ECE 5578 Multimedia Communication

Lec 03
Entropy and Coding II
Hoffman and Golomb Coding

Outline

- ❑ Lecture 02 ReCap
- ❑ Hoffman Coding
- ❑ Golomb Coding and JPEG 2000 Lossless Coding

Entropy

- ❑ Self Info of an event

$$i(X = x_k) = -\log(\Pr\{X = x_k\}) = -\log(p_k)$$

- ❑ Entropy of a source

$$H(X) = \sum_k p_k \log(\frac{1}{p_k})$$

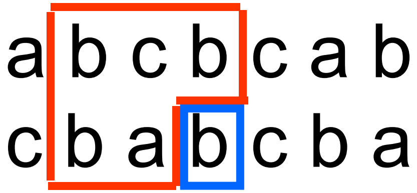
- ❑ Conditional Entropy, Mutual Information

$$H(X_1|X_2) = H(X_1, X_2) - H(X_2)$$
$$I(X_1, X_2) = H(X_1) + H(X_2) - H(X_1, X_2)$$

- ❑ Relative Entropy

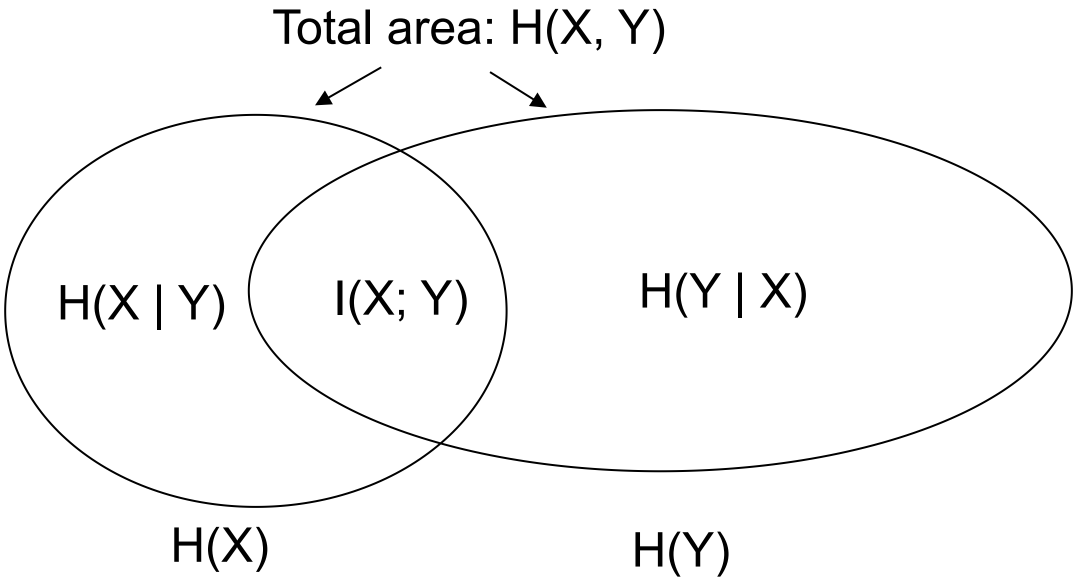
$$D(p||q) = \sum_k p_k \log\left(\frac{p_k}{q_k}\right)$$

Main application: Context Modeling



a b c b c a b

c b a b c b a



Total area: H(X, Y)

H(X | Y)

I(X; Y)

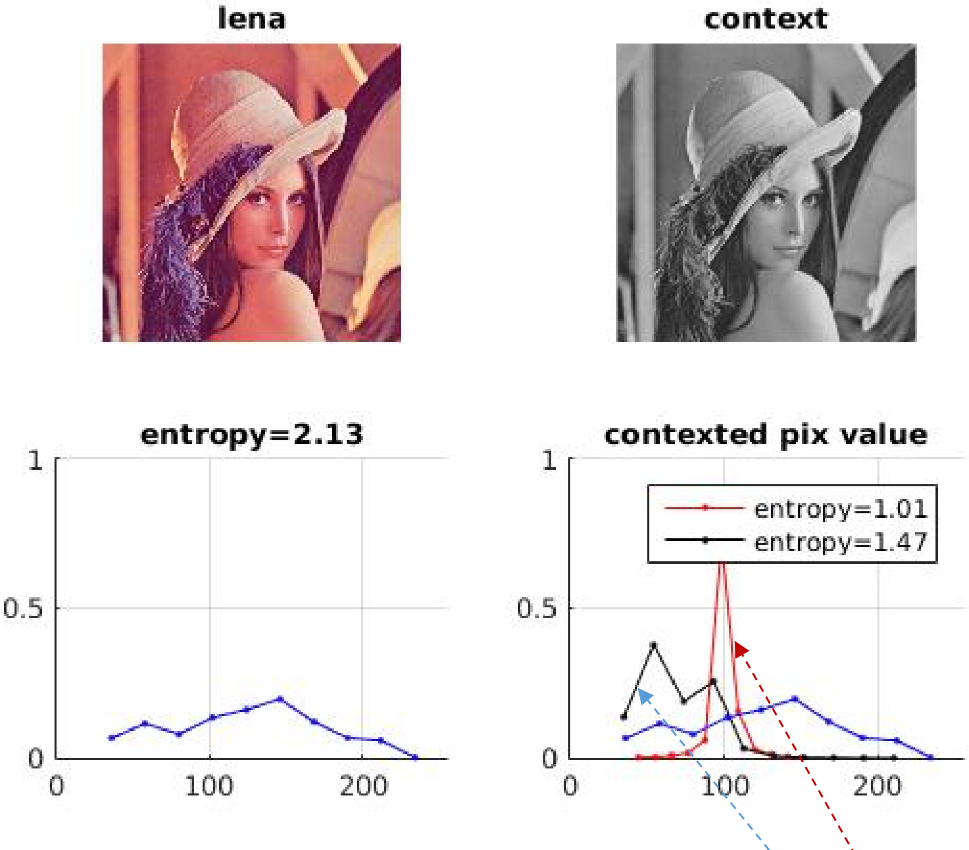
H(Y | X)

H(X)

H(Y)

Context Reduces Entropy Example

lenna.png



$f(x_4,x_3, x_2,x_1)$==100

$f(x_4,x_3, x_2,x_1)$<100

getEntropy.m, lossless_coding.m

Condition reduces entropy:

$$H(x_5) > H(x_5|x_4,x_3, x_2,x_1)$$

$$H(x_5) > H(x_5|f(x_4,x_3, x_2,x_1))$$

Context:

x_1	x_2	x_3
x_4	x_5	

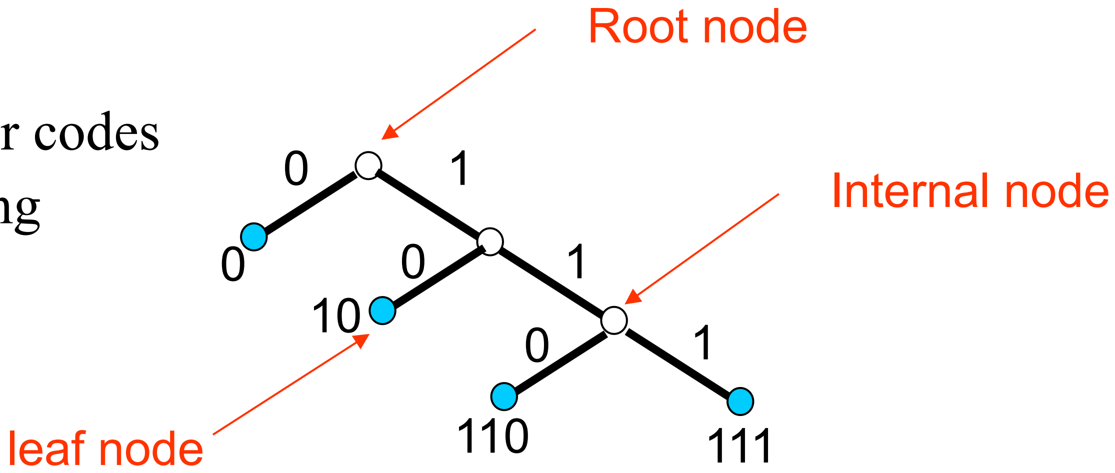
Context function:

