Dictionary Techniques (1)

∗ keep a list, or dictionary, of frequently occurring patterns

∗ encode those patterns with a reference (index) to the dictionary, respectively.

Example : A text consists of four-character words with the source alphabet

A = {"**a**", "**b**", …, "**z**", "**,**" , "**.**" , "**!**" , "**?**" , "**;**" , "**: "**}

①Fixed-length coding

$32^4 = 2^{20}$ patterns $\Rightarrow$ 20 bits /word

②Put 256 = 2^8 most frequently occurring words into a dictionary

➢If a word is in the dictionary (with probability p)

(1-bit flag) + (8- bit index) = 9 bits

➢Else (1-bit flag) + (20- bit code) = 21 bits

$R = 9p + 21(1-p) = 21 - 12p$ bits/word $< 20 \quad \Rightarrow \quad p \geq 0.084$

♠ **Static dictionary**

- the dictionary is permanent (or allowing addition but no deletion)
- application-specific or data-specific

Dictionary Techniques (2)

Example :Digram coding

- input : abracadabra

- output : 101100110111101100000

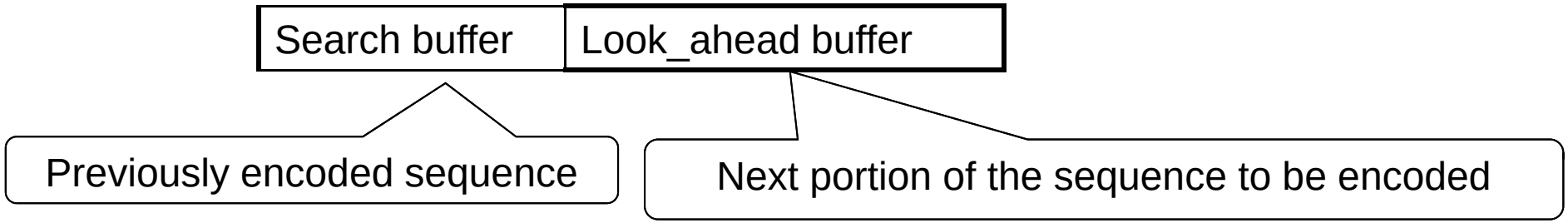
code	entry	code	entry
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

♠ **Adaptive dictionary**

- start with an empty (or alphabet only) dictionary
- add entries as they are found in the input stream

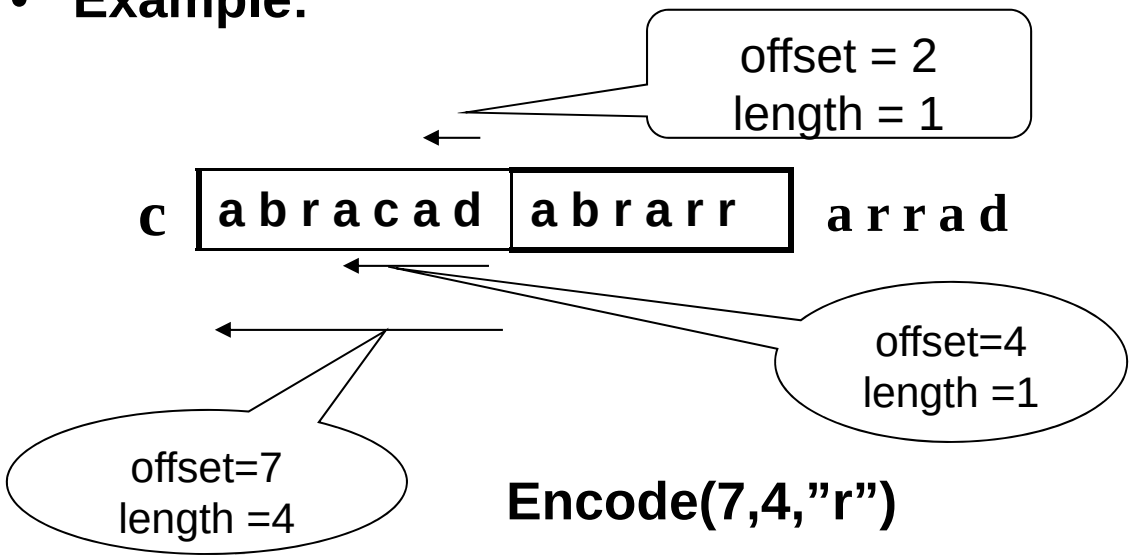
- **LZ77 (LZ1)**
  - J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. on Infor. Theory,* vol. 23, pp. 337-343, **1977**.
  - **LZSS**
    - J. A. Storer and T. G. Syzmanski, “Data Compression via Textual Substitution”,*Journal of the ACM*, vol. 29, pp. 928-951,1982.
- **LZ78 (LZ2)**
  - J. Ziv and A. Lempel, “Compression of individual Sequences via Variable- Rate Coding,” *IEEE transactions on Information Theory,* vol. 24, no. 5, pp. 530-536, Sep. **1978.**
  - **LZW**
    - T. A. Welch,”A technique for High-Performance Data Compression, ” *IEEE computer*, pp. 8-19, Jun. 1984.

