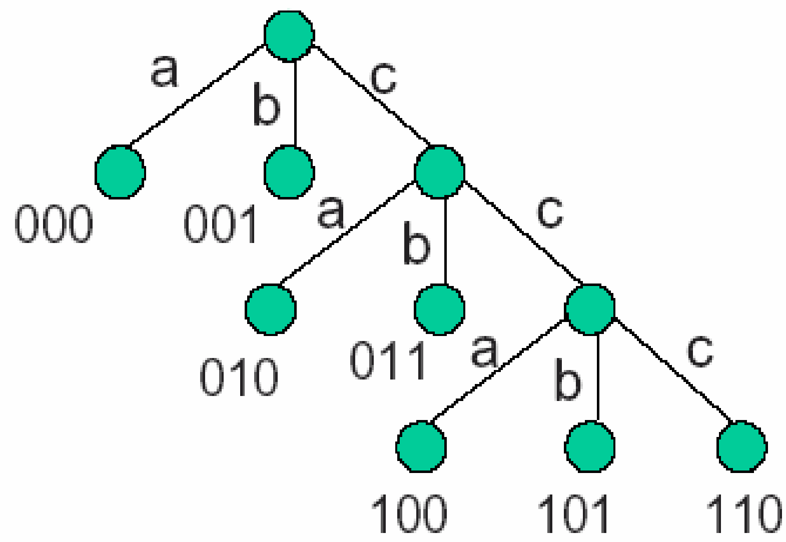
Tunstall Code Properties

- No input code is a prefix of another to assure unique *encodability*.

- Minimize the number of bits per symbol.

Prefix Code Properties

a	000
b	001
ca	010
cb	011
cca	100
ccb	101
ccc	110



Unused output code is 111.

Prefix Code Properties

- Consider the string “cc”. It does not have a code.
- Send the unused code and some fixed code for the cc.
- Generally, if there are k internal nodes in the prefix tree then there is a need for k-1 fixed codes.

Designing Tunstall Code

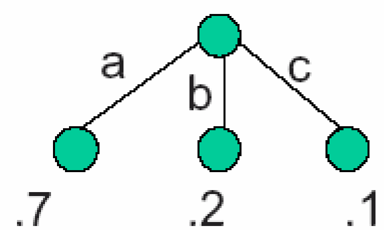
- Suppose there are m initial symbols.
- Choose a target output length n where $2^n > m$.

1. Form a tree with a root and m children with edges labeled with the symbols.
2. If the number of leaves is $> 2^n - m$ then halt.*
3. Find the leaf with highest probability and expand it to have m children.** Go to 2.

* In the next step we will add m-1 more leaves.
** The probability is the product of the probabilities of the symbols on the root to leaf path.

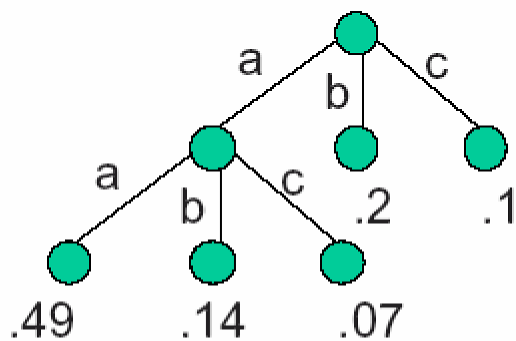
Example

- $P(a) = .7$, $P(b) = .2$, $P(c) = .1$
- $n = 3$



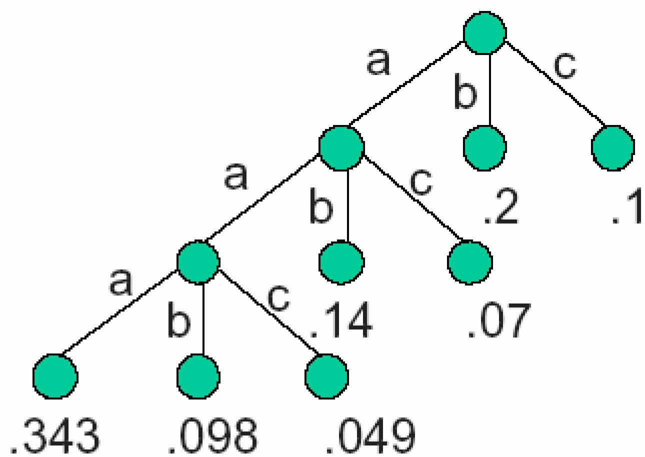
Example

- $P(a) = .7$, $P(b) = .2$, $P(c) = .1$
- $n = 3$



Example

- P(a) = .7, P(b) = .2, P(c) = .1
- n = 3

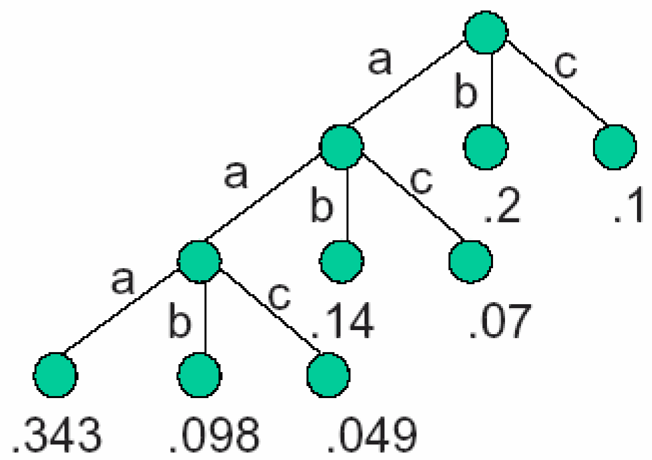


Compression of Tunstall Code

- The length of the output code divided by the average length of the input code.
- Let p_i be the probability of input code i and r_i the length of input code i ($1 \leq i \leq s$) and let n be the length of the output code.

$$\text{Average bit rate} = \frac{n}{\sum_{i=1}^{s} p_i r_i}$$

Average Bit Rate of Tunstall Code



ABR = 3/[3 (.343 + .098 + .049) + 2 (.14 + .07) + .2 + .1]
= 1.37 bits per symbol
Entropy = 1.16 bits per symbol

Notes on Tunstall Code

- Variable-to-fixed length code
- Error resilient
  - A flipped bit will introduce just one error in the output.
  - Huffman is not error resilient. A single bit flip can destroy the code.