Introduction to Dictionary Coding

Dr. Arup Kumar Pal
Department of Computer Science & Engineering
Indian School of Mines, Dhanbad
Jharkhand-826004
E-mail: arupkrpal@gmail.com

Dictionary Coding

- Construct a list of commonly occurring patterns and encode these patterns by transmitting their index in the list
- In general, dictionary-based techniques works well for highly correlated data (e.g. text), but less efficient for data with low correlation (e.g. i.i.d. sources)

Motivation behind Dictionary Coding

- Consider an 'English' source with 26 letters & six punctuation marks
  - Single-symbol FLC, fixed-length encoding: 5 bps
  - Four-symbol FLC, fixed-length encoding: 20 bps (32^4)
- If we assume uneven distribution of the symbols
  - Pick a dictionary witch contains the 256 most-frequent patterns (probability p) *and encode them with 8 bits*
  - Encode the rest with 20 bits
  - Use 1-bit prefix to distinguish the two cases
- Then, the average rate is $9p + 21(1 - p) = 21 - 12p$.

  If $p > 0.084$, $21 - 12p < 20$.

Categorization of Dictionary-Based Coding

- The heart of dictionary coding is the formulation of the dictionary.
- A successfully built dictionary results in data compression; the opposite case may lead to data expansion.
- According to the ways in which dictionaries are constructed, dictionary coding techniques can be classified as **static or adaptive**.

Static Dictionary Coding

- A fixed dictionary,
  - Produced before the coding process
  - Used at both the transmitting and receiving ends
  - It is possible when the knowledge about the source alphabet and the related strings of symbols, also known as phrases, is sufficient.
- Merit of the static approach: its simplicity.
- Its drawbacks lie on
  - Relatively lower coding efficiency
  - Less flexibility compared with adaptive dictionary techniques

Static Dictionary Coding

- An example of static algorithms occurs is ***diagram*** coding.
  - A simple and fast coding technique.
  - The dictionary contains:
    - all source symbols and
    - some frequently used pairs of symbols.
  - In encoding, two symbols are checked at once to see if they are in the dictionary.
    - If so, they are replaced by the index of the two symbols in the dictionary, and the next pair of symbols is encoded in the next step.

Static Dictionary Coding

- If not, then the index of the first symbol is used to encode the first symbol. The second symbol is combined with the third symbol to form a new pair, which is encoded in the next step.

- The diagram can be straightforwardly extended to ***n-gram***. In the extension, the size of the dictionary increases and so is its coding efficiency.

Static Dictionary Coding

Suppose we have a source with a five-letter alphabet $\mathcal{A} = \{a, b, c, d, r\}$. Based on knowledge about the source, we build the dictionary shown in Table 5.1.

TABLE 5.1 A sample dictionary.

Code	Entry	Code	Entry
000	*a*	100	*r*
001	*b*	101	*ab*
010	*c*	110	*ac*
011	*d*	111	*ad*

Suppose we wish to encode the sequence

abracadabra

101 100 110 111 101 100 000

Sliding Window (LZ77) Algorithms

- **Introduction**
  - The dictionary used is actually a portion of the input text, which has been recently encoded.
  - The text that needs to be encoded is compared with the strings of symbols in the dictionary.
  - The longest matched string in the dictionary is characterized by a *pointer* (sometimes called a *token*), which is represented by a triple of data items.
  - Note that this triple functions as an index to the dictionary.
  - In this way, a variable-length string of symbols is mapped to a fixed-length pointer.

