CSEP 590
Data Compression
Autumn 2007

Dictionary Coding

LZW, LZ77

Dictionary Coding

- Does not use statistical knowledge of data.
- Encoder: As the input is processed develop a dictionary and transmit the index of strings found in the dictionary.
- Decoder: As the code is processed reconstruct the dictionary to invert the process of encoding.
- Examples: LZW, LZ77, Sequitur,
- Applications: Unix Compress, gzip, GIF

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

a b a b a b a b a

0 a
1 b

LZW Encoding Example (2)

Dictionary

0 a
1 b
2 ab

<u>a</u> b a b a b a b a
0

LZW Encoding Example (3)

Dictionary

0 a
1 b
2 ab
3 ba

<u>a</u> <u>b</u> a b a b a b a
0 1

LZW Encoding Example (4)

Dictionary

0 a
1 b
2 ab
3 ba
4 aba

a b a b a b a b a
0 1 2

LZW Encoding Example (5)

Dictionary

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

a b a b a b a b a

0 1 2 4

LZW Encoding Example (6)

Dictionary

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

a b a b a b a b a
0 1 2 4 3

LZW Decoding Algorithm

- Emulate the encoder in building the dictionary. Decoder is slightly behind the encoder.

```
initialize dictionary;
decode first index to w;
put w? in dictionary;
repeat
   decode the first symbol s of the index;
   complete the previous dictionary entry with s;
   finish decoding the remainder of the index;
   put w? in the dictionary where w was just decoded;
```

LZW Decoding Example (1)

Dictionary

0 a
1 b
2 a?

0 1 2 4 3 6
a

LZW Decoding Example (2a)

Dictionary

0	a
1	b
2	ab

0 1 2 4 3 6

a b

LZW Decoding Example (2b)

Dictionary

0	a
1	b
2	ab
3	b?

<u>0</u> <u>1</u> 2 4 3 6
a b

LZW Decoding Example (3a)

Dictionary

0	a
1	b
2	ab
3	ba

0 1 2 4 3 6

a b a

LZW Decoding Example (3b)

Dictionary

0	a
1	b
2	ab
3	ba
4	ab?

0 1 2 4 3 6
a b ab

LZW Decoding Example (4a)

Dictionary

0	a
1	b
2	ab
3	ba
4	aba

0 1 2 4 3 6
a b ab a

LZW Decoding Example (4b)

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 (5a)

Dictionary

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

0 1 2 4 3 6

a b ab aba b

LZW Decoding Example (5b)

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 (6a)

Dictionary

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

<u>0</u> <u>1</u> <u>2</u> <u>4</u> <u>3</u> <u>6</u>

a b ab aba ba b

LZW Decoding Example (6b)

