CSEP 590
Data Compression
Autumn 2007

Course Policies
Introduction to Data Compression
Entropy
Variable Length Codes

Instructors

• Instructor
  – Richard Ladner
  – ladner@cs.washington.edu
  – 206 543-9347
• TA
  – Rahul Vanam
  – rahulv@uwashington.edu

Helpful Knowledge

• Algorithm Design and Analysis
• Probability

Resources

• Text Book
• Course Web Page
  – http://www.cs.washington.edu/courses/csep590a
• Papers and Sections from Books
• Discussion Board
  – For discussion

Engagement by Students

• Weekly Assignments
  – Understand compression methodology
  – Due in class on Fridays (except midterm Friday)
  – No late assignments accepted except with prior approval
• Programming Projects
  – Bi-level arithmetic coder and decoder,
  – Build code and experiment

Final Exam and Grading

• 6:30-8:20 p.m. Thursday, Dec. 13, 2007
• Percentages
  – Weekly assignments (50%)
  – Project (20%)
  – Final exam (30%)
Logistics

- I will be gone the week of October 15th. We’ll need to have a make up class.
- There is no class Thanksgiving week, November 19th.
- We have some guest speakers toward the end of the quarter.

Basic Data Compression Concepts

- **Lossless compression** $x = \hat{x}$
  - Also called entropy coding, reversible coding.
- **Lossy compression** $x \neq \hat{x}$
  - Also called irreversible coding.
- **Compression ratio** $\frac{|\hat{x}|}{|x|}$
  - $|x|$ is number of bits in $x$.

Why Compress

- **Conserve storage space**
- **Reduce time for transmission**
  - Faster to encode, send, then decode than to send the original
- **Progressive transmission**
  - Some compression techniques allow us to send the most important bits first so we can get a low resolution version of some data before getting the high fidelity version
- **Reduce computation**
  - Use less data to achieve an approximate answer

Braille

- System to read text by feeling raised dots on paper (or on electronic displays). Invented in 1820s by Louis Braille, a French blind man.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>the</td>
<td>with</td>
<td>mother</td>
</tr>
<tr>
<td>th</td>
<td>ch</td>
<td>gh</td>
<td></td>
</tr>
</tbody>
</table>

Braille Example

Clear text:
Call me Ishmael. Some years ago -- never mind how long precisely -- having \ little or no money in my purse, and nothing particular to interest me on shore, \ I thought I would sail about a little and see the watery part of the world. (238 characters)

Grade 2 Braille in ASCII.
```
call me .s ye$>$-$s ago -- n`e m$>$d h[ l$>$g precisely -- hav$>$ \ l$>$ or no m`i$>$y my pur$>$e! \ & no?+ `picul$>$-$s 6 9$t[el me on \%\%ore! \ ] j $7$p$>$-$s 8$p$>$ jw sail ab a ll \& see ! wat$>$y `p ( 4\,\,\,4 203 characters)
```

Compression ratio = 238/203 = 1.17

Lossless Compression

- Data is not lost - the original is really needed.
  - text compression
  - compression of computer binary files
- Compression ratio typically no better than 4:1 for lossless compression on many kinds of files.
- **Statistical Techniques**
  - Huffman coding
  - Arithmetic coding
  - Golomb coding
- **Dictionary techniques**
  - LZW, LZ77
  - Seqitur
  - Burrows-Wheeler Method
- **Standards** - Morse code, Braille, Unix compress, gzip, zip, bzip, GIF, JPEG, Lossless JPEG
Lossy Compression
• Data is lost, but not too much.
  – audio
  – video
  – still images, medical images, photographs
• Compression ratios of 10:1 often yield quite high fidelity results.
• Major techniques include
  – Vector Quantization
  – Wavelets
  – Block transforms
  – Standards - JPEG, JPEG2000, MPEG 2, H.264

Why is Data Compression Possible
• Most data from nature has redundancy
  – There is more data than the actual information contained in the data.
  – Squeezing out the excess data amounts to compression.
  – However, unsqueezing is necessary to be able to figure out what the data means.
• Information theory is needed to understand the limits of compression and give clues on how to compress well.