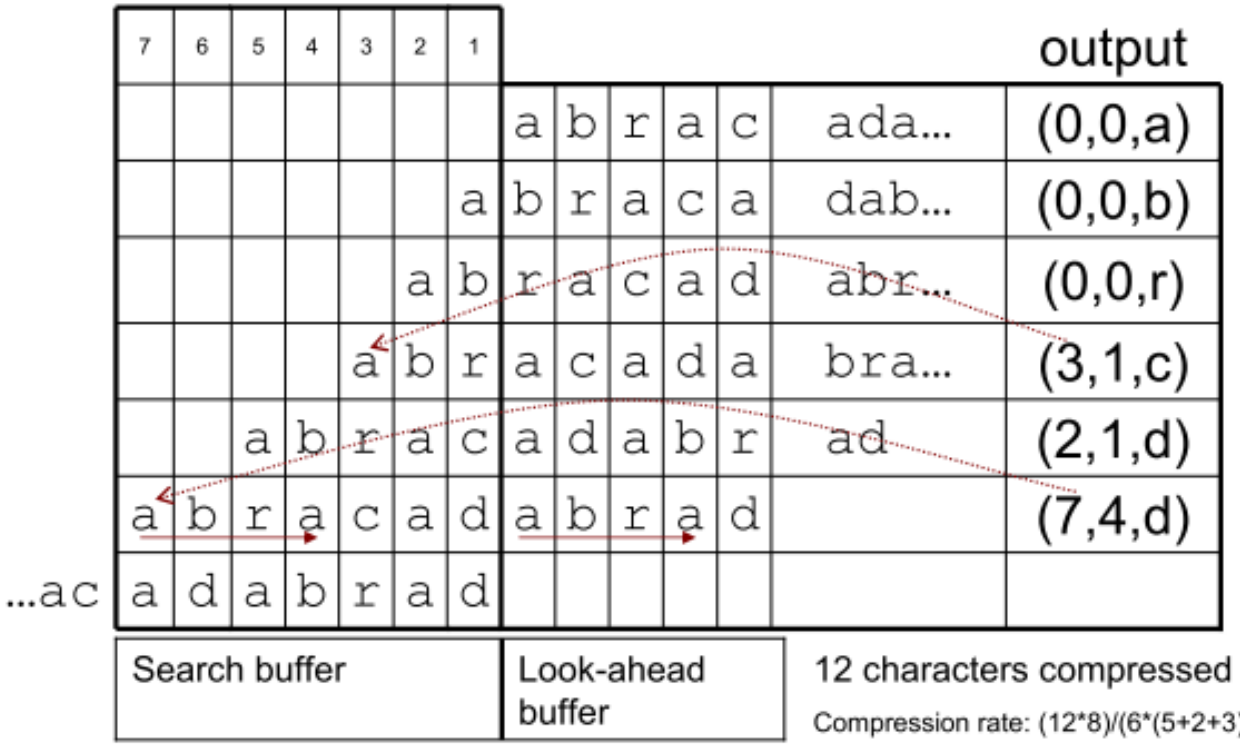
Introduction

- There is a sliding window in the LZ77 algorithms. The window consists of two parts: a search buffer and a look-ahead buffer.
  - The search buffer contains: the portion of the text stream that has recently been encoded --- the dictionary.
  - The look-ahead buffer contains: the text to be encoded next.
- The window slides through the input text stream from beginning to end during the entire encoding process.
- The size of the search buffer is much larger than that of the look-ahead buffer.
  - The sliding window: usually on the order of a few thousand symbols.
  - The look-ahead buffer: on the order of several tens to one hundred symbols.

LZ77 – Example (Encoding)

Encoding of the string:
abracadabrad

output tuple: (offset, length, symbol)



12 characters compressed into 6 tuples

Compression rate: (12*8)/(6*(5+2+3))=96/60=1,6=60%.

Example (Decoding)

input		7	6	5	4	3	2	1
(0,0,a)								a
(0,0,b)							a	b
(0,0,r)						a	b	r
(3,1,c)				a	b	r	a	c
(2,1,d)		a	b	r	a	c	a	d
(7,4,d)	abrac	a	d	a	b	r	a	d

Summary of the LZ77 Approach

- The first symbol in the look-ahead buffer is the symbol or the beginning of a string of symbols to be encoded at the moment. Let us call it the symbol *s*.
- In encoding, the search pointer moves to the left, away from the symbol *s*, to find a match of the symbol *s* in the search buffer. When there are multiple matches, the match that produces the longest matched string is chosen. The match is denoted by a triple <i, j, k>.

Summary of the LZ77 Approach

- The first item in the triple, “i”, is the **offset**, which is the distance between the pointer pointing to the symbol giving the maximum match and the symbol *s*.
- The second item, “j”, is the **length** of the matched string.
- The third item, “k”, is the codeword of the symbol following the matched string in the look-ahead buffer.

- At the very beginning, the content of the search buffer can be arbitrarily selected. For instance, the symbols in the search buffer may all be the space symbol.

Summary of the LZ77 Approach

- The limitation of the approach is that if the distance between the repeated patterns in the input text stream is larger than the size of the search buffer, then the approach cannot utilize the structure to compress the text.

LZ78 Algorithm

- **Limitations with LZ77:**
  - If the distance between two repeated patterns is larger than the size of the search buffer, then the LZ77 algorithms cannot work efficiently.
  - It cannot recognize the patterns occurring some time ago because they may have been shifted out from the buffer. In this situation, the patterns are ignored and no matches are found.
  - Increasing the sizes of the search buffer and the look-ahead buffer seemingly will resolve the problems. A close look, however, reveals that it also leads to increases in the number of bits required to encode the offset and matched string length as well as an increase in processing complexity.