Dictionary

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

<u>0</u> <u>1</u> <u>2</u> <u>4</u> <u>3</u> <u>6</u>
a b ab aba ba bab

Decoding Exercise

Base Dictionary

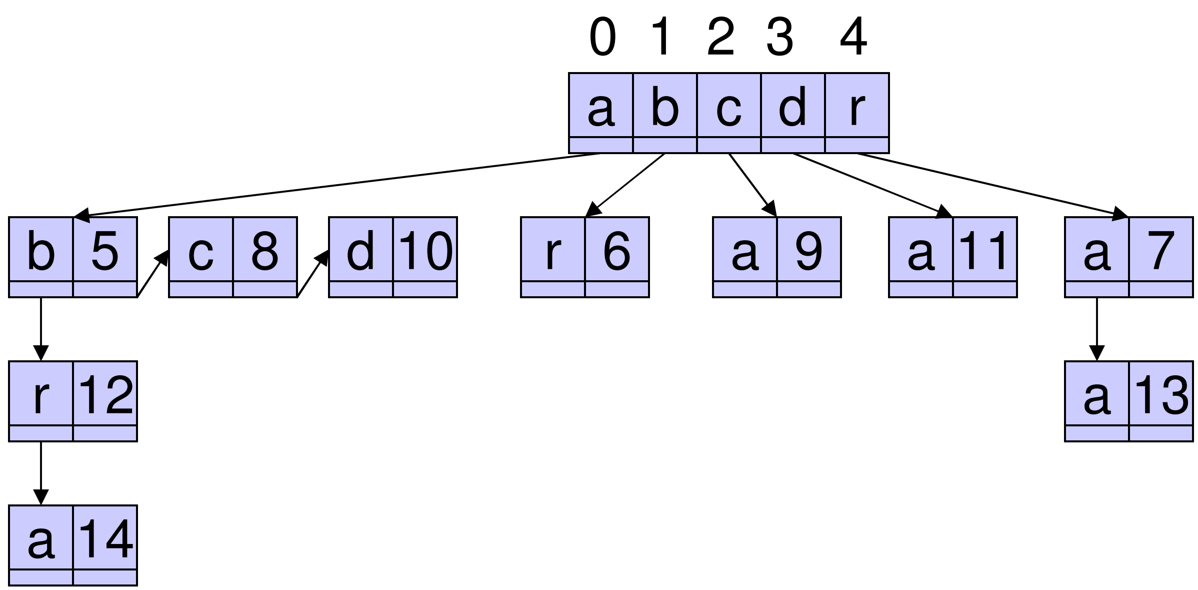
0	a
1	b
2	c
3	d
4	r

0 1 4 0 2 0 3 5 7

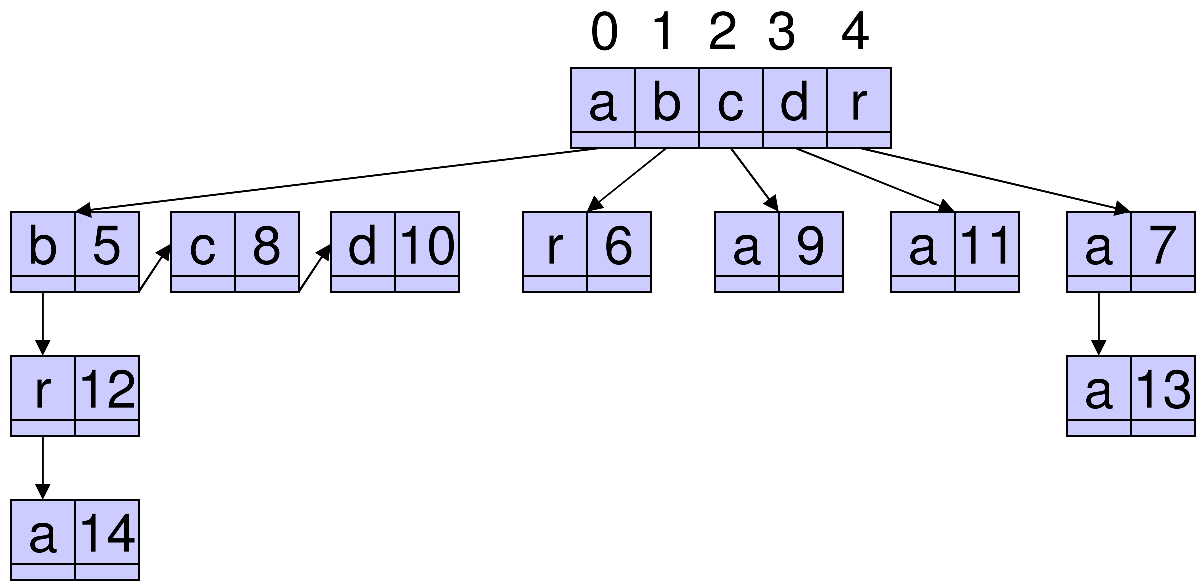
Trie Data Structure for Encoder's Dictionary

- Fredkin (1960)

