CMPN206: Multimedia

Lecture 4: Dictionary Coding

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2014

Agenda

- Dictionary Based Coding
- LZ78
- LZW
- LZ77

Acknowledgments: Most slides are adapted from Richard Ladner and from Li and Drew.

Dictionary Coding

- Does *not* use statistical information about the source
- Basic Idea:
  - Encoder: As input is processed, develop a *dictionary* of strings and transmit the index of the string in the dictionary
  - Decoder: As input is processed, reconstruct the dictionary to invert the process of encoding
- The intuition is to encode *frequent* strings with the indices, and infrequent strings with less efficient encoding
- Works well for sources that emit relatively few symbols over and over again e.g. text

Example

- Suppose we are encoding text of a book with only lowercase characters and punctuations with an *alphabet* of 32 symbols
- To encode each symbol we will need 5 bits
- Assume all the words are 4-character words → 20 bits/word
- If we have a dictionary of 256 entries storing the most frequent 256 words, when we see a word in the dictionary we can just transmit its *index* → 8 bits/word
- If the word is not in the dictionary, we can transmit 20 bits/word

Index	Word
0	cats
1	rats
255	bat!

Example

- To differentiate between the two cases, we transmit a 1-bit flag at the start, such that:
  - when the word is in the dictionary a value of 1 preceeds the 8 bits of the index → 9 bits/word
  - when the word is *not* in the dictionary a value of 0 preceeds the 20 bits coding the four letters → 21 bits/word
- The average number of bits per word = $9\,p + 21\,(1 - p)$ bits/word where p is the probability of finding a word in the dictionary

Index	Word
0	cats
1	rats
255	bat!

Example

- The average number of bits per word = $9\,p + 21\,(1 - p)$ where p is the probability of finding a word in the dictionary
- For this technique to make sense, this average should be lower than 20 bits i.e. $p \geq 0.084$
- We also need to build the dictionary to be used:
  - Static Dictionary: built ahead of time
  - Adaptive Dictionary: built from the data while it's being processed

Index	Word
0	cats
1	rats
255	bat!

Adaptive Dictionary

- Two families of algorithms by Ziv and Lempel:
  - LZ78 and its variations, specially LZW
  - LZ77 and its variations

LZ78

- Introduced in 1978
- Builds the dictionary adaptively as the data is processed
- Both the *encoder* and the *decoder* build the dictionary
- Works best when the data contains *patterns* that are frequently repeated in the input stream

LZ78 Encoder

```
Repeat
   find the longest match w in the dictionary
      and its index i
   output the double ⟨i, c(a)⟩ where c(a) was the
      code unmatched symbol
   add "wa" in the dictionary
```

LZ78 Encoder Example

Dictionary

Input sequence

a b a b a b a b a

Encoded Output

LZ78 Encoder Example

Dictionary

1 a

Input sequence

a b a b a b a b a

Encoded Output

<0, C(a)>

a doesn't match anything, so add it
to the dictionary and output <0, C(a)>
to denote that there is no match

LZ78 Encoder Example

Dictionary

1 a
2 b

Input sequence

a b a b a b a b a

Encoded Output

<0, C(a)>
<0, C(b)>

b doesn't match anything, so add it
to the dictionary and output <0, C(b)>
to denote that there is no match

LZ78 Encoder Example

Dictionary

1 a
2 b
3 ab

Input sequence

a b a b a b a b a

Encoded Output

<0, C(a)>
<0, C(b)>
<1, C(b)>

a matches entry 1, so output <1, C(b)>
where b is the next symbol and add
ab to the dictionary

LZ78 Encoder Example

Dictionary

1 a
2 b
3 ab
4 aba

Input sequence

a b a b a b a b a

Encoded Output

<0, C(a)>
<0, C(b)>
<1, C(b)>
<3, C(a)>

ab matches entry 3, so output <3, C(a)>
where a is the next symbol and add
aba to the dictionary

