ARITHMETIC CODING

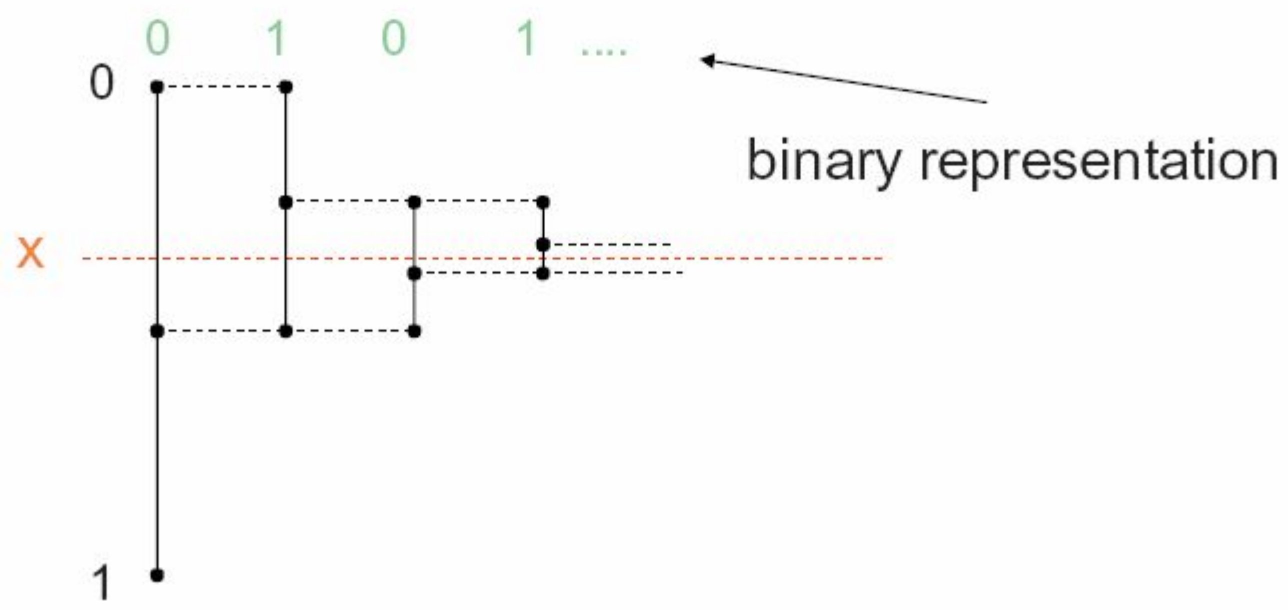
Prof. Pintu Chauhan, Assistant Professor
Information Technology Engineering

CHAPTER-6

Arithmetic Coding

Representation of Real Numbers 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.



Real-To-Binary Conversion Algortihm

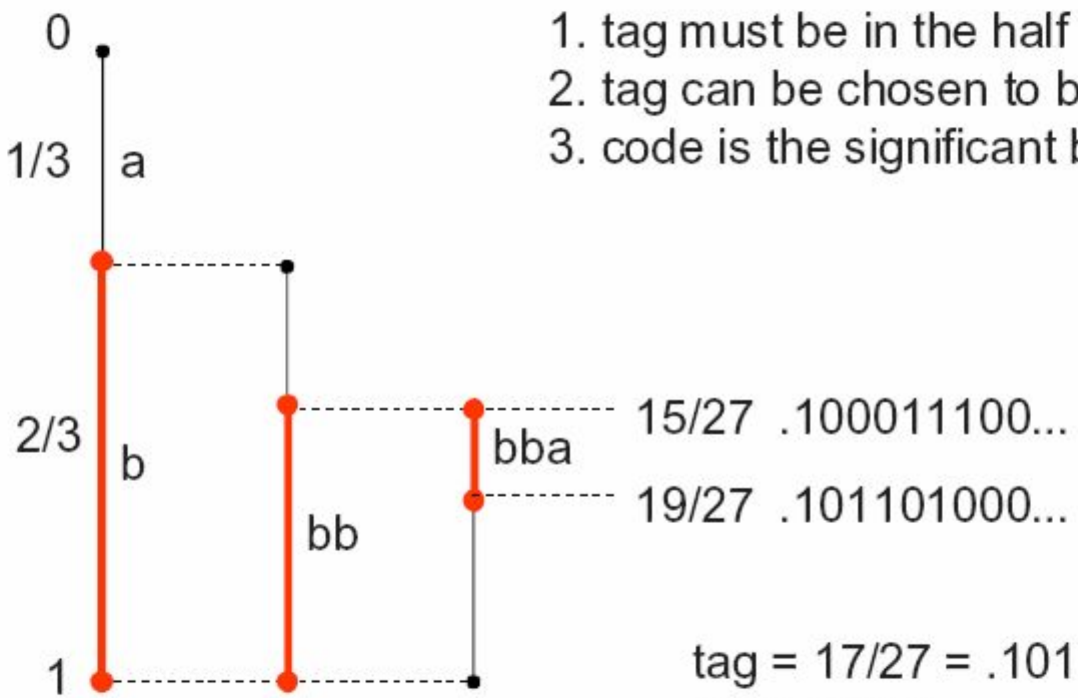
$L := 0$; $R := 1$; $i := 1$
while $x > L$ *
    if $x < (L+R)/2$ then $b_i := 0$; $R := (L+R)/2$;
    if $x \geq (L+R)/2$ then $b_i := 1$; $L := (L+R)/2$;
    $i := i + 1$
end{while}
$b_j := 0$ for all $j \geq i$

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

Arithmetic Coding

- Basic idea in arithmetic coding (Shannon-Fano- Elias):
  - 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 $.t_1t_2t_3$... is the code of x. That is, . $.t_1t_2t_3...t_k000...$ is in the interval [L,R).

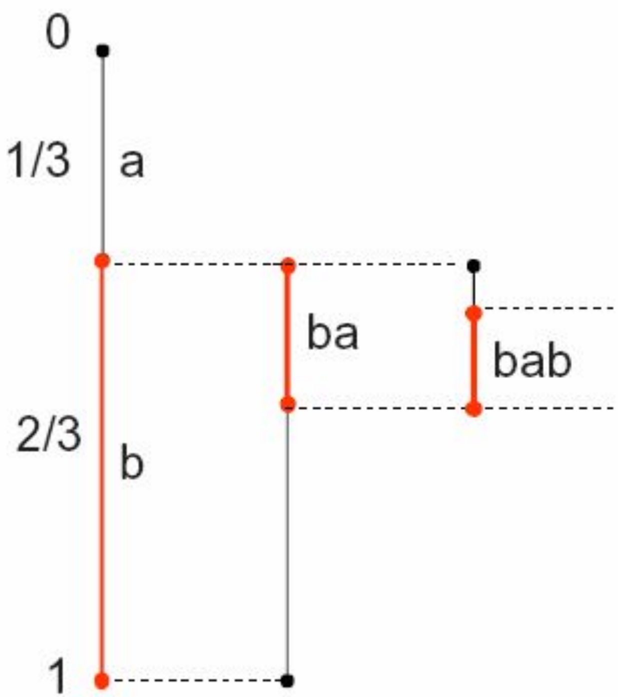
Example of Arithmetic Coding



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



11/27 .011010000...