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



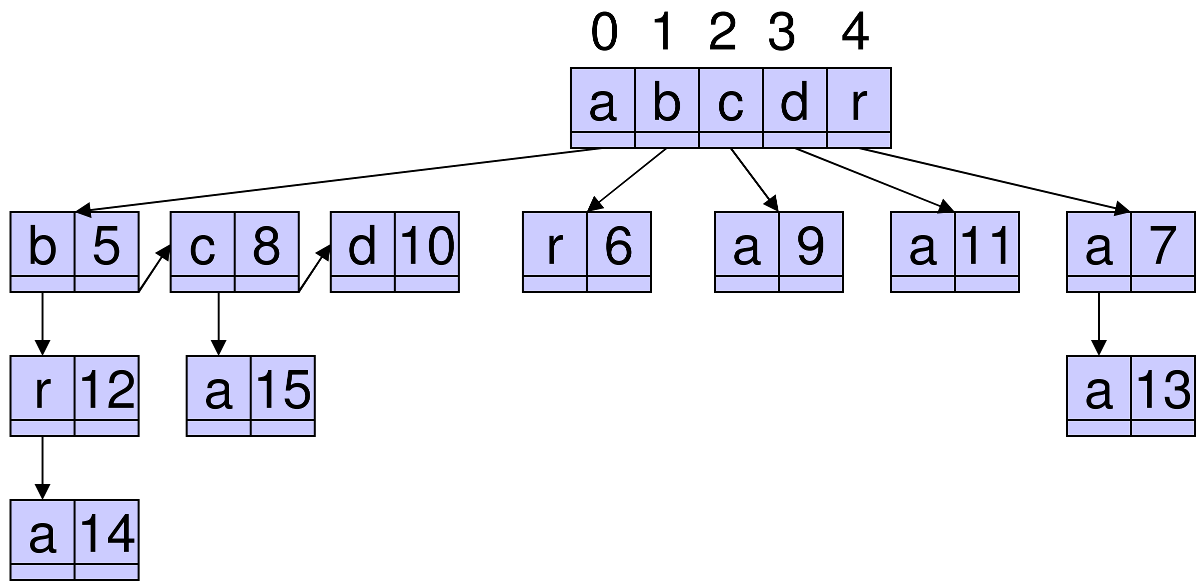
Encoder Uses a Trie (1)



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

0 1 4 0 2 0 3 5 7 12

Encoder Uses a Trie (2)