What is Information
• Analog data
  – Also called continuous data
  – Represented by real numbers (or complex numbers)
• Digital data
  – Finite set of symbols \( \{a_1, a_2, \ldots, a_m\} \)
  – All data represented as sequences (strings) in the symbol set.
  – Example: \( \{a,b,c,d,r\} \) abracadabra
  – Digital data can be an approximation to analog data

Symbols
• Roman alphabet plus punctuation
• ASCII - 256 symbols
• Binary - \( \{0,1\} \)
  – 0 and 1 are called bits
  – All digital information can be represented efficiently in binary
  – \( \{a,b,c,d\} \) fixed length representation
  
<table>
<thead>
<tr>
<th>symbol</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
  – 2 bits per symbol

Exercise - How Many Bits Per Symbol?
• Suppose we have \( n \) symbols. How many bits (as a function of \( n \) ) are needed in to represent a symbol in binary?
  – First try \( n \) a power of 2.

Discussion: Non-Powers of Two
• Can we do better than a fixed length representation for non-powers of two?
Information Theory

- Developed by Shannon in the 1940's and 50's
- Attempts to explain the limits of communication using probability theory.
- Example: Suppose English text is being sent
  - It is much more likely to receive an "e" than a "z".
  - In some sense "z" has more information than "e".

First-order Information

- Suppose we are given symbols \( \{a_1, a_2, \ldots, a_m\} \).
- \( P(a_i) \) = probability of symbol \( a_i \) occurring in the absence of any other information.
- \( P(a_i) + P(a_{i-1}) + \ldots + P(a_1) = 1 \)
- \( \text{inf}(a_i) = \log_2 \left( \frac{1}{P(a_i)} \right) \) bits is the information of \( a_i \) in bits.

Example

- \( \{a, b, c\} \) with \( P(a) = 1/8, P(b) = 1/4, P(c) = 5/8 \)
  - \( \text{inf}(a) = \log_2(8) = 3 \)
  - \( \text{inf}(b) = \log_2(4) = 2 \)
  - \( \text{inf}(c) = \log_2(8/5) = .678 \)
- Receiving an "a" has more information than receiving a "b" or "c".

First Order Entropy

- The first order entropy is defined for a probability distribution over symbols \( \{a_1, a_2, \ldots, a_m\} \).
- \( H = \sum_{i=1}^{m} P(a_i) \log_2 \left( \frac{1}{P(a_i)} \right) \)
- \( H \) is the average number of bits required to code up a symbol, given all we know is the probability distribution of the symbols.
- \( H \) is the Shannon lower bound on the average number of bits to code a symbol in this "source model".
- Stronger models of entropy include context.

Entropy Examples

- \( \{a, b, c\} \) with a 1/8, b 1/4, c 5/8.
  - \( H = 1/8 * 3 + 1/4 * 2 + 5/8 * .678 = 1.3 \text{ bits/symbol} \)
- \( \{a, b, c\} \) with a 1/3, b 1/3, c 1/3. (worst case)
  - \( H = 3 * (1/3) * \log_2(3) = 1.6 \text{ bits/symbol} \)
- Note that a standard code takes 2 bits per symbol

<table>
<thead>
<tr>
<th>symbol</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary code</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

An Extreme Case

- \( \{a, b, c\} \) with a 1, b 0, c 0
  - \( H = ? \)
Entropy Curve

- Suppose we have two symbols with probabilities $x$ and $1-x$, respectively.

$$-x \log x - (1-x) \log(1-x)$$

A Simple Prefix Code

- $\{a, b, c\}$ with $a \frac{1}{8}$, $b \frac{1}{4}$, $c \frac{5}{8}$.
- A prefix code is defined by a binary tree.
- Prefix code property:
  - no output is a prefix of another

Binary Tree Terminology

1. Each node, except the root, has a unique parent.
2. Each internal node has exactly two children.

Decoding a Prefix Code

```
repeat
  start at root of tree
  repeat
    if read bit = 1 then go right
    else go left
  until node is a leaf
  report leaf
  until end of the code
```