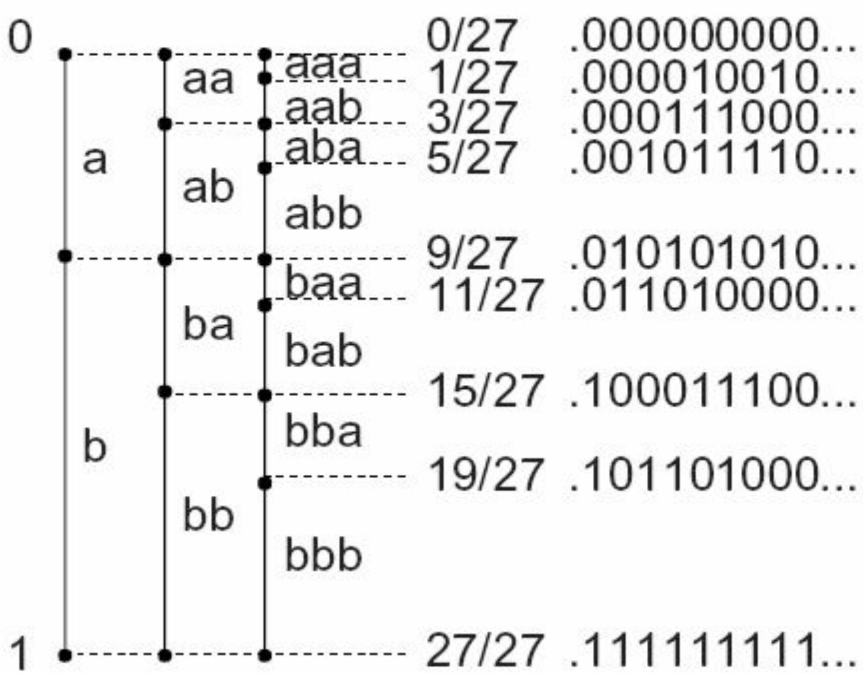
15/27 .100011100...

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

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

Example

- 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 Tags

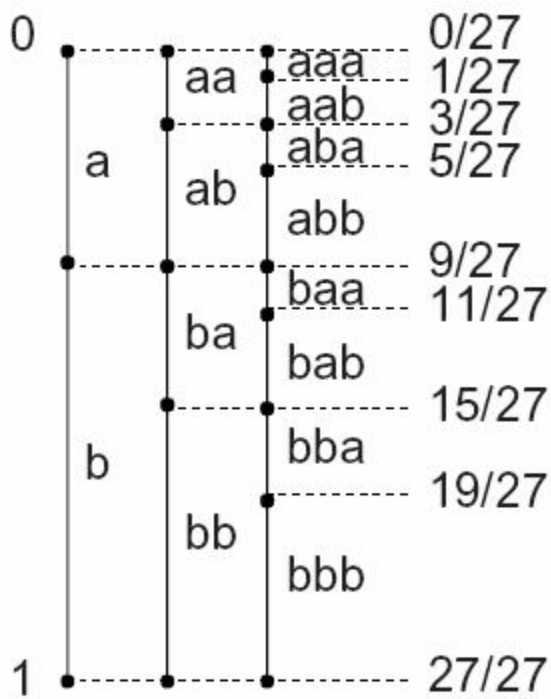
- If binary tag is $t_1t_2t_3\ldots.\ = (L+R)/2\ in\ [L,\ R)$ then we want to choose k form the code $t_1t_2\ \ldots.\ t_k$.

- Short code :
  - Choose k to be as small as possible so that
    $L\ \leq\ .t_1t_2\ \ldots.\ t_k000\ldots\ldots\ < R$

- Guarantee Code :
  - Choose $k = \lceil \log_2 (1/(R-L)) \rceil + 1$
  - $L\ \leq\ .t_1t_2\ \ldots.\ t_k\, b_1b_2b_3\ldots\ldots\ < R$ for any bits $b_1b_2b_3\ldots$
  - For fixed length strings provides a god prefix code.
  - Example: $[.0000000\ldots.\ ,\ .000010010\ldots),\ tag = .000001001\ldots.$
    short code : 0
    Guaranteed code: 000001

Guarantee 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

$C(x_i) = P(x_0) + P(x_1) + \ldots P(x_i)$

```
Initialize L: = 0 and R: = 1;
For i = 1 to n do
        W:= R – L;
        L: = L + W*C(x_{i-1});
        R := L + W*C(x_i);
T: = (L+R)/2;
Choose code for the tag
```

Example

P(A) = ¼, P(b) = ½ , P(c) = ¼

C(a) = ¼, C(b) = ¾, C(c) = 1

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

Example

P(A) = ¼, P(b) = ½ , P(c) = ¼

C(a) = ¼, C(b) = ¾, C(c) = 1

bbbb

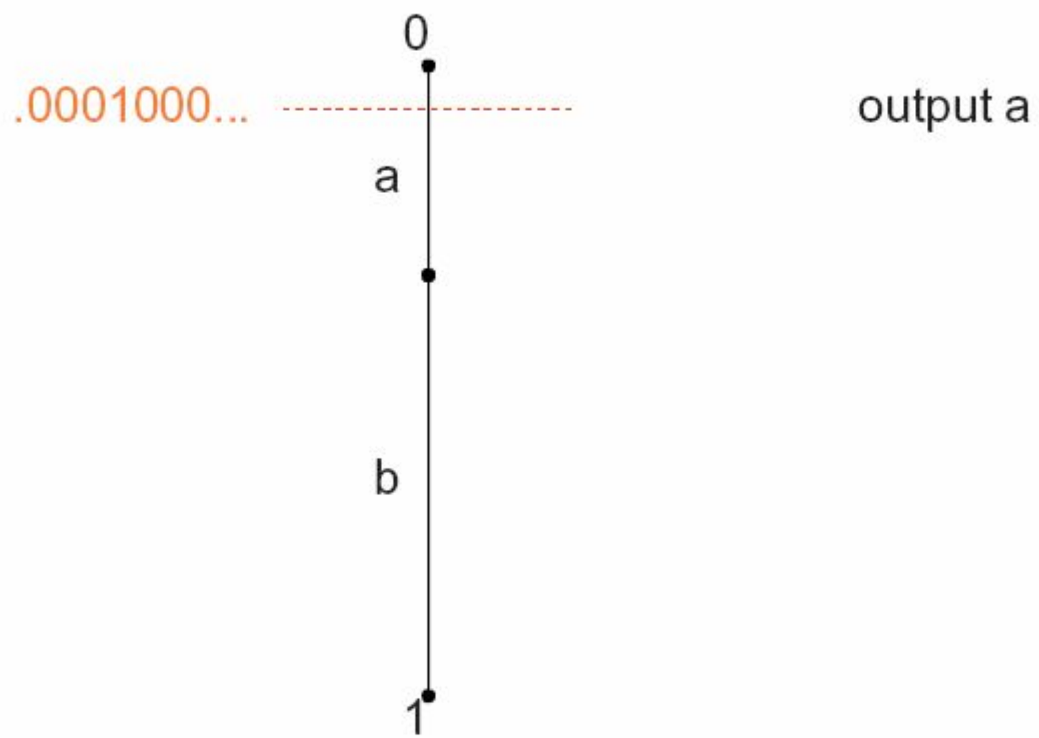
W := R - L;
L := L + W C(x);
R := L + W P(x)