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

0 1 4 0 2 0 3 5 7 12 8

Decoder's Data Structure

- Simply an array of strings

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

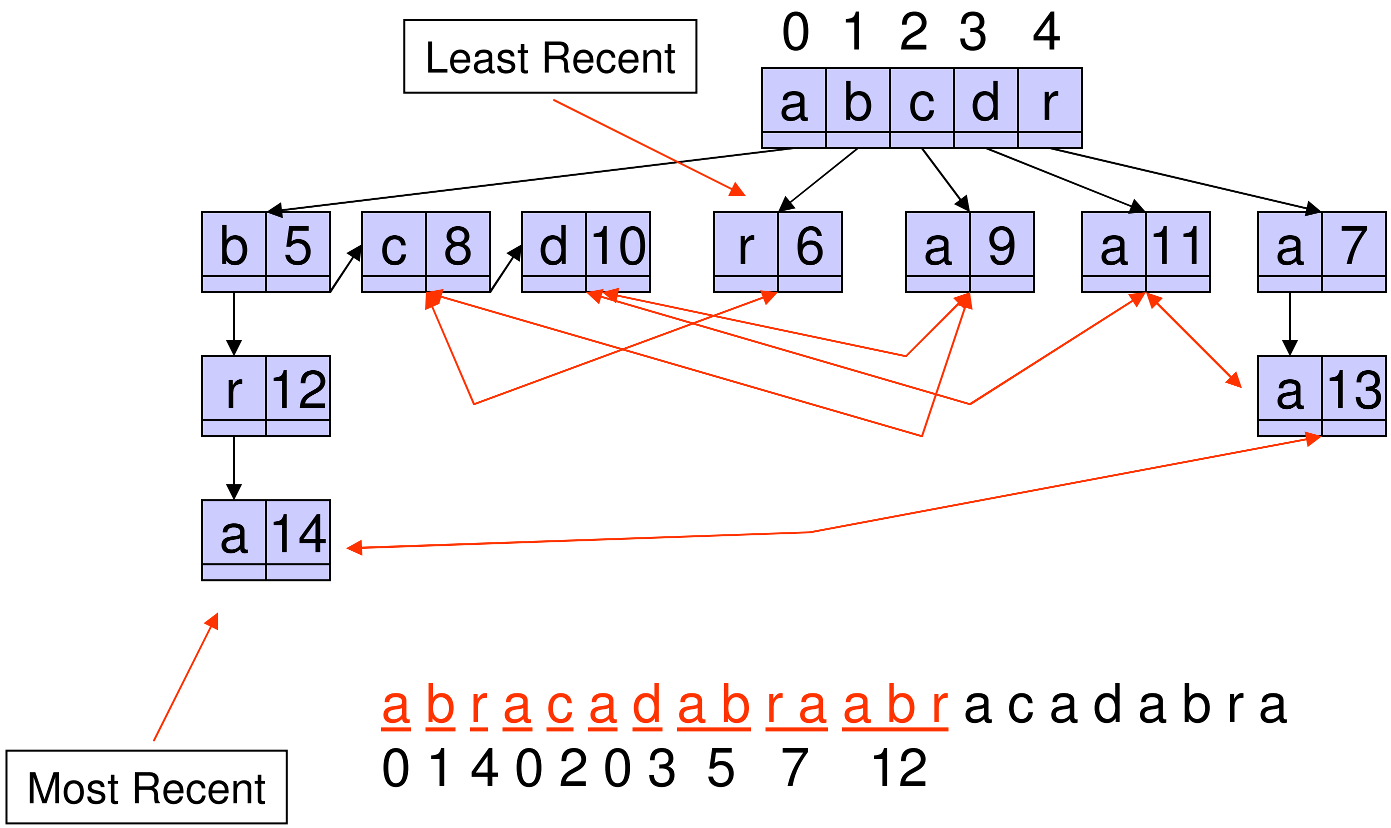
0 1 4 0 2 0 3 5 7 12 8 ...

