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 retrieve and transmit data.

Lossless And Lossy Compression

- compressedData = compress(originalData)
- decompressedData = decompress(compressedData)
- When originalData = decompressedData, the compression is lossless.
- When originalData != 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 = abababbabaabbabbaabba
- 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 = abababbabaabbabbaabba
- 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 = abababbabaabbabbaabba
- 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 = ababababbabaabbabbaabba
- 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 = abababbabaabbabbaabba
- 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 = abababbabaabbabbaabba
- 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 = abababbabaabbabbaabba
- 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 = abababbabaaabbabbaaabba
- 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 = abababbabaabbabbaabba
- 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 : find(key) and insert(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	abbaa

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

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.

LZW Decompression

code	0	1	2
key	a	b	ab

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588
- 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 = ababababbabaabbabbaabba
- 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 = abababbabaabbabbaabba
- 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 = abababbabaaabbabbaabba
- 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 = ababababbabbabbaabba
- Compressed text = 012233588
- 8 represents abba
- Decompressed text = ababababbabbabbaabba.
- 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 (key, element) = (code, what the code represents) = (code, codeKey).
  - Operations are : find(key) and insert(key, code)
- Keys are integers 0, 1, 2, …
- Use a 1D array codeTable.
  - codeTable[code] = codeKey.
  - Each code key has the form pc, where the string p is a code key that is already in the table.
  - Replace pc with (pCode)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.

Prepare to be confused

How to pack 12 bit codes into bytes

```
1. void output(long pcode)
2. {// Output 8 bits, save rest in leftOver.
3.    int c, d;
4.    if (bitsLeftOver)
5.    {// half byte remains from before
6.       d = int(pcode & MASK1); // right ByteSize bits
7.       c = int((leftOver << EXCESS) | (pcode >> BYTE_SIZE));
8.       out.put(c);
9.       out.put(d);
10.      bitsLeftOver = false;
11.   }
12.   else
13.   {// no bits remain from before
14.      leftOver = pcode & MASK2; // right EXCESS bits
15.      c = int(pcode >> EXCESS);
16.      out.put(c);
17.      bitsLeftOver = true;
18.   }
19.}
```

MASK1 = 15 (1111)
EXCESS = 4
BYTE_SIZE = 8
MASK2 = 255 (11111111)

Take 8 right bits from 12-bit number

Combine leftover bits with 4 left bits from new 12-bit number

Chop 4 right bits off 12-bit number and save for later

Output 8 left bits

Compression - Initializing

```
1. hashChains<long, int> h(DIVISOR);
2.    for (int i = 0; i < ALPHA; i++)
3.       h.insert(pairType(i, i));
4.    int codesUsed = ALPHA;
5.
```

Note that ‘a’ is going to be something like 65 and not 0 as suggested by the earlier example.

Compression

```
1. int c = in.get(); // first character of input file
2. if (c != EOF) {// input file is not empty
3.    long pcode = c; // prefix code
4.    while ((c = in.get()) != EOF) {// process character c
5.       long theKey = (pcode << BYTE_SIZE) + c;
6.       // see if code for theKey is in the dictionary
7.       pairType* thePair = h.find(theKey);
8.       if (thePair == NULL) {// theKey is not in the table
9.          output(pcode);
10.         if (codesUsed < MAX_CODES) // create new code
11.            h.insert(pairType((pcode << BYTE_SIZE) | c,
                                codesUsed++));
12.         pcode = c;
13.      } else pcode = thePair->second; // theKey is in table
14.   }
15.
16.   // output last code(s)
17.   output(pcode);
18.   if (bitsLeftOver)
19.      out.put(leftOver << EXCESS);
```

Decompression

$$text(p) = text(q)c$$

```
1. typedef pair<int, char> pairType;
2. pairType ht[MAX_CODES];

3. void output(int code)
4. {// Output string corresponding to code.
5.    size = -1;
6.    while (code >= ALPHA)
7.    {// suffix in dictionary
8.       s[++size] = ht[code].second;
9.       code = ht[code].first;
10.   }
11.   s[++size] = code; // code < ALPHA
12.
13.   // decompressed string is s[size] ... s[0]
14.   for (int i = size; i >= 0; i--)
15.      out.put(s[i]);
16.}
```

Decompression

Turning bytes into 12-bit codes

```
1. bool getCode(int& code)
2. {// Put next code in compressed file into code.
3. // Return false if no more codes.
4.    int c, d;
5.    if ((c = in.get()) == EOF)
6.       return false; // no more codes
7.
8.    // see if any left over bits from before
9.    // if yes, concatenate with left over 4 bits
10.   if (bitsLeftOver)
11.      code = (leftOver << BYTE_SIZE) | c;
12.   else
13.   {// no left over bits, need four more bits
14.    // to complete code
15.    d = in.get(); // another 8 bits
16.    code = (c << EXCESS) | (d >> EXCESS);
17.    leftOver = d & MASK; // save 4 bits
18.   }
19.   bitsLeftOver = !bitsLeftOver;
20.   return true;
21.}
```

EXCESS = 4

MASK = 15 (1111)

Decompression

```
1. if (getCode(pcode)) {// file is not empty
2.     s[0] = pcode;    // character for pcode
3.     out.put(s[0]);  // output string for pcode
4.     size = 0; //s[size] is first character of last string
5.
6.     while(getCode(ccode)) {// there is another code
7.        if (ccode < codesUsed) {// ccode is defined
8.           output(ccode);
9.           if (codesUsed < MAX_CODES) {// create new code
10.             ht[codesUsed].first = pcode;
11.             ht[codesUsed++].second = s[size];
12.          }
13.       } else {// special case, undefined code
14.          ht[codesUsed].first = pcode;
15.          ht[codesUsed++].second = s[size];
16.          output(ccode);
17.       }
18.       pcode = ccode;
19.    }
20.}
```