Decoding a Prefix Code

```
1 1 0 0 0 1 1 1 0 1 1 1 1
```

Decoding a Prefix Code

```
1 1 0 0 0 1 1 1 0 1 1 1 1
```
Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100

Decoding a Prefix Code

11000111100
Decoding a Prefix Code

```
0 1
0 1 0
0 1 0
1 1 1
1 1 1
1 1 0
1 1 0
```

1100011100
cca

1100011100
ccab

Decoding a Prefix Code

```
0 1
0 1 0
0 1 0
1 1 1
1 1 1
1 1 0
1 1 0
```

```
ccab
ccca
```

Exercise Encode/Decode

- Player 1: Encode a symbol string
- Player 2: Decode the string
- Check for equality

```
• Player 1: Encode a symbol string
• Player 2: Decode the string
• Check for equality
```

How Good is the Code

```
0 1
0 1 0
0 1 0
1 1 1
1 1 1
1 1 0
1 1 0
```

```
bit rate = (1/8)2 + (1/4)2 + (5/8)1 = 11/8 = 1.375 bps
Entropy = 1.3 bps
Standard code = 2 bps
```

(bps = bits per symbol)

Design a Prefix Code 1

- abracadabra
- Design a prefix code for the 5 symbols {a,b,r,c,d} which compresses this string the most.
Design a Prefix Code 2

- Suppose we have \( n \) symbols each with probability \( 1/n \). Design a prefix code with minimum average bit rate.
- Consider \( n = 2, 3, 4, 5, 6 \) first.

Huffman Coding

- Huffman (1951)
- Uses frequencies of symbols in a string to build a variable rate prefix code.
  - Each symbol is mapped to a binary string.
  - More frequent symbols have shorter codes.
  - No code is a prefix of another.
- Example:

```
  a  0
  b  100
  c  101
  d  11
```

Variable Rate Code Example

- Example: \( a, 0, b, 100, c, 101, d, 11 \)
- Coding:
  - \( aabddca \) = 16 bits
  - \( 0 0 100 11 11 101 0 0 \) = 14 bits
- Prefix code ensures unique decodability.
  - \( 0010011110100 \)
  - \( \text{a a b d d c a} \)

Cost of a Huffman Tree

- Let \( p_1, p_2, \ldots, p_m \) be the probabilities for the symbols \( a_1, a_2, \ldots, a_m \), respectively.
- Define the cost of the Huffman tree \( T \) to be
  \[
  C(T) = \sum_{i=1}^{m} p_i r_i
  \]
  where \( r_i \) is the length of the path from the root to \( a_i \).
  - \( C(T) \) is the expected length of the code of a symbol coded by the tree \( T \). \( C(T) \) is the bit rate of the code.

Example of Cost

- Example: \( a, 1/2, b, 1/8, c, 1/8, d, 1/4 \)
- \( T \)

```
  C(T) = 1 \times 1/2 + 3 \times 1/8 + 3 \times 1/8 + 2 \times 1/4 = 1.75
  \text{a \ b \ c \ d}
```

Huffman Tree

- Input: Probabilities \( p_1, p_2, \ldots, p_m \) for symbols \( a_1, a_2, \ldots, a_m \), respectively.
- Output: A tree that minimizes the average number of bits (bit rate) to code a symbol. That is, minimizes
  \[
  HC(T) = \sum_{i=1}^{m} p_i r_i
  \]
  where \( r_i \) is the length of the path from the root to \( a_i \). This is the Huffman tree or Huffman code.
Optimality Principle 1
• In a Huffman tree a lowest probability symbol has maximum distance from the root.
  – If not exchanging a lowest probability symbol with one at maximum distance will lower the cost.

\[
C(T) = C(T) + hp - hq + kq - kp = C(T) - (h-k)(q-p) < C(T)
\]

Optimality Principle 2
• The second lowest probability is a sibling of the smallest in some Huffman tree.
  – If not, we can move it there without raising the cost.

\[
C(T') = C(T) + hq - hr + kr - kq = C(T) - (h-k)(r-q) < C(T)
\]