a b r a c a d ab ra abr

Bounded Size Dictionary

- Bounded Size Dictionary
  - n bits of index allows a dictionary of size 2^n
  - Doubtful that long entries in the dictionary will be useful.
- Strategies when the dictionary reaches its limit.
  1. Don't add more, just use what is there.
  2. Throw it away and start a new dictionary.
  3. Double the dictionary, adding one more bit to indices.
  4. Throw out the least recently visited entry to make room for the new entry.

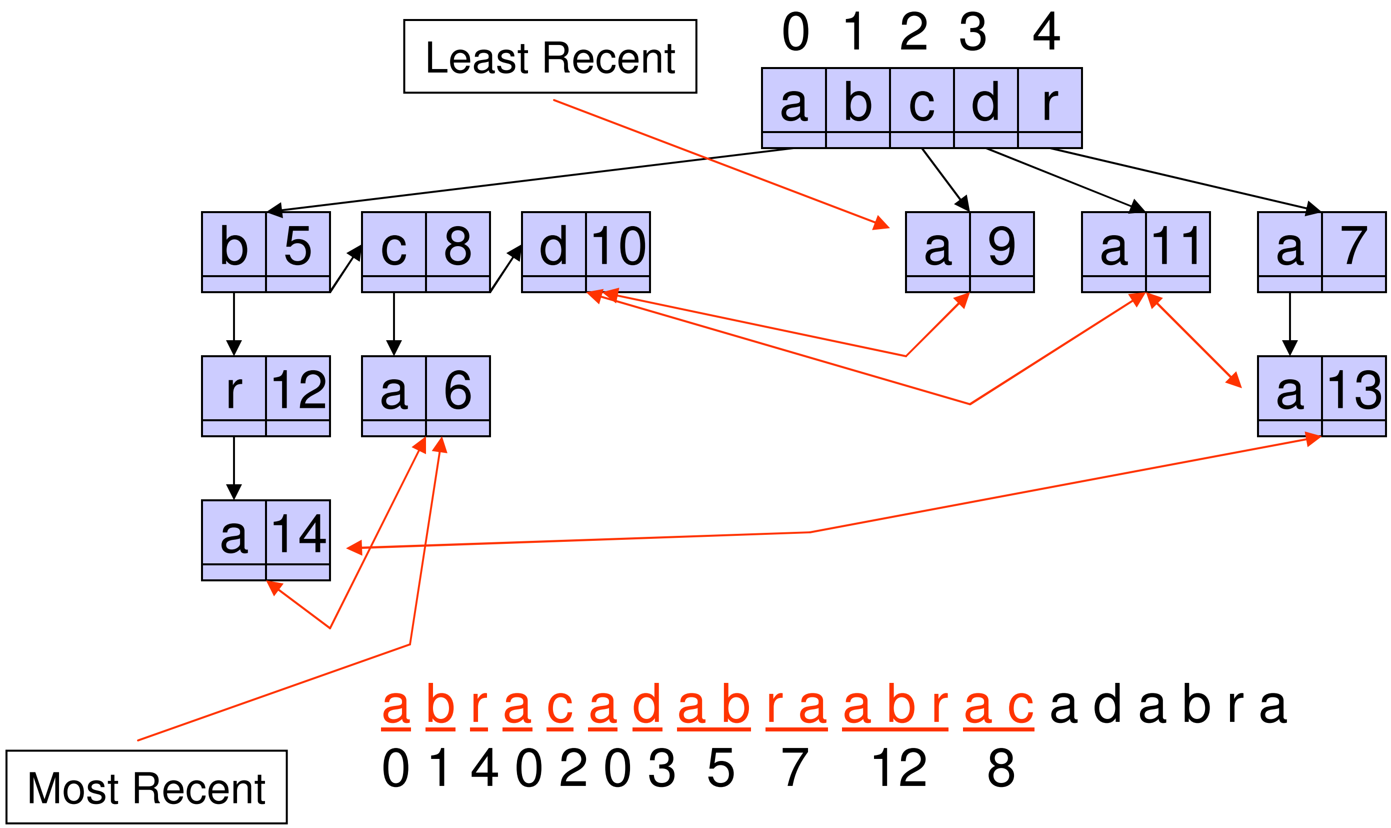
Implementing the LRV Strategy



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

0 1 4 0 2 0 3 5 7 12

Implementing the LRV Strategy



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

0 1 4 0 2 0 3 5 7 12 8

Notes on LZW

- Extremely effective when there are repeated patterns in the data that are widely spread.
- Negative: Creates entries in the dictionary that may never be used.
- Applications:
  - Unix compress, GIF, V.42 bis modem standard

The Dictionary is Implicit

- Ziv and Lempel, 1977
- Use the string coded so far as a dictionary.
- Given that $x_1x_2...x_n$ has been coded we want to code $x_{n+1}x_{n+2}...x_{n+k}$ for the largest k possible.

Solution A

- If $x_{n+1}x_{n+2}...x_{n+k}$ is a substring of $x_1x_2...x_n$ then $x_{n+1}x_{n+2}...x_{n+k}$ can be coded by <j,k> where j is the beginning of the match.
- Example

ababababa babababababababab....

coded

ababababa babababa babababab....

<2,8>

Solution A Problem

- What if there is no match at all in the dictionary?

ababababa cabababababababab....

coded

- Solution B. Send tuples <j,k,x> where
  - If k = 0 then x is the unmatched symbol
  - If k > 0 then the match starts at j and is k long and the unmatched symbol is x.

Solution B

- If $x_{n+1}x_{n+2}...x_{n+k}$ is a substring of $x_1x_2...x_n$ and $x_{n+1}x_{n+2}...\ x_{n+k}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}...x_{n+k}\ x_{n+k+1}$ can be coded by

  $$<j, k, x_{n+k+1}>$$

  where j is the beginning of the match.
- Examples

ababababa cabababababababab....

ababababa c abababab ababab....
<0,0,c> <1,9,b>

Solution B Example

a bababababababababababab.....

<0,0,a>

a b ababababababababababab.....

<0,0,b>

a b aba bababababababababab.....

<1,2,a>

a b aba babab ababababababab.....

<2,4,b>

a b aba babab ababababab bab.....

