Lempel-Ziv Compression Techniques

- 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 in1977 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:

LZ77 Variants	LZR	LZSS	LZB	LZH		
LZ78 Variants	LZW	LZC	LZT	LZMW	LZJ	LZFG

- 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:

(0, char)	if one-character pattern is not in Dictionary.
(DictionaryPrefixIndex, lastPatternCharacter)	if multi-character pattern is not in Dictionary.
(DictionaryPrefixIndex,)	if the last input character or the last pattern is in the Dictionary.

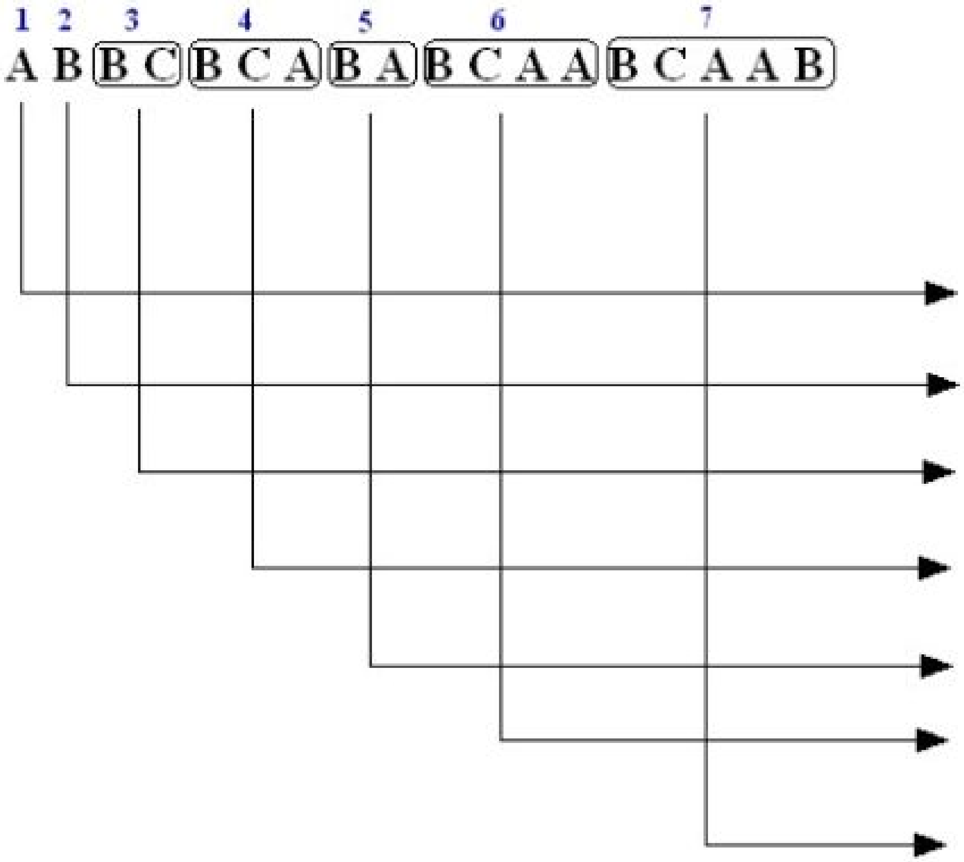
Note: The dictionary is usually implemented as a hash table.

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 **ABBCBCABABCAABCAAB** using the LZ78 algorithm.



	Dictionary	
output	index	string
(0, A)	1	A
(0, B)	2	B
(2, C)	3	BC
(3, A)	4	BCA
(2, A)	5	BA
(4, A)	6	BCAA
(6, B)	7	BCAAB

The compressed message is: **(0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)**