$f(x_4,x_3, x_2,x_1)$= *sum*(x_4,x_3, x_2,x_1)

Lossless Coding

❑ Prefix Coding

- Codes on leaves
- No code is prefix of other codes
- Simple encoding/decoding



❑Kraft- McMillan Inequality:

- For a coding scheme with code length: l_1, l_2, …l_n,

$$\sum_k 2^{-l_k} \leq 1$$

- Given a set of integer length {l_1, l_2, …l_n} that satisfy above inequality, we can always find a prefix code with code length l_1, l_2, …l_n

Outline

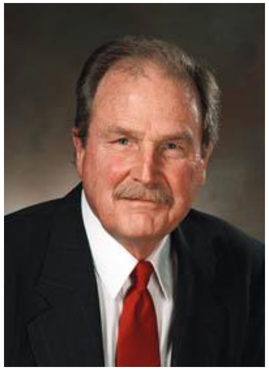
❑ Lecture 02 ReCap

❑ Hoffman Coding

❑ Golomb Coding and JPEG 2000 Lossless

Huffman Coding

- ❑A procedure to construct optimal prefix code
- ❑Result of David Huffman's term paper in 1952 when he was a PhD student at MIT
  - Shannon → Fano → Huffman (1925-1999)

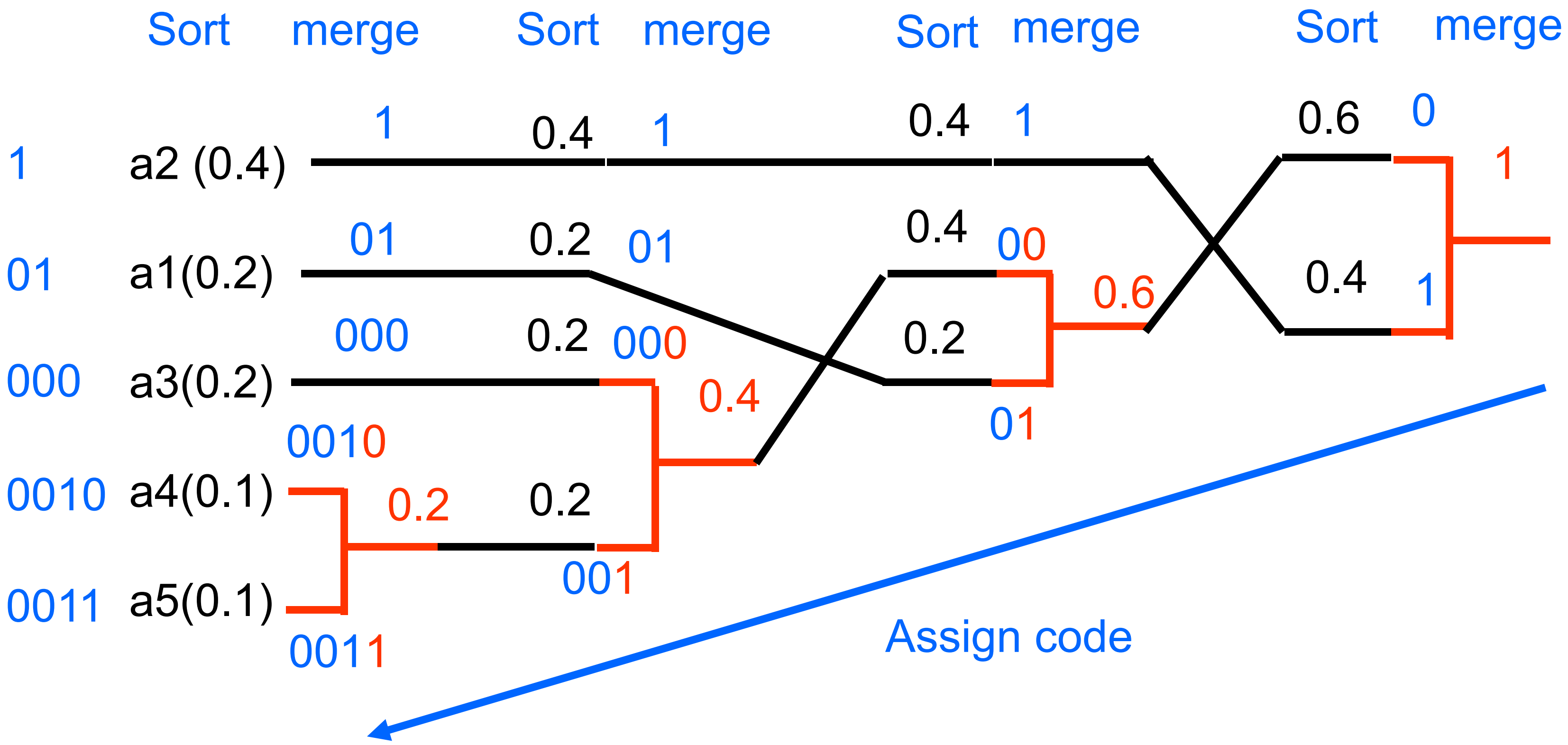


Huffman Code Design

- Requirement:
  - The source probability distribution.
    (But not available in many cases)
- Procedure:
  1. Sort the probability of all source symbols in a descending order.
  2. Merge the last two into a new symbol, add their probabilities.
  3. Repeat Step 1, 2 until only one symbol (the root) is left.
  4. Code assignment:
     Traverse the tree from the root to each leaf node,
     assign 0 to the top branch and 1 to the bottom branch.