symbol	W	L	R
		0	1
b	1		
b			
b			
b			

tag =
L =
R =
code =
prefix code =

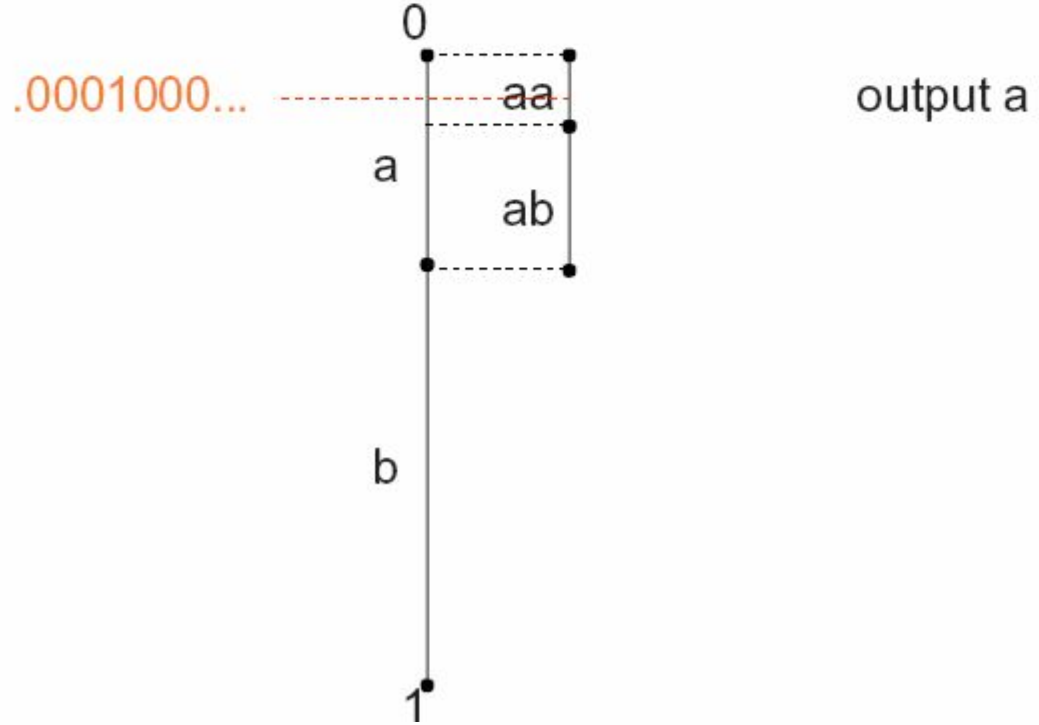
Decoding

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



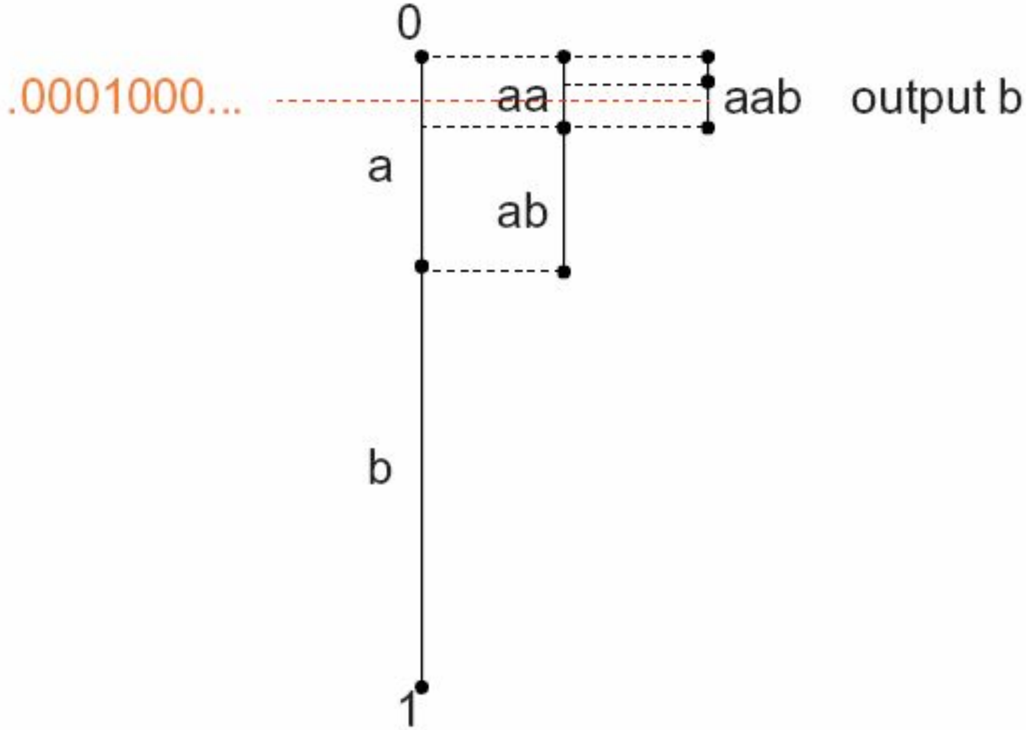
Decoding

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



Decoding

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



Arithmetic Decoding Algortihm

$C(x_i) = P(x_0) + P(x_1) + \dots P(x_i)$

Decode $b_1 b_2 \dots b_m$, the number of symbols in n

Initialize L:= 0 and R := 1;
t:= $.b_1 b_2 \dots b_m$ For i =

1 to n do

W := R-L;
Find j such that $L + W*C(x_{j-1}) <= t < L + W*C(x_j)$ Output x_j;

$L := L + W*C(x_{j-1})$;

$R := L + W*C(x_j)$;

Decoding Example

P(a) = ¼, P(b) = ½ , P(c) = ¼

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

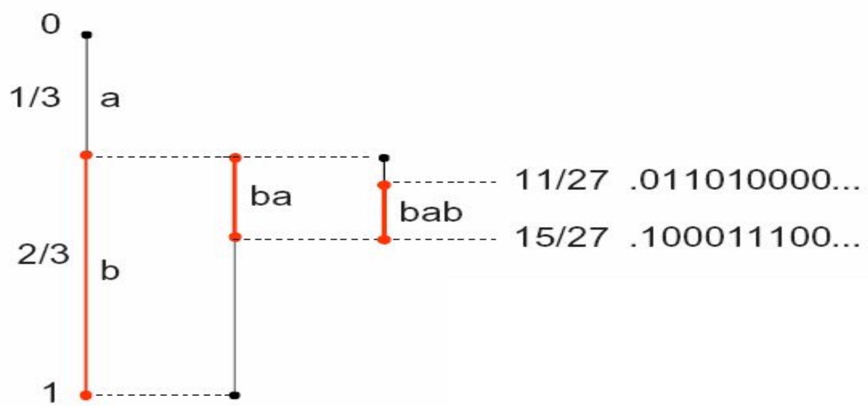
- There are two ways for the decoder to know when to stop decoding.
  1. Transmit the length of the string
  2. Transmit a unique end of string symbol

Practical Arithmetic Coding

- 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.
- Integer arithmetic coding avoids floating point all together.

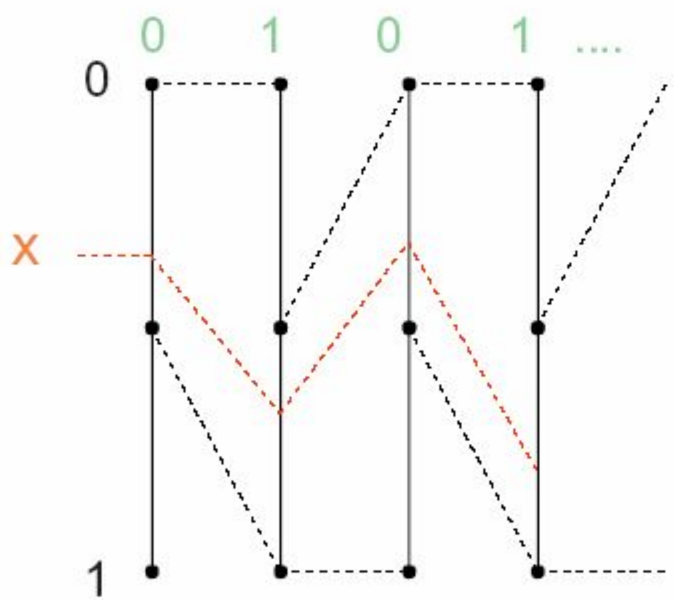
Issues with Arithmetic Coding

- The intervals are getting smaller as the sequence of symbols is getting longer.
- Arithmetic's (computations) on very small numbers results in underflow!
- Need to rescale at every step!



