

Multimedia Communications

Dictionary Coding



Dictionary Coding

- In many applications, output of source consists of recurring patterns.
- Approach: keep a list (dictionary) of frequently occurring patterns
- Encode a pattern in the dictionary by an index.
- If the pattern does not appear in the dictionary, it is coded using some other (less efficient) method
- Do not require knowledge of the probability distribution of the input.
- Variations (Jacob Ziv, Abraham Lempel, Terry Welch)
 - LZ77 (PKZip, LHarc, ARJ, zip, gzip)
 - LZ78
 - LZW (compress, arc, V.42bis)

Dictionary Coding

- How do different dictionary coders differ?
 - How is the dictionary built?
 - How is the dictionary search performed?
 - How is an index encoded?
 - What happens if a match is not found in the dictionary?

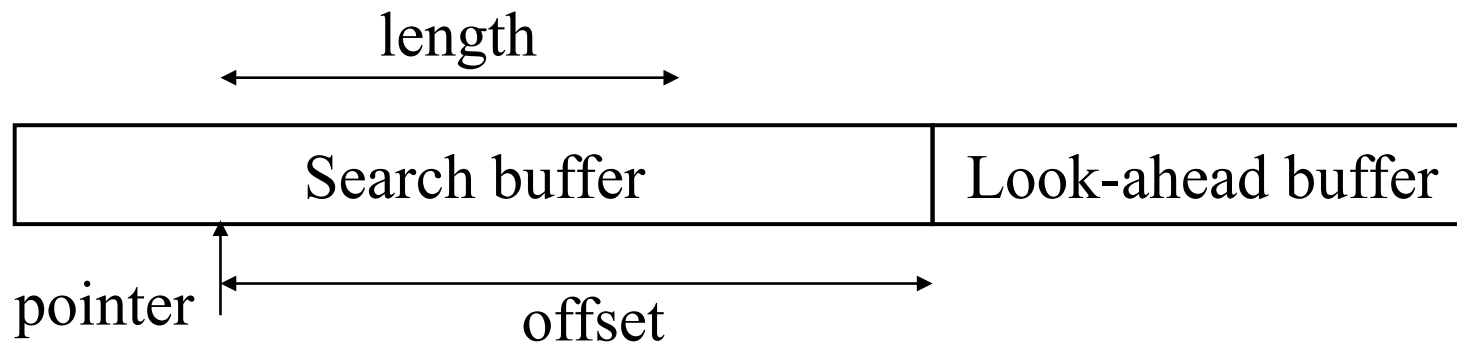
LZ77

- Dictionary is a portion of previously encoded sequence
- Encoder examines the input sequence through a sliding window
- The window has two parts: search buffer and look-ahead buffer
- Search buffer: a portion of recently encoded sequence
- Look-ahead buffer: next portion of the sequence to be coded



LZ77

- To encode a sequence in the look-ahead buffer, the encoder moves a search pointer through the search buffer until it finds the longest match



- Once the longest match has been found, encoder emits a triplet $\langle o, l, c \rangle$ where o = offset; l = length; c = code word of symbol following the match.
- The match might exceed the length of the search buffer and enters the look-ahead buffer

LZ77: Example

...cabracadabrarrarrad...

Encoder: 3 situations

1. No match for the next character,
transmit $\langle 0,0,c(d) \rangle$

c a b r a c a	d a b r a r
---------------	-------------

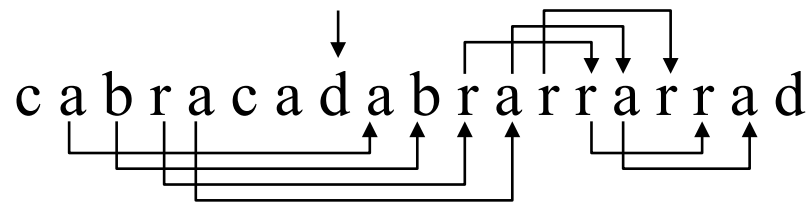
2. Match completely included in
search buffer $\langle 7,4,c(r) \rangle$

a b r a c a d	a b r a r r
---------------	-------------

3. Match starts in search buffer and
ends in look-ahead buffer. $\langle 3,5,c(d) \rangle$

a d a b r a r	r a r r a d
---------------	-------------

Decoder:

