EE 6427 Video Signal Processing

Instructor: Dr. Chau Lap Pui

Room: S2-B2a-09

Email: elpchau@ntu.edu.sg

Tel: 67904239

Topics

1. Entropy Coding
2. Sampling and DCT
3. JPEG
4. Motion compensation coder
5. Motion estimation
6. Wavelet
7. Image filter
8. Error control
9. 3D video
10. Bit rate control

Weightage	Assessment
80%	Final Examination
20%	Continuous Assessment: Take-home assignment (Individual)

Text Books

- Textbooks:

1. Y. Wang, J. Ostermann, and Y.-Q. Zhang, Video Processing and Communications, Prentice Hall, 2002.
2. Yun Q. Shi and Huifang Sun, Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards, CRC Press, 2nd Edition (2008).

- Error Control part:

1. Y. Wang, J. Ostermann, and Y.-Q. Zhang, Video Processing and Communications (Chapter 14), Prentice Hall, 2002.
2. Yao Wang, Lecture Materials on "Error control," http://eeweb.poly.edu/~yao/EL6123old/error_control_new.pdf

References

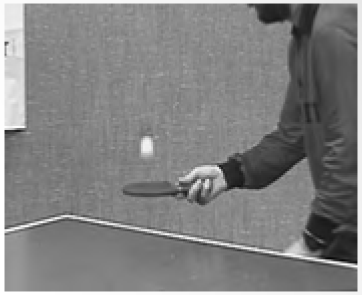
1. Iain E.G. Richardson, H.264 and MPEG-4 Video. Compression. Video Coding for Next-generation Multimedia, John Wiley & Sons, 2003
2. Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, 4th Edition
3. K.S. Thyagarajan, Still Image and Video Compression with MATLAB, Wiley, 2011
4. Oge Marques, Practical Image and Video Processing using MATLAB, Wiley, 2011
5. John W. Woods, Multidimensional Signal, Image, and Video Processing and Coding, Academic Press, 2012
6. [MPEG-1] ISO/IEC. IS 11172: Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s, 1993.
7. [MPEG-2 Video] ISO/IEC. IS 13818-2: Information technology - Generic coding of moving pictures and associated audio information, 1995.
8. [MPEG-4 Video] ISO/IEC. IS 14496-2: Information technology - coding of audio- visual objects, 1999.
9. ITU-T Recommendation H.264 & ISO/IEC 14496-10 (MPEG-4) AVC, Advanced Video Coding for Generic Audiovisual Services, version 3: 2005.
10. Digital Video Broadcasting (DVB), Frame Compatible Plano-Stereoscopic 3DTV (DVB-3DTV) DVB Document A154, February 2011

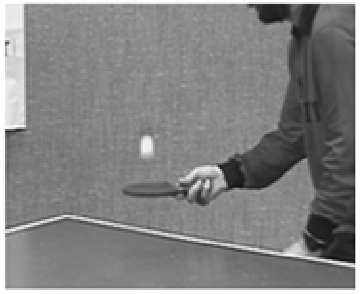
Video Compression Fundamental

- For an uncompressed 4K video signal with 3840 pixels times 2160 lines (16:9), 8 bits per pixel for each colour component, and 30 frames per second, it requires a bandwidth of 3840x2160x8x3x30=5971.97Mbits per second (Mbps) for transmission.
- For transmitting the video through a 1Gbps broadband network with an assured throughput of 100Mbps, it requires a compression ratio of at least 60:1 for real-time transmission.
- For storing one hour of the uncompressed video signal in harddisk, it requires 2.69TG bytes disk space.
- Owing to these high compression requirements, the lossy compression techniques are much suitable compared with the lossless ones.

Video Compression Fundamental

- The most basic approach to compress a digital video signal is on a frame by frame basis. This approach achieves compression by exploiting the spatial, spectral and psychovisual redundancies of a single frame.
- The compression ratio is limited since it does not exploit the temporal redundancies between neighbor frames.
- Thus the interframe techniques are widely adopted various video compression standards to reduce temporal redundancies of successive frames, in addition to the intraframe techniques.
- Among various inter/intra-frame compression techniques, the motion compensated transform coding technique is the most popular one, which is adopted by many video coding standards such as MPEG-1/2/4 and H.261/262/263/264/265, owing to its high compression efficiency.


Frame 39


Animated Frame 39-41

Basics of Information Theory

- According to Shannon, the entropy of an information source S is defined as:

$$H(S) = \sum_i p_i \log_2 \frac{1}{p_i} = -\sum_i p_i \log_2 p_i$$

where p_i is the probability that symbol S_i in S will occur $\sum_i p_i = 1$.
$\log_2(1/p_i) = -\log_2(p_i)$ indicates the amount of information contained in S_i, i.e., the number of bits needed to code S_i.

