CS 1501: Algorithm Implementation

LZW Data Compression

Data Compression

- Reduce the size of data
  - **Reduces** storage **space** and hence the **cost**
  - Reduces time to retrieve and transmit data

Compression ratio = original data size / compressed data size

Lossless and Lossy Compression

- compressedData = **compress**(originalData)
- decompressedData = **decompress**(compressedData)
- **Lossless** compression ➔ originalData = decompressedData
- **Lossy** compression ➔ originalData ≠ decompressedData

Lossless and Lossy Compression

- **Lossy compressors** generally obtain much higher compression ratios as compared to **lossless compressors**

  e.g. 100 vs. 2

- **Lossless compression** is essential in applications such as **text** file compression.

- **Lossy compression** is acceptable in many **audio** and **imaging** applications

  - In video transmission, a slight loss in the transmitted video is not noticable 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

Compression

```
public static void compress() {
    String input = BinaryStdIn.readString();
    TST<Integer> st = new TST<Integer>();
    for (int i = 0; i < R; i++)
        st.put("" + (char) i, i);
    int code = R+1;  // R is codeword for EOF
    while (input.length() > 0) {
        String s = st.longestPrefixOf(input);  // Find max prefix match s.
        BinaryStdOut.write(st.get(s), W);      // Print s's encoding.
        int t = s.length();
        if (t < input.length() && code < L)    // Add s to symbol table.
            st.put(input.substring(0, t + 1), code++);
        input = input.substring(t);            // Scan past s in input.
    }
    BinaryStdOut.write(R, W);
    BinaryStdOut.close();
}
```

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 to be compressed.

LZW Compression

code	0	1	2
key	a	b	ab

- Original text = **ababababbabaabbabbaabba**

- p = a   pCode = 0   Compressed text = 0
- c = b

- Enter p | c = ab into the code table.

code	0	1	2	3
key	a	b	ab	ba

- Original text = ababababbaabbabbaabba
- Compressed text = 0

- p = b pCode = 1 Compressed text = 01
- c = a
- Enter p | c = ba into the code table.

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

- Original text = abababbabaabbabbaabba
- Compressed text = 01

- p = ab pCode = 2 Compressed text = 012
- c = a
- Enter aba into the code table

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

- Original text = ababbbabaabbabbaabba
- Compressed text = 012

- p = ab   pCode = 2   Compressed text = 0122
- c = b

- Enter abb into the code table.

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 Compressed text = 01223
- c = b
- Enter bab into the code table.

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 Compressed text = 012233
- c = a

- Enter baa into the code table.

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   Compressed text = 0122335
- c = a
- Enter abba into the code table.

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 = 0122335

- p = abba pCode = 8 Compressed text = 01223358
- c = a
- Enter abbaa into the code table.

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

- Original text = ababababbabaabbabbaabba
- Compressed text = 01223358

- p = abba   pCode = 8   Compressed text = 012233588
- c = null

- No need to enter anything to the table

LZW Decompression

code	0	1
key	a	b

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588

- Convert codes to text from left to right.
- pCode=0 represents p=a DecompressedText = a

Expand

```
public static void expand() {
    String[] st = new String[L];
    int i; // next available codeword value
    // initialize symbol table with all 1-character strings
    for (i = 0; i < R; i++)
        st[i] = "" + (char) i;
    st[i++] = "";                    // (unused) lookahead for EOF
    int codeword = BinaryStdIn.readInt(W);
    if (codeword == R) return;       // expanded message is empty string
    String val = st[codeword];
    while (true) {
        BinaryStdOut.write(val);
        codeword = BinaryStdIn.readInt(W);
        if (codeword == R) break;
        String s = st[codeword];
        if (i == codeword) s = val + val.charAt(0);   // special case hack
        if (i < L) st[i++] = val + s.charAt(0);
        val = s;
    }
    BinaryStdOut.close();
}
```

LZW Decompression

code	0	1	2
key	a	b	ab

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588

- pCode=1 represents p = b   DecompressedText = ab
- lastP = a followed by first character of p is entered into the code table (a | b)

code	0	1	2	3
key	a	b	ab	ba

- Original text = ababbabaabbabbaabba
- Compressed text = 012233588

- pCode = 2 represents p=ab   DecompressedText = abab
- lastP = b followed by first character of p is entered into the code table (b | a)

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

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588

- pCode = 2 represents p = ab   DecompressedText = ababab.

- lastP = ab followed by first character of p is entered into the code table (ab | a)

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

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588

- pCode = 3 represents p = ba   DecompressedText = abababba.
- lastP = ab followed by first character of p is entered into the code table

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

- Original text = abababbabaabbabbaabba
- Compressed text = 012233588

- pCode = 3 represents p = ba DecompressedText = abababbaba.
- lastP = ba followed by first character of p is entered into the code table.

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

- Original text = ababababbabaabbabbaabba
- Compressed text = 012233588

- pCode = 5 represents p = abb DecompressedText = abababbabaabb.
- lastP = ba followed by first character of p is entered into the code table.

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

- Original text = ababababbabaabbabbaabba
- Compressed text = 012233588

- 8 represents ???
- When a code is not in the table (then, it is the last one entered), and its key is lastP followed by first character of lastP

- lastP = abb
- So 8 represents p = abba   Decompressed text = abababbabaabbabba

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

- Original text = ababababbabaabbabbaabba
- Compressed text = 012233588

- pCode= 8 represents p=abba DecompressedText=ababababbabaabbabbaabba.
- lastP = abba followed by first character of p is entered into the code table (abba | a)

Time Complexity

- Compression

  Expected time = **O(n)**

  where n is the length of the text that is being compressed.

- Decompression

  O(n), where n is the length of the decompressed text.