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@u.washington.edu
Helpful Knowledge

• Algorithm Design and Analysis
• Probability
Resources

• Text Book

• Course Web Page
  – http://www.cs.washington.edu/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 15\textsuperscript{th}. We’ll need to have a make up class.
• There is no class Thanksgiving week, November 19\textsuperscript{th}.
• 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** \( = \frac{|x|}{|y|} \)
  - \(|x|\) 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.
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 ,i\%mael4 ,``s ye>$>s ago -- n``e m9d h[l;g precisely -- hav+ \ ll or no m``oy 9 my purse1 \& no?+ ``picul$>$ 6 9t]e/ me on \%ore1 \ ,i $?$``$|$ ,i wd sail ab a ll \& see ! wat]y ``p ( ! \_w4 (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
  – Sequitur
  – Burrows-Wheeler Method

• Standards - Morse code, Braille, Unix compress, gzip, zip, bzip, GIF, JBIG, 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
    - 2 bits per symbol

<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>
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.
  \[P(a_1) + P(a_2) + \ldots + P(a_m) = 1\]
• \(\inf(a_i) = \log_2(1/P(a_i))\) 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.
  \[ H = 1/8 \times 3 + 1/4 \times 2 + 5/8 \times .678 = 1.3 \text{ bits/symbol} \]

• \{a, b, c\} with a 1/3, b 1/3, c 1/3. (worst case)
  \[ H = 3 \times (1/3) \times \log_2(3) = 1.6 \text{ bits/symbol} \]

• 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>00</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
An Extreme Case

• \{a, b, c\} with a 1, b 0, c 0
  – H = ?
Suppose we have two symbols with probabilities \( x \) and \( 1-x \), respectively.

The entropy of these two symbols is given by

\[
-(x \log x + (1-x)\log(1-x))
\]

The maximum entropy occurs at \( x = 0.5 \).
A Simple Prefix Code

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

```
input | output
-----|-------
a    | 00    
b    | 01    
c    | 1     
```

```
cabcccbccc  
1 1 00 01 1 1 01 1 1 1
```
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
```

11000111100
Decoding a Prefix Code

11000111100
Decoding a Prefix Code

11000111100

C
Decoding a Prefix Code

11000111100

c
Decoding a Prefix Code

```
1 000111100
```

cc
Decoding a Prefix Code

11000111100

cc
Decoding a Prefix Code

11000111100

cc
Decoding a Prefix Code

11000111100

cca
Decoding a Prefix Code

11000111100

cca
Decoding a Prefix Code

11000111100

cca
Decoding a Prefix Code

11000111100

ccab
Decoding a Prefix Code

11000111100

ccabccccca
Exercise Encode/Decode

- Player 1: Encode a symbol string
- Player 2: Decode the string
- Check for equality
How Good is the Code

\[ \text{bit rate} = (1/8)2 + (1/4)2 + (5/8)1 = 11/8 = 1.375 \text{ bps} \]
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:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>100</td>
</tr>
<tr>
<td>c</td>
<td>101</td>
</tr>
<tr>
<td>d</td>
<td>11</td>
</tr>
</tbody>
</table>

![Huffman Coding Tree](image)
Variable Rate Code Example

• Example: a 0, b 100, c 101, d 11
• Coding:
  – aabddcaa = 16 bits
  – 0 0 100 11 11 101 0 0 = 14 bits
• Prefix code ensures unique decodability.
  – 00100111110100
  – a a b d d c a 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: \( a \frac{1}{2}, b \frac{1}{8}, c \frac{1}{8}, d \frac{1}{4} \)

\[
C(T) = 1 \times \frac{1}{2} + 3 \times \frac{1}{8} + 3 \times \frac{1}{8} + 2 \times \frac{1}{4} = 1.75
\]
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

\[
HC(T) = \sum_{i=1}^{m} p_i r_i \quad \text{bit rate}
\]

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 not raising the cost.

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

\[ C(T'') = C(T') + p + q < C(T') + p + q = C(T) \] 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 $a_i$ with weight $p_i$; insert the nodes in a min priority queue ordered by probability; while the priority queue has more than one element do
  min1 := delete-min;
  min2 := delete-min;
  create a new node $n$;
  $n$.weight := $min1$.weight + $min2$.weight;
  $n$.left := $min1$;
  $n$.right := $min2$;
  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
\[0.4 \times 1 + 0.1 \times 4 + 0.3 \times 2 + 0.1 \times 3 + 0.1 \times 4 = 2.1\]

a 0
b 1110
c 10
d 110
e 1111
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 \( p_1 = p_2 = 2^{-k} \) 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

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

\[ = -(p_1 + p_2) \log_2 (p_1 + p_2) - \sum_{i=3}^{n} p_i \log_2 (p_i) \]

\[ = -2^{-k+1} \log_2 (2^{-k+1}) - \sum_{i=3}^{n} p_i \log_2 (p_i) \]

\[ = -2^{-k+1} (\log_2 (2^{-k}) + 1) - \sum_{i=3}^{n} p_i \log_2 (p_i) \]

\[ = -2^{-k} \log_2 (2^{-k}) - 2^{-k} \log_2 (2^{-k}) - \sum_{i=3}^{n} p_i \log_2 (p_i) - 2^{-k} - 2^{-k} \]

\[ = -\sum_{i=1}^{n} p_i \log_2 (p_i) - (p_1 + p_2) \]

\[ = H(p_1, p_2, ..., p_n) - (p_1 + p_2) \]
Powers of Two (Cont.)

By the previous page,

\[ \text{HC}(p_1 + p_2, p_3, \ldots, p_n) = H(p_1, p_2, \ldots, p_n) - (p_1 + p_2) \]

By the properties of Huffman trees (principle 3),

\[ \text{HC}(p_1, p_2, \ldots, p_n) = \text{HC}(p_1 + p_2, p_3, \ldots, p_n) + (p_1 + p_2) \]

Hence,

\[ \text{HC}(p_1, p_2, \ldots, p_n) = H(p_1, p_2, \ldots, p_n) \]
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 bits/symbol (2 bit symbol)
= .515 bits/bit

Still not that close to $H = .08$ 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>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>.4</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td>b</td>
<td>.1</td>
<td>.9</td>
<td>.0</td>
</tr>
<tr>
<td>c</td>
<td>.1</td>
<td>.1</td>
<td>.8</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>a</td>
<td>.4</td>
</tr>
<tr>
<td>b</td>
<td>.1</td>
</tr>
<tr>
<td>c</td>
<td>.1</td>
</tr>
</tbody>
</table>

Code for first symbol

- a 00
- b 01
- c 10

- a b b a c c

00 00 0 1 01 0
Average Bit Rate for Code

- $P(a) = 0.4 \ P(a) + 0.1 \ P(b) + 0.1 \ P(c)$
- $P(b) = 0.2 \ P(a) + 0.9 \ P(b) + 0.1 \ P(c)$
- $1 = P(a) + P(b) + P(c)$
- $0 = -0.6 \ P(a) + 0.1 \ P(b) + 0.1 \ P(c)$
- $0 = 0.2 \ P(a) - 0.1 \ P(b) + 0.1 \ P(c)$
- $1 = P(a) + P(b) + P(c)$
- $P(a) = \frac{1}{7}, \ P(b) = \frac{4}{7}, \ P(c) = \frac{2}{7}$
Average Bit Rate for Code

\[
\text{ABR} = \frac{1}{7} (0.6 \times 2 + 0.4) + \frac{4}{7} (1) + \frac{2}{7} (0.2 \times 2 + 0.8)
\]
\[= \frac{8}{7} = 1.14 \text{ bps} \]
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
    00000010000000001000000000010001001.....
  – 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: \( m = 5 \)