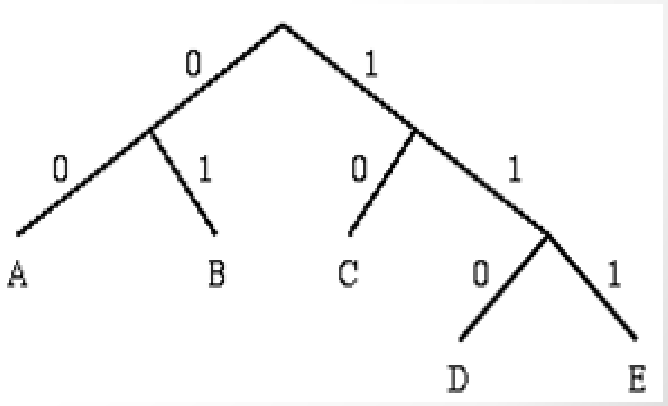
Higher probability symbols use fewer bits to represent it, that achieve compression. Shannon entropy tells us the theoretical lower bound.

Shannon-Fano Algorithm

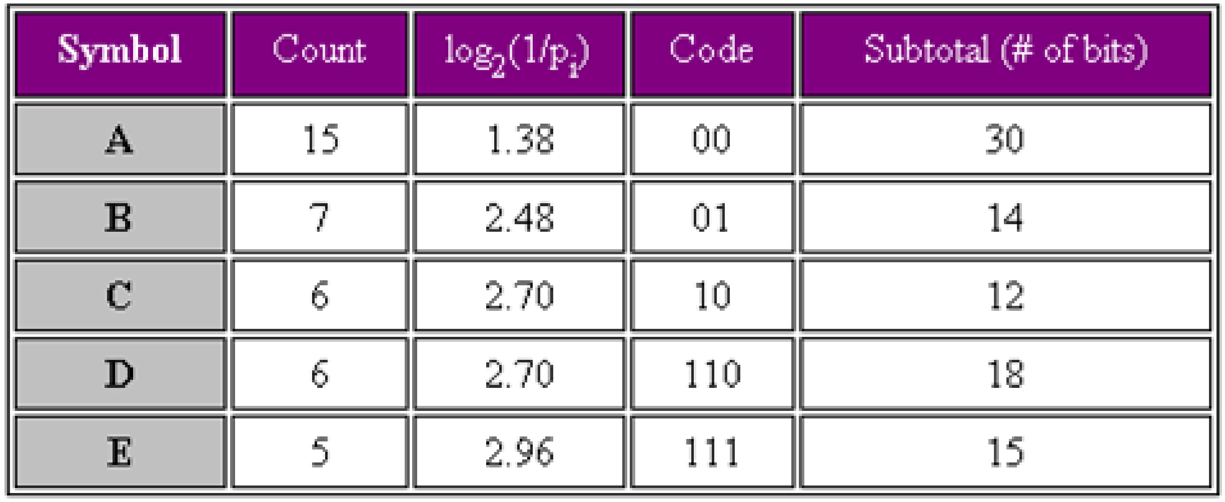
- A simple example will be used to illustrate the algorithm:

Symbol	A	B	C	D	E
Count	15	7	6	6	5

- Encoding for the Shannon-Fano Algorithm:
- A top-down approach
  1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.
  2. Recursively divide into two parts, each with approx. same number of counts.



Shannon-Fano Algorithm



Symbol	Count	$\log_2(1/p_i)$	Code	Subtotal (# of bits)
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15