Representation of Real Number in Binary

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



Binary Conversion with Scaling

$y := x; \;\; i := 0$
$while\; y > 0^*$
$\qquad i := i + 1;$
$\qquad if\; y < 1/2 \;\; then\; b_i \;:= 0; y := 2y;$
$\qquad if\; y \geq 1/2 \;\; then\; b_i \;:= 1; y := 2y - 1;$
$end\{while\}$
$b_j \;:= 0\; for\; all\;\; j \geq i + 1;$

*** Invariant :** $\boldsymbol{x = .\, b_1 b_2 \ldots .\, b_i + y/2^i}$

Proof of Invariant

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

$$
\begin{aligned}
.b_1b_2 \ldots b_i\, b_{i+1} + y'/2^{i+1} &= .b_1b_2 \ldots b_i\, 0 + 2y/2^{i+1} \\
&= .b_1b_2 \ldots b_i + y/2^i \\
&= x
\end{aligned}
$$

  - Case 2. $y \geq 1/2$. $b_{i+1} = 1$ and $y' = 2y - 1$

$$
\begin{aligned}
.b_1b_2 \ldots b_i\, b_{i+1} + y'/2^{i+1} &= .b_1b_2 \ldots b_i\, 1 + (2y-1)/2^{i+1} \\
&= .b_1b_2 \ldots b_i + 1/2^{i+1} + 2y/2^{i+1} - 1/2^{i+1} \\
&= .b_1b_2 \ldots b_i + y/2^i \\
&= x
\end{aligned}
$$

Excercise

$x = 1/3$

y	i	b
1/3	1	0
2/3	2	1
1/3	3	0
2/3	4	1
...	...	...

$x = 17/27$

y	i	b
17/27	1	1

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 Algorithm for Arithmetic Coding

- **Lower Half**

  $while\ [L, R) is\ contained\ in\ [0, .5) then$

  $L := 2L\ ; R := 2R$

  $output\ 0,\ followed\ by\ C\ 1's$

  $C := 0.$

- **Upper Half**

  $while\ [L, R) is\ contained\ in\ [.5, 1) then$

  $L := 2L - 1\ ; R := 2R - 1$

  $output\ 1,\ followed\ by\ C\ 0's$

  $C := 0.$

- **Lower Half**

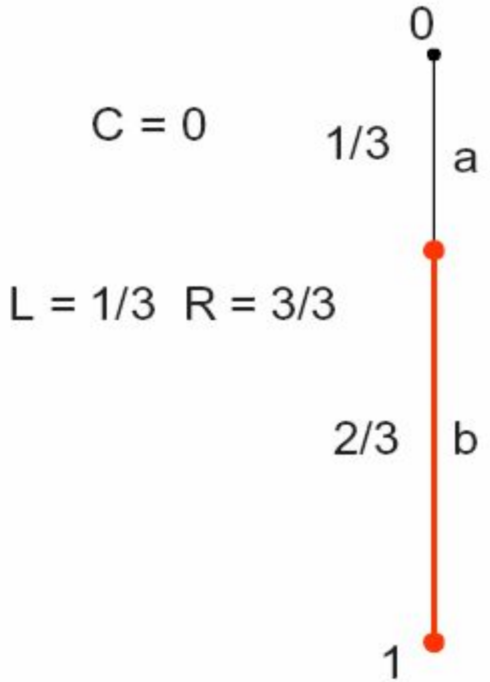
  $while\ [L, R) is\ contained\ in\ [.25, .75) then$