LZ78 Algorithm

- LZ78 are developed to maintain a dictionary that allow patterns to remain as entries permanently during the whole encoding process.

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.

Example: LZ78 Compression

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

ABBCBCABABCAABCAAB

1

A

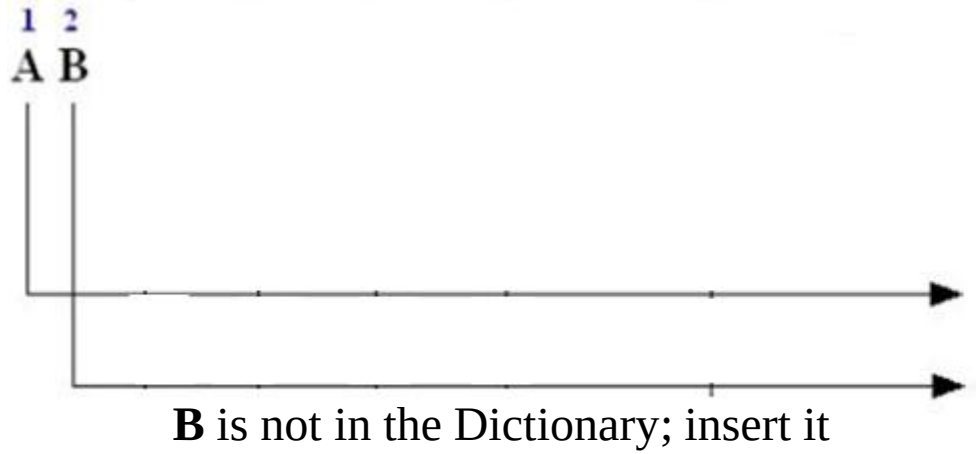
A is not in the Dictionary; insert it

	Dictionary	
output	index	string
(0, A)	1	A

Example: LZ78 Compression

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

ABBCBCABABCAABCAAB



	Dictionary	
output	index	string
(0, A)	1	A
(0, B)	2	B

B is not in the Dictionary; insert it

Example: LZ78 Compression

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

ABBCBCABABCAABCAAB

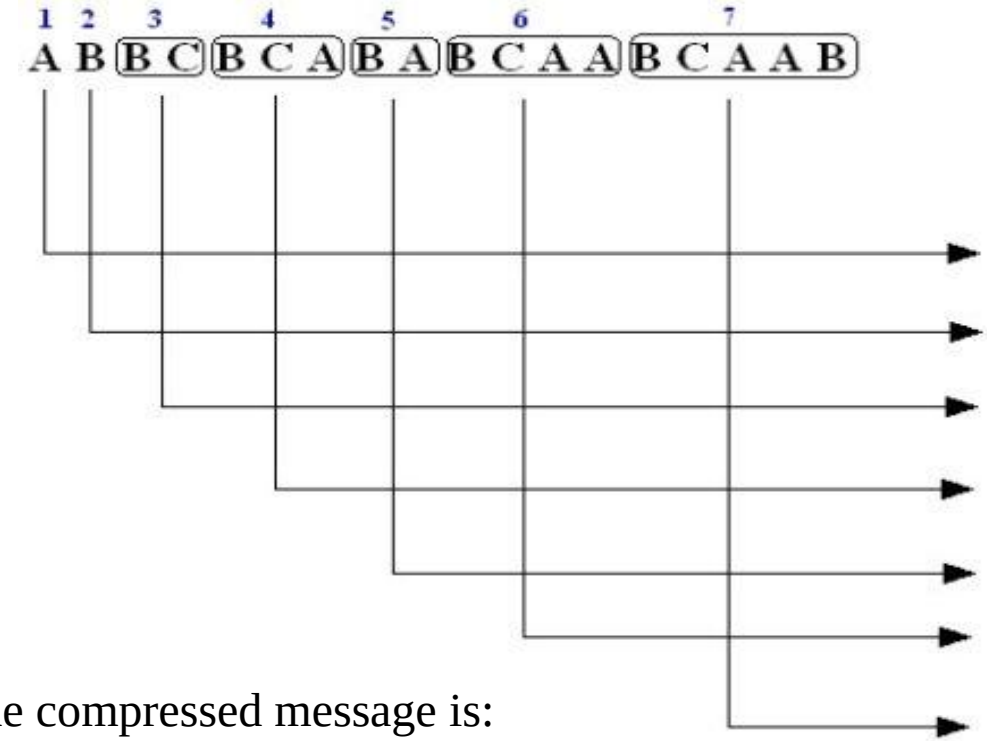
	Dictionary	
output	index	string
(0, A)	1	A
(0, B)	2	B
(2, C)	3	BC

B is in the Dictionary.
BC is not in the Dictionary; insert it.