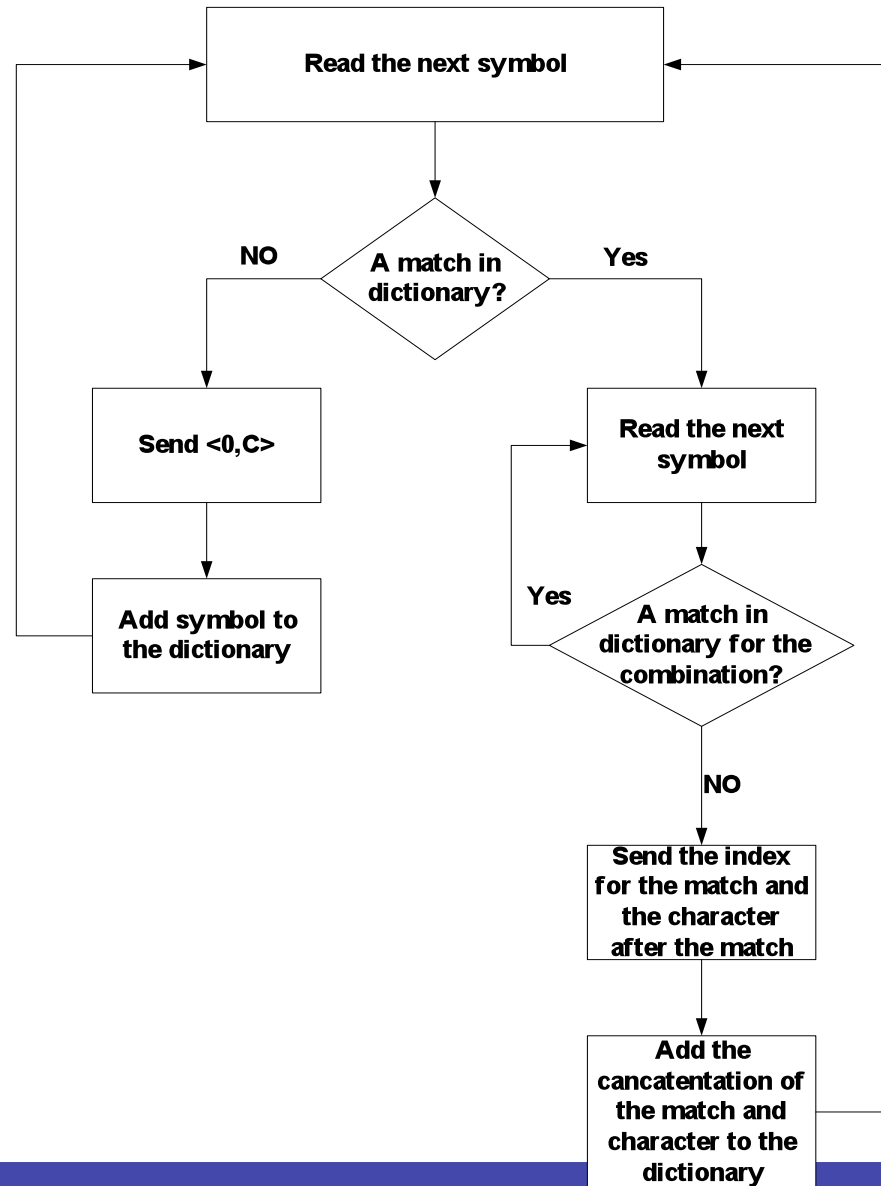


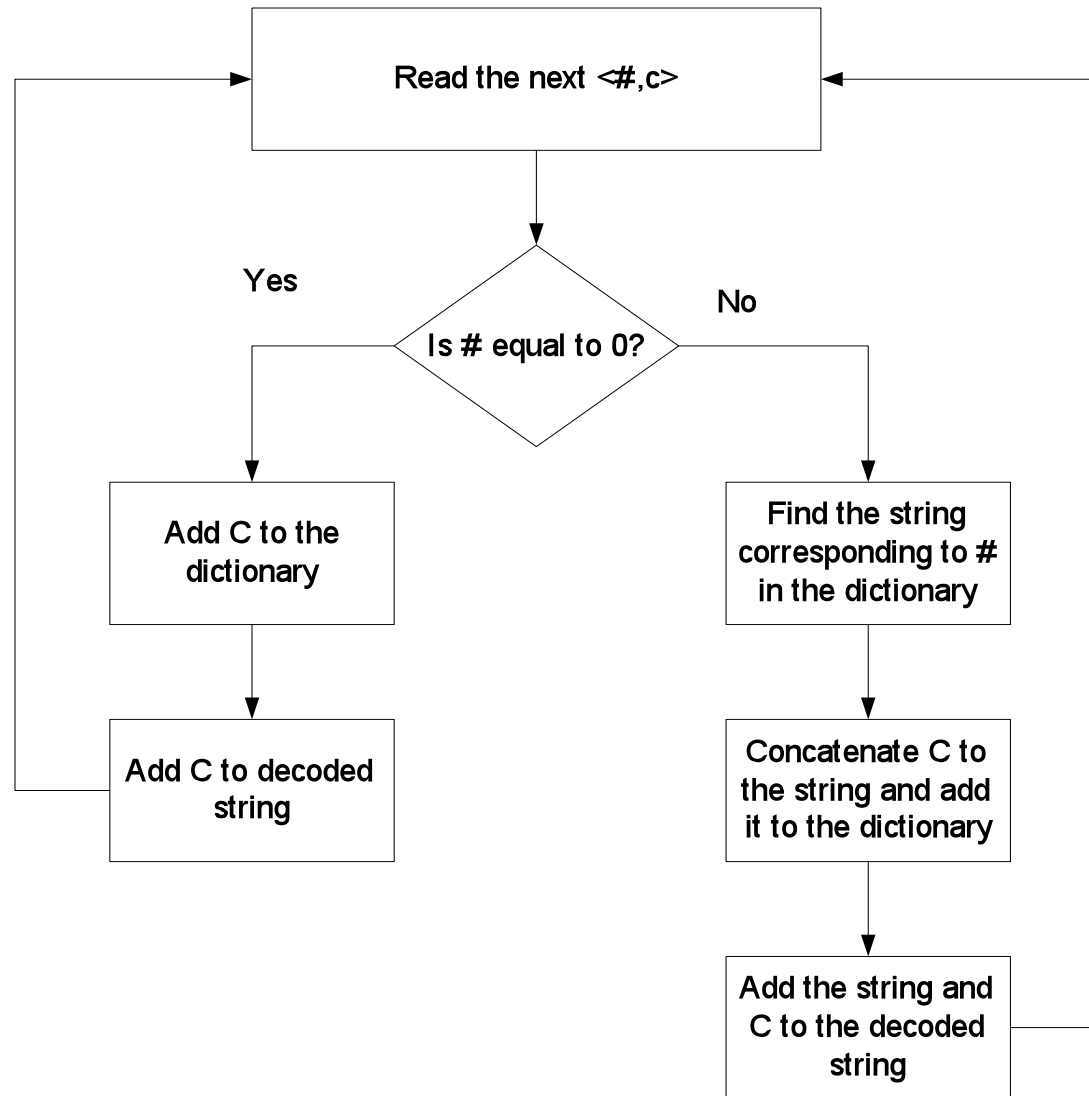
LZ77: Problems

- Patterns should occur close together
 - This assumption was removed in LZ78.
- The coding of $\langle o, l, c \rangle$ is not efficient \rightarrow this is solved in practical implementations (PKZip, LHarc, ARJ) by variable-length coding of the triplet.
- Performance increases with the size of the buffers. Large buffers require efficient search strategies.
- Using a triplet to encode a single character is inefficient. LZSS uses a flag bit to signal if a single character is sent.
- By using the flag bit, we get rid of the third element of the triplet

LZ78

- **Encoder:** Makes use of an explicit dictionary.
- Phrases in the dictionary are built one at a time, adding a new symbol to an existing phrase when a match occurs.
- Inputs are coded as $\langle i, c \rangle$, where
 - i = index in the dictionary to the longest matching phrase.
 - c = code word of symbol following the match.
 - index value of 0 is used in the case of no match
- **Decoder:** The dictionary is built in a similar way.
- Problem: dictionary keeps growing (dictionary management).