LZ78 Encoder Example

Dictionary

1	a
2	b
3	ab
4	aba
5	ba

Input sequence

a b a b a b a b a

Encoded Output

<0, C(a)>
<0, C(b)>
<1, C(b)>
<3, C(a)>
<2, C(a)>

b matches entry 2, so output <2, C(a)>
where a is the next symbol and add
ba to the dictionary

LZW

- One problem with LZ78 is that we need to encode the *second* character and send the pair $\langle i, C(a) \rangle$
- Welch improved LZ78 by introducing LZW where the encoder only sends the index i in the dictionary
- For this to work, the dictionary is *initialized* with the symbols of the alphabet

LZW Encoder

```
Initialize the dictionary with the symbols
Repeat
   find the longest match w in the dictionary
      and its index i
   output the index i
   add "wa" in the dictionary
```

LZW Encoding Example

Dictionary

0 a
1 b

Input sequence

a b a b a b a b a

Encoded Output

```
Initialize the dictionary with the symbols
Repeat
    find the longest match w in the dictionary
        and its index i
    output the index i
    add "wa" in the dictionary
```

LZW Encoding Example

Dictionary

0 a
1 b
2 ab

Input sequence

a b a b a b a b a

Encoded Output

0

a matches entry 0, so output 0 and add ab to the dictionary

```
Initialize the dictionary with the symbols
Repeat
    find the longest match w in the dictionary
        and its index i
    output the index i
    add "wa" in the dictionary
```

LZW Encoding Example

Dictionary

0 a
1 b
2 ab
3 ba

Input sequence

a b a b a b a b a

Encoded Output

0
1

b matches entry 1, so output 1 and add ba to the dictionary

```
Initialize the dictionary with the symbols
Repeat
    find the longest match w in the dictionary
        and its index i
    output the index i
    add "wa" in the dictionary
```

LZW Encoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba

Input sequence

a b a b a b a b a

Encoded Output

0
1
2

ab matches entry 2, so output 2 and add aba to the dictionary

```
Initialize the dictionary with the symbols
Repeat
    find the longest match w in the dictionary
        and its index i
    output the index i
    add "wa" in the dictionary
```

LZW Encoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba
5	abab

Input sequence

a b a b a b a b a

Encoded Output

0
1
2
4

aba matches entry 4, so output 4 and add abab to the dictionary

```
Initialize the dictionary with the symbols
Repeat
    find the longest match w in the dictionary
        and its index i
    output the index i
    add "wa" in the dictionary
```

LZW Encoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba
5	abab

Input sequence

a b a b a b a b a

Encoded Output

0
1
2
4
3

ba matches entry 3, so output 3

```
Initialize the dictionary with the symbols
Repeat
    find the longest match w in the dictionary
        and its index i
    output the index i
    add "wa" in the dictionary
```

LZW Decoder

- Build the same dictionary as the *encoder*

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
   Decode the first symbol s of the index
   Complete last entry in dictionary to ws
   Finish decoding the rest of the index
   Add w? to the dictionary where w
      was just decoded
```

LZW Decoding Example

Dictionary

0 a
1 b

Input sequence

0 1 2 4 3

Decoded Output

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0 a
1 b
2 a?

Input sequence

0 1 2 4 3

Decoded Output

a

0 is decoded as a, and add a? to the dictionary

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0 a
1 b
2 ab

Input sequence

0 1 2 4 3

Decoded Output

a b

1 is decoded as b, and update ab in the dictionary
i.e. s = b

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0 a
1 b
2 ab
3 b?

Input sequence

0 1 2 4 3

Decoded Output

a b

add b? to the dictionary

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0	a
1	b
2	ab
3	ba

Input sequence

0 1 2 4 3

Decoded Output

a b a

2 is decoded as a... , and update ba in the dictionary
i.e. s = a

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0 a
1 b
2 ab
3 ba
4 ab?

Input sequence

0 1 2 4 3

Decoded Output

a b ab

Finish decoding 2 and add ab? in the dictionary
i.e. w = ab

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba

Input sequence

0 1 2 4 3

Decoded Output

a b ab a

4 is decoded as a.., and update aba in the dictionary
i.e. s = a

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba
5	aba?

Input sequence

0 1 2 4 3

Decoded Output

a b ab aba

Finish decoding 4 as aba, and add aba? in the dictionary
i.e. w = aba

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba
5	abab

Input sequence

0 1 2 4 3

Decoded Output

a b ab aba b

3 is decoded as b.., and update abab in the dictionary
i.e. s = b

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Decoding Example

Dictionary

0 a
1 b
2 ab
3 ba
4 aba
5 abab
6 ba?

Input sequence

0 1 2 4 3

Decoded Output

a b ab aba ba

Finish decoding 3 as ba, and add ba? in the dictionary
i.e. w = ba