Example: LZ78 Compression

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

ABBCBCABABCAABCAAB



	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;

LZ78 Compression: Number of bits transmitted

Uncompressed String: **ABBCBCABABCAABCAAB**

Number of bits = Total number of characters ×8

$= 18 \times 8$

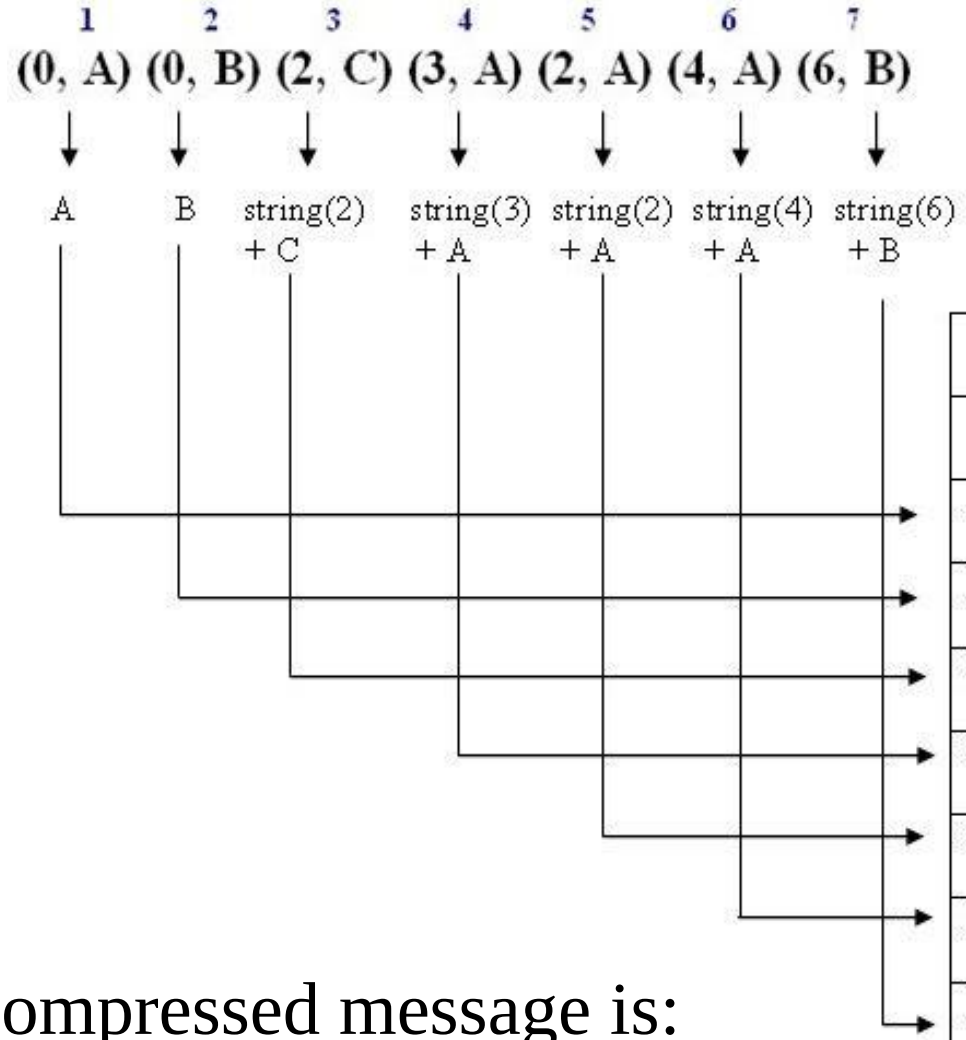
$= 144$ bits

Compressed string(codewords):

(0, A) (0, B) (2, C) (3, A) (2, A) (4, A) (6, B)

- Each code word consists of an integer and a character:
  - The character is represented by **8** bits.
  - The integer part of the codeword is represented by **3** bits.
  - Total Bit=11×7=77bits

LZ78 Decompression



	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

Summary of the LZ78 Approach

- No use of the sliding window.
- **Instead of the triples used in the LZ77, only pairs are used in the LZ78.**
- In LZ78, only the position of the pointer to the matched string and the symbol following the matched string need to be encoded.
- In theory the dictionary can keep the patterns forever after they have been seen once. In practice, the size of the dictionary cannot grow indefinitely. Some patterns may need to be reinstalled when the dictionary is full.

