LZ78 Implementation

- Classification of Lossless Compression techniques
- Introduction to Lempel-Ziv Encoding: LZ77 & LZ78
- LZ78 Encoding Algorithm
- LZ78 Decoding Algorithm
CLASSIFICATION OF LOSSLESS COMPRESSION TECHNIQUES

Recall what we studied before:

- Lossless Compression techniques are classified into static, adaptive (or dynamic), and hybrid.
- Static coding requires two passes: one pass to compute probabilities (or frequencies) and determine the mapping, and a second pass to encode.
- **Examples of Static techniques:** Static Huffman Coding
- All of the adaptive methods are *one-pass* methods; only one scan of the message is required.
- **Examples of adaptive techniques:** LZ77, LZ78, LZW, and Adaptive Huffman Coding

INTRODUCTION TO LEMPEL-ZIV ENCODING

- Data compression up until the late 1970’s mainly directed towards creating better methodologies for Huffman coding.
- An innovative, radically different method was introduced in 1977 by Abraham Lempel and Jacob Ziv.
- This technique (called Lempel-Ziv) actually consists of two considerably different algorithms, LZ77 and LZ78.
- Due to patents, LZ77 and LZ78 led to many variants:

<table>
<thead>
<tr>
<th>LZ77 Variants</th>
<th>LZR</th>
<th>LZSS</th>
<th>LZB</th>
<th>LZH</th>
</tr>
</thead>
<tbody>
<tr>
<td>LZ78 Variants</td>
<td>LZW</td>
<td>LZC</td>
<td>LZT</td>
<td>LZMW</td>
</tr>
</tbody>
</table>

- The *zip* and *unzip* use the LZH technique while UNIX’s *compress* methods belong to the LZW and LZC classes.
LZ78 COMPRESSION ALGORITHM

LZ78 inserts one- or multi-character, non-overlapping, distinct patterns of the message to be encoded in a Dictionary.

The multi-character patterns are of the form: $C_0C_1 \ldots C_{n-1}C_n$. The prefix of a pattern consists of all the pattern characters except the last: $C_0C_1 \ldots C_{n-1}$.

**LZ78 Output:**

<table>
<thead>
<tr>
<th>Pattern Description</th>
<th>Code Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single character</td>
<td>$(0, \text{ char})$ if one-character pattern is not in Dictionary.</td>
</tr>
<tr>
<td>Multi-character pattern</td>
<td>$(\text{DictionaryPrefixIndex}, \text{ lastPatternCharacter})$ if multi-character pattern is not in Dictionary.</td>
</tr>
<tr>
<td>Last character pattern</td>
<td>$(\text{DictionaryPrefixIndex}, \text{ )}$ if the last input character or the last pattern is in the Dictionary.</td>
</tr>
</tbody>
</table>

Note: The dictionary is usually implemented as a hash table.

**LZ78 COMPRESSION ALGORITHM (CONT’D)**

```
Dictionary ← empty; Prefix ← empty; DictionaryIndex ← 1;
While (characterStream is not empty)
{
    Char ← next character in characterStream;
    if(Prefix + Char exists in the Dictionary)
        Prefix ← Prefix + Char;
    else
    {
        if(Prefix is empty)
            CodeWordForPrefix ← 0;
        else
            CodeWordForPrefix ← DictionaryIndex for Prefix;
        Output: (CodeWordForPrefix, Char);
        InsertInDictionary( ( DictionaryIndex, Prefix + Char ) );
        DictionaryIndex++;
        Prefix ← empty;
    }
}
if(Prefix is not empty)
{
    CodeWordForPrefix ← DictionaryIndex for Prefix;
    Output: (CodeWordForPrefix, );
}
```
EXAMPLE 1: LZ78 COMPRESSION

Encode (i.e., compress) the string \text{ABBCBCABABCAABCAAB} using the LZ78 algorithm.

The compressed message is: \text{(0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)}

Note: The above is just a representation, the commas and parentheses are not transmitted; we will discuss the actual form of the compressed message later on in slide 13.

EXAMPLE 1: LZ78 COMPRESSION (CONT’D)

1. \text{A} is not in the Dictionary; insert it
2. \text{B} is not in the Dictionary; insert it
3. \text{B} is in the Dictionary.
   \text{BC} is not in the Dictionary; insert it.
4. \text{B} is in the Dictionary.
   \text{BC} is in the Dictionary.
  \text{BCA} is not in the Dictionary; insert it.
5. \text{B} is in the Dictionary.
  \text{BA} is not in the Dictionary; insert it.
6. \text{B} is in the Dictionary.
  \text{BC} is in the Dictionary.
  \text{BCA} is in the Dictionary.
  \text{BCAA} is not in the Dictionary; insert it.
7. \text{B} is in the Dictionary.
  \text{BC} is in the Dictionary.
  \text{BCA} is in the Dictionary.
  \text{BCAA} is in the Dictionary.
  \text{BCAAB} is not in the Dictionary; insert it.
EXAMPLE 2: LZ78 COMPRESSION

Encode (i.e., compress) the string BABAABRRRA using the LZ78 algorithm.

The compressed message is: (0, B)(0, A)(1, A)(2, B)(0, R)(5, R)(2, )

EXAMPLE 2: LZ78 COMPRESSION (CONT’D)

1. B is not in the Dictionary; insert it
2. A is not in the Dictionary; insert it
3. B is in the Dictionary.
   BA is not in the Dictionary; insert it.
4. A is in the Dictionary.
   AB is not in the Dictionary; insert it.
5. R is not in the Dictionary; insert it.
6. R is in the Dictionary.
   RR is not in the Dictionary; insert it.
7. A is in the Dictionary and it is the last input character; output a pair containing its index: (2, )
EXAMPLE 3: LZ78 COMPRESSION

Encode (i.e., compress) the string `AAAAAAAAA` using the LZ78 algorithm.