Optimality Principle 3
• Assuming we have a Huffman tree \( T \) whose two lowest probability symbols are siblings at maximum depth, they can be replaced by a new symbol whose probability is the sum of their probabilities.
  – The resulting tree is optimal for the new symbol set.

\[
C(T') = C(T) + (h-1)(p+q) - hp -hq = C(T) - (p+q)
\]

Optimality Principle 3 (cont’)
• If \( T' \) were not optimal then we could find a lower cost tree \( T'' \). This will lead to a lower cost tree \( T''' \) for the original alphabet.

\[
C(T''') = C(T'') + p + q < C(T') + p + q = C(T) \text{ which is a contradiction}
\]

Recursive Huffman Tree Algorithm
1. If there is just one symbol, a tree with one node is optimal. Otherwise
2. Find the two lowest probability symbols with probabilities \( p \) and \( q \) respectively.
3. Replace these with a new symbol with probability \( p + q \).
4. Solve the problem recursively for new symbols.
5. Replace the leaf with the new symbol with an internal node with two children with the old symbols.

Iterative Huffman Tree Algorithm
- form a node for each symbol \( a \) with weight \( p_a \);
- insert the nodes in a min priority queue ordered by probability;
- while the priority queue has more than one element do
  - \( \text{min1} := \text{delete-min}; \)
  - \( \text{min2} := \text{delete-min}; \)
  - create a new node \( n \);
  - \( n \text{.weight} := \text{min1.weight} + \text{min2.weight}; \)
  - \( n \text{.left} := \text{min1}; \)
  - \( n \text{.right} := \text{min2}; \)
  - \( \text{insert}(n) \)
- return the last node in the priority queue.
Example of Huffman Tree Algorithm (1)
- \( P(a) = .4, P(b) = .1, P(c) = .3, P(d) = .1, P(e) = .1 \)

Example of Huffman Tree Algorithm (2)

Example of Huffman Tree Algorithm (3)

Example of Huffman Tree Algorithm (4)

Huffman Code
- average number of bits per symbol is
  \[ .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4 = 2.1 \]

Optimal Huffman Code vs. Entropy
- \( P(a) = .4, P(b) = .1, P(c) = .3, P(d) = .1, P(e) = .1 \)
- Entropy
  \[ H = -.4 \times \log_2(.4) + .1 \times \log_2(.1) + .3 \times \log_2(.3) + .1 \times \log_2(.1) + .1 \times \log_2(.1) \]
  \[ = 2.05 \text{ bits per symbol} \]
- Huffman Code
  \[ HC = .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4 \]
  \[ = 2.1 \text{ bits per symbol} \]
  pretty good!
In Class Exercise

- P(a) = 1/2, P(b) = 1/4, P(c) = 1/8, P(d) = 1/16, P(e) = 1/16
- Compute the Optimal Huffman tree and its average bit rate.
- Compute the Entropy
- Compare
- Hint: For the tree change probabilities to be integers: a:8, b:4, c:2, d:1, e:1. Normalize at the end.

Quality of the Huffman Code

- The Huffman code is within one bit of the entropy lower bound.
  \[ H \leq HC \leq H + 1 \]
- Huffman code does not work well with a two symbol alphabet.
  - Example: P(0) = 1/100, P(1) = 99/100
  - HC = 1 bits/symbol

  \[ H = -(1/100) \log_2 (1/100) + (99/100) \log_2 (99/100) = .08 \text{ bits/symbol} \]

Powers of Two

- If all the probabilities are powers of two then
  \[ HC = H \]
- Proof by induction on the number of symbols.
  Let \( p_1 \leq p_2 \leq ... \leq p_n \) be the probabilities that add up to 1.
  If \( n = 1 \) then \( HC = H \) (both are zero).
  If \( n > 1 \) then \( p_i = 2^{-k} \) for some \( k \), otherwise the sum cannot add up to 1.
  Combine the first two symbols into a new symbol of probability \( 2^{-k} + 2^{-k} = 2^{-k+1} \).

Powers of Two (Cont.)

