ECI750 Multimedia Data Compression

Lectures 9 - 10

Dictionary Techniques

Dr. Shadan Khattak

Department of Electrical Engineering

COMSATS Institute of Information Technology - Abbottabad

Dictionary Techniques (1)

- In the previous lectures, we considered coding techniques which assumed that the source generated independent symbols.
- We also saw that most sources are correlated
  - and that modelling the data (or de-correlating it), generally, helped in compression.
- In this lecture, we will look at techniques that *build a list of commonly occurring patterns and encode these patterns by transmitting their index in the list (Dictionary Techniques)*.
  - Most useful with sources that generate a relatively small number of patterns quite frequently e.g., text, computer commands etc.

Dictionary Techniques (2)

- ***Main Idea***
  - Split the input into two classes:
    - Frequently occurring patterns
    - Infrequently occurring patterns
  - Encode the frequently occurring patterns more efficiently
  - Use a less efficient encoding technique for the rest of the patterns.
  - Hopefully (not necessarily), the average bits per symbol will get smaller.
  - *For a dictionary technique to be useful, the class of frequently occurring patterns (and thus the size of the dictionary) should be much smaller than the number of all possible patterns.*

Dictionary Techniques (3)

Example:

- Consider an English source with 26 letters & six punctuation marks (so we have a total of $26 + 6 = 32$ symbols in our alphabet)
- Suppose a text contains four-letter words (i.e., three letters and one punctuation mark)
  - Encoding each character independently (assuming each character is equally likely)
    - We will have $32^1 = 32$ symbols and will need $\lceil \log_2(32) \rceil = 5$ bits/symbol using FLC.
  - Encoding each word (four letters) independently (assuming each four-character pattern is equally likely)
    - We will have $32^4 = 1{,}048{,}576$ symbols and will need $\lceil \log_2(1{,}048{,}576) \rceil = 20$ bits/symbol using FLC.

Dictionary Techniques (4)

Example (2):

- Assume uneven distribution of the symbols
  - Create a list (dictionary) of 256 most frequently occurring patterns (with probability p) and encode them with $\lceil \log_2 256 \rceil = 8$ bits
  - Encode the rest with $20 + 1 = 21$ bits.
    - The additional 1-bit prefix is used as a flag to indicate that the pattern is not available in the dictionary.
  - So, the average rate is $9p + 21(1 - p) = 21 - 12p$
  - The scheme will be more efficient compared to FLC, if $21 - 12p < 20$.
    - For this to happen, $p > 0.084$

Dictionary Techniques (5)

Static vs. Adaptive (or Dynamic) Dictionary

- **Static**
  - It is permanent.
  - May allow addition of strings to the list, but no deletions.
  - If we have sufficient prior knowledge about the structure of the source before encoding, we use static dictionary.
- **Adaptive (or Dynamic)**
  - It allows both addition and deletion of strings as new input is being read.
  - If we do not have prior information about the structure of the source, we need to acquire this information during encoding.

Dictionary Techniques (6)

Static Dictionary

- It is especially suitable for use in specific applications.
- For example, if we are compressing student records at university
  - We know ahead of time that words such as “Name”, “Student ID”, and “GPA” etc. will occur quite often.
  - Hence, static-dictionary is well-suited to work for this application.
  - The same dictionary may not work for another application.
  - For a new (another suitable) application, if the same dictionary is used, it might cause an expansion of data, rather than compression.
- Digram coding is an example of a static-dictionary technique which is slightly less application specific.

Dictionary Techniques (7)

Digram Coding

- The dictionary is composed of
  - All letters from the alphabet
  - Plus, as many digrams (pairs of letters) as possible.
- For example, if we want to compress a document containing ASCII characters, we can design a dictionary of size 256 entries, and
  - Source alphabet: 95 printable ASCII symbols
  - Digrams: $256 - 95 = 161$ digrams (most common pairs).