LZ78: Example

“wabba_wabba_wabba_wabba_woo_woo_woo”

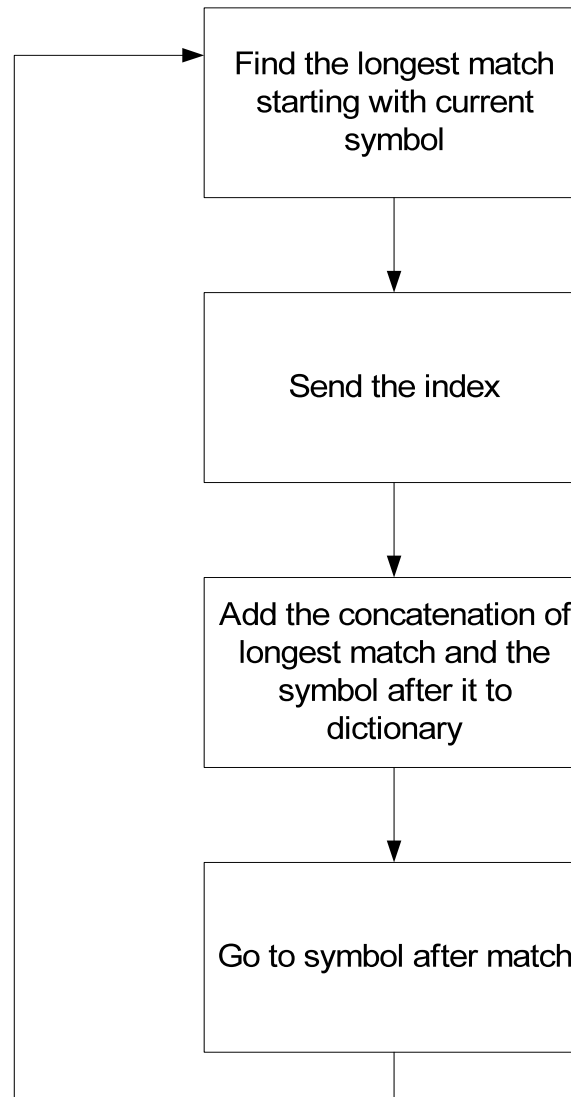
1. $\langle 0, c(w) \rangle$	10. $\langle 4, c(_) \rangle$	1. w	10. ba_
2. $\langle 0, c(a) \rangle$	11. $\langle 9, c(b) \rangle$	2. a	11. wabb
3. $\langle 0, c(b) \rangle$	12. $\langle 8, c(w) \rangle$	3. b	12. a_w
4. $\langle 3, c(a) \rangle$	13. $\langle 0, c(o) \rangle$	4. ba	13. o
5. $\langle 0, c(_) \rangle$	14. $\langle 13, c(_) \rangle$	5. _	14. o_
6. $\langle 1, c(a) \rangle$	15. $\langle 1, c(o) \rangle$	6. wa	15. wo
7. $\langle 3, c(b) \rangle$	16. $\langle 14, c(w) \rangle$	7. bb	16. o_w
8. $\langle 2, c(_) \rangle$	17. $\langle 13, c(o) \rangle$	8. a_	17. oo
9. $\langle 6, c(b) \rangle$		9. wab	