  $L := 2L - .5\ ; R := 2R - .5$

  $C := C + 1.$

Example

- baa

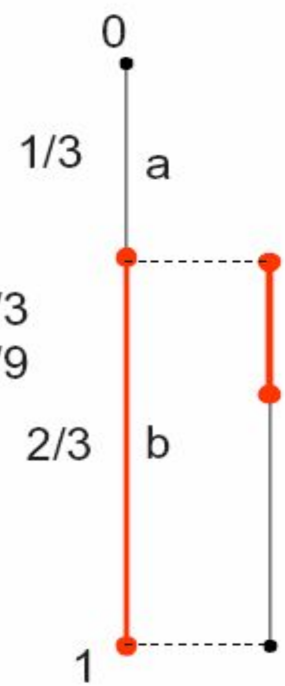


Example

- baa

C = 0

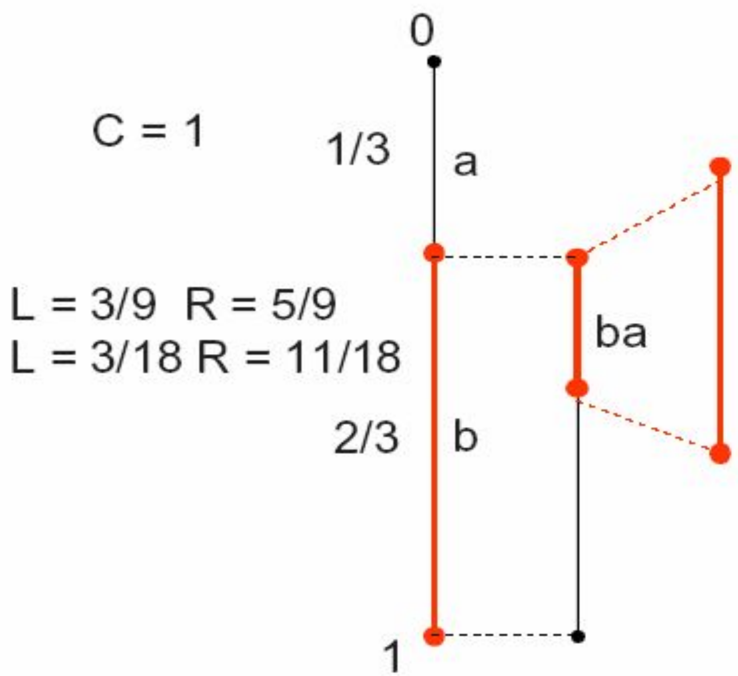
L = 1/3 R = 3/3
L = 3/9 R = 5/9



Scale middle half

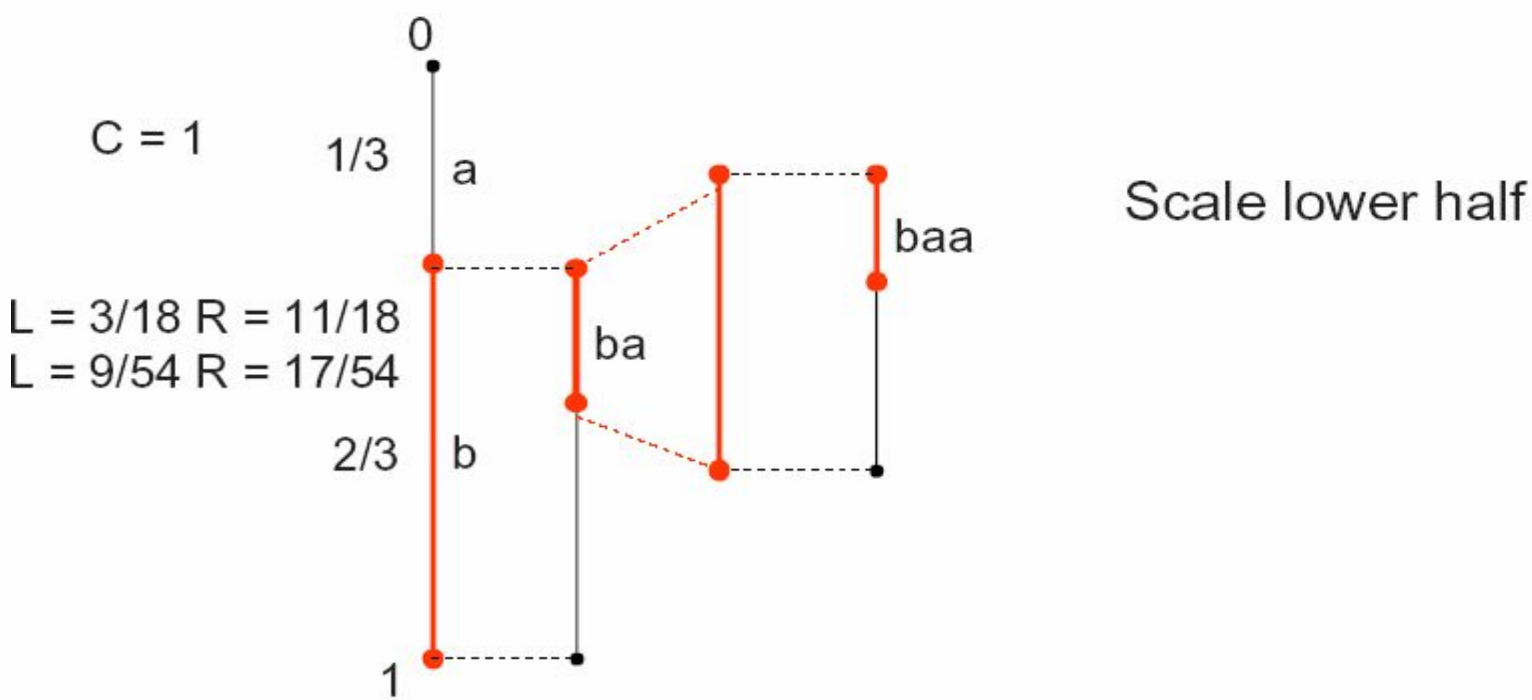
Example

- baa



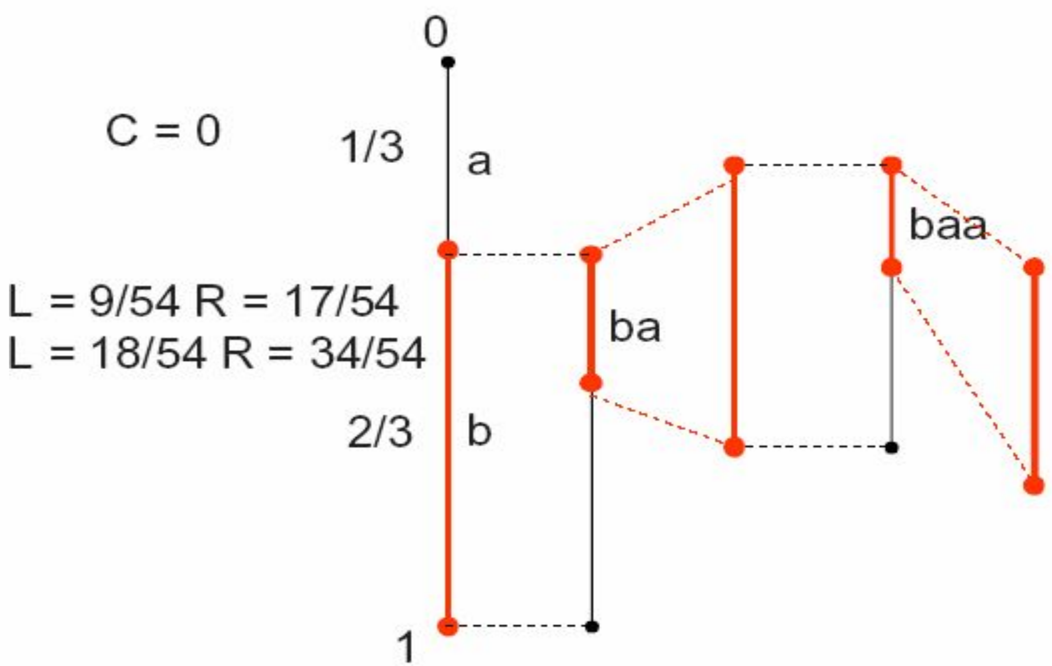
Example

- baa



Example

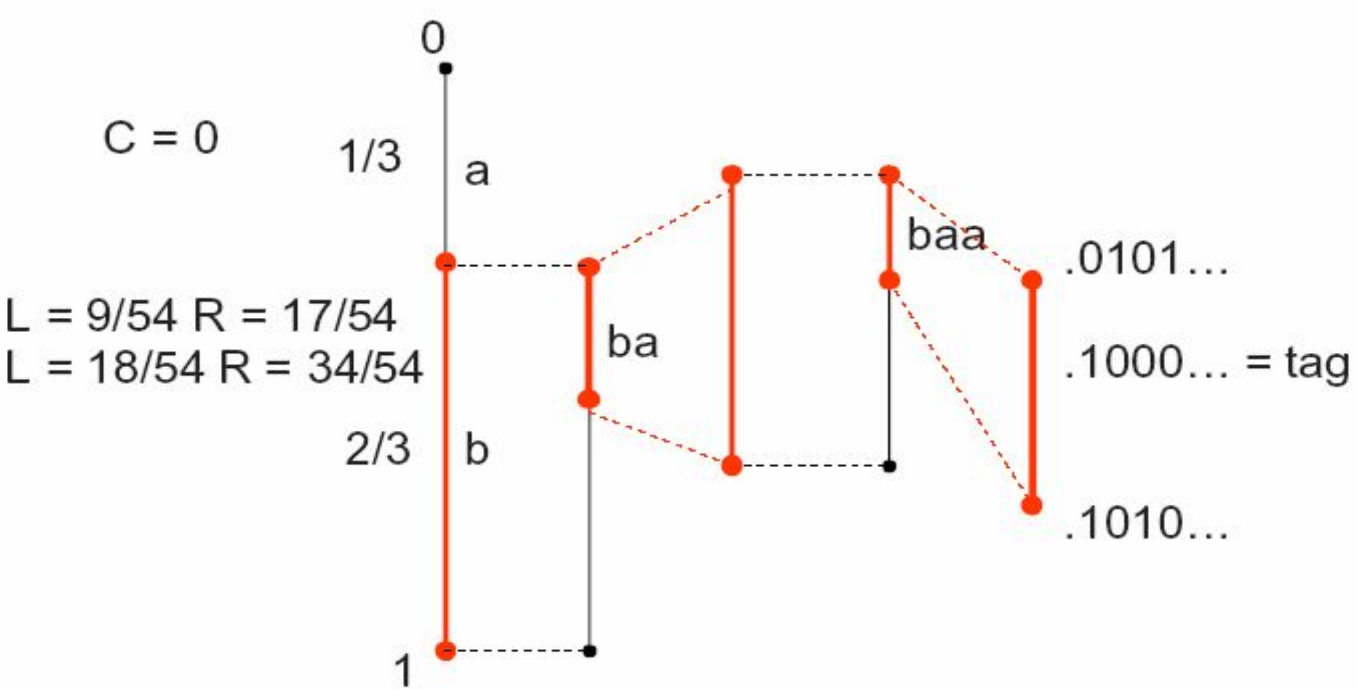
- baa 01



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!

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.

Arithmetic Coding with Context

- 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

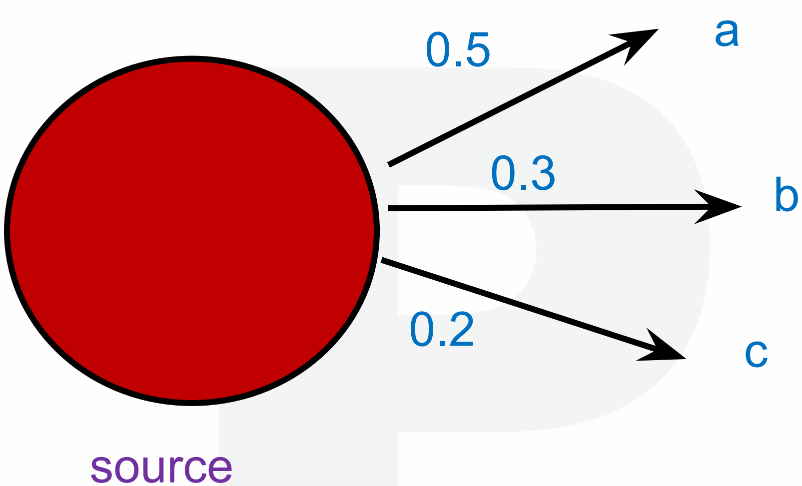
Arithmetic Coding with Context

- 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.