Total count 39, p_A=15/39

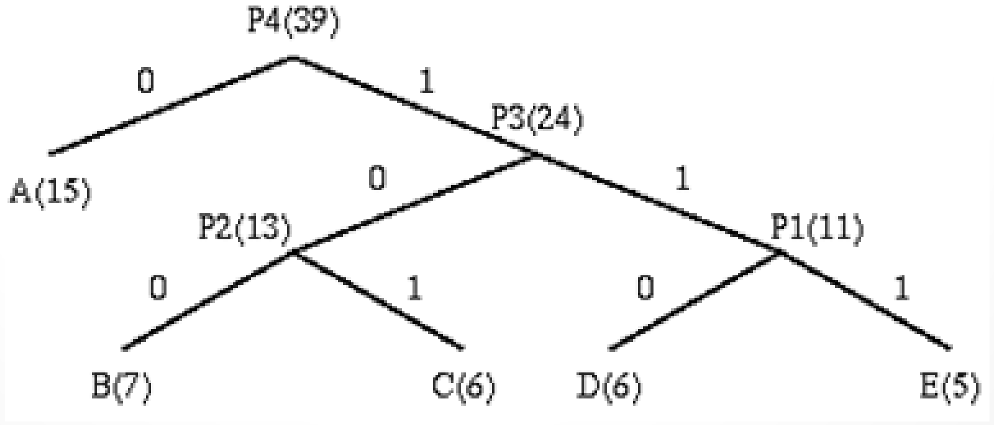
TOTAL (# of bits): 89

Huffman Coding

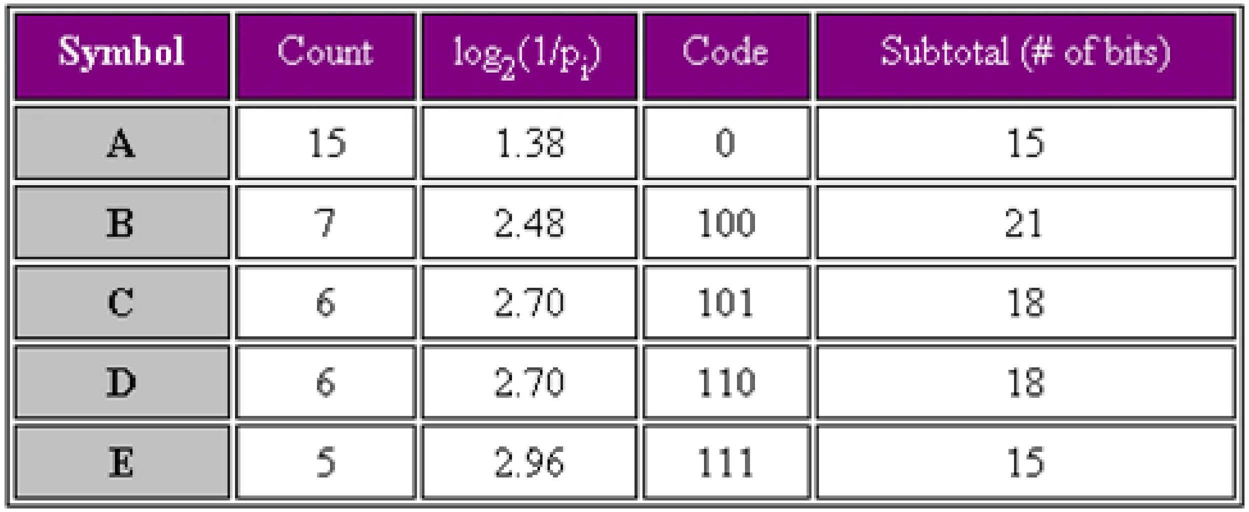
- **Encoding for Huffman Algorithm:**
- A bottom-up approach
- 1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).

Huffman Coding

- 2. Repeat until the OPEN list has only one node left:
  - From OPEN pick two nodes having the lowest frequencies/ probabilities, create a parent node of them.
  - Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.
  - Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.



Huffman Coding

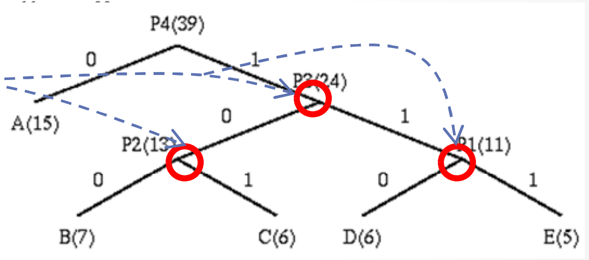


Symbol	Count	$\log_2(1/p_i)$	Code	Subtotal (# of bits)
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15

TOTAL (# of bits): 87

Discussions

- Decoding for the above two algorithms is trivial as long as the coding table (the statistics) is sent before the data. (There is a bit overhead for sending this, negligible if the data file is big.)
- Unique Prefix Property: no code is a prefix to any other code (all symbols are at the leaf nodes). E.g. {9, 59, 55} has the prefix property, but {9, 5, 59, 55} does not, because “5” is a prefix of both “59” and
  --> great for decoder, unambiguous.
- No symbols is at parent node