```
Initialize the dictionary with the symbols
Decode first index to w
Add w? in the dictionary
Repeat
    Decode the first symbol s of the index
    Complete last entry in dictionary to ws
    Finish decoding the rest of the index
    Add w? to the dictionary where w
        was just decoded
```

LZW Encoding Example

Dictionary

0	a
1	b
2	ab
3	ba
4	aba
5	abab

Input sequence

a b a b a b a b a

Encoded Output

0
1
2
4
3

We get the same dictionary and the original sequence!

Dictionary Size

- How do we represent these dictionary *indices*?
  - Use *fixed-length* coding: indices of n bits allows a dictionary with 2^n entries
  - Use a *two-pass* process: first computing the indices and then use *variable-length* coding e.g. Arithmetic Coding
- What happens when the dictionary gets full?
  - *Double* the size of the dictionary and increase the width of the indices by 1 bit. The length of the indices are usually in a range $[n_{\min}, n_{\max}]$
  - *Flush* the dictionary and start a new one
  - *Remove* the *least recently visited* LRV entries

LZW Applications: Unix `compress`

- Starts with a dictionary of 512 entries i.e. 9 bits
- Once it fills up, the size doubles
- This process continues to a maximum of 64K entries i.e. 16 bits
- The dictionary becomes fixed after it reaches the maximum
- The program monitors the compression ratio, if it falls below some threshold, the dictionary is discarded the process restarts

LZW Applications: GIF

- Graphics Interchange Format