CHAPTER-7

Dictionary Coding

Review of Entropy Coding



Minimize the number of bits to code a, b, c based on the statistical properties of the source

Dictionary Coding

Encoder codes the index

index	pattern
1	a
2	b
3	ab
…	
n	abc

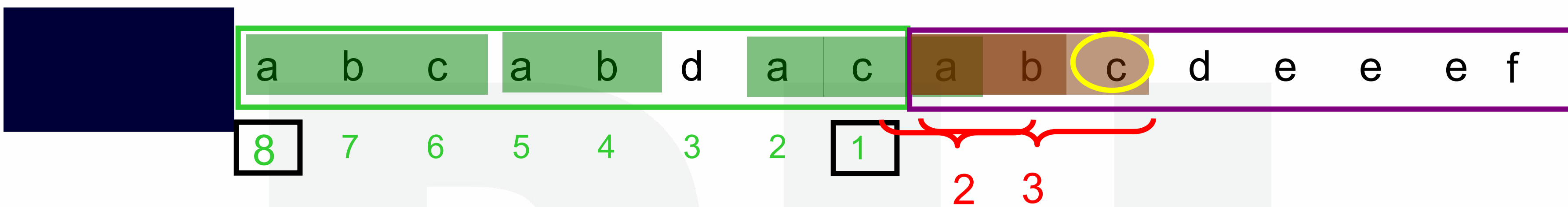
Encoder $\xrightarrow{\text{Indices}}$ Decoder

Both encoder and decoder are assumed to have the same dictionary (table)

Ziv-Lempel Coding (ZL or LZ)

- Named after J. Ziv and A. Lempel (1977).
- Adaptive dictionary technique.
  - Store previously coded symbols in a buffer.
  - Search for the current sequence of symbols to code.
  - If found, transmit buffer offset and length.

LZ77



Output triplet <offset, length, next>

Transmitted to decoder: 8 3 d 0 0 e 1 2 f

If the size of the search buffer is *N* and the size of the alphabet is *M* we need

$$\lceil \log(N+1) \rceil + \lceil \log(N+1) \rceil + \lceil \log M \rceil$$

bits to code a triplet.

Variation: Use a VLC to code the triplets!

PKZip, Zip, Lharc, PNG, gzip, ARJ

Drawback with LZ77

- Repetetive patterns with a period longer than the search buffer size are not found.

- If the search buffer size is 4, the sequence
  a b c d e a b c d e a b c d e a b c d e …
  will be expanded, not compressed.

LZ78

- Store patterns in a dictionary
- Transmit a tuple <dictionary index, next>

LZ78

a b c a b a b c

Output tuple <dictionary index, next>

Transmitted to decoder: 0 a 0 b 0 c 1 b 4 c

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

Strategy needed for limiting dictionary size!

LZW

- Modification to LZ78 by Terry Welch, 1984.
- Applications: GIF, v42bis
- Patented by UniSys Corp.
- Transmit only the dictionary index.
- The alphabet is stored in the dictionary in advance.

LZW

Input sequence:

a	b	c	a	b	a	b	c

Output: dictionary index

Transmitted:

1	2	3	4	5

Decoded:

a	b	c	a b	a b

1	a	6	bc
2	b	7	ca
3	c	8	aba
4	d	9	abc
5	a b		

Encoder dictionary:

1	a	6	bc
2	b	7	ca
3	c	8	aba
4	d		
5	a b		

Decoder dictionary:

GIF

- CompuServe Graphics Interchange Format (1987, 89).
- Features:
  - Designed for up/downloading images to/from BBSes via PSTN.
  - 1-, 4-, or 8-bit *colour palettes*.
  - Interlace for *progressive decoding* (four passes, starts with every 8th row).
  - *Transparent colour* for non-rectangular images.
  - Supports multiple images in one file (”animated GIFs”).

GIF : Method

- Compression by LZW.
- Dictionary size 2^{b+1} 8-bit symbols
  - b is the number of bits in the palette.
- Dictionary size doubled if filled (max 4096).
- Works well on computer generated images.

Parul® University

DIGITAL LEARNING CONTENT

GIF : Problems

- Unsuitable for natural images (photos):
    - Maximum 256 colors () bad quality).
    - Repetetive patterns uncommon () bad compression).
- LZW patented by UniSys Corp.
- Alternative: PNG

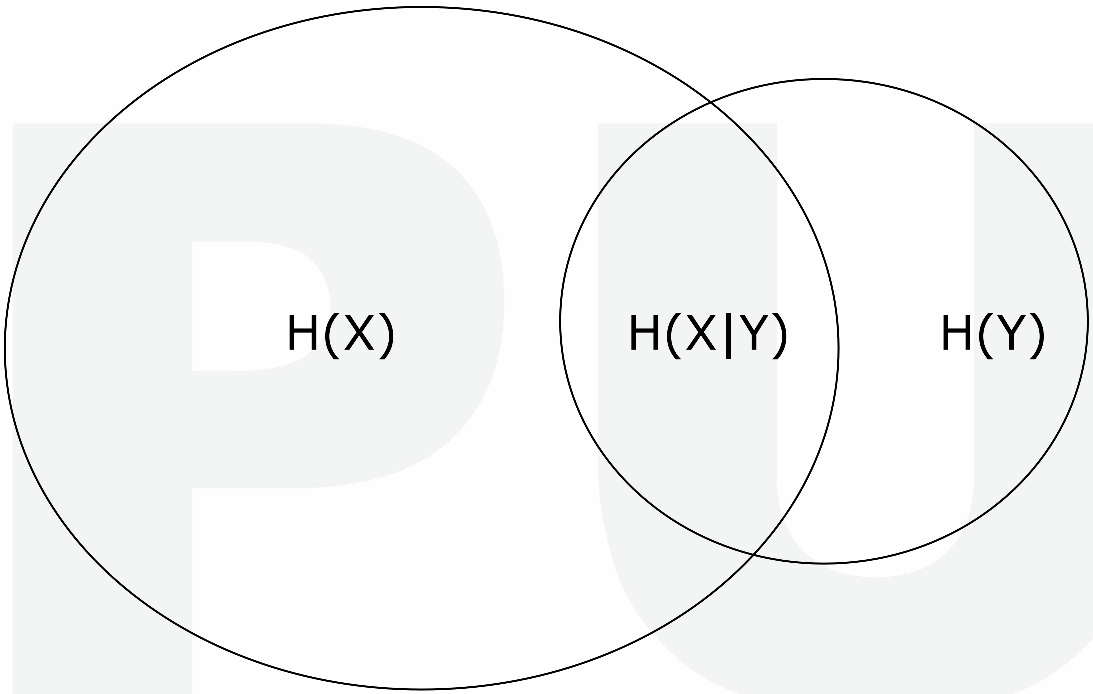
PNG : Portable Network Graphics

- Designed to replace GIF.
- Some features:
  - Indexed *or* true-colour images (· 16 bits per plane).
  - Alpha channel.
  - Gamma information.
  - Error detection.
- No support for multiple images in one file.
  - Use MNG for that.
- Method:
  - Compression by LZ77 using a 32KB search buffer

CHAPTER-6

Context Coding

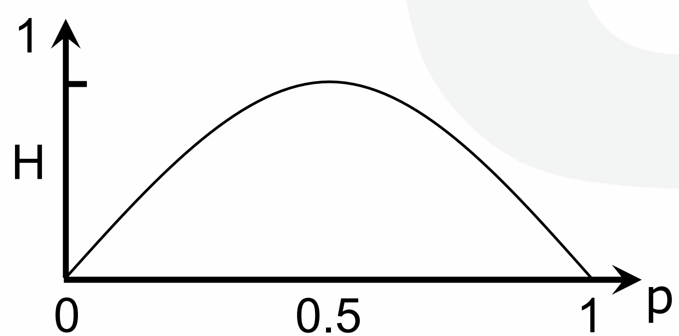
Context Coding



H(X|Y) <= H(X) ⟶ H(X|Y) takes fewer bits to code than H(X)

Context Coding

- The distribution of the next symbol based on the current context (past symbols) is skewed.
  - Next symbol is likely to be a certain alphabet the than others.
  - Less information
  - Easier to code.