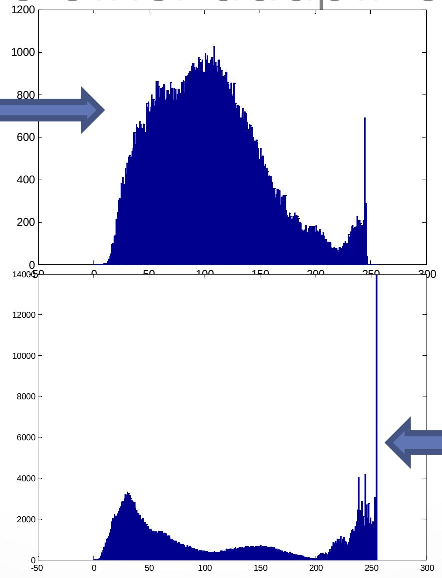


- If prior statistics are available and accurate, then Huffman coding is very good.
- In the above example: entropy = (15 x 1.38 + 7 x 2.48 + 6 x 2.7 + 6 x 2.7 + 5 x 2.96) / 39 = 85.26 / 39 = 2.19
- Number of bits needed for Huffman coding is: 87 / 39 = 2.23
- Number of bits needed for Shannon-Fano coding is: 89 / 39 = 2.28

Adaptive Huffman Coding

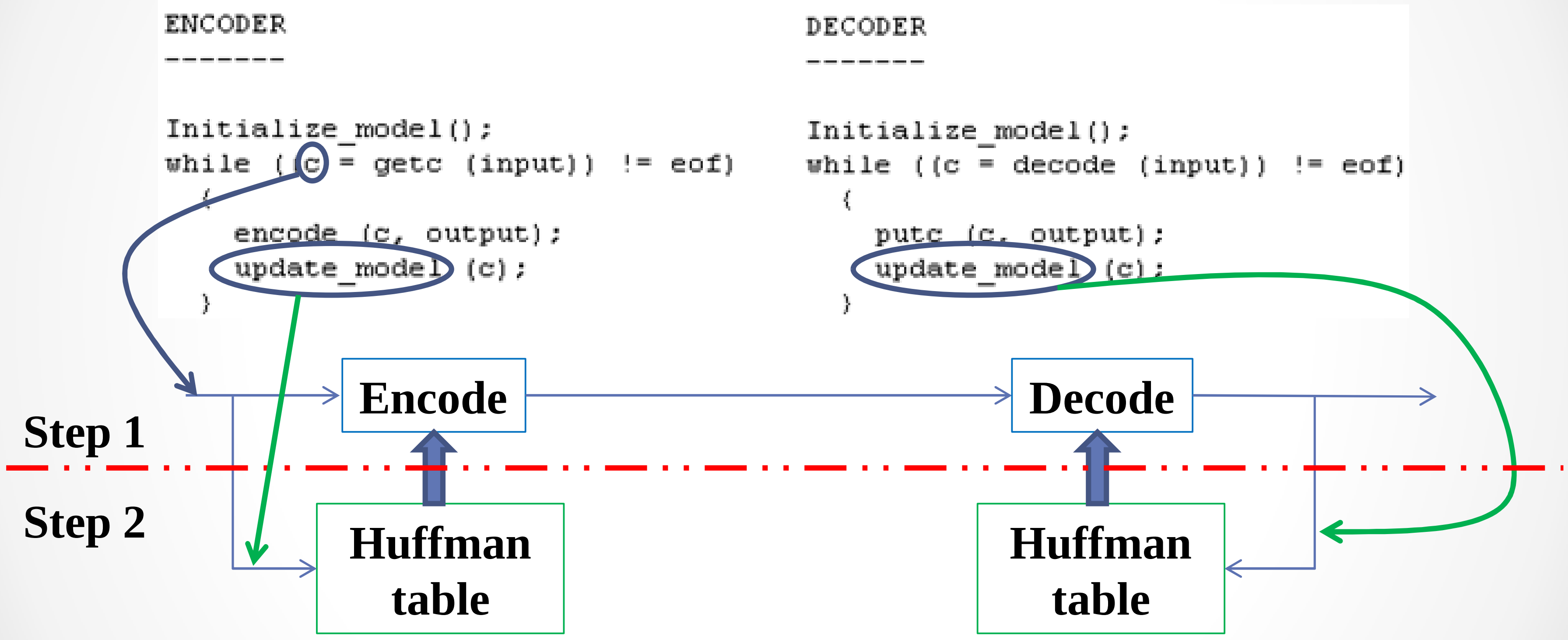
- **Motivations:**
  - o The previous algorithms require the statistical knowledge which is often not available (e.g., live audio, video).
  - o Even when it is available, it could be a heavy overhead especially when many tables had to be sent.
- The solution is to use adaptive algorithms. As an example, the Adaptive Huffman Coding is examined below. The idea is however applicable to other adaptive compression algorithms.

