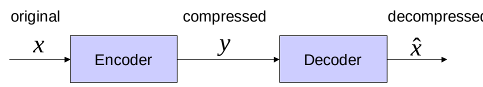
CSE 521
Algorithms
Spring 2003

Entropy
Arithmetic Coding

Basic Data Compression Concepts



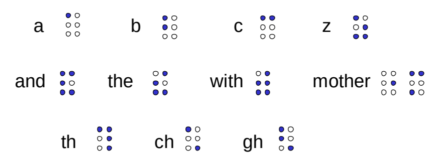
- Lossless compression $x = \hat{x}$
  - Also called entropy coding, reversible coding.
- Lossy compression $x \neq \hat{x}$
  - Also called irreversible coding.
- Compression ratio = $|x|/|y|$
  - $|x|$ is number of bits in x.

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 binaries to fit on a floppy
- 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

Why is Data Compression Possible

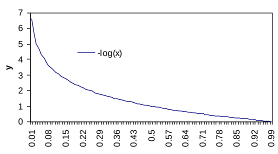
- Most data from nature has redundancy
  - There is more data than the actual information contained in the data.
  - Squeezing out the excess data amounts to compression.
  - However, unsqeezing out is necessary to be able to figure out what the data means.
- Information theory is needed to understand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

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
  - Suppose a "t" is received. Given English, the next symbol being a "z" has very low probability, the next symbol being a "h" has much higher probability. Receiving a "z" has much more information in it than receiving a "h". We already knew it was more likely we would receive an "h".

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.
  - $P(a_1) + P(a_2) + \ldots + P(a_m) = 1$
- $\inf(a_i) = -\log_2 P(a_i)$ 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
  - $\inf(a) = -\log_2(1/8) = 3$
  - $\inf(b) = -\log_2(1/4) = 2$
  - $\inf(c) = -\log_2(5/8) = .678$
- 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 $\{a_1, a_2, \ldots, a_m\}$.

$$H = -\sum_{i=1}^{m} P(a_i)\log_2(P(a_i))$$

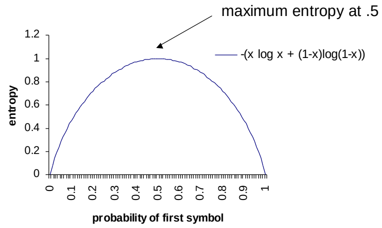
- H is the average number of bits required to code up a symbol, given all we know is the probability distribution of the symbols.
- H is the Shannon lower bound on the average number of bits to code a symbol in this "source model".
- Stronger models of entropy include context. We'll talk about this later.

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 bits/symbol
- {a, b, c} with a 1/3, b 1/3, c 1/3. (worst case)
  - $H = -3^* (1/3)^*\log_2(1/3) = 1.6$ bits/symbol
- {a, b, c} with a 1, b 0, c 0 (best case)
  - $H = -1^*\log_2(1) = 0$
- Note that the standard coding of 3 symbols takes 2 bits.

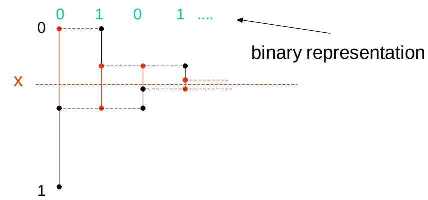
Entropy Curve

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



Reals in Binary

- Any real number x in the interval [0,1) can be represented in binary as $.b_1b_2...$ where b_i is a bit.



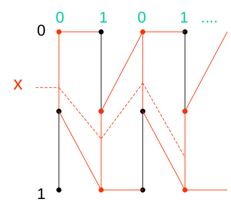
First Conversion

```
L := 0; R :=1; i := 1
while x > L *
   if x < (L+R)/2 then b_i := 0 ; R := (L+R)/2;
   if x ≥ (L+R)/2 then b_i := 1 ; L := (L+R)/2;
   i := i + 1
end{while}
b_j := 0 for all j ≥ i
```

* Invariant: x is always in the interval [L,R)

Conversion using Scaling

- Always scale the interval to unit size, but x must be changed as part of the scaling.