- The first byte stores the minimum number of bits per pixel b
- The initial dictionary size is 2^{b+1} and keeps doubling till a maximum of 4096 i.e. 12 bits
- Once this is reached, the dictionary becomes *static* i.e. fixed
- It compresses pixel values themselves without any preprocessing

TABLE 5.16 Comparison of GIF with arithmetic coding.

Image	GIF	Arithmetic Coding of Pixel Values	Arithmetic Coding of Pixel Differences
Sena	51,085	53,431	31,847
Sensin	60,649	58,306	37,126
Earth	34,276	38,248	32,137
Omaha	61,580	56,061	51,393

LZ77

- Developed by Lempel and Ziv in 1977
- Uses and *implicit* dictionary
- Instead of keeping an *explicit* dictionary, searches the immediate *history* for matches
- Idea: Given that $x_1 x_2 \ldots x_n$ have been coded, we want to code $x_{n+1} x_{n+2} \ldots x_{n+k}$ for the largest k possible i.e. find the largest match

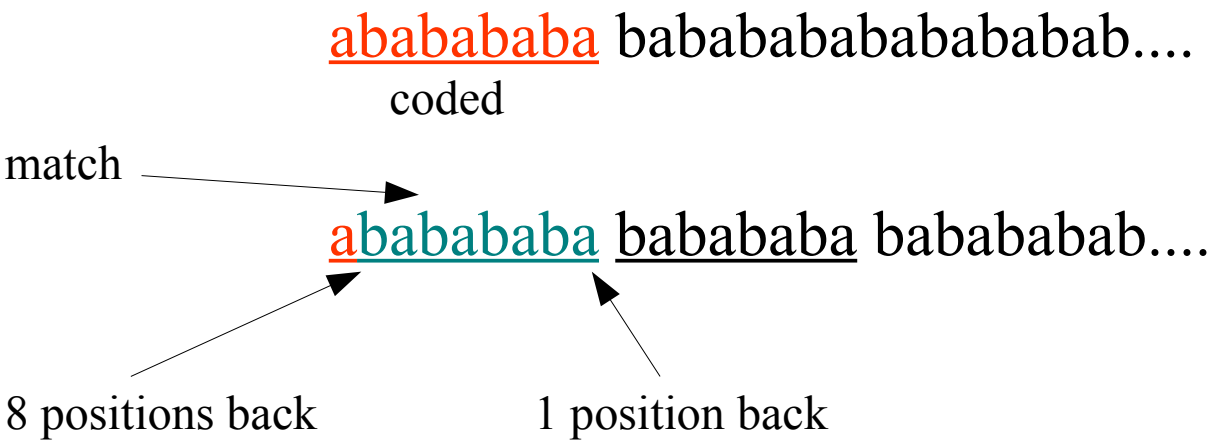
coded to be coded

$$x_1 x_2 \ldots x_n \; x_{n+1} x_{n+2} \ldots x_{n+k}$$

Solution

If $x_{n+1}x_{n+2}\ldots x_{n+k}$ is a substring of $x_1x_2\ldots x_n$ then $x_{n+1}x_{n+2}\ldots x_{n+k}$ can be coded using the pair $\langle j, k\rangle$ where j is the *offset* in $x_1x_2\ldots x_n$ looking backwards and k is the *match length*

- Example



Code: $\langle$**8, 8**$\rangle$ i.e. 8 matching characters starting 8 positions back

Problem

- What if there is *no* match in the coded string i.e. dictionary?

ababababa cabababababababab....
coded