<1,10,a>

Surprise Code!

a bababababababababababab$

<0,0,a>

a b ababababababababababab$

<0,0,b>

a b ababababababababababab$

<1,22,$>

Surprise Decoding

<0,0,a><0,0,b><1,22,$>

<0,0,a>	a
<0,0,b>	b
<1,22,$>	a
<2,21,$>	b
<3,20,$>	a
<4,19,$>	b
...	
<22,1,$>	b
<23,0,$>	$

Surprise Decoding

<0,0,a><0,0,b><1,22,$>

<0,0,a>	a
<0,0,b>	b
<1,22,$>	a
<2,21,$>	b
<3,20,$>	a
<4,19,$>	b
...	
<22,1,$>	b
<23,0,$>	$

Solution C

- The matching string can include part of itself!
- If $x_{n+1}x_{n+2}\ldots x_{n+k}$ is a substring of

  $$x_1x_2\ldots x_n\, x_{n+1}x_{n+2}\ldots x_{n+k}$$

  that begins at $j \leq n$ and $x_{n+1}x_{n+2}\ldots\, x_{n+k}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}\ldots x_{n+k}\, x_{n+k+1}$ can be coded by

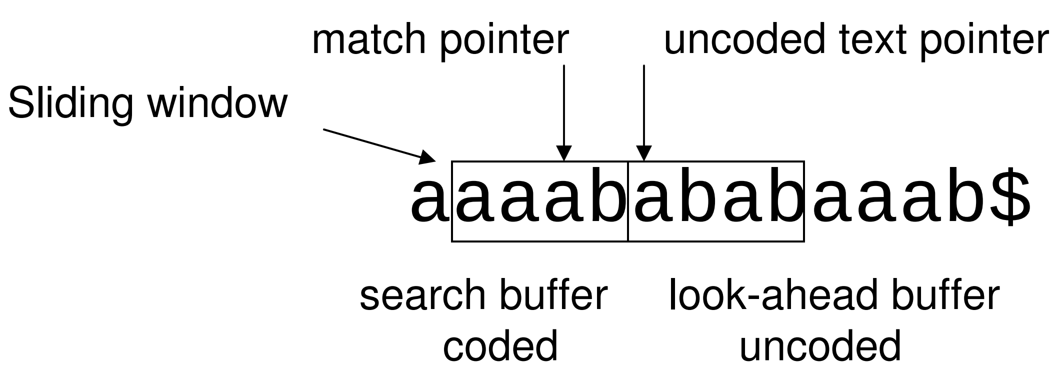
  $$<j,k,\, x_{n+k+1}>$$

In Class Exercise

- Use Solution C to code the string
  - abaabaaabaaaab$
  - aaaabaaabaabab$

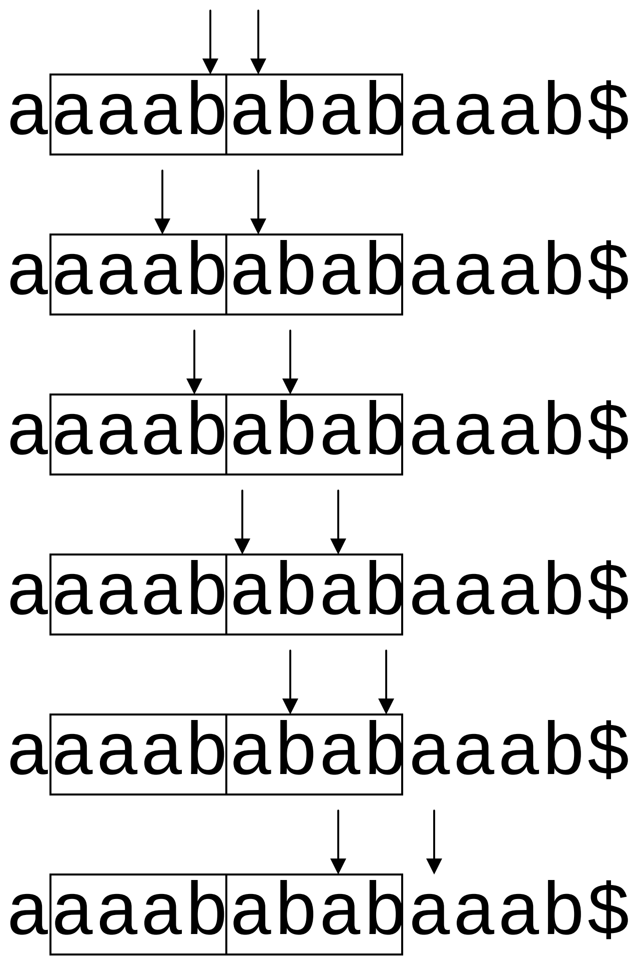
Bounded Buffer – Sliding Window

- We want the triples <j,k,x> to be of bounded size. To achieve this we use bounded buffers.
  - Search buffer of size s is the symbols $x_{n-s+1} \ldots x_n$
    j is then the offset into the buffer.
  - Look-ahead buffer of size t is the symbols $x_{n+1} \ldots x_{n+t}$