Binary Conversion with Scaling

```
y := x; i := 0
while y > 0 *
    i := i + 1;
    if y < 1/2 then b_i := 0; y := 2y;
    if y ≥ 1/2 then b_i := 1; y := 2y – 1;
end{while}
b_j := 0 for all j ≥ i + 1
```

* Invariant: $x = .b_1b_2 ... b_i + y/2^i$

Proof of the Invariant

- Initially $x = 0 + y/2^0$
- Assume $x = .b_1b_2 ... b_i + y/2^i$
  - Case 1. $y < 1/2$. $b_{i+1} = 0$ and $y' = 2y$
    $.b_1b_2 ... b_i b_{i+1} + y'/2^{i+1} = .b_1b_2 ... b_i 0 + 2y/2^{i+1}$
    $= .b_1b_2 ... b_i + y/2^i$
    $= x$
  - Case 2. $y \geq 1/2$. $b_{i+1} = 1$ and $y' = 2y – 1$
    $.b_1b_2 ... b_i b_{i+1} + y'/2^{i+1} = .b_1b_2 ... b_i 1 + (2y-1)/2^{i+1}$
    $= .b_1b_2 ... b_i + 1/2^{i+1} + 2y/2^{i+1} - 1/2^{i+1}$
    $= .b_1b_2 ... b_i + y/2^i$
    $= x$

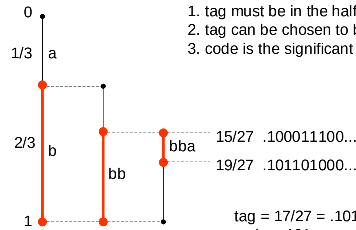
Arithmetic Coding

Basic idea in arithmetic coding:

- represent each string x of length n by a unique interval [L,R) in [0,1).
- The width r-l of the interval [L,R) represents the probability of x occurring.
- The interval [L,R) can itself be represented by any number, called a tag, within the half open interval.
- The k significant bits of the tag $.t_1t_2t_3\ldots$ is the code of x. That is, . $.t_1t_2t_3\ldots t_k000\ldots$ is in the interval [L,R).

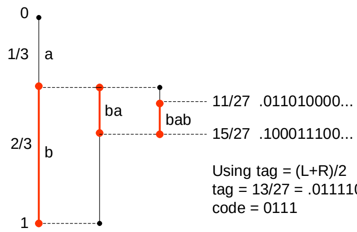
Example of Arithmetic Coding (1)

1. tag must be in the half open interval.
2. tag can be chosen to be (L+R)/2.
3. code is the significant bits of the tag.



tag = 17/27 = .101000010...
code = 101

Some Tags are Better than Others

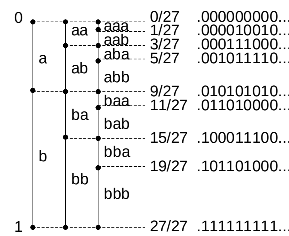


Using tag = (L+R)/2
tag = 13/27 = .011110110...
code = 0111

Alternative tag = 14/37 = .100001001...
code = 1

Example of Codes

- P(a) = 1/3, P(b) = 2/3.



tag = (L+R)/2	code	
.000001001...	0	aaa
.000100110...	0001	aab
.001001100...	001	aba
.010000101...	01	abb
.010111110...	01011	baa
.011110111...	0111	bab
.101000010...	101	bba
.110110100...	11	bbb

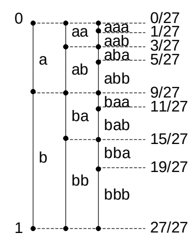
.95 bits/symbol
.92 entropy lower bound

Code Generation from Tag

- If binary tag is $.t_1t_2t_3\ldots = (L+R)/2$ in [L,R) then we want to choose k to form the code $t_1t_2\ldots t_k$.
- Short code:
  - choose k to be as small as possible so that $L \leq .t_1t_2\ldots t_k000\ldots < R$.
- Guaranteed code:
  - choose $k = \lceil \log_2 (1/(R-L)) \rceil + 1$
  - $L \leq .t_1t_2\ldots t_kb_1b_2b_3\ldots < R$ for any bits $b_1b_2b_3\ldots$
  - for fixed length strings provides a good prefix code.
  - example: [.000000000..., .000010010...), tag = .000001001...
    Short code: 0
    Guaranteed code: 000001

Guaranteed Code Example

- P(a) = 1/3, P(b) = 2/3.



tag = (L+R)/2	short code	Prefix code	
.000001001...	0	0000	aaa
.000100110...	0001	0001	aab
.001001100...	001	001	aba
.010000101...	01	0100	abb
.010111110...	01011	01011	baa
.011110111...	0111	0111	bab
.101000010...	101	101	bba
.110110100...	11	11	bbb

Arithmetic Coding Algorithm

- $P(a_1), P(a_2), \ldots, P(a_m)$
- $C(a_i) = P(a_1) + P(a_2) + \ldots + P(a_{i-1})$
- Encode $x_1x_2 \ldots x_n$

```
Initialize L := 0 and R:= 1;
for i = 1 to n do
   W := R - L;
    L := L + W * C(x_i);
    R := L + W * P(x_i);
t := (L+R)/2;
choose code for the tag
```


