CSE 589
Applied Algorithms
Spring 1999

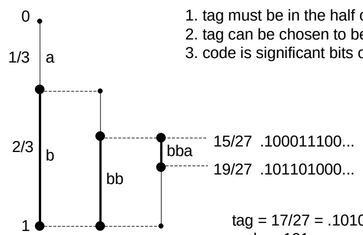
Arithmetic Coding
Dictionary Coding

Arithmetic Coding

- Huffman coding works well for larger alphabets and gets to within one bit of the entropy lower bound. Can we do better. Yes
- Basic idea in arithmetic coding:
  - represent each string x of length n by an 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).

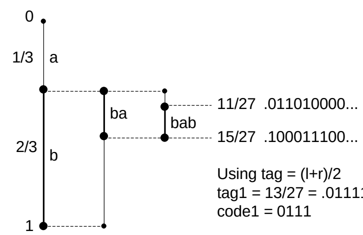
Example of Arithmetic Coding (1)

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



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

Some Tags are Better than Others

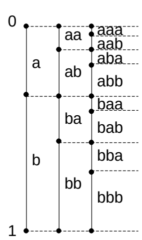


Using tag = (l+r)/2
tag1 = 13/27 = .011110110...
code1 = 0111

tag2 = 14/27 = .100001001...
code2 = 1

Example of Codes

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



0/27	.000000000...
1/27	.000010010...
3/27	.000111000...
5/27	.001011110...
9/27	.010101010...
11/27	.011010000...
15/27	.100011100...
19/27	.101101000...
27/27	.111111111...