Dictionary Techniques (8)

Digram Coding (2)

Example:

- $A = \{a, b, c, d, r\}$
- Dictionary:

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

- Sequence: $abracadabra$
- Output: 101100110111101100000 $(21\ bits)$
- FLC approach: $11 \times 3 = 33\ bits$

Dictionary Techniques (9)

Problem with Digram Coding

- Which digram to choose?

TABLE 5.2 Thirty most frequently occurring pairs of characters in a 41,364-character-long LaTeX document.

Pair	Count	Pair	Count
eƀ	1128	*ar*	314
ƀt	838	*at*	313
ƀƀ	823	*ƀw*	309
th	817	*te*	296
he	712	*ƀs*	295
in	512	*dƀ*	272
sƀ	494	*ƀo*	266
er	433	*io*	257
ƀa	425	*co*	256
tƀ	401	*re*	247
en	392	*ƀ$*	246
on	385	*rƀ*	239
nƀ	353	*di*	230
ti	322	*ic*	229
ƀi	317	*ct*	226

TABLE 5.3 Thirty most frequently occurring pairs of characters in a collection of C programs containing 64,983 characters.

Pair	Count	Pair	Count
ƀƀ	5728	*st*	442
nlƀ	1471	*le*	440
; *nl*	1133	*ut*	440
in	985	*f(*	416
nt	739	*ar*	381
=*ƀ*	687	*or*	374
ƀi	662	*rƀ*	373
tƀ	615	*en*	371
ƀ=	612	*er*	358
);	558	*ri*	357
,*ƀ*	554	*at*	352
nlnl	506	*pr*	351
ƀf	505	*te*	349
eƀ	500	*an*	348
ƀ∗	444	*lo*	347

Dictionary Techniques (10)

Adaptive Dictionary Technique

- Original ideas published by Jacob Ziv and Abraham Lempel in 1977 (LZ77/LZ1) and 1978 (LZ78/LZ2)

Dictionary Techniques (11)

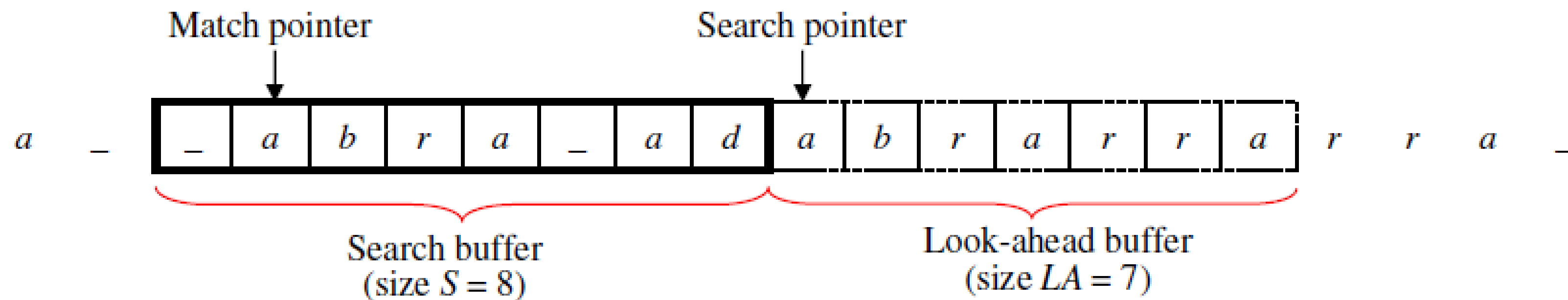
LZ77 Approach

- The dictionary is a portion of the previously encoded sequence.
- The encoder examines the input sequence through a ***sliding window.***
- The window consists of two parts:
  - ***Search buffer:*** contains a portion of the recently encoded sequence.
  - ***Lookahead buffer:*** contains the next portion of the sequence to be encoded.
- Find the maximum length match for the string pointed to by the search pointer in the search buffer, and encode it.

Dictionary Techniques (12)

LZ77 Approach (2)

- Sliding window size: $W = S + LA$



- In practice, the size of the buffers is significantly larger.

