

# Data Compression

- Reduce the size of data.
  - Reduces storage space and hence storage cost.
    - Compression ratio = original data size / compressed data size
  - Reduces time to transmit and retrieve data.
  - Reduces the storage requirement.  
(particularly useful in embedded systems, network bridges, routers etc.)

Adapted from Sahni's *Data Structures and Applications* slides.

# Lossless And Lossy Compression

- $\text{compressedData} = \text{compress}(\text{originalData})$
- $\text{decompressedData} = \text{decompress}(\text{compressedData})$
- When  $\text{originalData} = \text{decompressedData}$ , the compression is **lossless**.
- When  $\text{originalData} \neq \text{decompressedData}$ , the compression is **lossy**.

# Lossless And Lossy Compression

- Lossy compressors generally obtain much higher compression ratios than do lossless compressors.
  - Say 100 vs. 2.
- Lossless compression is essential in applications such as text file compression.
- Lossy compression is acceptable in many imaging applications.
  - In video transmission, a slight loss in the transmitted video is not noticed by the human eye.

# Text Compression

- Lossless compression is essential.
- Popular text compressors such as `zip` and Unix's `compress` are based on the LZW (`Lempel-Ziv-Welch`) method.

# LZW Compression

- Character sequences in the original text are replaced by codes that are dynamically determined.
- The code table is not encoded into the compressed text, because it may be reconstructed from the compressed text during decompression.

# LZW Compression

- Assume the letters in the text are limited to {a, b}.
  - In practice, the alphabet may be the 256 character ASCII set.
- The characters in the alphabet are assigned code numbers beginning at 0.
- The initial code table is:

code	0	1
key	a	b

# LZW Compression

code	0	1
key	a	b

- Original text = *abababbabababbaabba*
- Compression is done by scanning the original text from left to right.
- Find longest prefix *p* for which there is a code in the code table.
- Represent *p* by its code *pCode* and assign the next available code number to *pc*, where *c* is the next character in the text that is to be compressed.

# LZW Compression

code	0	1	2
key	a	b	ab

- Original text = abababbabaabbabbaabba
- $p = a$
- $pCode = 0$
- $c = b$
- Represent  $a$  by  $0$  and enter  $ab$  into the code table.
- Compressed text =  $0$

# LZW Compression

code	0	1	2	3
key	a	b	ab	ba

- Original text = abababbabababbabbaabba
- Compressed text = 0
  - $p = b$
  - $pCode = 1$
  - $c = a$
  - Represent  $b$  by  $1$  and enter  $ba$  into the code table.
- Compressed text = 01

# LZW Compression

code	0	1	2	3	4
key	a	b	ab	ba	aba

- Original text = abababbabababbabbaabba
- Compressed text = 01
  - $p = ab$
  - $pCode = 2$
  - $c = a$
  - Represent  $ab$  by  $2$  and enter  $aba$  into the code table.
  - Compressed text = 012

# LZW Compression

code	0	1	2	3	4	5
key	a	b	ab	ba	aba	abb

- Original text = abababbababaabbabbaabba
- Compressed text = 012
  - $p = ab$
  - $pCode = 2$
  - $c = b$
  - Represent  $ab$  by  $2$  and enter  $abb$  into the code table.
  - Compressed text = 0122

# LZW Compression

code	0	1	2	3	4	5	6
key	a	b	ab	ba	aba	abb	bab

- Original text = ababab**bab**aabbabbaabba
- Compressed text = 0122
- $p = ba$
- $pCode = 3$
- $c = b$
- Represent **ba** by 3 and enter **bab** into the code table.
- Compressed text = 01223

# LZW Compression

code	0	1	2	3	4	5	6	7
key	a	b	ab	ba	aba	abb	bab	baa

- Original text = abababba**ba**abbabbaabba
- Compressed text = 01223
- $p = ba$
- $pCode = 3$
- $c = a$
- Represent **ba** by **3** and enter **baa** into the code table.
- Compressed text = 012233

# LZW Compression

code	0	1	2	3	4	5	6	7	8
key	a	b	ab	ba	aba	abb	bab	baa	abba

- Original text = abababbabababbabbaabba
- Compressed text = 012233
  - $p = abb$
  - $pCode = 5$
  - $c = a$
  - Represent  $abb$  by  $5$  and enter  $abba$  into the code table.
  - Compressed text = 0122335

# LZW Compression

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

- Original text = abababbabababb**abba**abba
- Compressed text = 0122335
  - $p = abba$
  - $pCode = 8$
  - $c = a$
  - Represent **abba** by 8 and enter **abbaa** into the code table.
  - Compressed text = 01223358

# LZW Compression

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

- Original text = abababbabaabbabba**abba**
- Compressed text = 01223358
  - $p = abba$
  - $pCode = 8$
  - $c = null$
  - Represent **abba** by 8.
  - Compressed text = 012233588

# Code Table Representation

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

- Dictionary.
  - Pairs are  $(key, element) = (key, code)$ .
  - Operations are :  $get(key)$  and  $put(key, code)$
- Limit number of codes to  $2^{12}$ .
- Use a hash table.
  - Convert variable length keys into fixed length keys.
  - Each key has the form  $pc$ , where the string  $p$  is a key that is already in the table.
  - Replace  $pc$  with  $(pCode)c$ .