Arithmetic Coding Example

- P(a) = 1/4, P(b) = 1/2, P(c) = 1/4
- C(a) = 0, C(b) = 1/4, C(c) = 3/4
- abca

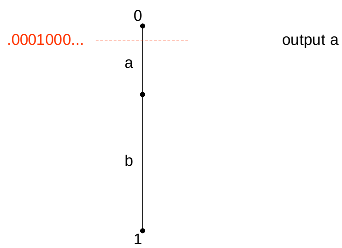
W := R - L;
L := L + W C(x);
R := L + W P(x)

symbol	W	L	R
		0	1
a	1	0	1/4
b	1/4	1/16	3/16
c	1/8	5/32	6/32
a	1/32	5/32	21/128

tag = (5/32 + 21/128)/2 = 41/256 = .001010010...
L = .001010000...
R = .001010100...
code = 00101
prefix code = 00101001

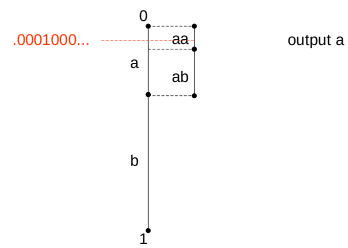
Decoding (1)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...



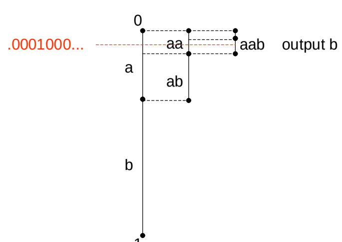
Decoding (2)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...



Decoding (3)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...



Arithmetic Decoding Algorithm

- $P(a_1), P(a_2), \ldots, P(a_m)$
- $C(a_i) = P(a_1) + P(a_2) + \ldots + P(a_{i-1})$
- Decode $b_1b_2 \ldots b_m$, number of symbols is n.

```
Initialize L := 0 and R := 1;
t := .b_1b_2...b_m000...
for i = 1 to n do
   W := R - L;
    find j such that L + W * C(a_j) ≤ t < L + W * (C(a_j)+P(a_j))
    output a_j;
    L := L + W * C(a_j);
    R := L + W * P(a_j);
```

Decoding Example

- P(a) = 1/4, P(b) = 1/2, P(c) = 1/4
- C(a) = 0, C(b) = 1/4, C(c) = 3/4
- 00101

tag = .00101000... = 5/32

W	L	R	output
	0	1	
1	0	1/4	a
1/4	1/16	3/16	b
1/8	5/32	6/32	c
1/32	5/32	21/128	a

Decoding Issues

- There are two ways for the decoder to know when to stop decoding.
  1. Transmit the length of the string
  2. Transmit a unique end of string symbol

More Issues

- Avoiding real arithmetic and scaling
- Context
- Adaptive
- Comparison with Huffman coding

Scaling

- Scaling:
  – By scaling we can keep L and R in a reasonable range of values so that W = R – L does not underflow.
  – The code can be produced progressively, not at the end.
  – Complicates decoding some.

Scaling Principle

Lower half
If [L,R) is contained in [0,.5) then
L := 2L; R := 2R
output 0, followed by C 1's
C := 0.

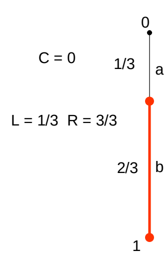
Upper half
If [L,R) is contained in [.5,1) then
L := 2L –1, R := 2R - 1
output 1, followed by C 0's
C := 0

Middle Half
If [L,R) is contained in [.25,.75) then
L := 2L –.5, R := 2R -.5
C := C + 1.

C keeps track of the number of bits needed when we learn which side of 1/2 the tag must be in.