LZ77 (1)



- **The encoder searches the search buffer for the longest matched pattern** $\Rightarrow$ encode **(Offset , max_match_Length , next_symbol_Codeword)**

- **Example:**



Encode(7,4,"r")

S: the size of search buffer
W: the size of search buffer and look-ahead buffer
A: the size of alphabet
Suppose fixed-length code was adopted for < O, L, C >

$\Rightarrow$

# of bits $=\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 A \rceil$

LZ77 (2)

	c a b r a c	a d a b r a r r a r r a d	⇒ (0, 0 , “c”)
c	a b r a c a	d a b r a r r a r r a d	⇒ (0, 0 , “a”)
c a	b r a c a d	a b r a r r a r r a d	⇒ (0, 0 , “b”)
c a b	r a c a d a	b r a r r a r r a d	⇒ (0, 0 , “r”)
c a b r	a c a d a b	r a r r a r r a d	⇒ (3, 1 , “c”)
c a b r a c	a d a b r a	r r a r r a d	⇒ (2, 1 , “d”)
c a b r a c a d	a b r a r r	a r r a d	⇒ (7, 4 , “r”)
c a b r a c a d a b r a r	r a r r a d		⇒ (3, 5 , “d”)

LZ77 (3)

- **Variations**
  - PKZip , Zip , LHarc, PNG, gzip, ARJ
    - encode the triples using variable length code
  - LZSS
    - more effective search strategies
    - encode two fields instead of three

      ① one flag bit"1"+ <O,L> : for matched case

      ② one flag bit"0"+codeword : for non-matched case

- **Deficiencies**
  - implicitly assumes that like patterns will occur close together and seriously relies on the search buffer size

    ⇒ by dropping the reliance on the search buffer and keeping an explicit dictionary (LZ78)
  - The size L of the look_ahead buffer is limited
  - The size S of the search buffer is limited
    - L↑(or S↑) ⇒ longer matches would be possible ⇒ compression efficiency ↑

      But, search for longer matches ⇒ speed ↓

      and field length↑ ⇒ compression efficiency ↓

LZ78 (1)

- **Keep an explicit dictionary of previously occurring patterns**
  - encode (Index , next_symbol_Codeword) < I, C>
- **Example**
  - input sequence : wabba wabba wabba wabba woo woo woo

Encode	Dictionary	
	Index	Entry
<0 , C(w)>	1	“w”
<0 , C(a)>	2	“a”
<0 , C(b)>	3	“b”
<3 , C(a)>	4	“ba”
<0 , C( )>	5	“ ”
<1 , C(a)>	6	“wa”
<3 , C(b)>	7	“bb”
<2 , C( )>	8	“a “
<6 , C(b)>	9	“wab”
<4 , C( )>	10	“ba “
<9 , C(b)>	11	“wabb”
<8 , C(w)>	12	“a w”
<0 , C(o) >	13	“o”
<13 , C( )>	14	“o “
<1 , C(o)>	15	“wo”
<14 , C(w) >	16	“o w”
<13 , C(o)>	17	“oo”

The dictionary will keep growing without bound

$\Rightarrow$ Either prune it back or treat the encoding as a fixed dictionary scheme

LZ78 (2)

- **Some possible solutions to stop the growth of the dictionary :**

  When the dictionary is full

  - freeze the dictionary
    - ➢ static dictionary-based method
  - delete some of the least recently used entries to make room for new ones
    - ➢ the dictionary always reflects recent patterns in the input
    - ➢ to search the dictionary for the first entry that is not a prefix to any other entry

      e.g.