Context Coding

- From information theory – The lower the information, the fewer bits are needed to code the symbol.

$$\inf(a) = \log_2\left(\frac{1}{P(a)}\right)$$

- Examples:
  - P(a) = 1023/1024, inf(a) = .000977
  - P(a) = 1/2, inf(a) = 1
  - P(a) = 1/1024, inf(a) = 10

Review of Entropy

- Entropy is the expected number of bit to code a symbol in the model with a_i having probability $P(a_i)$.

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

- Good coders should be close to this bound.
  - Arithmetic
  - Huffman
  - Golomb
  - Tunstall

Problem with Context Coding

- Context explosion!
  - Suppose we want to use 5-letter context to predict the next letter in an English paragraph.
  - Number of contexts = 24^{5}.
  - No storage for contexts.
  - Speed

Which Context to Use ?

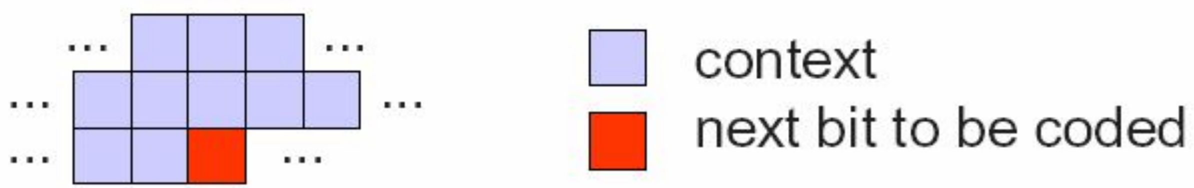
- Using previous table, which context for italicized letter?
  - “We pulled a he*a*vy wagon.”
  - “The the*a*tre was fun.”
  - “’Twas the*e*re haus!”

PPM- Prediction with Partial Matching

- Cleary and Witten (1984)
- Uses only current contexts (not all possible contexts)
- Uses arithmetic coding to code the context

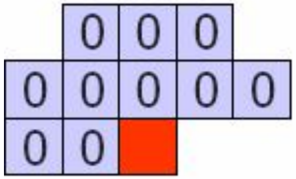
JBIG

- Coder for binary images
  - documents
  - graphics
- Codes in scan line order using context from the same and previous scan lines.



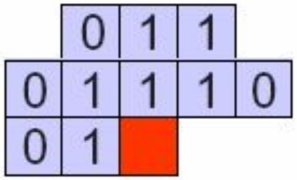
- Uses adaptive arithmetic coding with context.

JBIG Example



next bit	0	1
frequency	100	10

$$H = \frac{10}{110}\log(\frac{110}{10}) + \frac{100}{110}\log(\frac{110}{100}) = .44$$



next bit	0	1
frequency	15	50

$$H = \frac{15}{65}\log(\frac{65}{15}) + \frac{50}{65}\log(\frac{65}{50}) = .78$$

Issue with Context

- Context dilution
  - If there are too many contexts then too few symbols are coded in each context, making them in effective because of the zero-frequency problem.
- Context saturation
  - If there are too few contexts then the contexts might not be as good as having more context
- Wrong context
  - poor predictors.

Burrows – Wheeler Transform

- Burrows-Wheeler (1994)
- BW Transform creates a representation of the data which has a small working set.
- The transformed data is compressed with move to front compression.
- The decoder is quite different from the encoder.
- The algorithm requires processing the entire string at once (it is not on-line).
- It is a remarkably good compression method.

Encoding Example

- abracadabra

1. Create all cyclic shifts of the string.

0	abracadabra
1	bracadabraa
2	racadabraab
3	acadabraabr
4	cadabraabra
5	adabraabrac
6	dabraabraca
7	abraabracad
8	braabracada
9	raabracadab
10	aabracadabr

Encoding Example

2. Sort the strings alphabetically in to array A

			A	
0	abracadabra		0	aabracadabr
1	bracadabraa		1	abraabracad
2	racadabraab		2	abracadabra
3	acadabraabr		3	acadabraabr
4	cadabraabra	→	4	adabraabrac
5	adabraabrac		5	braabracada
6	dabraabraca		6	bracadabraa
7	abraabracad		7	cadabraabra
8	braabracada		8	dabraabraca
9	raabracadab		9	raabracadab
10	aabracadabr		10	racadabraab

Encoding Example

3. L = the last column

A	
0	aabracadabr
1	abraabracad
2	abracadabra
3	acadabraabr
4	adabraabrac
5	braabracada
6	bracadabraa
7	cadabraabra
8	dabraabraca
9	raabracadab
10	racadabraab

L = rdarcaaaabb

Encoding Example

4. Transmit X the index of the input in A and L (using move to front coding).

A

0	aabracadabr
1	abraabracad
2	abracadabra
3	acadabraabr
4	adabraabrac
5	braabracada
6	bracadabraa
7	cadabraabra
8	dabraabraca
9	raabracadab
10	racadabraab

L = rdarcaaaabb

X = 2

Why does BW Works ?

- Ignore decoding for the moment.
- The prefix of each shifted string is a context for the last symbol.
  - The last symbol appears just before the prefix in the original.
- By sorting, similar contexts are adjacent.
  - This means that the predicted last symbols are similar.

Decoding Examples ?

- We first decode assuming some information. We then show how compute the information.
- Let A^s be A shifted by 1

	A		A^s
0	aabracadabr	0	raabracadab
1	abraabracad	1	dabraabraca
2	abracadabra	2	aabracadabr
3	acadabraabr	3	racadabraab
4	adabraabrac	4	cadabraabra
5	braabracada	5	abraabracad
6	bracadabraa	6	abracadabra
7	cadabraabra	7	acadabraabr
8	dabraabraca	8	adabraabrac
9	raabracadab	9	braabracada
10	racadabraab	10	bracadabraa

Decoding Examples ?

- Assume we know the mapping T[i] is the index in A^s of the string i in A.
- T = [2 5 6 7 8 9 10 4 1 0 3]

A	
0	aabracadabr
1	abraabracad
2	abracadabra
3	acadabraabr
4	adabraabrac
5	braabracada
6	bracadabraa
7	cadabraabra
8	dabraabraca
9	raabracadab
10	racadabraab

A^s	
0	raabracadab
1	dabraabraca
2	aabracadabr
3	racadabraab
4	cadabraabra
5	abraabracad
6	abracadabra
7	acadabraabr
8	adabraabrac
9	braabracada
10	bracadabraa

Decoding Examples ?

- Let F be the first column of A, it is just L, sorted.

F =

0	1	2	3	4	5	6	7	8	9	10
a	a	a	a	a	b	b	c	d	r	r

T =

0	1	2	3	4	5	6	7	8	9	10
2	5	6	7	8	9	10	4	1	0	3

- Follow the pointers in T in F to recover the input starting with X.

Parul® University

DIGITAL LEARNING CONTENT

Decoding Examples ?

X = 2

F =

0	1	2	3	4	5	6	7	8	9	10
a	a	a	a	a	b	b	c	d	r	r

T =

0	1	2	3	4	5	6	7	8	9	10
2	5	6	7	8	9	10	4	1	0	3

a

Decoding Examples ?