Example

- ❑Source alphabet A = {a_1, a_2, a_3, a_4, a_5}
- ❑Probability distribution: {0.2, 0.4, 0.2, 0.1, 0.1}



Huffman code is prefix-free

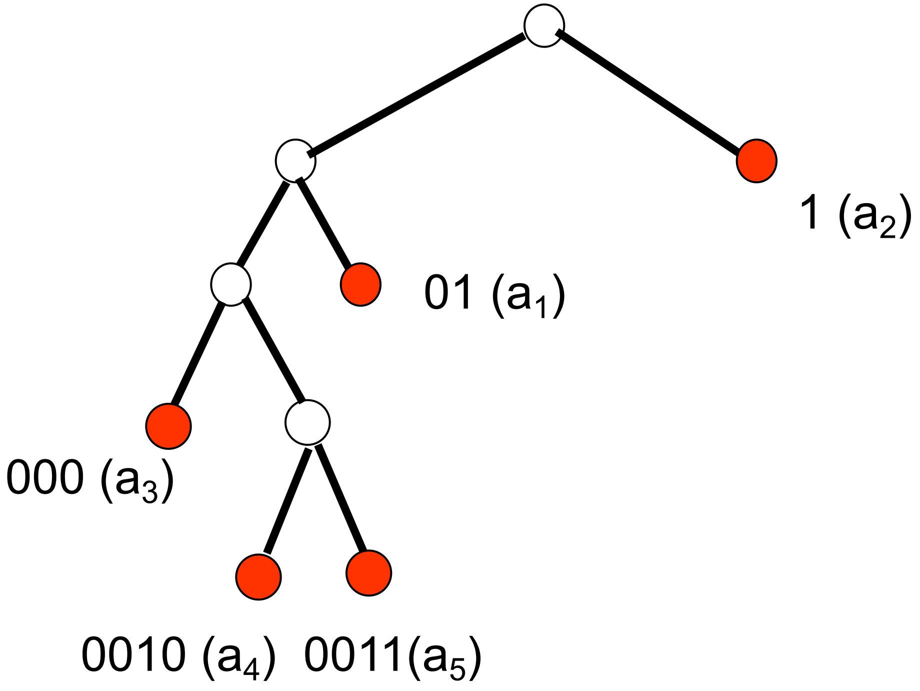
1

01

000

0010

0011



- All codewords are *leaf nodes*

  → No code is a prefix of any other code. (Prefix free)

Average Codeword Length vs Entropy

- Source alphabet A = {a, b, c, d, e}
- Probability distribution: {0.2, 0.4, 0.2, 0.1, 0.1}
- Code: {01, 1, 000, 0010, 0011}

- Entropy:

H(S) = - (0.2*log$_2$(0.2)*2 + 0.4*log$_2$(0.4)+0.1*log$_2$(0.1)*2)

= 2.122 bits / symbol

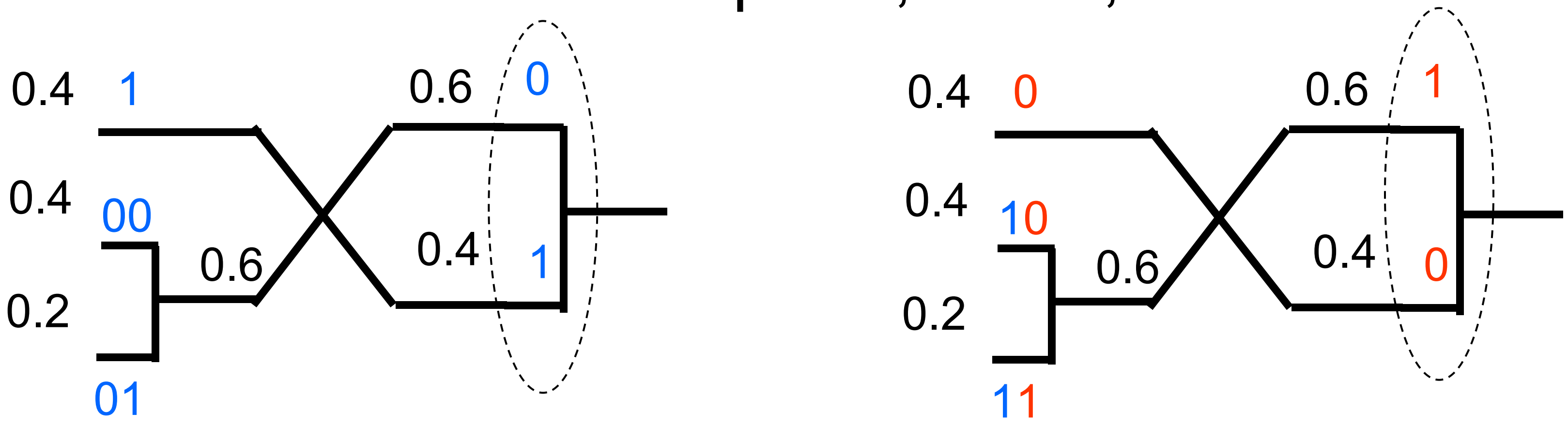
- Average Huffman codeword length:

  L = 0.2*2+0.4*1+0.2*3+0.1*4+0.1*4 = 2.2 bits / symbol

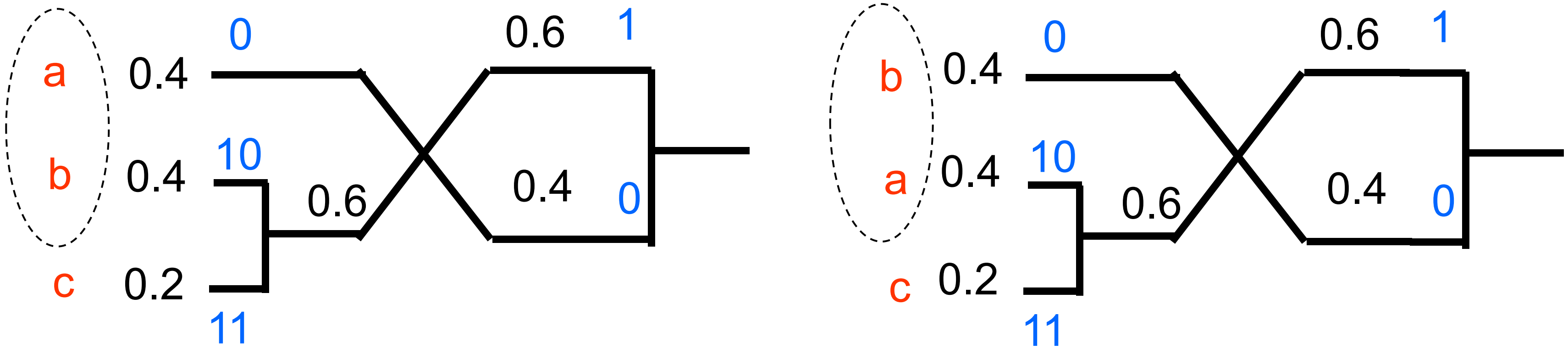
- This verifies H(S) ≤ L < H(S) + 1.

Huffman Code is not unique

- Two choices for each split: 0, 1 or 1, 0



- Multiple ordering choices for tied probabilities



Huffman Coding is Optimal

❑ Assume the probabilities are ordered:

- $p_1 \geq p_2 \geq \ldots\ldots p_m$.

❑ **Lemma**: For any distribution, there exists an optimal prefix code that satisfies:

- If $p_j \geq p_k$, then $l_j \leq l_k$: otherwise can swap codewords to reduce the average length.

❑ The two least probable letters have the same length: otherwise we can truncate the longer one without violating prefix-free condition.

❑ The two longest codewords differ only in the last bit and correspond to the two least probable symbols. Otherwise we can rearrange to achieve this.

❑ Proof skipped.