- Solution: Send a triplet $\langle j, k, x \rangle$ where j is the *offset* in $x_1 x_2 \ldots x_n$ looking backwards, k is the *match length*, and x is the codeword for the first unmatched symbol
  - If $j = k = 0$ then there is *no* match and x is the codeword of the first unmatched symbol
  - Otherwise, there is a match starting at j and is k symbols long and x is the codeword of the first unmatched symbol

Solution

If $x_{n+1}x_{n+2}\ldots x_{n+k}$ is a substring of $x_1x_2\ldots x_n$ and $x_{n+1}x_{n+2}\ldots x_{n+k+1}$ is *not* then $x_{n+1}x_{n+2}\ldots x_{n+k+1}$ can be coded using the triplet $\langle j, k, x_{n+k+1}\rangle$ where j is the *offset* in $x_1x_2\ldots x_n$ looking backwards and k is the *match length*

- Example

  abababa c abababababababab....
  coded

  Code: ⟨**0, 0, c**⟩

  abababac abababa bababab....
  coded

  Code: ⟨**10, 9, b**⟩

Example

a bababababababababababab.....

⟨0, 0, a⟩

a b abababababababababababab.....

⟨0, 0, b⟩

ab ab abababababababababababab.....

⟨2, 2, a⟩

ababa baba bababababababab.....

⟨4, 4, b⟩

abababababab abababababab abab.....

⟨10, 10, a⟩

red: coded green: being coded

Surprise Code

a bababababc

⟨0, 0, a⟩

a b abababababc

⟨0, 0, b⟩

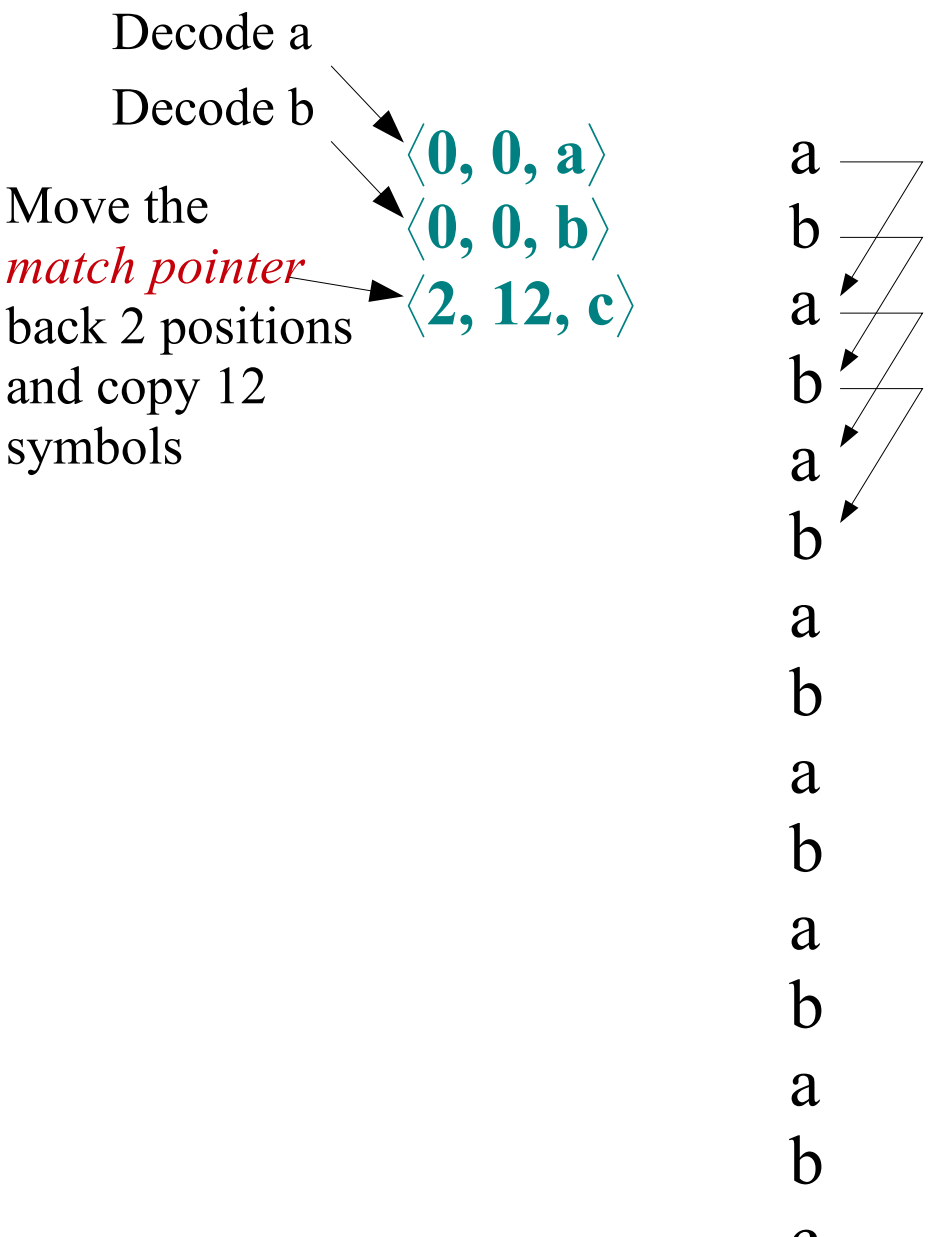
ab abababababab c

⟨2, 12, c⟩

How come the *match length* is more than the coded string?

Surprise Decoding

⟨0, 0, a⟩ ⟨0, 0, b⟩ ⟨2, 12, c⟩



Solution

