Lec 03
Entropy and Coding II
Hoffman and Golomb Coding

Zhu Li

---

**Entropy**

- **Self Info** of an event
  \[ i(X = x_k) = -\log(Pr(X = x_k)) = -\log(p_k) \]
- **Entropy** of a source
  \[ H(X) = \sum p_k \log \left( \frac{1}{p_k} \right) \]
- **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**
  \[ D(p||q) = \sum_k p_k \log \left( \frac{p_k}{q_k} \right) \]

---

**Context Reduces Entropy Example**

- **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 function**:
  \[ f(x_4, x_3, x_2, x_1) = \text{sum}(x_4, x_3, x_2, x_1) \]
  \[ f(x_4, x_3, x_2, x_1) < 100 \]

---

Outline

- Lecture 02 ReCap
- Hoffman Coding
- Golomb Coding and JPEG 2000 Lossless Coding
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: \( l_1, l_2, \ldots l_n \),
  \[ \sum_{i=1}^{n} 2^{-l_i} \leq 1 \]
  - Given a set of integer length \( \{l_1, l_2, \ldots l_n \} \) that satisfy above inequality, we can always find a prefix code with code length \( l_1, l_2, \ldots l_n \).

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 \( \rightarrow \) Fano \( \rightarrow \) 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: $\{0.2, 0.4, 0.2, 0.1, 0.1\}$

<table>
<thead>
<tr>
<th></th>
<th>Sort</th>
<th>merge</th>
<th>Sort</th>
<th>merge</th>
<th>Sort</th>
<th>merge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a2 (0.4)</td>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>a1 (0.2)</td>
<td>01</td>
<td>0.2</td>
<td>01</td>
<td>0.2</td>
<td>01</td>
</tr>
<tr>
<td>000</td>
<td>a3 (0.2)</td>
<td>000</td>
<td>0.2</td>
<td>000</td>
<td>0.2</td>
<td>000</td>
</tr>
<tr>
<td>0010</td>
<td>a4 (0.1)</td>
<td>0.4</td>
<td>0010</td>
<td>0.4</td>
<td>0010</td>
<td>0.4</td>
</tr>
<tr>
<td>0011</td>
<td>a5 (0.1)</td>
<td>0011</td>
<td>0.4</td>
<td>0011</td>
<td>0.4</td>
<td>0011</td>
</tr>
</tbody>
</table>

Assign code

Huffman code is prefix-free

- 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:
  \[
  H(S) = - (0.2 \log_2(0.2) \times 2 + 0.4 \log_2(0.4) + 0.1 \log_2(0.1) \times 2) \\
  = 2.122 \text{ bits / symbol}
  \]

- Average Huffman codeword length:
  \[
  L = 0.2 \times 2 + 0.4 \times 1 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 2.2 \text{ bits / symbol}
  \]

- This verifies $H(S) \leq L < H(S) + 1$.

Huffman Code is not unique

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

- Multiple ordering choices for tied probabilities
Huffman Coding is Optimal

- Assume the probabilities are ordered:
  - $p_1 \geq p_2 \geq \ldots \geq p_m$.

- **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

- Coding from length level $n$ to level $n+1$:
  - $C(n+1, 1) = 2 \cdot (C(n, last) + 1)$: append a 0 to the next available level-n code

  - First code of length $n+1$ Last code of length $n$

- If from length $n$ to $n + 2$ directly:
  - e.g., 1, 3, 3, 3, 3, 4
  - $C(n+2, 1) = 4 \cdot (C(n, last) + 1)$
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
- Huffman Coding
- Golomb Coding

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?

<table>
<thead>
<tr>
<th>( n )</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>1110</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
</tr>
<tr>
<td>5</td>
<td>111110</td>
</tr>
</tbody>
</table>

- When is this code optimal?
  - When probabilities are: 1/2, 1/4, 1/8, 1/16, 1/32 ... \( \Rightarrow \) D-adic
  - Huffman code becomes unary code in this case.