Canonical Huffman Code

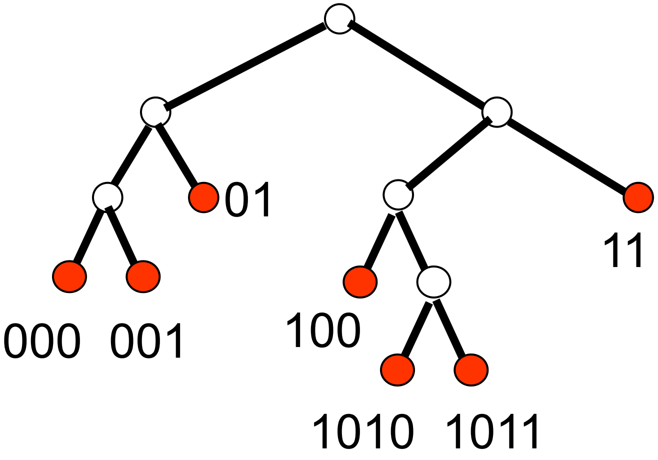
- ❑ Huffman algorithm is needed only to compute the optimal codeword lengths
  - ▪ The optimal codewords for a given data set are not unique
- ❑ Canonical Huffman code is well structured
- ❑ Given the codeword lengths, can find a canonical Huffman code
- ❑ Also known as slice code, alphabetic code.

Canonical Huffman Code

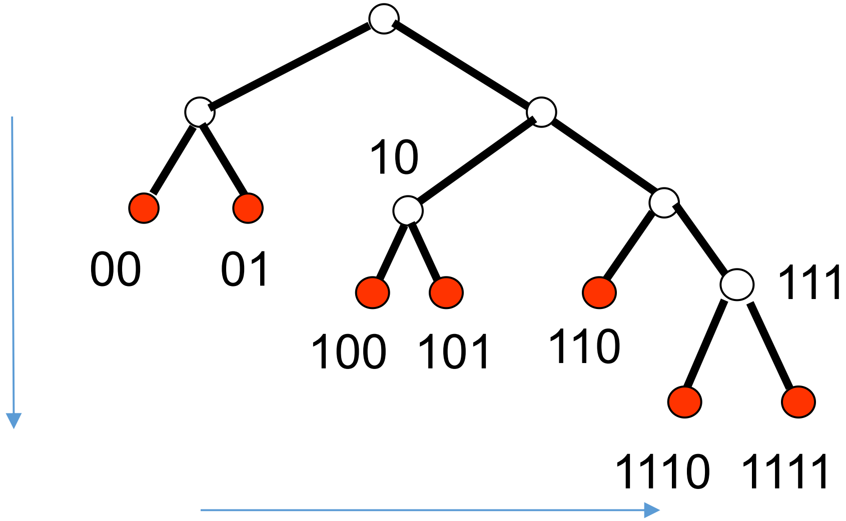
- Rules:
  - Assign 0 to left branch and 1 to right branch
  - Build the tree from left to right in increasing order of depth
  - Each leaf is placed at the first available position
- Example:
  - Codeword lengths: 2, 2, 3, 3, 3, 4, 4
  - Verify that it satisfies Kraft-McMillan inequality

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1$$

A non-canonical example



The Canonical Tree

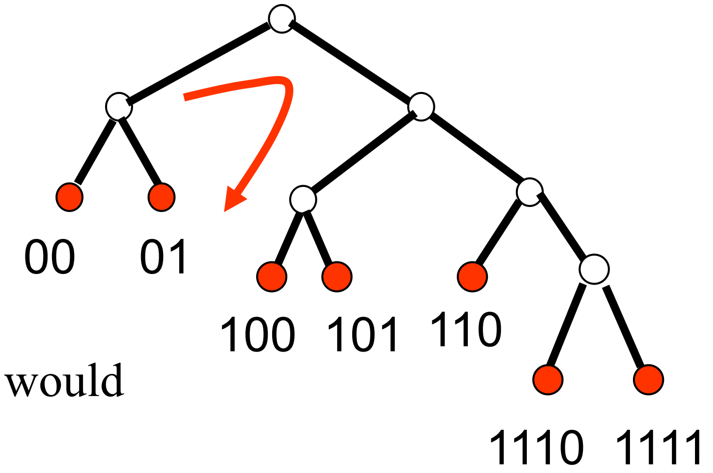


Canonical Huffman

❑Properties:

- The first code is a series of 0
- Codes of same length are consecutive: 100, 101, 110
- If we pad zeros to the right side such that all codewords have the same length, shorter codes would have lower value than longer codes:

0000 < 0100 < 1000 < 1010 < 1100 < 1110 < 1111

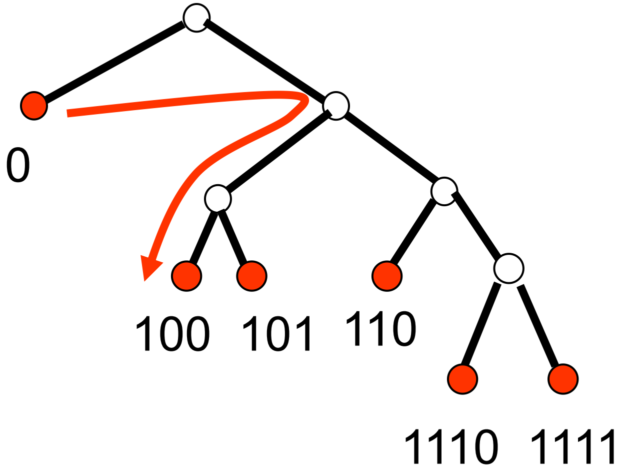


❑ Coding from length level n to level n+1:

- $\underbrace{C(n+1, 1)}_{\text{First code of length n+1}} = 2 (\underbrace{C(n, last) +}_{\text{Last code of length n}} 1)$: append a 0 to the next available level-n code

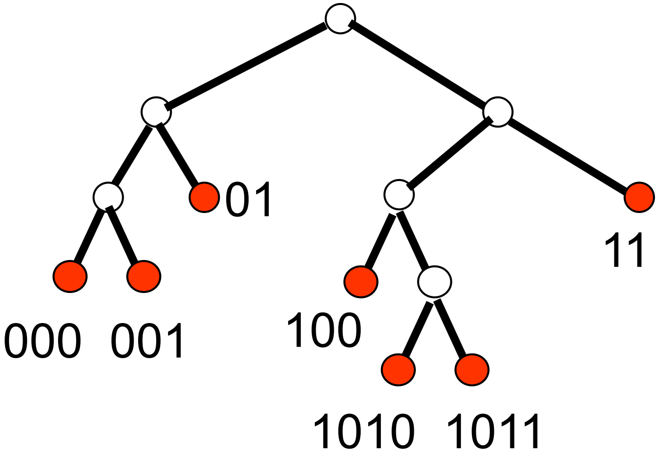
- If from length n to n + 2 directly:
  e.g., 1, 3, 3, 3, 4, 4