The LZW Algorithm

- LZW uses fixed-length indices to represent variable-length strings of symbols/characters that commonly occur together, e.g., words in English text.
- The LZW encoder and decoder build up the same dictionary dynamically while receiving the data.
- LZW places longer and longer repeated entries into a dictionary, and then emits the code for an element, rather than the string itself, if the element has already been placed in the dictionary.

Introduction to LZW

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries.
- Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks.
- Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code book.

LZW Encoding Algorithm

Repeat not end of input stream

Step 1: Find the longest match **w** in the dictionary

Step 2: Output the index of **w**

Step 3: Put **w+n** in the dictionary where **n** was the unmatched symbol

Example: Compression using LZW

Use the LZW algorithm to compress the string

BABAABAAA

ENCODER	OUTPUT	STRING	TABLE
output code	representing	Index	Entry
		1	A
		2	B

Example 1: LZW Compression

BABAABAAA

ENCODER	OUTPUT	STRING	TABLE
output code	representing	Index	Entry
		1	A
		2	B
2	B	3	BA

Example 1: LZW Compression

BABAABAAA

ENCODER	OUTPUT	STRING	TABLE
output code	representing	Index	Entry
		1	A
		2	B
2	B	3	BA
1	A	4	AB

Example 1: LZW Compression

BABAABAAA

ENCODER	OUTPUT	STRING	TABLE
output code	representing	Index	Entry
		1	A
		2	B
2	B	3	BA
1	A	4	AB
3	BA	5	BAA

Example 1: LZW Compression

BABAABAAA

ENCODER	OUTPUT	STRING	TABLE
output code	representing	Index	Entry
		1	A
		2	B
2	B	3	BA
1	A	4	AB
3	BA	5	BAA
4	AB	6	ABA

Example 1: LZW Compression

BABAABAAA

ENCODER OUTPUT		STRING TABLE	
output code	representing	Index	Entry
		1	A
		2	B
2	B	3	BA
1	A	4	AB
3	BA	5	BAA
4	AB	6	ABA
1	A	7	AA

Example 1: LZW Compression

BABAABAAA

ENCODER OUTPUT		STRING TABLE	
output code	representing	Index	Entry
		1	A
		2	B
2	B	3	BA
1	A	4	AB
3	BA	5	BAA
4	AB	6	ABA
1	A	7	AA
7	AA		

LZW Decoding Algorithm

Initialize dictionary
Decode first index to w
    Repeat not end of indices
    {
        Step 1: Decode next index to S
        Step 2: Add w with first character of S to the Dictionary
        Step 3: Update w by S
    }

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417 **w = B**

DECODER	OUTPUT	STRING	TABLE
Index	**representing**	**Index**	**Entry**
		1	**A**
2	**B**	**2**	**B**

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417

w = B S=A

DECODER	OUTPUT	STRING	TABLE
Index	representing	Index	Entry
		1	A
2	B	2	B
1	A	3	BA

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417

w = A S=BA

DECODER	OUTPUT	STRING	TABLE
Index	representing	Index	Entry
		1	A
2	B	2	B
1	A	3	BA
3	BA	4	AB

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417

w = BA S=AB

DECODER OUTPUT		STRING TABLE	
Index	**representing**	**Index**	**Entry**
		1	A
2	B	2	B
1	A	3	BA
3	BA	4	AB
4	AB	5	BAA

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417

w = AB S=A

DECODER OUTPUT		STRING TABLE	
Index	**representing**	**Index**	**Entry**
		1	A
2	B	2	B
1	A	3	BA
3	BA	4	AB
4	AB	5	BAA
1	A	6	ABA

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417 **w = A S=?**

DECODER OUTPUT		STRING TABLE	
Index	representing	Index	Entry
		1	A
2	B	2	B
1	A	3	BA
3	BA	4	AB
4	AB	5	BAA
1	A	6	ABA
7		7	A?

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417

w = A S=A

DECODER OUTPUT		STRING TABLE	
Index	**representing**	**Index**	**Entry**
		1	A
2	B	2	B
1	A	3	BA
3	BA	4	AB
4	AB	5	BAA
1	A	6	ABA
7		7	A?

Example: Decoding using LZW

Use the LZW algorithm to compress the string

213417

w = A S=A

DECODER OUTPUT		STRING TABLE	
Index	representing	Index	Entry
		1	A
2	B	2	B
1	A	3	BA
3	BA	4	AB
4	AB	5	BAA
1	A	6	ABA
7	AA	7	AA

LZW Compression

Observation

1. The compression algorithm and the decompression algorithm build an identical dictionary independently. The advantage of this is that the compression algorithm does not have to pass the dictionary to the decompressor.
2. The size of the dictionary may grow so quickly that an effective method of maintaining the dictionary would be essential.