If $x_{n+1}x_{n+2}\ldots x_{n+k}$ is a substring of $x_1x_2\ldots x_n\ldots x_{n+k}$ and $x_{n+1}x_{n+2}\ldots x_{n+k+1}$ is *not* then $x_{n+1}x_{n+2}\ldots x_{n+k+1}$ can be coded using the triplet $\langle j, k, x_{n+k+1}\rangle$ where j is the *offset* in $x_1x_2\ldots x_n$ looking backwards and k is the *match length*

Example

a aaabaaabaabc

⟨0, 0, a⟩

a aaa baaabaabc

⟨1, 3, b⟩

aaaab aaab aabc

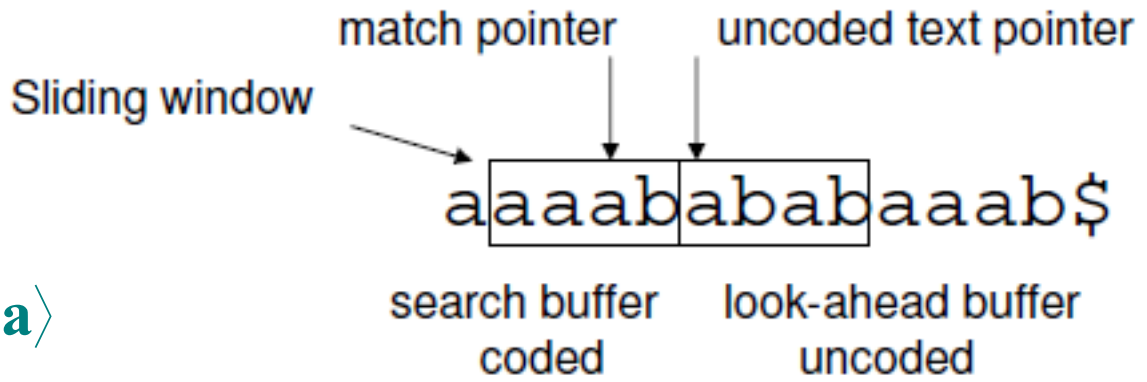
⟨4, 4, a⟩

aaaabaaaba ab c

⟨3, 2, c⟩

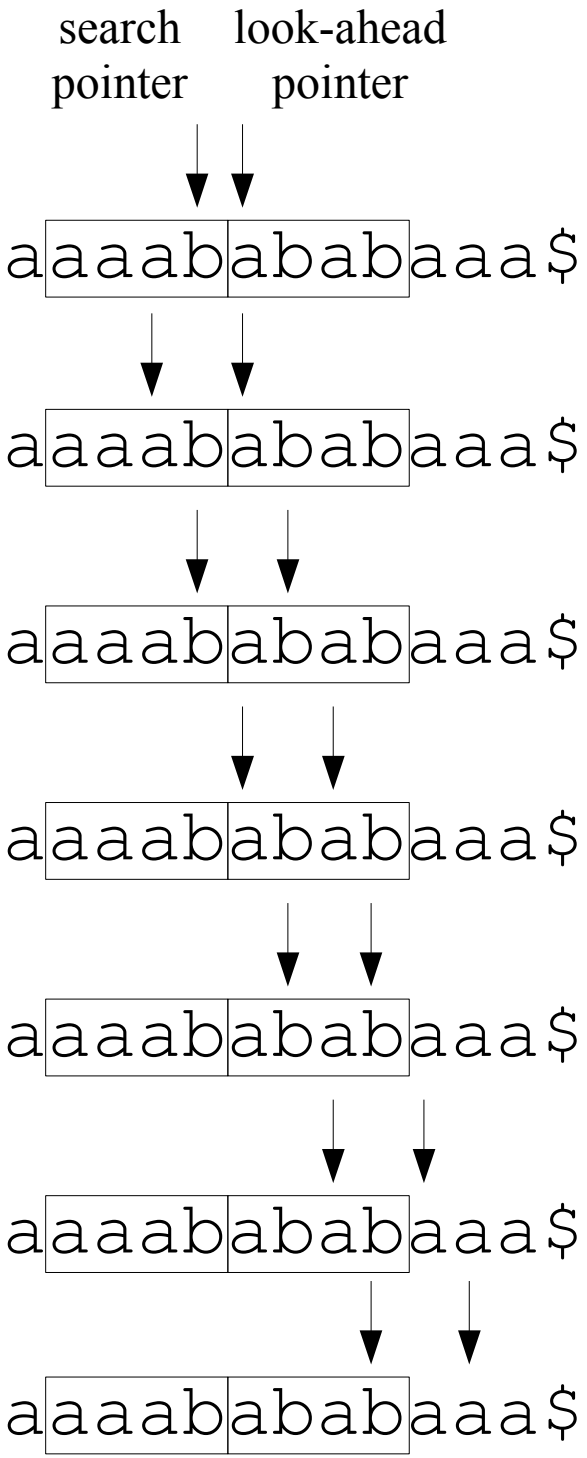
LZ77

- Uses a *sliding window*:
  - Search Buffer: of size s where we start looking for a match in the previously coded string $x_{n-s+1}\ldots x_n$
  - Look-ahead Buffer: of size t with the currently coded string $x_{n+1}\ldots x_{n+t}$
- The *match pointer* can start in the search buffer and go into the look-ahead buffer but no further



⟨2, 5, a⟩

Searching



offset	length	
1	0	Start with search pointer at the end of the search buffer and look-ahead pointer at the start of the look-ahead buffer
2	1	If no match then move search pointer back
2	2	If matching, advance *both* pointers forward
2	3	If matching, advance *both* pointers forward
2	4	If matching, advance *both* pointers forward
2	5	If matching, advance *both* pointers forward
2	5	No more matches, stop and reset look-ahead pointer to try another match

Encoding Example

$s = 4$ and $t = 4$

aaaababababaaab$	⟨0, 0, a⟩
aaaababababaaab$	⟨1, 3, b⟩
aaaababababaaab$	⟨2, 5, a⟩
aaaababababaaab$	⟨4, 2, $⟩

Coding the Triplets