- Match pointer can start in search buffer and go into the look-ahead buffer but no farther.



tuple
<2,5,a>

Search in the Sliding Window



offset	length
1	0
2	1
2	2
2	3
2	4
2	5

tuple
<2,5,a>

Coding Example

s = 4, t = 4, a = 3

	tuple
aaaabababaaab$	<0,0,a>
aaaabababaaab$	<1,3,b>
aaaabababaaab$	<2,5,a>
aaaabababaaab$	<4,2,$>

Coding the Tuples

- Simple fixed length code

$$\lceil \log_2(s+1) \rceil + \lceil \log_2(s+t+1) \rceil + \lceil \log_2 a \rceil$$

s = 4, t = 4, a = 3

tuple	fixed code
<2,5,a>	010 0101 00

- Variable length code using adaptive Huffman or arithmetic code on Tuples
  - Two passes, first to create the tuples, second to code the tuples
  - One pass, by pipelining tuples into a variable length coder

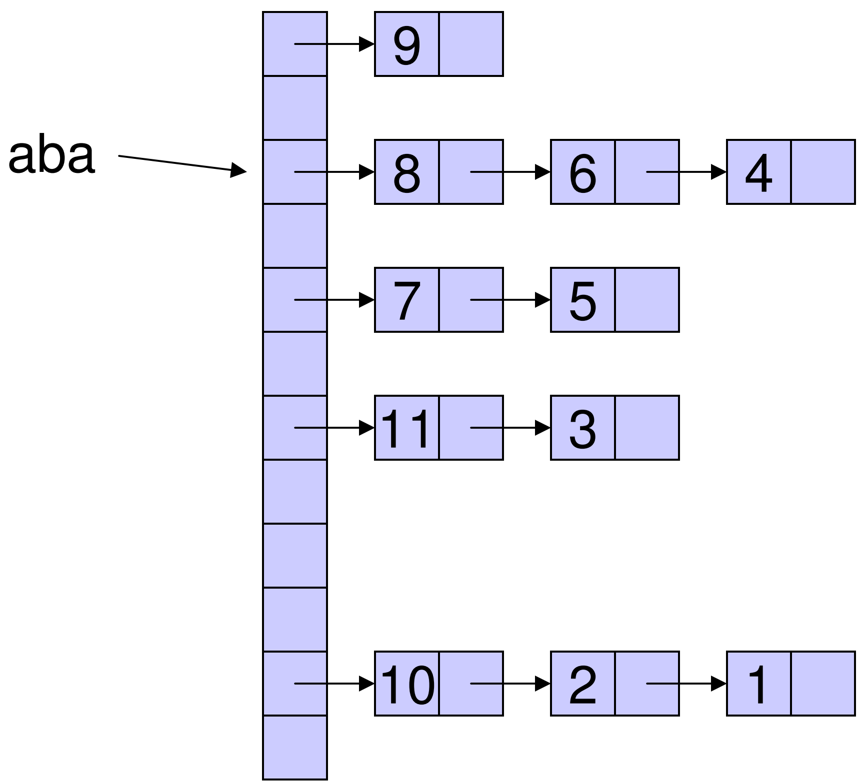
Zip and Gzip

- Search Window
  - Search buffer 32KB
  - Look-ahead buffer 258 Bytes
- How to store such a large dictionary
  - Hash table that stores the starting positions for all three byte sequences.
  - Hash table uses chaining with newest entries at the beginning of the chain. Stale entries can be ignored.
- Second pass for Huffman coding of tuples.
- Coding done in blocks to avoid disk accesses.

