Introduction to Information Theory and Coding

Chapter 13: Universal source coding

Lecturer: Prof. Si-Hyeon Lee

Overview

Universal code

- If the source distribution is known, we can use Huffman algorithm to construct an optimal code for that distribution.
- However, in many practical systems, the source distribution may be unknown and we cannot apply the Huffman algorithm directly.
- Chapter 13 is about universal source coding, which does not require the knowledge of source distribution.

Lempel-Ziv coding

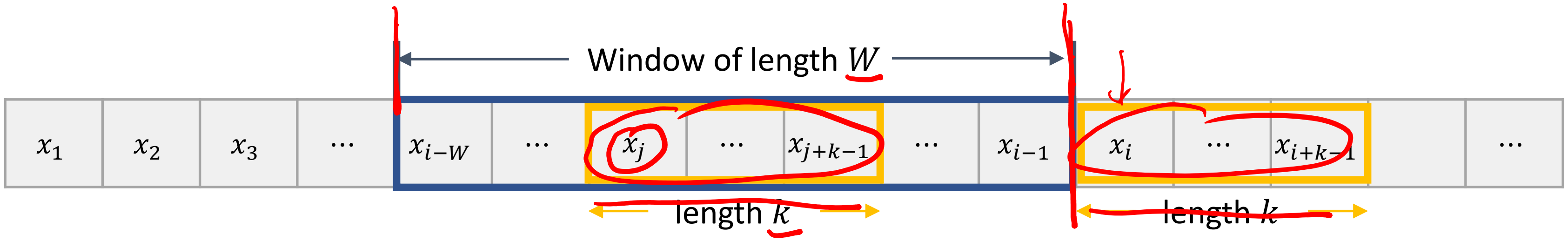
- Sliding window Lempel-Ziv (LZ77, LZ1)
  - Exploit repeated sequences in the previously encoded data through a sliding window mechanism.
  - Basis of ZIP, PNG compression
- Tree-structured Lempel-Ziv algorithm (LZ78, LZ2)
  - Construct a dictionary to encode newly encountered string patterns by assigning them unique indices.
  - Basis of GIF, TIFF compression

- Their asymptotic optimality can be proved (Chapter 13.5), but we will not cover the proof.

LZ77: Sliding-window Lempel-Ziv algorithm

Main idea: Encode a string by finding the longest match anywhere within a window of past symbols

- Represent the string by a pointer to location of the match within the window and the length of the match



Assume we have compressed up to x_{i-1}.

- Find the longest match in the window, i.e., find the largest k such that for some $j \in [i-W : i-1]$, the string of length k starting from x_j is equal to the string of length k starting from x_i.
- If you can find such match, the string $(x_i, \cdots, x_{i+k-1})$ is represented by $(1, i-j, k)$
- Otherwise, i.e., there is no symbol x_i in the window, x_i is represented by $(0, x_i)$.

Hence, the encoded tuples are of two types: (F, P, L) or (F, C)

- F: Flag bit showing whether there is a match in the window, ($F = 1$: there is a match, $F = 0$: there is no match)
- P: Location of the beginning of the match
- L: Length of the match
- C: Uncompressed character

LZ77: Sliding-window Lempel-Ziv algorithm

Example) $W = 4$

1	0	0	1	0	0	1	0	0	0	1	1	0	1	0	1

Encoding: → 1, 0, 0, 100100, 01, 1, 01, 01

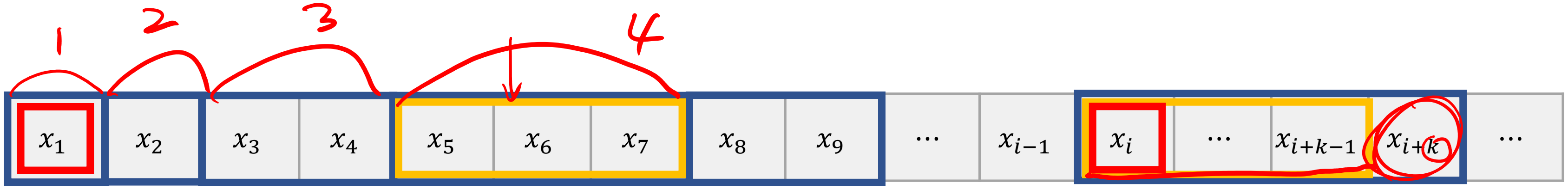
→ (0, 1), (0, 0), (1, 1, 1), (1, 3, 0), (1, 4, 2), (1, 1, 1), (1, 3, 2), (1, 2, 2)

Decoding: 1001001000110101

LZ78: Tree-structured Lempel-Ziv algorithm

Main idea: Parse a string into phrases, where each phrase is the shortest phrase not seen earlier.

- Build a dictionary in the form of a tree, where the nodes correspond to phrases seen so far.



Assume we have parsed the string up to x_{i-1} into phrases.

- Find the smallest k such that $x_i, \cdots, x_{i+k}$ does not correspond to one of the phrases before x_i.
- Note that $x_i, \cdots, x_{i+k-1}$ is one of the phrases that appeared before. Let's assume that it is the jth phrase.
- Then, the string $(x_i, \cdots, x_{i+k})$ is represented by (j, x_{i+k}).

Hence, the encoded tuples has the form of (D, C).

- D: Location of the prefix
- C: Value of the last bit

LZ78: Tree-structured Lempel-Ziv algorithm

Example)

1	0	0	1	0	0	1	0	0	0	1	1	0	1	0	1	1

Encoding: 1, 0, 01, 00, 10, 001, 101, 011

(0,1), (0,0), (2,1), (2,0), (1,0), (4,1), (5,1), (3,1)

Decoding: 10010010001101011

Dictionary

order	word
1	1
2	0
3	01
4	00
5	10
6	001
7	101
8	011

Example

0	0	1	0	1	0	0	1	1	1	1	1

LZ77 with $W = 3$

Encoding :

→ ~~(0,0)~~, ~~(1,1,1)~~, ~~(0,1)~~, ~~(1,2,3)~~, ~~(1,3,2)~~, (1,1,4)

Decoding :

0 0 1 0 1 0 0 1 1 1 1

Example

0	0	1	0	1	0	0	1	1	1	1	1

LZ78

Encoding:

→ ~~(0,0), (1,1), (2,0), (2,1), (0,1)~~, (5,1)   c(n)

Decoding: 001010011 111

$c(\log c + 1)$

order	word
1	0
2	01
3	010
4	011
5	1
6	11