- Simple *fixed-length* coding, need $\lceil \log_2 S + 1 \rceil + \lceil \log_2 S + T + 1 \rceil + \lceil \log_2 A \rceil$ bits where S is the length of the *search buffer*, T is the length of the *look-ahead buffer*, and A is the number of *symbols* in the alphabet
- Or *variable-length* coding using Huffman or Arithmetic coding to code the triplets:
  - either the *adaptive* version to encode as the triplets are generated
  - or use a *two-pass* approach to first generate all the triplets then encode them

LZ77

- Very popular
- The *deflate* algorithm is based on LZ77 and used in PKZip, Zip, Gzip, PNG, …
- Tends to work better than LZW

LZ77 Applications: PNG

- Portable Network Graphics
- Developed due to patent issues with the GIF format, which is based on the patented LZW algorithm
- A free and open standard based on the LZ77 algorithm
- Based on the *deflate* algorithm
- Match lengths between 3 and 258
- Offset between 1 and 32,768
- Encodes *predicted* pixel values instead of the pixel values e.g. pixel above, to the left, their average, …

LZ77 Applications: PNG

- Much better compression than GIF
- Similar to Arithmetic coding of pixel differences

TABLE 5.19 Comparison of PNG with GIF and arithmetic coding.

Image	PNG	GIF	Arithmetic Coding of Pixel Values	Arithmetic Coding of Pixel Differences
Sena	31,577	51,085	53,431	31,847
Sensin	34,488	60,649	58,306	37,126
Earth	26,995	34,276	38,248	32,137
Omaha	50,185	61,580	56,061	51,393

Recap

- Dictionary Based Coding
- LZ78
- LZW
- LZ77

- More information: Chapter 5 [**IDC**]