$$C(n+2, 1) = 4(C(n, last) + 1)$$

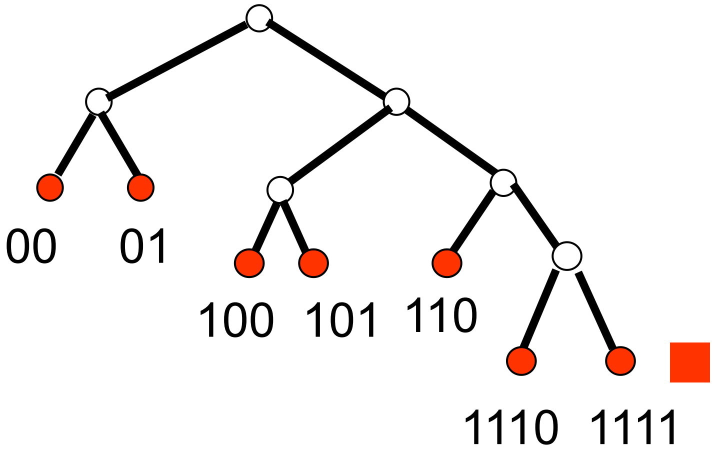


Advantages of Canonical Huffman

1. Reducing memory requirement



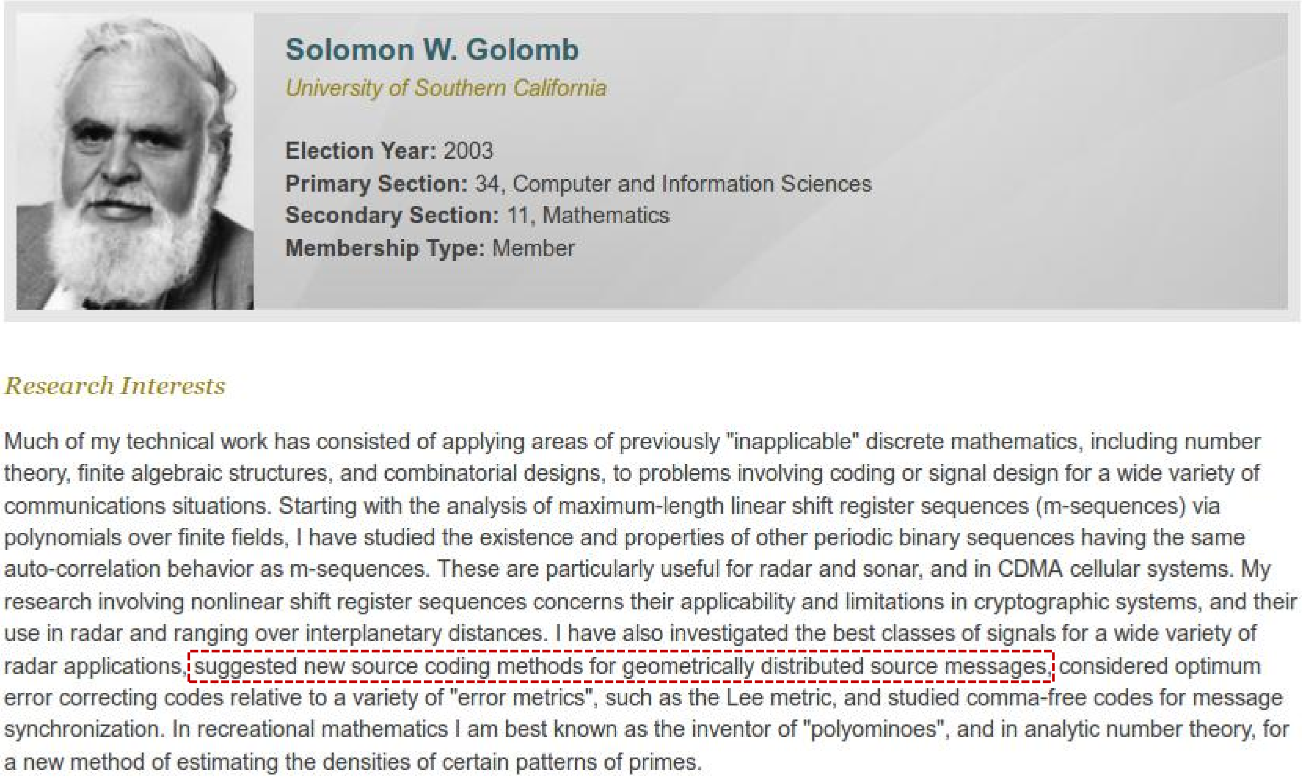
- Non-canonical tree needs:
  - □ All codewords
  - □ Lengths of all codewords
- Need a lot of space for large table



- Canonical tree only needs:
  - □ Min: shortest codeword length
  - □ Max: longest codeword length
  - □ Distribution:
    - Number of codewords in each level
- Min=2, Max=4,
  # in each level: 2, 3, 2

Outline

- □ Lecture 02 ReCap
- □ Hoffman Coding
- □ Golomb Coding



Solomon W. Golomb

University of Southern California

Election Year: 2003
Primary Section: 34, Computer and Information Sciences
Secondary Section: 11, Mathematics
Membership Type: Member

Research Interests

Much of my technical work has consisted of applying areas of previously "inapplicable" discrete mathematics, including number theory, finite algebraic structures, and combinatorial designs, to problems involving coding or signal design for a wide variety of communications situations. Starting with the analysis of maximum-length linear shift register sequences (m-sequences) via polynomials over finite fields, I have studied the existence and properties of other periodic binary sequences having the same auto-correlation behavior as m-sequences. These are particularly useful for radar and sonar, and in CDMA cellular systems. My research involving nonlinear shift register sequences concerns their applicability and limitations in cryptographic systems, and their use in radar and ranging over interplanetary distances. I have also investigated the best classes of signals for a wide variety of radar applications, suggested new source coding methods for geometrically distributed source messages, considered optimum error correcting codes relative to a variety of "error metrics", such as the Lee metric, and studied comma-free codes for message synchronization. In recreational mathematics I am best known as the inventor of "polyominoes", and in analytic number theory, for a new method of estimating the densities of certain patterns of primes.

NATIONAL ACADEMY OF ENGINEERING

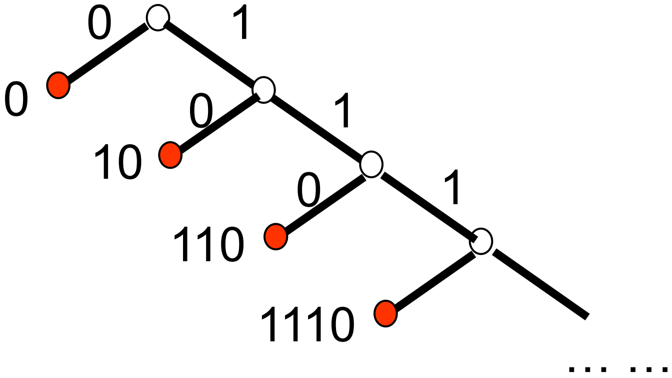
THE NATIONAL ACADEMY OF SCIENCES
IN SERVICE TO THE NATION

Unary Code (Comma Code)

❑Encode a nonnegative integer n by n 1's and a 0 (or n 0's and an 1).

- No need to store codeword table, very simple
- Is this code prefix-free?

n	Codeword
0	0
1	10
2	110
3	1110
4	11110
5	111110
… …	



- When is this code optimal?
  - ❑ When probabilities are: 1/2, 1/4, 1/8, 1/16, 1/32 … ➔ D-adic
  - ❑ Huffman code becomes unary code in this case.