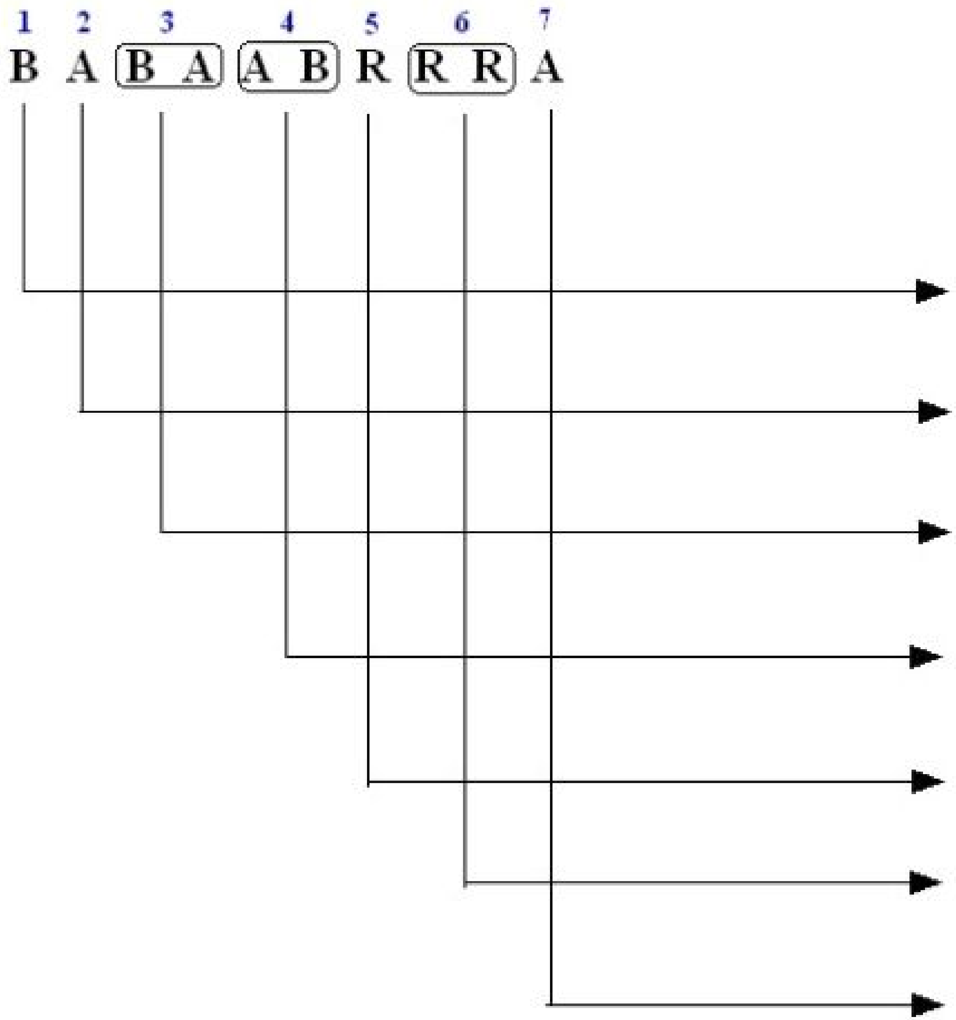
Note: The above is just a representation, the commas and parentheses are not transmitted; we will discuss the actual form of the compressed message later on in slide 12.

Example 1: LZ78 Compression (cont'd)

1. A is not in the Dictionary; insert it
2. B is not in the Dictionary; insert it
3. B is in the Dictionary.
BC is not in the Dictionary; insert it.
4. B is in the Dictionary.
BC is in the Dictionary.
BCA is not in the Dictionary; insert it.
5. B is in the Dictionary.
BA is not in the Dictionary; insert it.
6. B is in the Dictionary.
BC is in the Dictionary.
BCA is in the Dictionary.
BCAA is not in the Dictionary; insert it.
7. B is in the Dictionary.
BC is in the Dictionary.
BCA is in the Dictionary.
BCAA is in the Dictionary.
BCAAB is not in the Dictionary; insert it.

Example 2: LZ78 Compression

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



	Dictionary	
output	index	string
(0, B)	1	B
(0, A)	2	A
(1, A)	3	BA
(2, B)	4	AB
(0, R)	5	R
(5, R)	6	RR
(2,)		

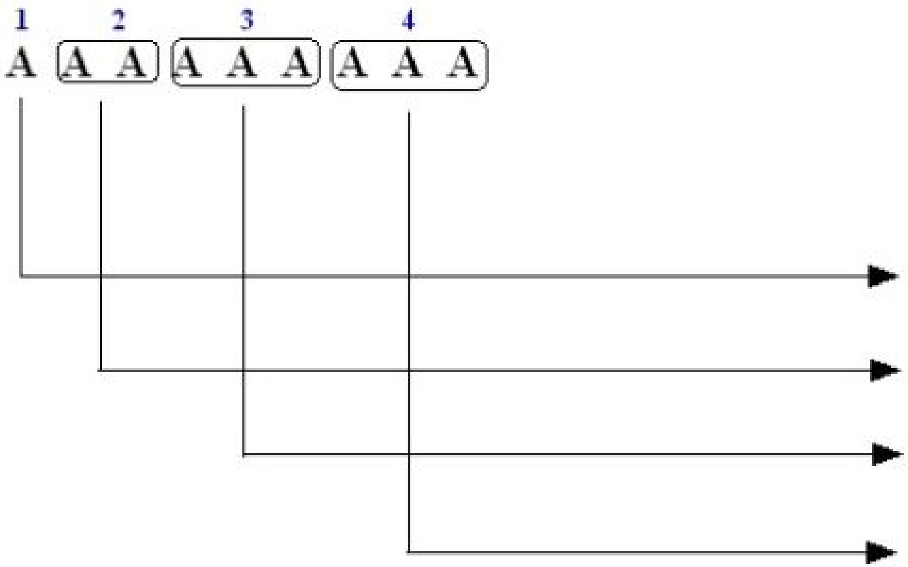
The compressed message is: (0,B)(0,A)(1,A)(2,B)(0,R)(5,R)(2,)

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.