F =

0	1	2	3	4	5	6	7	8	9	10
a	a	a	a	a	b	b	c	d	r	r

T =

0	1	2	3	4	5	6	7	8	9	10
2	5	6	7	8	9	10	4	1	0	3

ab

Decoding Examples ?

$$F = \begin{array}{ccccccccccc} 0 & 1 & \underline{2} & 3 & 4 & 5 & \underline{6} & 7 & 8 & 9 & \underline{10} \\ a & a & a & a & a & b & b & c & d & r & r \end{array}$$

$$T = \begin{array}{ccccccccccc} 0 & 1 & \underline{2} & 3 & 4 & 5 & \underline{6} & 7 & 8 & 9 & \underline{10} \\ 2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \end{array}$$

abr

Decoding Examples ?

- Why does this work?
- The first symbol of A[T[i]] is the second symbol of A[i] because $A^s[T[i]] = A[i]$.

A

0	aabracadabr
1	abraabracad
2	abracadabra
3	acadabraabr
4	adabraabrac
5	braabracada
6	bracadabraa
7	cadabraabra
8	dabraabraca
9	raabracadab
10	racadabraab

A^s

0	raabracadab
1	dabraabraca
2	aabracadabr
3	racadabraab
4	cadabraabra
5	abraabracad
6	abracadabra
7	acadabraabr
8	adabraabrac
9	braabracada
10	bracadabraa

Decoding Examples ?

- How do we compute T from L and X?

	0	1	2	3	4	5	6	7	8	9	10
F =	a	a	a	a	a	b	b	c	d	r	r
L =	r	d	a	r	c	a	a	a	a	b	b

Note that L is the first column of A^s and A^s is in the same order as A.

If *i* is the *k*-th *x* in F then T[i] is the *k*-th *x* in L.

Decoding Examples ?

	0	1	2	3	4	5	6	7	8	9	10
F =	a	a	a	a	a	b	b	c	d	r	r
L =	r	d	a	r	c	a	a	a	a	b	b

T= 0 1 2 3 4 5 6 7 8 9 10

2 5 6 7 8

Decoding Examples ?

	0	1	2	3	4	5	6	7	8	9	10
F =	a	a	a	a	a	b	b	c	d	r	r
L =	r	d	a	r	c	a	a	a	a	b	b

T=	0	1	2	3	4	5	6	7	8	9	10
	2	5	6	7	8	9	10				

Decoding Examples ?

	0	1	2	3	4	5	6	7	8	9	10
F =	a	a	a	a	a	b	b	c	d	r	r
L =	r	d	a	r	c	a	a	a	a	b	b

T=	0	1	2	3	4	5	6	7	8	9	10
	2	5	6	7	8	9	10	4			

Decoding Examples ?

	0	1	2	3	4	5	6	7	8	9	10
F =	a	a	a	a	a	b	b	c	d	r	r
L =	r	d	a	r	c	a	a	a	a	b	b

	0	1	2	3	4	5	6	7	8	9	10
T=	2	5	6	7	8	9	10	4	1		

Decoding Examples ?

	0	1	2	3	4	5	6	7	8	9	10
F =	a	a	a	a	a	b	b	c	d	r	r
L =	r	d	a	r	c	a	a	a	a	b	b

	0	1	2	3	4	5	6	7	8	9	10
T =	2	5	6	7	8	9	10	4	1	0	3

Notes on BW

- Alphabetic sorting does not need the entire cyclic shifted inputs. You just have to look at long enough prefixes.
  - A bucket sort will work here.
- Requires *entire* input. In practice, that's impossible.
  - Break input into blocks.
- There are high quality practical implementations:
  - Bzip
  - Bzip2 (public domain)

Move To Front Algorithm

- MTF is part of Burrows-Wheeler, basis for bzip2!
- Non-numerical data.
- The data have a relatively small working set that changes over the sequence.
  - Example: a b a b a a b c c b b c c c c b d b c c
- Move to Front algorithm:
  - Symbols are kept in a list indexed 0 to m-1.
  - To code a symbol output its index and move the symbol to the front of the list.

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0

0	1	2	3
a	b	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1

0	1	2	3
a	b	c	d

↓

0	1	2	3
b	a	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1

0	1	2	3
b	a	c	d

↓

0	1	2	3
a	b	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1

0	1	2	3
a	b	c	d

↓

0	1	2	3
b	a	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1 1

0	1	2	3
b	a	c	d

↓

0	1	2	3
a	b	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c

  0 1 1 1 1 0

0	1	2	3
a	b	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1 1 0 1

0	1	2	3
a	b	c	d

↓

0	1	2	3
b	a	c	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1 1 0 1 2

0	1	2	3
b	a	c	d

↓

0	1	2	3
c	b	a	d

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1 1 0 1 2 0 1 0 1 0 00 1 3 1 2 0

0	1	2	3
c	b	d	a

Example

- Example: a b a b a a b c c b b c c c c b d b c c
  0 1 1 1 1 0 1 2 0 1 0 1 0 00 1 3 1 2 0

Frequencies of {a, b, c, d}

a	b	c	d
4	7	8	1

Frequencies of {0, 1, 2, 3}

0	1	2	3
8	9	2	1

Example

Input:

aaaaaaaaaabbbbbbbbbbccccccccccdddddddddd

Output

0000000000100000000020000000003000000000

Frequencies of a b c d

a	b	c	d
10	10	10	10

Frequencies of 0 1 2 3

0	1	2	3
37	1	1	1

Example

Input:

aaaaaaaaaabbbbbbbbbbccccccccccdddddddddd

Output

0000000000100000000020000000003000000000

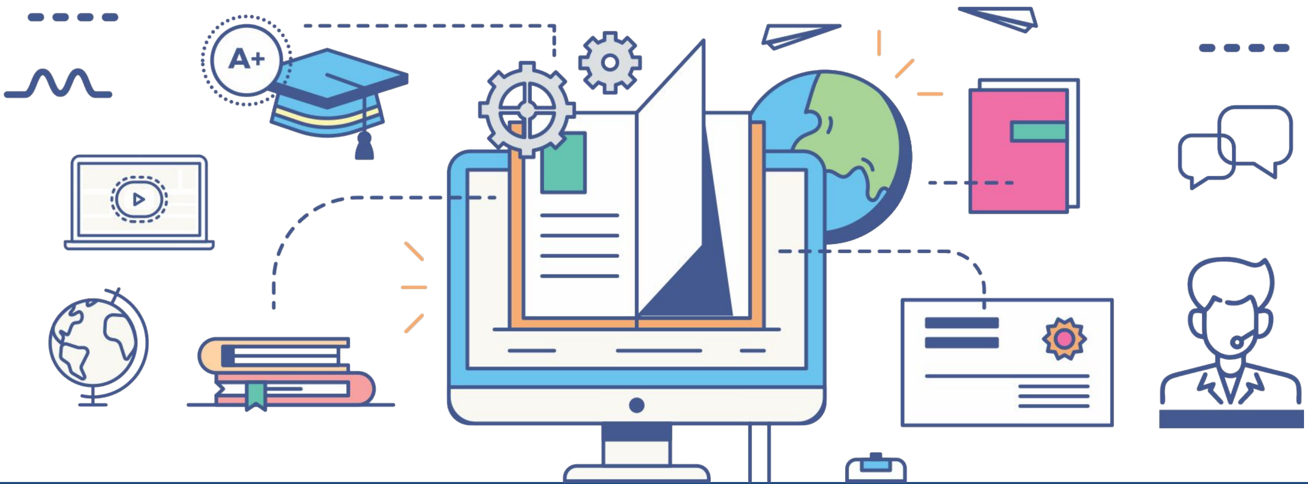
Frequencies of a b c d

a	b	c	d
10	10	10	10

Frequencies of 0 1 2 3

0	1	2	3
37	1	1	1

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in