Implementation – Very Efficient

- Encoding:

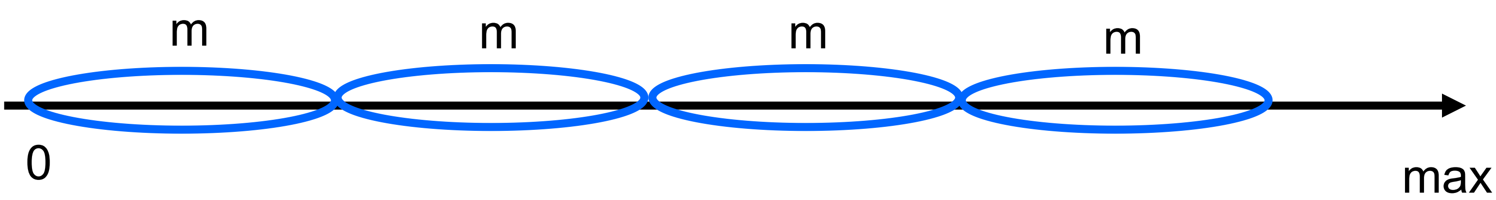
```
UnaryEncode(n) {
  while (n > 0) {
    WriteBit(1);
    n--;
  }
  WriteBit(0);
}
```

- Decoding:

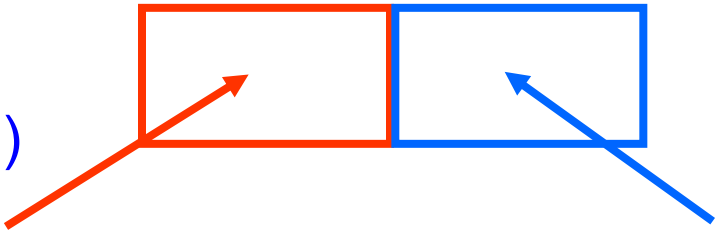
```
UnaryDecode() {
  n = 0;
  while (ReadBit(1) == 1) {
    n++;
  }
  return n;
}
```

Golomb Code [Golomb, 1966]

- A multi-resolutional approach:
  - Divide all numbers into groups of equal size m
    - Denote as Golomb(m) or Golomb-m
  - Groups with smaller symbol values have shorter codes
  - Symbols in the same group has codewords of similar lengths
    - The codeword length grows much slower than in unary code



- Codeword :
  - (Unary, fixed-length)



Group ID:
Unary code

Index within each group:

Golomb Code

$$n = qm + r = \left\lfloor \frac{n}{m} \right\rfloor m + r$$

- q: Quotient, used unary code

q	Codeword
0	0
1	10
2	110
3	1110
4	11110
5	111110
6	1111110
…	…

- r: remainder, "fixed-length" code
- K bits if m = 2^k
  - ❑ m=8: 000, 001, ……, 111
- If m ≠ 2^k: (not desired)

$\lfloor \log_2 m \rfloor$ bits for smaller r

$\lceil \log_2 m \rceil$ bits for larger r

m = 5: 00, 01, 10, 110, 111

Golomb Code with m = 5 (Golomb-5)

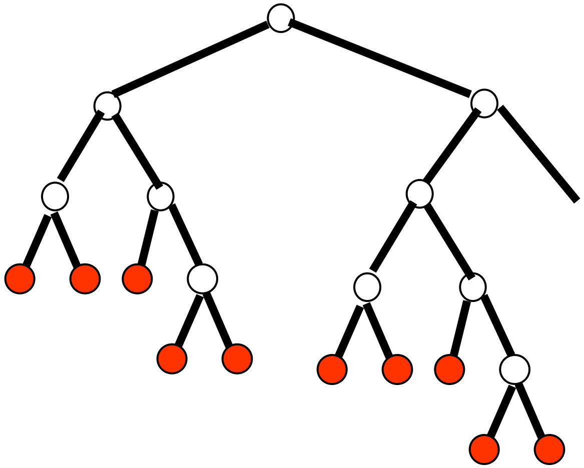
n	q	r	code
0	0	0	000
1	0	1	001
2	0	2	010
3	0	3	0110
4	0	4	0111

n	q	r	code
5	1	0	1000
6	1	1	1001
7	1	2	1010
8	1	3	10110
9	1	4	10111

n	q	r	code
10	2	0	11000
11	2	1	11001
12	2	2	11010
13	2	3	110110
14	2	4	110111

Golomb vs Canonical Huffman

- Codewords: 000, 001, 010, 0110, 0111, 1000, 1001, 1010, 10110, 10111
- Canonical form:
  - From left to right
  - From short to long
  - Take first valid spot



- Golomb code is a canonical Huffman
  - With more properties

Golobm-Rice Code

- A special Golomb code with m= 2^k
- The remainder r is the fixed k LSB bits of n

- m = 8

n	q	r	code
0	0	0	0000
1	0	1	0001
2	0	2	0010
3	0	3	0011
4	0	4	0100
5	0	5	0101
6	0	6	0110
7	0	7	0111

n	q	r	code
8	1	0	10000
9	1	1	10001
10	1	2	10010
11	1	3	10011
12	1	4	10100
13	1	5	10101
14	1	6	10110
15	1	7	10111

Implementation

- Encoding:

Remainder bits:
RBits = 3 for m = 8

```
GolombEncode(n, RBits) {
  q = n >> RBits;
  UnaryCode(q);
  WriteBits(n, RBits);
}
```

Output the lower (RBits) bits of n.

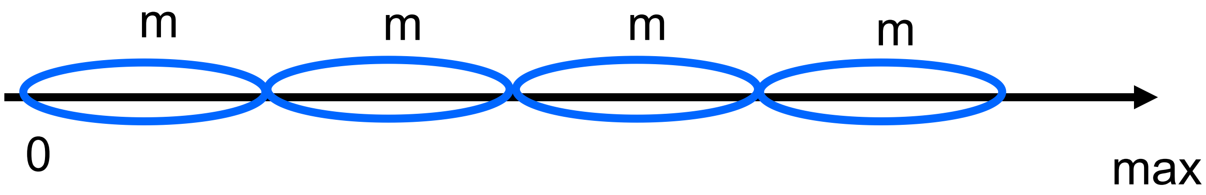
n	q	r	code
0	0	0	0000
1	0	1	0001
2	0	2	0010
3	0	3	0011
4	0	4	0100
5	0	5	0101
6	0	6	0110
7	0	7	0111

- Decoding:

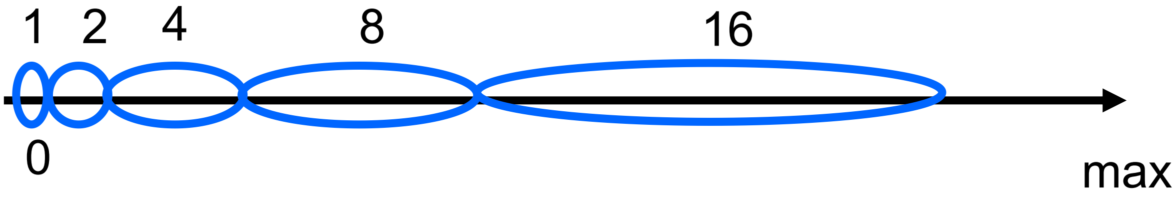
```
GolombDecode(RBits) {
  q = UnaryDecode();
  n = (q << RBits) + ReadBits(RBits);
  return n;
}
```

Exponential Golomb Code (Exp-Golomb)