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

Arithmetic Coding Algorithm

- $P(a_1), P(a_2), \dots, P(a_m)$
- $C(a_i) = P(a_1) + P(a_2) + \dots + P(a_{i-1})$
- Encode $x_1 x_2 \dots x_n$

```
Initialize l := 0 and r := 1;
for i = 1 to n do
   w := r - l;
   l := l + wC(x_i);
   r := l + wP(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

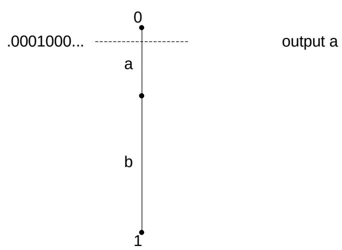
```
w := r - l;
l := l + w C(x);
r := l + w P(x)
```

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

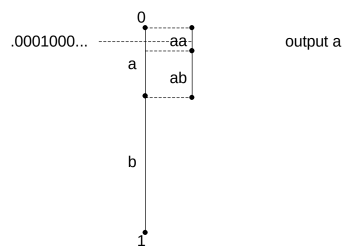
Decoding (1)

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



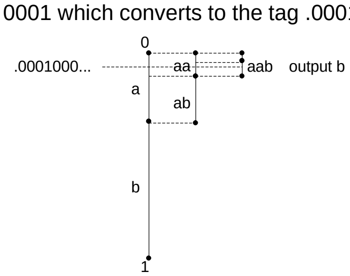
Decoding (2)

- 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

- $P(a_1), P(a_2), \dots, P(a_m)$
- $C(a_i) = P(a_1) + P(a_2) + \dots + P(a_{i-1})$
- Decode $b_1 b_2 \dots b_m$, 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 + wC(a_j) ≤ t < l + w(C(a_j)+P(a_j))
   output a_j;
   l := l + wC(a_j);
   r := l + wP(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

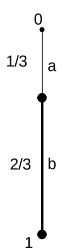
Practical Arithmetic Coding

- 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 altogether.

Coding with Scaling (1)

- Assume the length is known to be 3.
- bba

l = 2/3 r = 1



Scaling Principle:
If [r,l) is contained in [0,.5) the double, r := 2r; l := 2l and output 0.

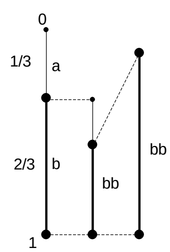
If [r,l) is contained in [.5,1) then shift by .5 and double, r := 2(r -.5); l := 2(l-.5) and output 1.

Coding with Scaling (2)

- Assume the length is known to be 3.
- bba 1

output 1 and scale

l = 5/9 r = 1
l = 1/9 r = 1

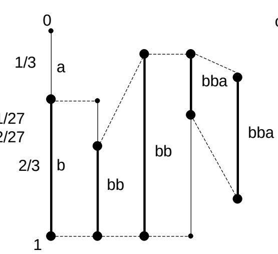


Coding with Scaling (3)

- Assume the length is known to be 3.
- bba 10

output 0 and scale

l = 1/9 r = 11/27
l = 2/9 r = 22/27



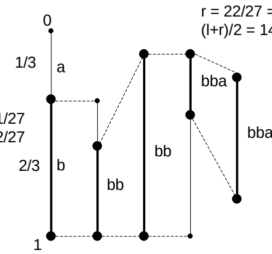
Coding with Scaling (4)

- Assume the length is known to be 3.
- bba 101

l = 2/9 = .000100101...
r = 22/27 = .110100001
(l+r)/2 = 14/27 = .100001001...

output 1

l = 1/9 r = 11/27
l = 2/9 r = 22/27



Notes on Arithmetic Coding

- Arithmetic codes come close to the entropy lower bound.
- Grouping symbols is effective for arithmetic coding.
- Arithmetic codes can be used effectively on small symbol sets. Advantage over Huffman.
- Context can be added so that more than one probability distribution can be used.
  - The best coders in the world use this method.
- There are very effective adaptive arithmetic coding methods.

Dictionary Coding

- Most popular methods are based on Ziv and Lempel's seminal work in 1977 and 1978.
- Basic idea: Maintain a dictionary of commonly used strings. Each commonly used string has an index.
  - Static dictionary, fixed and does not change.
  - Dynamic dictionary, adapts to the changing string.

Static Dictionary

0	a	6	bc
1	b	7	bcc
2	c	8	ada
3	d	9	abc
4	aa	10	dda
5	ab	11	aaaa

Encoding: from the current position find the longest string in source string that matches a string in the dictionary. Output its index.
Decoding: for each index output the corresponding string in the dictionary.

Static Dictionary Example

0	a	6	bc
1	b	7	bcc
2	c	8	ada
3	d	9	abc
4	aa	10	dda
5	ab	11	aaaa

a a b c c a d b a a a a d d a   30 bits with 2 bits/symbol

<u>a a</u> <u>b c c</u> <u>a</u> <u>d</u> <u>b</u> <u>a a a a</u> <u>d d a</u>
4 7 0 3 1 11 10   28 bits at 4 bits/symbol

Dynamic Dictionary

- For a static dictionary both the encoder and decoder have to have the dictionary.
- Dynamic dictionary
  - The encoder builds the dictionary as it scans the input.
  - The decoder emulates the encoder, building the same dictionary as it decodes the string.

LZW Compression

- Invented by Ziv and Lempel in 1978 and improved upon by Welch in 1984.
- Unix compress and GIF are based on LZW
- In LZW both encoder and decoder share the same indexes of the symbol alphabet ahead of time.
  - For standard symbols sets like ASCII this is no problem.

LZW Encoding Algorithm

Repeat
  find the longest match w in the dictionary
  output the index of w
  put wa in the dictionary where a was the unmatched symbol

LZW Encoding Example (1)

Dictionary

0 a
1 b
2 c
3 d
4 r

a b r a c a d a b a r a b r a

LZW Encoding Example (2)

Dictionary

0 a
1 b
2 c
3 d
4 r
5 ab

a b r a c a d a b a r a b r a

0

LZW Encoding Example (3)

Dictionary

0 a
1 b
2 c
3 d
4 r
5 ab
6 br

a b r a c a d a b a r a b r a

0 1

LZW Encoding Example (4)

Dictionary

0 a
1 b
2 c
3 d
4 r
5 ab
6 br
7 ra

a b r a c a d a b a r a b r a

0 1 4

LZW Encoding Example (5)

Dictionary

0 a
1 b
2 c
3 d
4 r
5 ab
6 br
7 ra
8 ac

a b r a c a d a b a r a b r a

0 1 4 0

LZW Encoding Example (6)

Dictionary

0	a	9	ca
1	b		
2	c		
3	d		
4	r		
5	ab		
6	br		
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2

LZW Encoding Example (7)

Dictionary

0	a	9	ca
1	b	10	ad
2	c		
3	d		
4	r		
5	ab		
6	br		
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0

LZW Encoding Example (8)

Dictionary

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d		
4	r		
5	ab		
6	br		
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0 3

LZW Encoding Example (9)

Dictionary

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	aba
4	r		
5	ab		
6	br		
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0 3 5

LZW Encoding Example (10)

Dictionary

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	aba
4	r	13	ar
5	ab		
6	br		
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0 3 5 0

LZW Encoding Example (11)

Dictionary

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	aba
4	r	13	ar
5	ab	14	rab
6	br		
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0 3 5 0 7

LZW Encoding Example (12)

Dictionary

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	aba
4	r	13	ar
5	ab	14	rab
6	br	15	bra
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0 3 5 0 7 6

LZW Encoding Example (13)

Dictionary

0	a	9	ca
1	b	10	ad
2	c	11	da
3	d	12	aba
4	r	13	ar
5	ab	14	rab
6	br	15	bra
7	ra		
8	ac		

a b r a c a d a b a r a b r a

0 1 4 0 2 0 3 5 0 7 6 0

LZW Decoding Algorithm

- Emulate the Encoder in building the dictionary.
- Decode each index according to its index.
- Problem: the current index have an incomplete entry because it is currently being added to the dictionary.
  - The problem is solved because there is enough information in the incomplete entry to continue decoding.

LZW Decoding Example (1)

Dictionary

0 a
1 b

0 1 2 4 3 6

LZW Decoding Example (2)

Dictionary

0 a
1 b
2 a...

0 1 2 4 3 6

a

LZW Decoding Example (3)

Dictionary

0 a
1 b
2 ab
3 b...

0 1 2 4 3 6

a b

LZW Decoding Example (4)

Dictionary

0 a
1 b
2 ab
3 ba
4 ab...

0 1 2 4 3 6

a b ab

The next index is 4, but it is incomplete!

LZW Decoding Example (5)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

0 1 2 4 3 6

a b ab

The entry has a first symbol which is all we need to complete it.

LZW Decoding Example (6)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 aba...

0 1 2 4 3 6

a b ab aba

LZW Decoding Example (7)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 ba...

0 1 2 4 3 6

a b ab aba ba

LZW Decoding Example (8)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 bab

0 1 2 4 3 6

a b ab aba ba

complete 6

LZW Decoding Example (9)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 bab
7 bab...

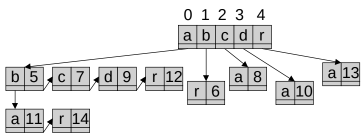
0 1 2 4 3 6

a b ab aba ba bab

Trie Data Structure for Dictionary

- Fredkin (1960)

0	a	9	ad
1	b	10	da
2	c	11	aba
3	d	12	ar
4	r	13	ra
5	ab	14	abr
6	br		
7	ac		
8	ca		



Depending on the size of the dictionary it might be wise to have two array levels to minimize searching.

Notes on Dictionary Coding

- Extremely effective when there are repeated patterns in the data that are widely spread. Where local context is not as significant.
  - text
  - some graphics
  - program sources or binaries
- Variants of LZW are pervasive.