Dictionary Techniques (13)

LZ77 Approach (3)

- Offset = search pointer – match pointer ($o = 7$)
- Length of match = number of consecutive letters matched ($l = 4$)
- Codeword ($c = C(r)$), where $C(x)$ is the codeword for x.
- Encode the triple: (o, l, c) = $(7, 4, C(r))$
- If FLC is used and the alphabet size is A,
  - (o, l, c) can be encoded with $\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 A \rceil$ bits

Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Dictionary Techniques (14)

LZ77 Approach: Possible Cases for Triples

- There can be three possible cases that can be encountered during the encoding process:
    1. There is no match for the next character to be encoded in the window
    2. There is a match
    3. The matched string extends inside the lookahead buffer.
- For each of these cases, we have a triple to signal the case to the decoder.

Dictionary Techniques (15)

LZ77 Encoding: Example

- Sequence to be encoded: $\ldots cabracadabrarrarrad \ldots$
  - $W = 13, LA = 6, S = 7$
  - Current condition: $cabraca|dabrar$
  - Encode "d" as $< 0, 0, C(d) >$
  - New buffer condition: $abracad|abrarr$
  - Encode "abra" as $< 7, 4, C(r) >$
  - New buffer condition: $adabrar|rarrad$
  - Encode "rar" as $< 3, 3, C(r) >$
    - How about encoding "rar" as $< 3, 5, C(d) >$?

Dictionary Techniques (16)

LZ77 Decoding: Example

- Sequence already decoded: $cabraca$
- Sequence to be decoded: $< 0, 0, C(d) >, < 7, 4, C(r) >, < 3, 5, C(r) >$
  - Decode $< 0, 0, C(d) >$ as: $c|abracad|$
  - Decode $< 7, 4, C(r) >$ as: $cabrac|adabrar|$
  - Decode $< 3, 5, C(d) >$ as: $cabracadabra|rrarard|$

Dictionary Techniques (17)

LZ77 Decoding: Advantages and Disadvantages

- Simple
- Requires no prior knowledge of the source
- Assumes that the recurring patterns are close to each other

Dictionary Techniques (18)

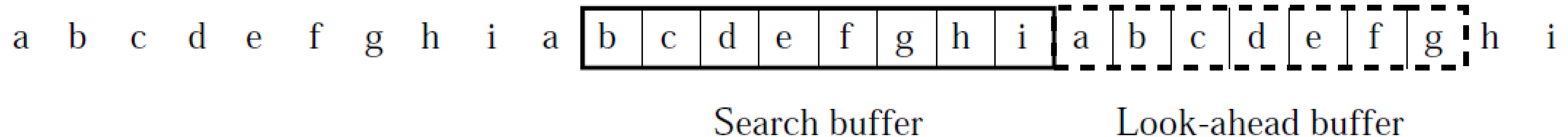
LZ77 Approach: Some Variants

- Can be made efficient in a number of ways:
  - Encode the triples using variable length codes
    - Zip, PNG among applications that use LZ77-based algorithm followed by a variable length coding scheme.
  - Varying the size of the search window.
    - Making the search buffer large, requires the development of more effective search strategies.
  - Eliminate the situation where we use a triple to encode a single character.
    - E.g., by using a flag bit to indicate whether what follows is the codeword for a single symbol.
    - The flag bit can also get rid of the third element of the triple.
    - This modification is known as LZSS.

Dictionary Techniques (19)

LZ77 Approach: Problem

- If the recurring patterns happen with a period larger than the search window, the performance is bad.
- For example,



Dictionary Techniques (20)

The LZ78 Approach:

- No search buffer.
- Dictionary has to be built at both the encoder and decoder synchronously.
- The inputs are encoded as a double $< i, c >$
  - i: index corresponding to the dictionary entry that was the longest match to the input.
  - c: code for the character in the input following the matched portion of the input
  - The index value of 0 is used in the case of no match.
  - Each new entry into the dictionary is one new symbol concatenated with an existing dictionary entry.