- ❑Golomb code divides the alphabet into groups of equal size



- In Exp-Golomb code, the group size increases exponentially
- Codes still contain two parts:
  - ❑ Unary code followed by *fixed-length* code



- Proposed by Teuhola in 1978

n	code
0	0
1	100
2	101
3	11000
4	11001
5	11010
6	11011
7	1110000
8	1110001
9	1110010
10	1110011
11	1110100
12	1110101
13	1110110
14	1110111
15	111100000

Implementation

❑Decoding

```
ExpGolombDecode() {
    GroupID = UnaryDecode();
    if (GroupID == 0) {
            return 0;
    } else {
        Base = (1 << GroupID) - 1;
        Index = ReadBits(GroupID);
        return (Base + Index);
    }
  }
}
```

n	code	Group ID
0	0	0
1	100	1
2	101	
3	11000	2
4	11001	
5	11010	
6	11011	
7	1110000	3
8	1110001	
9	1110010	
10	1110011	
11	1110100	
12	1110101	
13	1110110	
14	1110111	

Outline

- ❑Golomb Code Family:
  - Unary Code
  - Golomb Code
  - Golomb-Rice Code
  - Exponential Golomb Code
- ❑Why Golomb code?

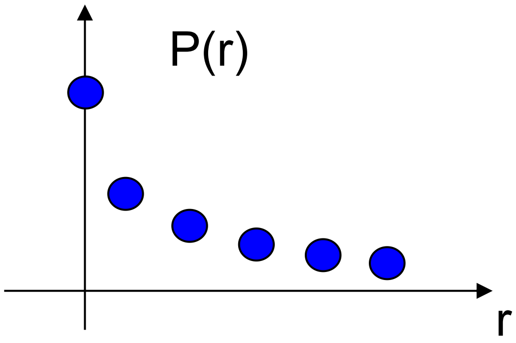
Geometric Distribution (GD)

- Geometric distribution with parameter ρ:
  - $P(x) = \rho^x (1 - \rho)$, $x \geq 0$, integer.
  - Prob of the number of failures before the first success in a series of independent Yes/No experiments (Bernoulli trials).
- Unary code is the optimal prefix code for geometric distribution with $\rho \leq 1/2$:
- $\rho = 1/4$: P(x): 0.75, 0.19, 0.05, 0.01, 0.003, …
  - Huffman coding never needs to re-order ➔ equivalent to unary code.
  - Unary code is the optimal prefix code, but not efficient

    (avg length >> entropy)
- $\rho = 3/4$: P(x): 0.25, 0.19, 0.14, 0.11, 0.08, …
  - Reordering is needed for Huffman code, unary code not optimal prefix code.
- $\rho = 1/2$: Expected length = entropy.
  - Unary code is not only the optimal prefix code, but also optimal among all entropy coding (including arithmetic coding).

Geometric Distribution (GD)

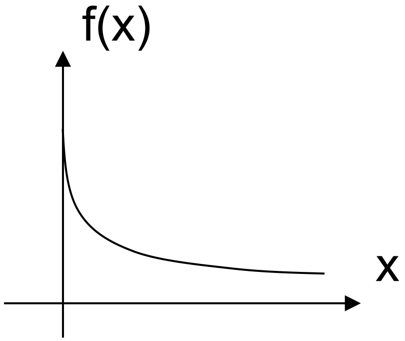
- ❑Geometric distribution is very useful for image/video compression
- ❑Example 1: run-length coding
  - Binary sequence with i.i.d. distribution
  - $P(0) = \rho \approx 1$:
  - Example: 000001000000001000011000001
  - Entropy << 1, prefix code has poor performance.
  - Run-length coding is efficient to compress the data:
    - r: Number of consecutive 0's between two 1's
    - ➔ run-length representation of the sequence: 5, 8, 4, 0, 6
  - Probability distribution of the run-length r:
    - $P(r = n) = \rho^n (1- \rho)$: n 0's followed by an 1.
    - ➔ The run has one-sided geometric distribution with parameter ρ.



Geometric Distribution

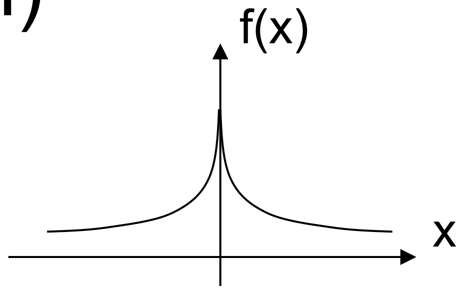
- GD is the discrete analogy of the Exponential distribution

$$f(x) = \lambda\, e^{-\lambda x}$$



- Two-sided geometric distribution is the discrete analogy of the Laplacian distribution (also called double exponential distribution)

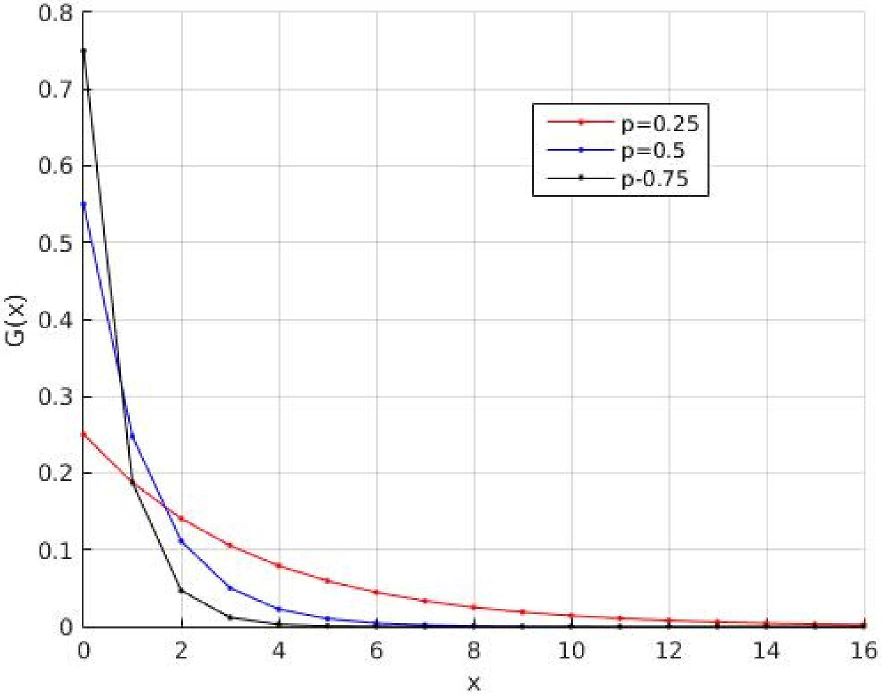
$$f(x) = \tfrac{1}{2}\lambda\, e^{-\lambda|x|}$$



Why Golomb Code?

❑Significance of Golomb code:

- For any geometric distribution (GD), Golomb code is optimal prefix code and is as close to the entropy as possible (among all prefix codes)
- How to determine the Golomb parameter?
- How to apply it into practical codec?