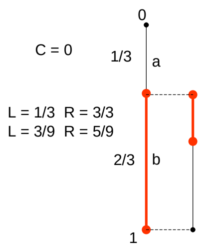
Example

- <u>b</u>aa



Example

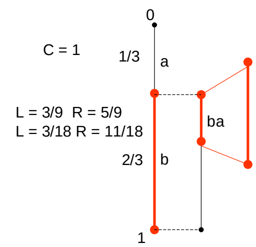
- b<u>a</u>a



Scale middle half

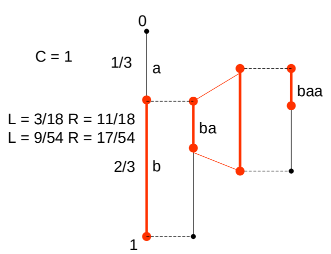
Example

- ba<u>a</u>



Example

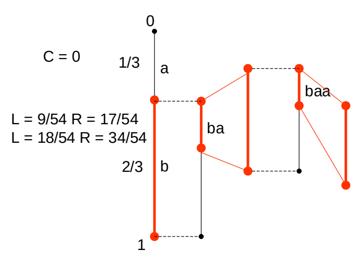
- ba<u>a</u>



Scale lower half

Example

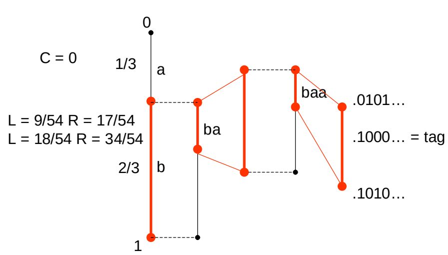
- ba<u>a</u> <u>01</u>



Example

- baa 01<u>1</u>

In end L < ½ < R, choose tag to be 1/2



Integer Implementation

- m bit integers
  - Represent 0 with 000…0 (m times)
  - Represent 1 with 111…1 (m times)
- Probabilities represented by frequencies
  - n_i is the number of times that symbol a_i occurs
  - $C_i = n_1 + n_2 + \ldots + n_{i-1}$
  - $N = n_1 + n_2 + \ldots + n_m$

$$W := R - L + 1$$

$$L' := L + \left\lfloor \frac{W \cdot C_i}{N} \right\rfloor$$

$$R := L + \left\lfloor \frac{W \cdot C_{i+1}}{N} \right\rfloor - 1$$

$$L := L'$$

Coding the i-th symbol using integer calculations. Must use scaling!

Context

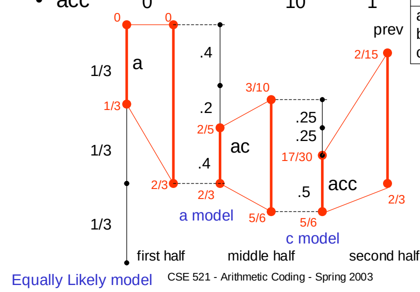
- Consider 1 symbol context.
- Example: 3 contexts.

next

prev	a	b	c
a	.4	.2	.4
b	.1	.8	.1
c	.25	.25	.5

Example with Scaling

- acc 0 10 1



next

prev	a	b	c
a	.4	.2	.4
b	.1	.8	.1
c	.25	.25	.5

Code = 0101

Arithmetic Coding with Context

- Maintain the probabilities for each context.
- For the first symbol use the equal probability model
- For each successive symbol use the model for the previous symbol.

Adaptation

- Simple solution – Equally Probable Model.
  - Initially all symbols have frequency 1.
  - After symbol x is coded, increment its frequency by 1
  - Use the new model for coding the next symbol
- Example in alphabet a,b,c,d

		a	a	b	a	a	c
a	1	2	3	3	4	5	5
b	1	1	1	2	2	2	2
c	1	1	1	1	1	1	2
d	1	1	1	1	1	1	1

After aabaac is encoded
The probability model is
a 5/10 b 2/10
c 2/10 d 1/10

Zero Frequency Problem

- How do we weight symbols that have not occurred yet.
  - Equal weights? Not so good with many symbols
  - Escape symbol, but what should its weight be?
  - When a new symbol is encountered send the <esc>, followed by the symbol in the equally probable model. (Both encoded arithmetically.)

		a	a	b	a	a	c
a	0	1	2	2	3	4	4
b	0	0	0	1	1	1	1
c	0	0	0	0	0	0	1
d	0	0	0	0	0	0	0
<esc>	1	1	1	1	1	1	1

After aabaac is encoded
The probability model is
a 4/7 b 1/7
c 1/7 d 0
<esc> 1/7

PPM

- Prediction with Partial Matching
  - Cleary and Witten (1984)
- State of the art arithmetic coder
  - Arbitrary order context
  - The context chosen is one that does a good prediction given the past
  - Adaptive
- Example
  - Context "the" does not predict the next symbol "a" well. Move to the context "he" which does.

Arithmetic vs. Huffman

- Both compress very well. For m symbol grouping.
  - Huffman is within 1/m of entropy.
  - Arithmetic is within 2/m of entropy.
- Context
  - Huffman needs a tree for every context.
  - Arithmetic needs a small table of frequencies for every context.
- Adaptation
  - Huffman has an elaborate adaptive algorithm
  - Arithmetic has a simple adaptive mechanism.
- Bottom Line – Arithmetic is more flexible than Huffman.

Applications of Arithmetic Coding

- JPEG 2000
  - Image compression
  - Wavelet transform
  - Bit-planes of the transformed image is adaptively arithmetic coded.
  - Contexts relate to structure of wavelet coefficients
- JBIG
  - Binary image compression
  - Context is about 10 nearby pixels already coded.