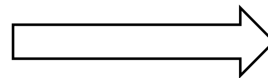
Dictionary Techniques (21)

The LZ78 Approach: Example

- String to be encoded: *wabbab̸wabbab̸wabbab̸wabbab̸woob̸woob̸woo*

The initial dictionary

Index	Entry
1	*w*
2	*a*
3	*b*



Encoder Output	Dictionary Index	Dictionary Entry
$\langle 0, C(w)\rangle$	1	*w*
$\langle 0, C(a)\rangle$	2	*a*
$\langle 0, C(b)\rangle$	3	*b*
$\langle 3, C(a)\rangle$	4	*ba*
$\langle 0, C(b̸)\rangle$	5	*b̸*
$\langle 1, C(a)\rangle$	6	*wa*
$\langle 3, C(b)\rangle$	7	*bb*
$\langle 2, C(b̸)\rangle$	8	*ab̸*
$\langle 6, C(b)\rangle$	9	*wab*
$\langle 4, C(b̸)\rangle$	10	*bab̸*
$\langle 9, C(b)\rangle$	11	*wabb*
$\langle 8, C(w)\rangle$	12	*ab̸w*
$\langle 0, C(o)\rangle$	13	*o*
$\langle 13, C(b̸)\rangle$	14	*ob̸*
$\langle 1, C(o)\rangle$	15	*wo*
$\langle 14, C(w)\rangle$	16	*ob̸w*
$\langle 13, C(o)\rangle$	17	*oo*

Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Dictionary Techniques (22)

<u>The LZ78 Approach:</u>

- The dictionary keeps growing without bounds
- Possible solutions:
  - Stop growing the dictionary
  - Prune it
    - Based on usage statistics
  - Reset it
    - Start all over again

Dictionary Techniques (23)

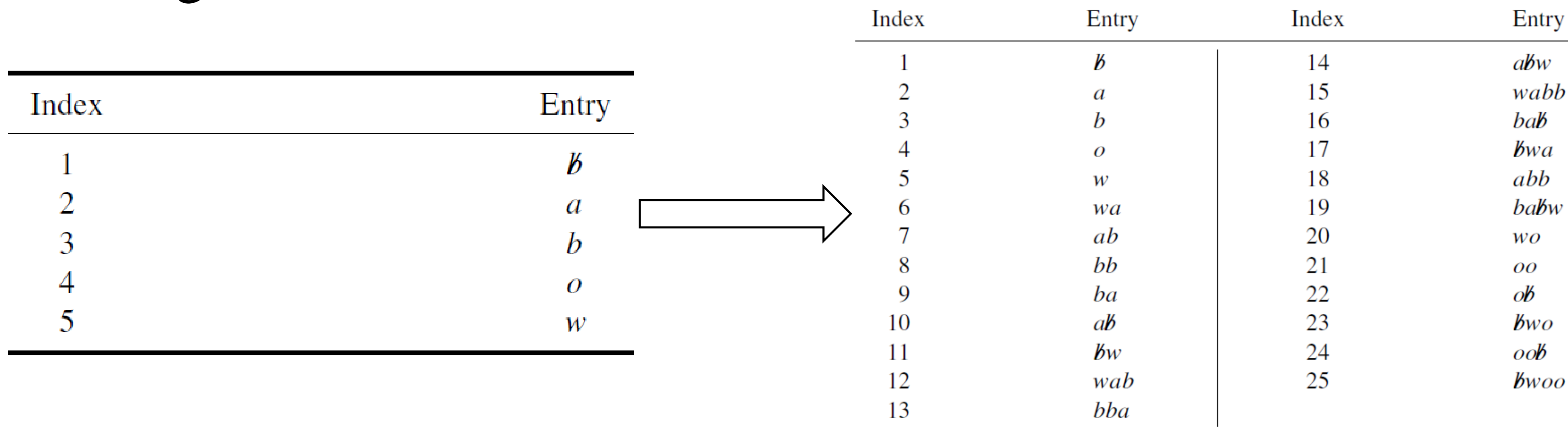
Variant of the LZ78 Approach: The LZW Algorithm