Example

12

aaaababab**abaa**|**abaa**aababababaaabba$

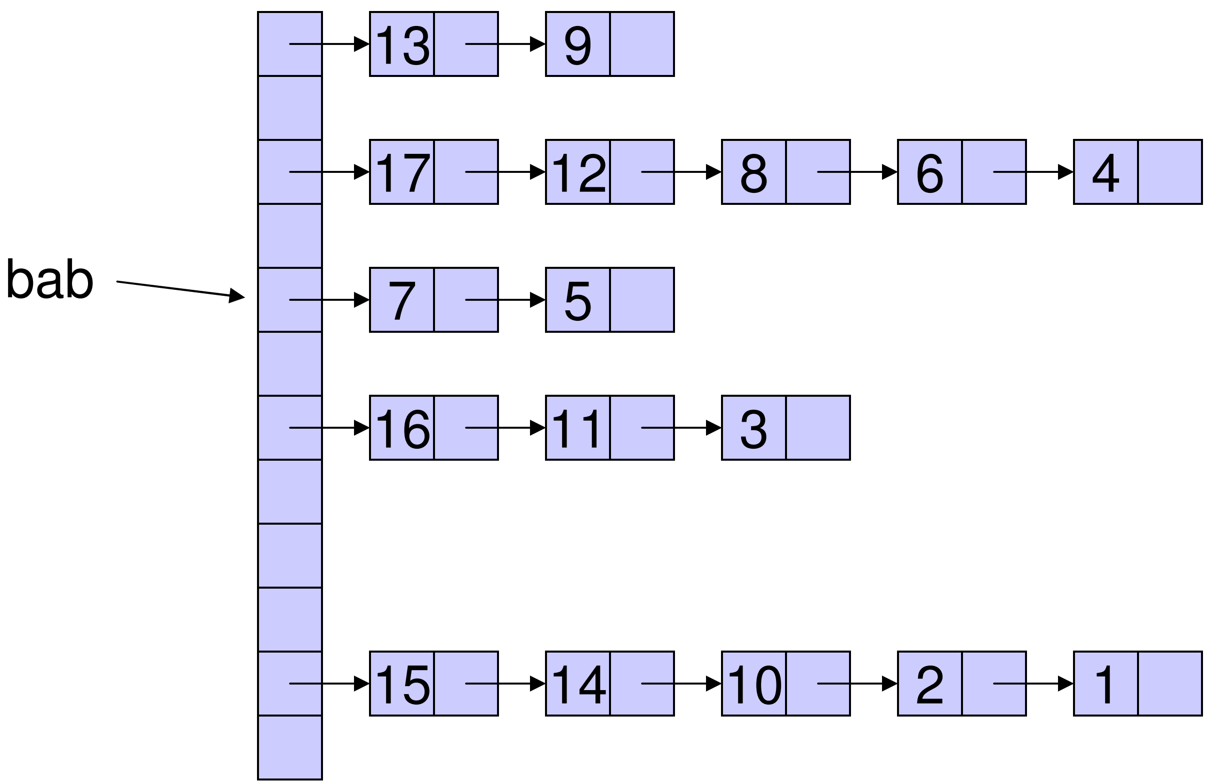


Offset =12 – 8 = 4
Length = 5
Tuple = <4,5,a>

Example

18

aaaababababaaabaaaababababaaabba$



No match

Tuple = <0,0,b>

Notes on LZ77

- Very popular especially in unix world
- Many variants and implementations
  - Zip, Gzip, PNG, PKZip,Lharc, ARJ
- Tends to work better than LZW
  - LZW has dictionary entries that are never used
  - LZW has past strings that are not in the dictionary
  - LZ77 has an implicit dictionary. Common tuples are coded with few bits.