	Dictionary	
output	**index**	**string**
(0, A)	1	A
(1, A)	2	AA
(2, A)	3	AAA
(3,)		

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,)

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:

$$n = \begin{cases} 1 & \text{if } \quad i = 1 \\ \lceil \log_2 i \rceil & \text{if } i > 1 \end{cases}$$

- 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 Compression: Number of bits transmitted (cont'd)

index	index - 1	bits	Number of significant bits
1	0	0	1
2	1	1	
3	2	10	2
4	3	11	
5	4	100	3
6	5	101	
7	6	110	
8	7	111	
9	8	1000	4
10	9	1001	
11	10	1010	
12	11	1011	
13	12	1100	
14	13	1101	
15	14	1110	
16	15	1111	

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) + (3 + 8) + (3 + 8) + (3 + 8) = 71 bits

The actual compressed message is: 0A0B10C11A010A100A110B

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

LZ78 Decompression Algorithm

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

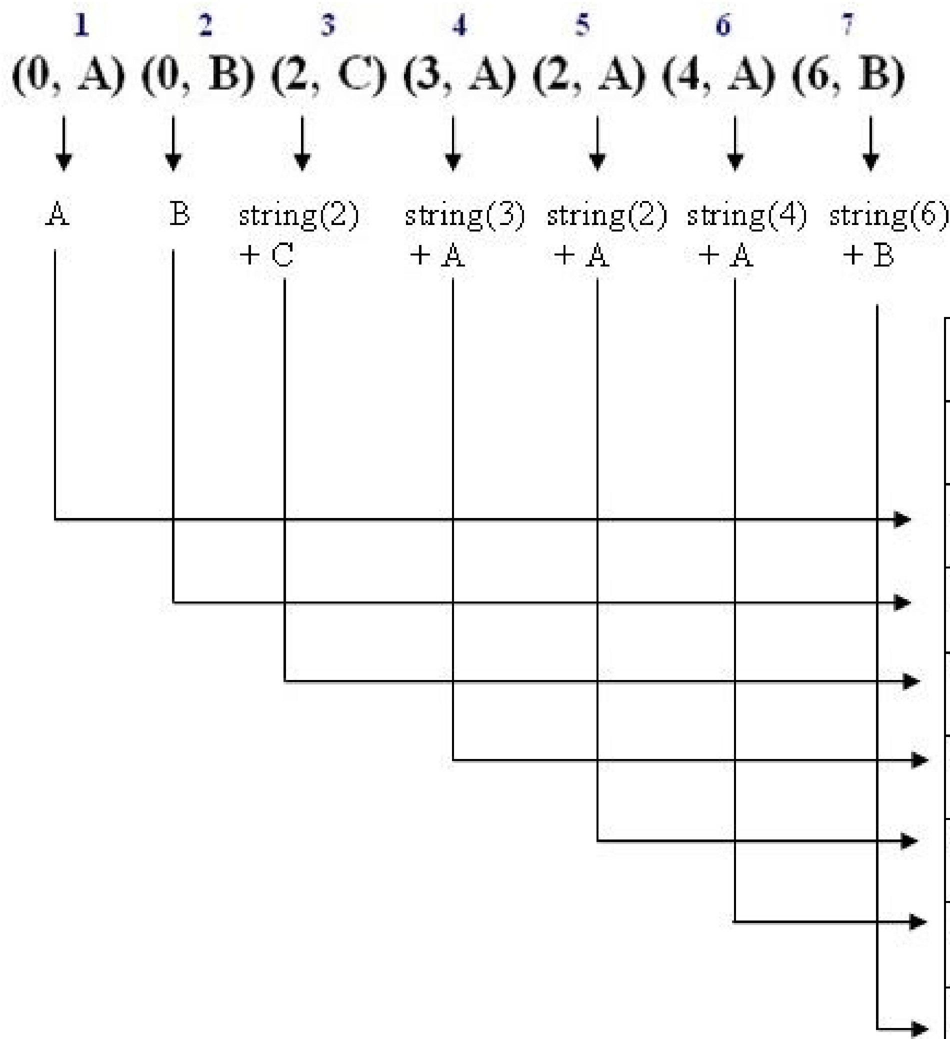
Summary:

- **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)



	Dictionary	
output	index	string
A	1	A
B	2	B
BC	3	BC
BCA	4	BCA
BA	5	BA
BCAA	6	BCAA
BCAAB	7	BCAAB

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,)

	Dictionary	
output	index	string
B	1	B
A	2	A
BA	3	BA
AB	4	AB
R	5	R
RR	6	RR
A		

The decompressed message is: BABAABRRRA

Example 3: LZ78 Decompression

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

	Dictionary	
output	index	string
A	1	A
AA	2	AA
AAA	3	AAA
AAA		

The decompressed message is: AAAAAAAAA

Exercises

1. Use LZ78 to trace encoding the string

SATATASACITASA.

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

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