- Proposed by Terry Welch
- The necessity of encoding the second element of the pair $< i, c >$ can be removed.
  - i.e., only send the index to the dictionary.
  - The dictionary has to be initialized with all the letters of the source alphabet.
  - The input to the encoder is accumulated in a pattern p as long as p is contained in the dictionary.
  - If the addition of another letter a results in a pattern $p * a$ ($*$ represents concatenation) that is not in the dictionary, then
    - the index of p is transmitted to the receiver
    - The pattern $p * a$ is added to the dictionary
    - Start another pattern with the letter a

Dictionary Techniques (24)

The LZW Algorithm: Encoding Example

- String to be encoded: *wabbab̸wabbab̸wabbab̸wabbab̸woob̸woob̸woo*



Index	Entry
1	*b̸*
2	*a*
3	*b*
4	*o*
5	*w*

Index	Entry	Index	Entry
1	*b̸*	14	*ab̸w*
2	*a*	15	*wabb*
3	*b*	16	*bab̸*
4	*o*	17	*b̸wa*
5	*w*	18	*abb*
6	*wa*	19	*bab̸w*
7	*ab*	20	*wo*
8	*bb*	21	*oo*
9	*ba*	22	*ob̸*
10	*ab̸*	23	*b̸wo*
11	*b̸w*	24	*oob̸*
12	*wab*	25	*b̸woo*
13	*bba*		

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

Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Dictionary Techniques (25)

The LZW Algorithm: Decoding Example

- To decode: 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4
- Start with the same initial dictionary

Index	Entry
1	$\not b$
2	a
3	b
4	o
5	w

Dictionary Techniques (26)

The LZW Algorithm: Decoding Example (2)

Index	Entry
1	*b̸*
2	*a*
3	*b*
4	*o*
5	*w*
6	*wa*
7	*ab*
8	*bb*
9	*ba*
10	*ab̸*
11	*b̸*. . .

Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Dictionary Techniques (27)

The LZW Algorithm: Decoding Example

Special Case in Decoding:

- $A = \{a, b\}$
- Encode the sequence beginning with $ababababab$ ...

Index	Entry
1	a
2	b

Index	Entry
1	a
2	b
3	ab
4	ba
5	aba
6	$abab$
7	$b\ldots$

Index	Entry
1	a
2	b
3	ab
4	ba
5	$a\ldots$

Dictionary Techniques (28)

The LZW Algorithm: Decoding Example

Special Case in Decoding (2):

Index	Entry
1	a
2	b
3	ab
4	ba
5	$ab\ldots$

Index	Entry
1	a
2	b
3	ab
4	ba
5	aba
6	$a\ldots$

- ***Problem:*** We may run into a situation that the indexed entry is in partial construction
- ***Solution:*** the current dictionary entry under construction is in p, we should allow reading partial data out of p during decoding.

Dictionary Techniques (29)

Applications of Dictionary Schemes:

- Applications of LZ78 (LZW) include:
  - GIF
  - V.42 bis
- LZ78 is patented while LZ77 is patent-free
  - Hence, more interest has been shown in LZ77
- Most popular implementation of the LZ77 algorithm is the *deflate* algorithm.
  - Part of the popular *zlib* and *gzip* algorithm.
  - Used in the popular *PNG* compression.

Dictionary Techniques (30)

Applications of Dictionary Schemes:

1. ***File Compression – UNIX `compress`***

- One of the earlier applications of LZW.
- Start with a dictionary of size 512.
  - Hence the transmitted codewords are $\log_2 512 = 9$ bits long.
- The size of the dictionary is progressively doubled as it fills up.
- The maximum size of the codeword, b_{max}, can be set by the user to between 9 and 16, with 16 bits being the default.
- When the dictionary reaches $2^{b_{max}}$ entries, it becomes a static dictionary encoder.
- If compression ratio falls below a threshold, dictionary is reset.

Dictionary Techniques (31)

Applications of Dictionary Schemes:

2. ***Image Compression – The Graphics Interchange Format (GIF)***

- Another implementation of LZW, very similar to `compress`.
- The compressed image is stored with the first byte being the minimum number of bits b per pixel in the original image.
- The binary number 2^b is defined to be the ***clear code***.
  - It is used to reset all compression and decompression parameters to a start-up state.
- The initial size of the dictionary is 2^{b+1}.
  - When the dictionary fills up, this size is doubled until the maximum dictionary size of 4096 is reached.