By the induction hypothesis

\[ HC(p_1, p_2, ..., p_n) = H(p_1, p_2, ..., p_n) - (p_1 + p_2) \]

By the properties of Huffman trees (principle 3),

\[ HC(p_1, p_2, ..., p_n) = HC(p_1 + p_2, p_3, ..., p_n) + (p_1 + p_2) \]

Hence,

\[ HC(p_1, p_2, ..., p_n) = H(p_1, p_2, ..., p_n) \]

Extending the Alphabet

- Assuming independence \( P(ab) = P(a)P(b) \), so we can lump symbols together.
- Example: P(0) = 1/100, P(1) = 99/100
  - P(00) = 1/10000, P(01) = P(10) = 99/10000,
  - P(11) = 9801/10000.

\[ HC = 1.03 \text{ bits/symbol (2 bit symbol)} = .515 \text{ bits/bit} \]

Still not that close to \( H = .08 \text{ bits/bit} \)
Quality of Extended Alphabet

- Suppose we extend the alphabet to symbols of length k then
  \[ H \leq HC \leq H + \frac{1}{k} \]

- Pros and Cons of Extending the alphabet
  + Better compression
    - 2^k symbols
    - padding needed to make the length of the input divisible by k

Huffman Codes with Context

- Suppose we add a one symbol context. That is in compressing a string \( x_1 x_2 \ldots x_n \) we want to take into account \( x_{k-1} \) when encoding \( x_k \).
  - New model, so entropy based on just independent probabilities of the symbols doesn’t hold. The new entropy model (2nd order entropy) has for each symbol a probability for each other symbol following it.
  - Example: \( \{a, b, c\} \)

<table>
<thead>
<tr>
<th>prev</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>4.2</td>
</tr>
<tr>
<td>c</td>
<td>.4</td>
</tr>
</tbody>
</table>

Multiple Codes

<table>
<thead>
<tr>
<th>prev</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>

Code for first symbol

| a    | .4   |
| b    | .2   |
| c    | .4   |

Average Bit Rate for Code

\[
\begin{align*}
P(a) &= .4 \cdot P(a) + .1 \cdot P(b) + .1 \cdot P(c) \\
P(b) &= .2 \cdot P(a) + .9 \cdot P(b) + .1 \cdot P(c) \\
1 &= P(a) + P(b) + P(c) \\
0 &= -.6 \cdot P(a) + .1 \cdot P(b) + .1 \cdot P(c) \\
0 &= .2 \cdot P(a) + .1 \cdot P(b) + .1 \cdot P(c) \\
1 &= P(a) + P(b) + P(c) \\
P(a) &= \frac{1}{7}, P(b) = \frac{4}{7}, P(c) = \frac{2}{7}
\end{align*}
\]

ABR = \( \frac{1}{7} \cdot (.6 \cdot 2 + .4) + \frac{4}{7} \cdot (1) + \frac{2}{7} \cdot (.2 \cdot 2 + .8) \) = 1.14 bps

Complexity of Huffman Code Design

- Time to design Huffman Code is \( O(n \log n) \) where \( n \) is the number of symbols.
  - Each step consists of a constant number of priority queue operations (2 deletemin’s and 1 insert)
Approaches to Huffman Codes

1. Frequencies computed for each input
   - Must transmit the Huffman code or frequencies as well as the compressed input
   - Requires two passes
2. Fixed Huffman tree designed from training data
   - Do not have to transmit the Huffman tree because it is known to the decoder.
   - H.263 video coder
3. Adaptive Huffman code
   - One pass
   - Huffman tree changes as frequencies change

Run-Length Coding

- Lots of 0’s and not too many 1’s.
  - Fax of letters
  - Graphics
- Simple run-length code
  - Input: 00000010000000010000000000010001001....
  - Symbols: 6 9 10 3 2...
  - Code the bits as a sequence of integers
  - Problem: How long should the integers be?