Geometric Distribution

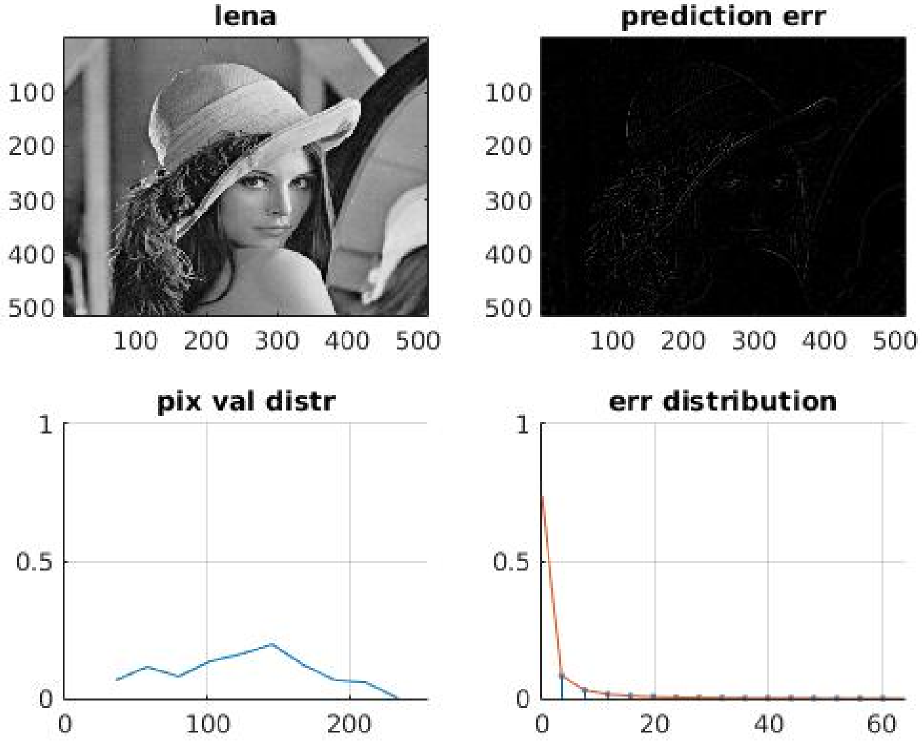
❑Example 2: GD is a also good model for Prediction error

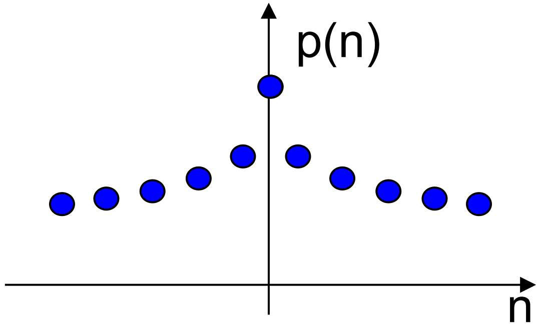
e(n) = x(n) – pred(x(1), …, x(n-1)).

$$p(n) = \frac{1-\rho}{1+\rho}\rho^{|n|}, \quad 0 < \rho < 1$$

❑Most e(n)'s have smaller values around 0:

❑➔ can be modeled by geometric distribution.





x_1	x_2	x_3
x_4	x_5	

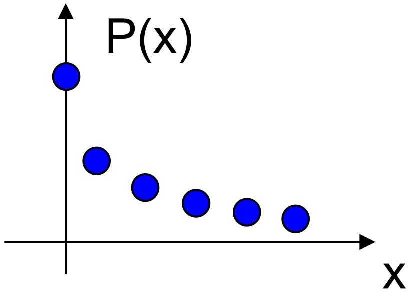
0.2	0.3	0.2
0.2	0	0
0	0	0

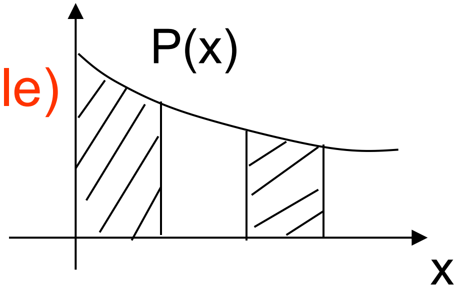
$$\hat{x_5} = \sum_{k=1}^{4} w_k * x_k$$

Optimal Code for Geometric Distribution

- Geometric distribution with parameter ρ:
  - ❑ P(X=n) = $\rho^n (1 - \rho)$
- Unary code is optimal prefix code when ρ ≤ 1/2.
  - ❑ Also optimal among all entropy coding for ρ = 1/2.
- How to design the optimal code when ρ > 1/2 ?
  - ❑ Transform into GD with ρ ≤ 1/2 (as close as possible)

    How? By grouping m events together!





Each x can be written as $x = x_q m + x_r$

$$P_{X_q}(q) = \sum_{r=0}^{m-1} P_X(qm+r) = \sum_{r=0}^{m-1} (1-\rho)\rho^{qm+r} = (1-\rho)\rho^{qm}\frac{1-\rho^m}{1-\rho} = \rho^{mq}(1-\rho^m)$$

➔ x_q has geometric dist with parameter ρ^m.

Unary code is optimal for x_q if $\rho^m \le 1/2$ ➔

$$m \ge -\frac{1}{\log_2 \rho} \qquad m = \left\lceil -\frac{1}{\log_2 \rho} \right\rceil \text{ is the minimal possible integer.}$$

Golomb Parameter Estimation (J2K book: pp. 55)

- Goal of adaptive Golomb code:
  - For the given data, find the best m such that $\rho^m \leq 1/2$.
- How to find ρ from the statistics of past data?

$$P(x) = (1-\rho)\rho^x$$

$$\Rightarrow \quad E(x) = \sum_{x=0}^{\infty}(1-\rho)x\rho^x = (1-\rho)\frac{\rho}{(1-\rho)^2} = \boxed{\frac{\rho}{1-\rho}}$$

$$\Rightarrow \quad \rho = \frac{E(x)}{1+E(x)}.$$

$$\rho^m = \left(\frac{E(x)}{1+E(x)}\right)^m \leq \frac{1}{2} \quad \Rightarrow \quad m \geq 1/\log\left(\frac{1+E(x)}{E(x)}\right)$$

Let m=2^k $\Rightarrow$ $k \geq \log_2\left(1/\log_2\left(\frac{1+E(x)}{E(x)}\right)\right).$ Too costly to compute

Golomb Parameter Estimation (J2K book: pp. 55)

$$E(x) = \frac{\rho}{1-\rho}$$

A faster method: Assume ρ ≈ 1, 1 – ρ ≈ 0.

$$\rho^m = \left(1-(1-\rho)\right)^m \approx 1 - m(1-\rho) \approx 1 - m\frac{1-\rho}{\rho} = 1 - \frac{m}{E(x)}$$

$\rho^m \le 1/2$ ➡ $m = 2^k \ge \frac{1}{2}E(x)$

$$\boxed{k = \max\left\{0, \ \left\lceil \log_2\left(\frac{1}{2}E(x)\right)\right\rceil\right\}.}$$

Summary

- ❑ Hoffman Coding
  - A prefix code that is optimal in code length (average)
  - Canonical form to reduce variation of the code length
  - Widely used
- ❑ Golomb Coding
  - Suitable for coding prediction errors in image
  - Optimal for Geometrical Distribution of p=0.5
  - Simple to encode and decode
  - Many practical applications, e.g., JPEG-2000 lossless.

Q&A