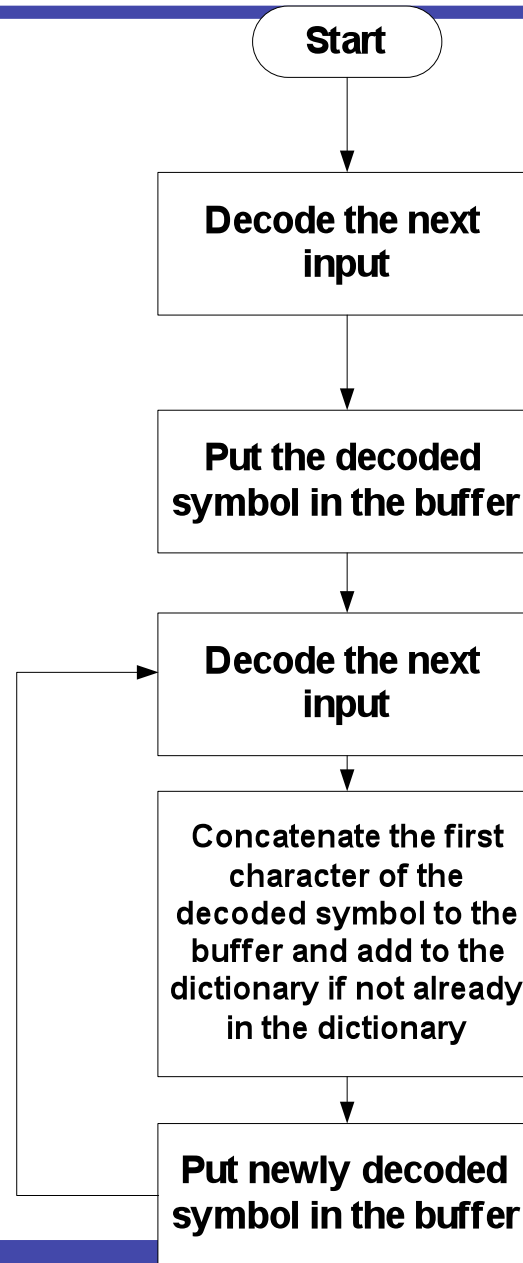
LZW

- LZW is based on LZ78.
- In LZW in the pair $\langle i, c \rangle$, c is not coded. The encoder and decoder dictionaries are initialized with letters of the alphabet.
- The dictionary is expanded by adding new entries
- As the length of matched phrases increases; the dictionary captures increasingly more of the structure of the sequence.

LZW

- Encoder:
 - 1. Find largest match “m”, ‘c’ follows
 - 2. Encode match “m”
 - 3. Add (“m”, ‘c’) to dictionary
 - 4. Repeat from 1, starting with ‘c’.
- Decoder:
 - 1. Decode and output pattern
 - 2. Add (previous pattern, first symbol in current pattern)
 - 3. Go to 2.





LZW: Example 1

Encode “wabba_wabba_wabba_wabba_woo_woo_woo”

Output: 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4

Dictionary:

1. _	6. wa	11. _w	16. ba_	
	21. oo			
2. a	7. ab	12. wab	17. _wa	22. o_
3. b	8. bb	13. bba	18. abb	
	23. _wo			
4. o	9. ba	14. a_w	19. ba_w	24. oo_
5. w	10. a_	15. wabb	20. wo	25. _woo

LZW: Example 2

- Encode: “abababab...”. Encoded: 1235...

Encoder dictionary

1. a

2. b

3. ab

4. ba

5. aba

6. abab

7. b...

Decoder dictionary

1. a

2. b

3. ab

4. ba

5. aba

6. abab

7. ...

The LZW decoder must handle incomplete dictionary entries.

LZW

- Looking back at the coding stage, it can be seen that the exceptional case occurs when the newest dictionary entry is used as the output.
- Two ways to solve this problem:
 1. Exceptional handler: if encounter an undefined code add the first character of the string in the buffer to the end of this string and put as a new entry in the dictionary
 2. Avoid exceptional case: the newest entry of the dictionary is not considered as output.
 - Using this approach in the previous example, instead of transmitting codeword 5, the encoder will send codewords 3 and 1.

LZW: Applications

Unix “compress”

- Adaptive dictionary size: $2^9 \dots 2^{16}$. So we start with 9-bit codewords and increase code word size as dictionary increases.
- Once dictionary reaches maximum size, it becomes a static dictionary.
- Dictionary is flushed (reinitialized) if compression performance degrades.

LZW: Applications

GIF graphics format

- First byte = number of bits/pixel.
- Consider grayscale images, $b = 8$.
- Initial dictionary size = $2^{b+1} = 512$. Can increase to 4096.
- The code 2^b corresponds to the “clear code” which is used to reset all the coding parameters to the initial state.
- Works well with computer-generated images.

LZW: Applications

V.42bis

- ITU-T recommendation for modem communications.
- Transparent and compressed (LZW) modes.
- Standard suggests periodic testing of compression performance to see if transparent mode should be used.
- Dictionary size: negotiated: minimum 512, recommended 2048.