Golomb Code of Order m

Variable Length Code for Integers

- 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: \( m = 5 \)

<table>
<thead>
<tr>
<th>code for ( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

Example (\( m = 5 \)):

<table>
<thead>
<tr>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>010100110111110001111010</td>
</tr>
</tbody>
</table>

Alternative Explanation

Golomb Code of order 5

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>1</td>
</tr>
<tr>
<td>00001</td>
<td>0111</td>
</tr>
<tr>
<td>00010</td>
<td>0110</td>
</tr>
<tr>
<td>00100</td>
<td>010</td>
</tr>
<tr>
<td>01000</td>
<td>001</td>
</tr>
<tr>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>00010</td>
<td>1</td>
</tr>
<tr>
<td>00110</td>
<td>0</td>
</tr>
<tr>
<td>01010</td>
<td>0</td>
</tr>
<tr>
<td>10110</td>
<td>0</td>
</tr>
</tbody>
</table>

Run Length Example: \( m = 5 \)

<table>
<thead>
<tr>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010000000010000000000010001001....</td>
</tr>
<tr>
<td>000000010000000000000000010001001....</td>
</tr>
<tr>
<td>00000000010000000000000000000000010001001....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
</tr>
<tr>
<td>00001</td>
</tr>
<tr>
<td>00010</td>
</tr>
<tr>
<td>00100</td>
</tr>
<tr>
<td>01000</td>
</tr>
<tr>
<td>10000</td>
</tr>
<tr>
<td>00010</td>
</tr>
<tr>
<td>00110</td>
</tr>
<tr>
<td>01010</td>
</tr>
<tr>
<td>10110</td>
</tr>
</tbody>
</table>

In this example we coded 17 bits in only 9 bits.
Choosing m

- Suppose that 0 has the probability \( p \) and 1 has probability \( 1-p \).
- The probability of 0\(^n\)1 is \( p^n(1-p) \). The Golomb code of order \( m = \left\lfloor \frac{1}{\log_2 p} \right\rfloor \) is optimal.
- Example: \( p = \frac{127}{128} \).

\[
m = \left\lfloor \frac{1}{\log_2 (127/128)} \right\rfloor = 89
\]

Average Bit Rate for Golomb Code

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

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

- \( ABR = 5p^4 + 3p^3(1-p) + 3p^2(1-p) + 3p(1-p) + 3(1-p) \) if \( m = 4 \).
- \( ABR = 4p^4 + 4p^3(1-p) + 3p^2(1-p) + 2p(1-p) + (1-p) \) if \( m = 4 \).

Comparison of GC with Entropy

Notes on Golomb codes

- 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 Codes

- Variable-to-fixed length code
- Example

\[
\begin{array}{c|c}
\text{input} & \text{output} \\
\hline
a & 000 \\
b & 001 \\
a & 010 \\
cb & 011 \\
cca & 100 \\
cb & 101 \\
cc & 110 \\
\end{array}
\]

Tunstall code Properties

1. No input code is a prefix of another to assure unique encodability.
2. Minimize the number of bits per symbol.
Prefix Code Property

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>001</td>
</tr>
<tr>
<td>ca</td>
<td>010</td>
</tr>
<tr>
<td>cb</td>
<td>011</td>
</tr>
<tr>
<td>cca</td>
<td>100</td>
</tr>
<tr>
<td>ccb</td>
<td>101</td>
</tr>
<tr>
<td>cccc</td>
<td>110</td>
</tr>
</tbody>
</table>

Unused output code is 111.

Use for unused code

- Consider the string “cc”, if it occurs at the end of the data. 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 a 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
Bit Rate of Tunstall

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

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

Example

<table>
<thead>
<tr>
<th>Input</th>
<th>Code</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>100</td>
<td>0.343</td>
</tr>
<tr>
<td>$b$</td>
<td>101</td>
<td>0.098</td>
</tr>
<tr>
<td>$c$</td>
<td>110</td>
<td>0.049</td>
</tr>
</tbody>
</table>

$$\text{ABR} = \frac{3}{3} (0.343 + 0.098 + 0.049) + 2 (0.14 + 0.07) + 0.2 + 0.1$$

$$= 1.37 \text{ bits per symbol}$$

Entropy = 1.16 bits per symbol

Notes on Tunstall Codes

- 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.