Dictionary Techniques (32)

Applications of Dictionary Schemes:

2. ***Image Compression – The Graphics Interchange Format (GIF)***

- Format:
  - Codewords stored in blocks of 8-bit characters
  - Each block begins with a header with a size count up to 255, and ends with a block terminator symbol (8 zero bits)
  - The last block has a end-of-information code, $2^b + 1$, before the block terminator.

Dictionary Techniques (33)

Applications of Dictionary Schemes:

2. ***Image Compression – The Graphics Interchange Format (GIF)***

- Works well for computer-generated graphical images

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

Dictionary Techniques (34)

Applications of Dictionary Schemes:

3. ***Image Compression – Portable Network Graphics (PNG)***

- In 1994, Unisys and Compuserve (patent-holders of LZW) started charging royalties to authors of software that included support for GIF.
- Within three months, a patent-free replacement for GIF, PNG, was born.
- Compression algorithm is based on LZ77.

Dictionary Techniques (35)

Applications of Dictionary Schemes:

3. ***Image Compression – Portable Network Graphics (PNG)***

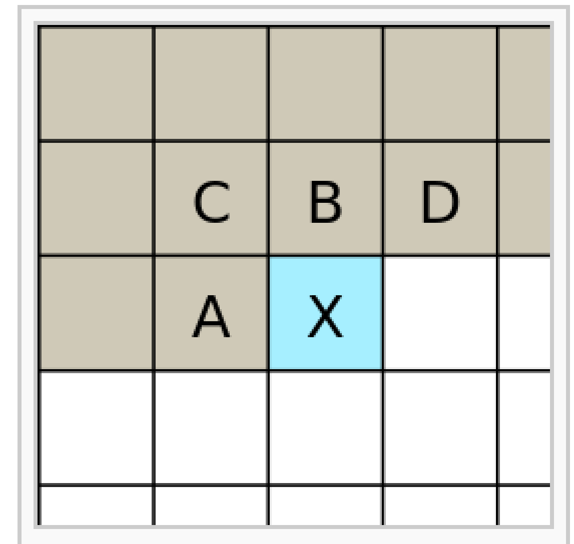
- Based on the *deflate* implementation of LZ77
- Designed specifically for lossless image compression
- Modes: true colour, grayscale, 8-bit palette.
- Two autonomous compression components
  - Filtering – lossless transformations of byte-level image data.
  - Deflate (RFC 1951) – LZ77-style dictionary compression algorithm plus Huffman coding.

Dictionary Techniques (36)

Applications of Dictionary Schemes:

3. Image Compression – Portable Network Graphics (PNG)

- ***Filtering***
  - Applied on a scanline-by-scanline basis
  - All algorithms applied to bytes (not pixels)
  - Filter types:
    - None: Unmodified value
    - Sub: difference from previous byte value (A)
    - Up: difference from the byte value above (B)
    - Average: subtract average of the left and the above bytes ($\frac{A+B}{2}$)
    - Paeth:
      - Compute initial estimate by A+B-C
      - The value of A, B, or C that is closest to the initial estimate is used as the estimate



Dictionary Techniques (37)

Applications of Dictionary Schemes:

3. Image Compression – Portable Network Graphics (PNG)

- ***Deflate*** = LZ77 + Huffman
- Three types of data blocks
  - Uncompressed, LZ77+fixed Huffman, LZ77+adaptive Huffman
- Match length is between 3 and 258 bytes
  - A sliding window of at least 3-byte long is examined
  - If match is not found, encode the first byte and slide window
  - At each step, LZ77 either outputs a codeword for a literal or a paired value of $< match_{length}, offset >$
    - Match length is encoded by index code (257~285) and a selector code (0~5 bits)
    - Offset (1~32768) is encoded using Huffman code.

Dictionary Techniques (38)

Applications of Dictionary Schemes:

3. ***Image Compression – Portable Network Graphics (PNG)***

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