1. A is not in the Dictionary; insert it
2. A is in the Dictionary
   AA is not in the Dictionary; insert it
3. A is in the Dictionary.
   AA is in the Dictionary.
   AAA is not in the Dictionary; insert it.
4. A is in the Dictionary.
   AA is in the Dictionary.
   AAA is in the Dictionary and it is the last pattern; output a pair containing its index: (3, )

```
Dictionary

<table>
<thead>
<tr>
<th>output</th>
<th>Index</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, A)</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>(1, A)</td>
<td>2</td>
<td>AA</td>
</tr>
<tr>
<td>(2, A)</td>
<td>3</td>
<td>AAA</td>
</tr>
<tr>
<td>(3, )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

LZ78 COMPRESSION: NUMBER OF BITS TRANSMITTED

- Example: Uncompressed String: `ABBCBCABABCAABCAAB`
  Number of bits = Total number of characters * 8
  = 18 * 8
  = 144 bits

- Suppose the codewords are indexed starting from 1:
  Compressed string (codewords): (0, A) (0, B) (2, C) (3, A) (2, A) (4, A) (6, B)
  Codeword index 1 2 3 4 5 6 7

Each code word consists of an integer and a character:
The character is represented by 8 bits.
The number of bits n required to represent the integer part of the codeword with index i is given by:

\[
n = \begin{cases} 
1 & \text{if } i = 1 \\
\left\lceil \log_2 i \right\rceil & \text{if } i > 1 
\end{cases}
\]

Alternatively number of bits required to represent the integer part of the codeword with index i is the number of significant bits required to represent the integer i – 1.
LZ78 COMPRESSON: NUMBER OF BITS TRANSMITTED (CONT'D)

<table>
<thead>
<tr>
<th>index</th>
<th>index - 1</th>
<th>bits</th>
<th>Number of significant bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>1101</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>

Codeword | (0, A) | (0, B) | (2, C) | (3, A) | (2, A) | (4, A) | (6, B) |
---------------------------
index | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
---------------------------

Bits: \((1 + 8) + (1 + 8) + (2 + 8) + (2 + 8) + (2 + 8) + (3 + 8) + (3 + 8) = 70 \text{ bits}\)

The actual compressed message is: 0A 0B 10C 11A 10A 100A 110B

where each character is replaced by its binary 8-bit ASCII code.

LZ78 DECOMPRESSION ALGORITHM

Dictionary \(\leftarrow\) empty; DictionaryIndex \(\leftarrow\) 1;
while(there are more (CodeWord, Char) pairs in codestream) {
    CodeWord \(\leftarrow\) next CodeWord in codestream;
    Char \(\leftarrow\) character corresponding to CodeWord;
    if(CodeWord = = 0)
        String \(\leftarrow\) empty;
    else
        String \(\leftarrow\) string at index CodeWord in Dictionary;
    Output: String + Char;
    insertInDictionary( (DictionaryIndex, String + Char) )
    DictionaryIndex++;
}

Summary:

- **Input:** (CW, character) pairs
- **Output:**
  - if(CW == 0)
    - output: currentCharacter
  - else
    - output: stringAtIndex CW + currentCharacter
- **Insert:** current output in dictionary
### EXAMPLE 1: LZ78 DECOMPRESSION

Decode (i.e., decompress) the sequence \((0, A) \ (0, B) \ (2, C) \ (3, A) \ (2, A) \ (4, A) \ (6, B)\)

<table>
<thead>
<tr>
<th>Output</th>
<th>Index</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>BC</td>
<td>3</td>
<td>BC</td>
</tr>
<tr>
<td>BCA</td>
<td>4</td>
<td>BCA</td>
</tr>
<tr>
<td>BA</td>
<td>5</td>
<td>BA</td>
</tr>
<tr>
<td>BCAA</td>
<td>6</td>
<td>BCAA</td>
</tr>
<tr>
<td>BCAAB</td>
<td>7</td>
<td>BCAAB</td>
</tr>
</tbody>
</table>

The decompressed message is: **ABBCBCABABCAABCAAB**

### EXAMPLE 2: LZ78 DECOMPRESSION

Decode (i.e., decompress) the sequence \((0, B) \ (0, A) \ (1, A) \ (2, B) \ (0, R) \ (5, R) \ (2, )\)

<table>
<thead>
<tr>
<th>Output</th>
<th>Index</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>BA</td>
<td>3</td>
<td>BA</td>
</tr>
<tr>
<td>AB</td>
<td>4</td>
<td>AB</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>R</td>
</tr>
<tr>
<td>RR</td>
<td>6</td>
<td>RR</td>
</tr>
<tr>
<td>A</td>
<td>7</td>
<td>A</td>
</tr>
</tbody>
</table>

The decompressed message is: **BABAABRRRA**
EXAMPLE 3: LZ78 DECOMPRESSION

Decode (i.e., decompress) the sequence (0, A) (1, A) (2, A) (3, )

<table>
<thead>
<tr>
<th>output</th>
<th>index</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>AA</td>
<td>2</td>
<td>AA</td>
</tr>
<tr>
<td>AAA</td>
<td>3</td>
<td>AAA</td>
</tr>
<tr>
<td>AAA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The decompressed message is: AAAAAAAAAA

EXERCISES

1. Use LZ78 to trace encoding the string SATATASACITASA.

2. Write a program that encodes a given string using LZ78.

3. Write a program that decodes a given set of encoded codewords using LZ78.