# Code Table Representation

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbad

code	0	1	2	3	4	5	6	7	8	9
key	a	b	0b	1a	2a	2b	3b	3a	5a	8a

# Implementation of LZW algorithm

```
void Compress()  
{// Lempel-Ziv-Welch compressor.  
    ChainHashTable<element, long> h(D);  
    element e;  
    for (int i = 0; i < alpha; i++) {// initialize  
        e.key = i;  
        e.code = i;  
        h.Insert(e);  
    }  
    int used = alpha; // codes used  
// input and compress  
    unsigned char c;  
    in.get(c);  
    long pcode = c; // prefix code
```

# Implementation of LZW algorithm

```
if (!in.eof()) { // file length is > 1
    do { // process rest of file
        in.get(c);
        if (in.eof()) break; // finished
        long k = (pcode << ByteSize) + c;
        // see if code for k in dictionary
        if (h.Search(k, e)) pcode = e.code; // yes
        else { // k not in table
            output(pcode);
            if (used < codes) // create new code
            {e.code = used++;
             e.key = (pcode << ByteSize) | c;
             h.Insert(e);}
            pcode = c;}
        } while (true);
        output(pcode);
        if (status) {c = LeftOver << excess;
                    out.put(c);}
    }
    out.close(); in.close();
}
```

# LZW Decompression

code	0	1
key	a	b

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588
- Convert codes to text from left to right.
- 0 represents a.
- Decompressed text = a
- pCode = 0 and p = a.
- p = a followed by next text character (c) is entered into the code table.

# LZW Decompression

code	0	1	2
key	a	b	ab

- Original text = **a**bababbabaabbabbaabba
- Compressed text = **0**12233588
- 1 represents **b**.
- Decompressed text = **ab**
- **pCode** = 1 and **p** = **b**.
- **lastP** = **a** followed by first character of **p** is entered into the code table.

# LZW Decompression

code	0	1	2	3
key	a	b	ab	ba

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588
- 2 represents ab.
- Decompressed text = abab
- pCode = 2 and p = ab.
- lastP = b followed by first character of p is entered into the code table.

# LZW Decompression

code	0	1	2	3	4
key	a	b	ab	ba	aba

- Original text = **abab**abbabababbaabba
- Compressed text = **012233588**
  - **2** represents **ab**
  - Decompressed text = **ababab**.
  - **pCode = 2** and **p = ab**.
  - **lastP = ab** followed by first character of **p** is entered into the code table.

# LZW Decompression

code	0	1	2	3	4	5
key	a	b	ab	ba	aba	abb

- Original text = **abababb**babaabbabbaabba
- Compressed text = **012233588**
- **3** represents **ba**
- Decompressed text = **abababba**.
- **pCode = 3** and **p = ba**.
- **lastP = ab** followed by first character of **p** is entered into the code table.

# LZW Decompression

code	0	1	2	3	4	5	6
key	a	b	ab	ba	aba	abb	bab

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588
- 3 represents ba
- Decompressed text = abababbaba.
- pCode = 3 and p = ba.
- lastP = ba followed by first character of p is entered into the code table.

# LZW Decompression

code	0	1	2	3	4	5	6	7
key	a	b	ab	ba	aba	abb	bab	baa

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588
- 5 represents abb
- Decompressed text = abababbabaabb.
- pCode = 5 and p = abb.
- lastP = ba followed by first character of p is entered into the code table.

# LZW Decompression

code	0	1	2	3	4	5	6	7	8
key	a	b	ab	ba	aba	abb	bab	baa	abba

- Original text = abababbabababbabbaabba
- Compressed text = 012233588
- 8 represents ???
- When a code is not in the table, its key is lastP followed by first character of lastP.
- lastP = abb
- So 8 represents abba.



# LZW Decompression

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588
- 8 represents abba
- Decompressed text = abababbabaabbabbaabba.
- pCode = 8 and p = abba.
- lastP = abba followed by first character of p is entered into the code table.

# Code Table Representation

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

- Dictionary.
  - Pairs are  $(\text{key}, \text{element}) = (\text{code}, \text{what the code represents}) = (\text{code}, \text{codeKey})$ .
  - Operations are :  $\text{get}(\text{key})$  and  $\text{put}(\text{key}, \text{code})$
- Keys are integers  $0, 1, 2, \dots$
- Use a 1D array  $\text{codeTable}$ .
  - $\text{codeTable}[\text{code}] = \text{codeKey}$ .
  - Each code key has the form  $\text{pc}$ , where the string  $\text{p}$  is a code key that is already in the table.
  - Replace  $\text{pc}$  with  $(\text{pCode})\text{c}$ .

# Time Complexity

- Compression.
  - $O(n)$  expected time, where  $n$  is the length of the text that is being compressed.
- Decompression.
  - $O(n)$  time, where  $n$  is the length of the decompressed text.