      - suppose "**abcd**" is found & there are no entries of "**abcd…**"

        $\Rightarrow$ delete the entry "**abcd**"
  - delete the entire dictionary once it gets full & start with a new empty dictionary

(UNIX "compress" command) (LZW+adaptive dictionary size)

- ➢ the size of the dictionary is progressively doubled as it fills up
- ➢ when the largest allowed dictionary (2^{16} entries) fills up
  - if the compression ratio < threshold $\Rightarrow$ delete the dictionary and start with a new initialized-size (2^9 entries) dictionary
  - otherwise, $\Rightarrow$ adopt static dictionary method

LZW (1)

編碼法則： (S, C)　S : 字串　C : 字元

1. 先在字典裏存入輸入字元序列裏所有可能出現的字元，每一個字元在字典裏之對應索引值， 即為該字元爾後會被編碼成的值。
2. 從目前字典裏的內容找出最長字串S，假設其長度為 k，且該字串S與目前尚未被編碼之字元序列的前k個字元相同； C為目前未被編碼字元序列之第k+1個字元。
3. 輸出字串S在字典裏的索引值。
4. 將新的字串（SC）存入字典裏，且爾後該字串SC會被編碼成其在字典裏之對應索引值。
5. 對C字元(含)以後之剩餘字元序列繼續作編碼(即重覆法則2)。

解碼法則： (S1, S2(=CS3))　S1, S2: 字串　C : 字元　S3 : 字串或空字串

1. 如同編碼法則1，先建構起含輸入字元序列裏所有可能出現字元的字典，每一個字元在字典裏之對應索引值，即為該字元在編碼器裏會被編碼成的值。
2. 依序從收到的索引值，根據目前字典裏的內容，可解碼出相對應的字串。
3. 假設目前收到的索引值，其被解碼出字串S2(=CS3)，且上一個收到的索引值，其被解碼成字串S1。則將新的字串（S1C）存入字典裏，而其對應之新索引值，爾後將會被解碼出 S1C。
4. 重覆法則 2。

LZW (2)

編碼器：輸入字元序列　ABBBABAAB

輸出索引序列　124313

編碼步驟	(S,C)	已被編碼字串	待編碼字元序列	輸出	字典 索引	字典內容
0					1	A
0			ABBBABAAB		2	B
1	(A,B)	A	BBBABAAB	1	3	AB
2	(B,B)	B	BBABAAB	2	4	BB
3	(BB,A)	BB	ABAAB	4	5	BBA
4	(AB,A)	AB	AAB	3	6	ABA
5	(A,A)	A	AB	1	7	AA
6	(AB,)	AB		3		

LZW (3)

解碼器：輸入索引序列　124313　　輸出字元序列 ABBBABAAB

編碼步驟	已被解碼索引	待解碼索引序列	輸出	(S1, S2)	字典索引	字典內容
0					1	A
0		124313			2	B
1	1	24313	A			
2	2	4313	B	(A, B)	3	AB
3	4	313	BB	(B, BB)	4	BB
4	3	13	AB	(BB, AB)	5	BBA
5	1	3	A	(AB, A)	6	ABA
6	3		AB	(A, AB)	7	AA

針對收到的索引值4，但目前字典裏的內容只有三項，第四項仍在建構中，此時似乎無法繼續解碼。但根據解碼法則3(或者編碼法則4)，字典裏第四項的內容將會是B*，*表示未知字元，所以索引值4將會被解碼出B*，但再根據解碼法則3，由索引值2及索引值4分別被解碼出的字串B及B*，我們可以得知，字典裏第四項的內容將會是BB，即S1=B及C=B。所以此步驟裏將會解碼出BB。