Implementation – Very Efficient

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

- Decoding:
  ```java
  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 have codewords of similar lengths
    - The codeword length grows much slower than in unary code

- Codeword:
  - (Unary, fixed-length)

Golomb Code

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

- $q$: Quotient, used unary code
- $r$: remainder, “fixed-length” code
- $m=8$: 000, 001, …, 111
- $m \neq 2^k$: (not desired)
  - $\lceil \log_2 m \rceil$ 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)

<table>
<thead>
<tr>
<th>n</th>
<th>q</th>
<th>r</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0111</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>1010</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>10110</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>4</td>
<td>10111</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>11000</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>11001</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>2</td>
<td>11010</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>3</td>
<td>110110</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>4</td>
<td>110111</td>
</tr>
</tbody>
</table>

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
A special Golomb code with $m = 2^k$

- The remainder $r$ is the fixed $k$ LSB bits of $n$

$m = 8$

<table>
<thead>
<tr>
<th>n</th>
<th>q</th>
<th>r</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>q</th>
<th>r</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10001</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>10010</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>10011</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>10100</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>10101</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>10110</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>10111</td>
</tr>
</tbody>
</table>

**Golomb-Rice Code**

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

$m = 8$

**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

**Implementation**

**Encoding**:

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

**Decoding**:

```c
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

**Implementation**

**Decoding**:

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

- Golomb Code Family:
  - Unary Code
  - Golomb Code
  - Golomb-Rice Code
  - Exponential Golomb Code

- Why Golomb code?

Geometric Distribution (GD)

- Geometric distribution with parameter $\rho$:
  - $P(x) = \rho^x (1 - \rho), \ x \geq 0, \ \text{integer}$.
  - 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 $\Rightarrow$ 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 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)
  $$f(x) = \frac{1}{2} \lambda e^{-\lambda |x|}$$
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
  $$e(n) = x(n) - \text{pred}(x(1), \ldots, x(n-1)).$$
- Most $$e(n)$$'s have smaller values around 0:
  - can be modeled by geometric distribution.

Optimal Code for Geometric Distribution

- Geometric distribution with parameter $$\rho$$:
  - $$P(X=n) = \rho^n (1 - \rho)$$
  - Unary code is optimal prefix code when $$\rho \leq 1/2$$.
  - Also optimal among all entropy coding for $$\rho = 1/2$$.
  - How to design the optimal code when $$\rho > 1/2$$?
  - Transform into GD with $$\rho \leq 1/2$$ (as close as possible)
  - How? By grouping m events together!
  - Each x can be written as $$x = x_q m + x_r$$
  $$P_{x_q}(q) = \sum_{r=0}^{m-1} P_x(qm + r) = \sum_{r=0}^{m-1}(1 - \rho)\rho^{qm+r} = (1 - \rho)\rho^m \frac{1 - \rho^m}{1 - \rho} = \rho^m (1 - \rho^m)$$
  - $$x_q$$ has geometric dist with parameter $$\rho^m$$.
  - Unary code is optimal for $$x_q$$ if $$\rho^m \leq 1/2$$
  - $$m \geq -\frac{1}{\log \rho} \Rightarrow m = \left\lceil -\frac{1}{\log \rho} \right\rceil$$ 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 $$\rho$$ from the statistics of past data?
  - $$P(x) = (1 - \rho)\rho^x$$
  - $$E(x) = \sum_{x=0}^{\infty} (1 - \rho)x\rho^x = (1 - \rho)\frac{\rho}{(1 - \rho)^2} = \frac{\rho}{1 - \rho}$$
  - $$\rho = \frac{E(x)}{1 + E(x)}$$
  - $$\rho^m = \left(\frac{E(x)}{1 + E(x)}\right)^m \leq \frac{1}{2} \Rightarrow m \geq 1\log \left(\frac{1 + E(x)}{E(x)}\right)$$
  - Too costly to compute
Golomb Parameter Estimation (J2K book: pp. 55)

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

A faster method: Assume \( \rho \approx 1, 1 - \rho \approx 0. \)

\[ \rho^n = (1 - (1 - \rho))^n \approx 1 - m(1 - \rho) \approx 1 - m \frac{1 - \rho}{\rho} = 1 - \frac{m}{E(x)} \]

\[ \rho^m \leq 1/2 \quad \Rightarrow \quad m = 2^k \geq \frac{1}{2} E(x) \]

\[ k = \max \left\{ 0, \left\lfloor \log_2 \left( \frac{1}{2} E(x) \right) \right\rfloor \right\} \]

Q&A

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.