```
 0 1 2
0 0
1 1
```

code for \( r \)
Example

- \( n = qm + r \) is represented by:
  \[
  \underbrace{q}_{11 \ldots 10\hat{r}}
  \]
  – where \( \hat{r} \) is the fixed prefix code for \( r \)

- Example (\( m = 5 \)):

  \[
  \begin{array}{cccccc}
  2 & 6 & 9 & 10 & 27 \\
  010 & 1001 & 10111 & 11000 & 11111010 \\
  \end{array}
  \]
Alternative Explanation
Golomb Code of order 5

Variable length to variable length code.
Run Length Example: $m = 5$

```
000000100000000001000000000010001001.....
1
000000100000000001000000000010001001.....
001
000000100000000001000000000010001001.....
1
000000100000000001000000000010001001.....
0111
```

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 $0^n1$ is $p^n(1-p)$. The Golomb code of order
  \[ m = \left\lceil \frac{-1}{\log_2 p} \right\rceil \]
  is optimal.
- Example: $p = 127/128$.
  \[ m = \left\lceil \frac{-1}{\log_2 (127/128)} \right\rceil = 89 \]
Average Bit Rate for Golomb Code

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.

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

<table>
<thead>
<tr>
<th>Output</th>
<th>011</th>
<th>010</th>
<th>001</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input weight</td>
<td>0000</td>
<td>0001</td>
<td>001</td>
<td>01</td>
</tr>
<tr>
<td>p^4</td>
<td>p^3(1-p)</td>
<td>p^2(1-p)</td>
<td>p(1-p)</td>
<td>1-p</td>
</tr>
</tbody>
</table>
Comparison of GC with Entropy

order

\[ p \]

GC – entropy

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

<table>
<thead>
<tr>
<th>input</th>
<th>output</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>ccc</td>
<td>110</td>
</tr>
</tbody>
</table>

input | output |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>cca</td>
<td>cb</td>
</tr>
<tr>
<td>ccc</td>
<td></td>
</tr>
</tbody>
</table>

... 000 001 110 011 110 ...

... a b cca cb ccc ...
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>a</th>
<th>000</th>
</tr>
</thead>
<tbody>
<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>ccc</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 \( 2^n > m \).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Form a tree with a root and ( m ) children with edges labeled with the symbols.</td>
</tr>
<tr>
<td>2.</td>
<td>If the number of leaves is &gt; ( 2^n - m ) then halt.*</td>
</tr>
<tr>
<td>3.</td>
<td>Find the leaf with highest probability and expand it to have ( m ) children.** Go to 2.</td>
</tr>
</tbody>
</table>

* 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

- $P(a) = 0.7$, $P(b) = 0.2$, $P(c) = 0.1$
- $n = 3$
Example

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

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>.343</td>
<td>.2</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>.14</td>
<td>.07</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>.098</td>
<td>.049</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>.2</td>
<td>.1</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>.343</td>
<td>.098</td>
<td>.049</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>.14</td>
<td>.07</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>ac</td>
</tr>
<tr>
<td>.101</td>
<td>.100</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>101</td>
<td>110</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aac</td>
<td>aab</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aaa</td>
<td>000</td>
</tr>
</tbody>
</table>

CSEP 590 - Lecture 1 - Autumn 2007
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

```
ab    .14   011
ac    .07   010
b     .2    101
c     .1    110
```

ABR = $3 \left[ \frac{1}{3} (0.343 + 0.098 + 0.049) + 2 (0.14 + 0.07) + 0.2 + 0.1 \right]$

= 1.37 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.