CSE 490 GZ
Introduction to Data Compression
Winter 2002

Dictionary Coding

LZ77

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

  abababab<u></u>a babababababababab....
  coded

  ababababa bababab 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.
- Example

  ababababa cabababababababab....

  ababababa c ababababab 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 abababababa 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

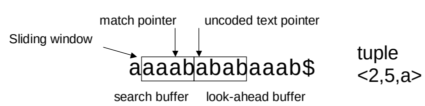
- The matching string can include part of itself!
- If $x_{n+1}x_{n+2}...x_{n+k}$ is a substring of $x_1x_2...x_n\ x_{n+1}x_{n+2}...x_{n+k}$ that begins at $j \leq n$ and $x_{n+1}x_{n+2}...\ x_{n+k}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}...x_{n+k}\ x_{n+k+1}$ can be coded by $\langle j,k,\ x_{n+k+1} \rangle$

In Class Exercise

- Use Solution C to code the string
  - abaabaaabaaaab$

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 $x_{n-s+1}...x_n$ j is then the offset into the buffer.
  - Look-ahead buffer of size t is the symbols $x_{n+1}...x_{n+t}$
- Match pointer can start in search buffer and go into the look-ahead buffer but no farther.



Search in the Sliding Window

string	offset	length	
aaaabababaaab$	1	0	
aaaabababaaab$	2	1	
aaaabababaaab$	2	2	
aaaabababaaab$	2	3	
aaaabababaaab$	2	4	
aaaabababaaab$	2	5	tuple <2,5,a>

Coding Example

s = 4, t = 4, a = 3

string	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

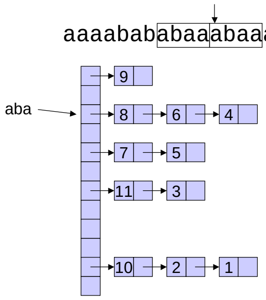
Zip and Gzip

- 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

aaaabababaaaabaaaababababaaaabba$

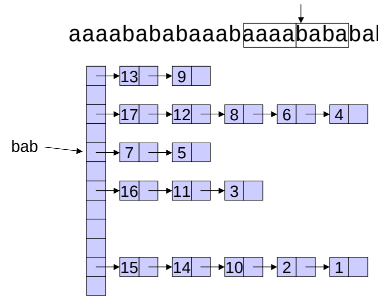


Offset =12 – 8 = 4
Length = 5
Tuple = <4,5,a>

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.