Indoor



Outdoor

Adaptive Huffman Coding



```
ENCODER
-------

Initialize_model();
while ((c = getc (input)) != eof)
  {
    encode (c, output);
    update_model (c);
  }
```

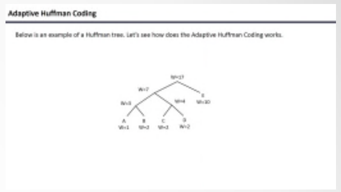
```
DECODER
-------

Initialize_model();
while ((c = decode (input)) != eof)
  {
    putc (c, output);
    update_model (c);
  }
```

- The key is to have both encoder and decoder to use exactly the same *initialization* and *update_model* routines.

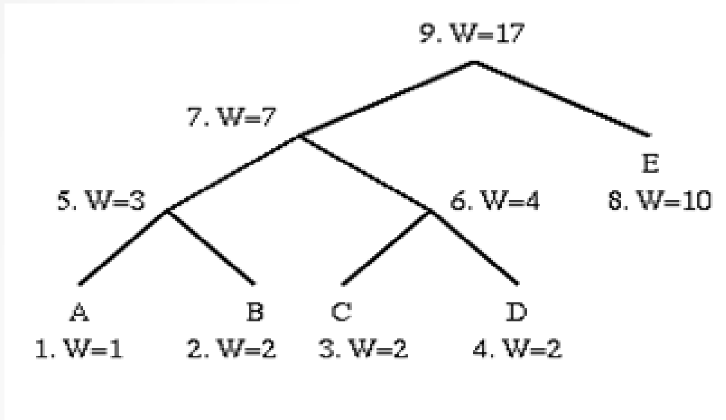
Adaptive Huffman Coding

- *update_model* does two things: (a) increment the count, (b) update the Huffman tree.
  - During the updates, the Huffman tree will be maintained its *sibling property*, i.e. the nodes (internal and leaf) are arranged in order of increasing weights (see figure).
  - When *swapping* is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1.
    Note: If the node with weight W has a subtree beneath it, then the subtree will go with it.
  - The Huffman tree could look very different after node swapping, e.g., in the third tree, node A is again swapped and becomes the #5 node. It is now encoded using only 2 bits.

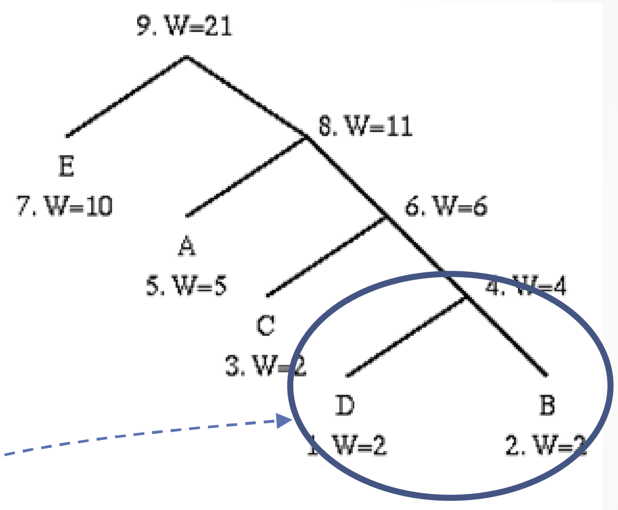
Adaptive Huffman Coding


Adaptive Huffman Coding

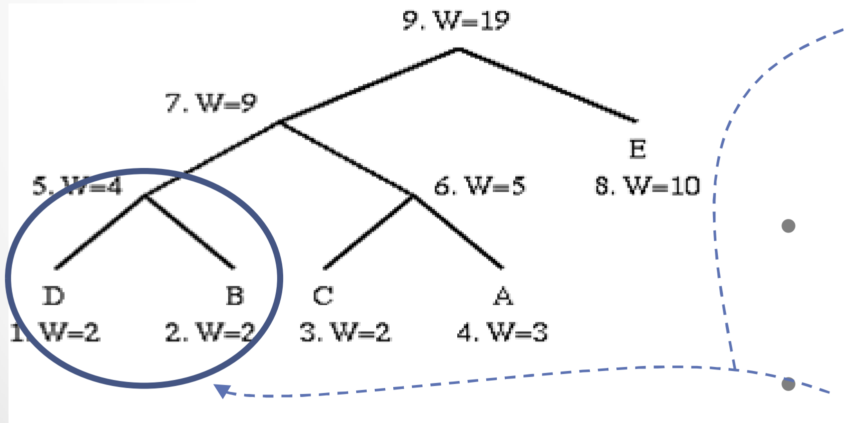
Below is an example of a Huffman tree. Let's see how does the Adaptive Huffman Coding works.



A Huffman Tree



After A was incremented two more times



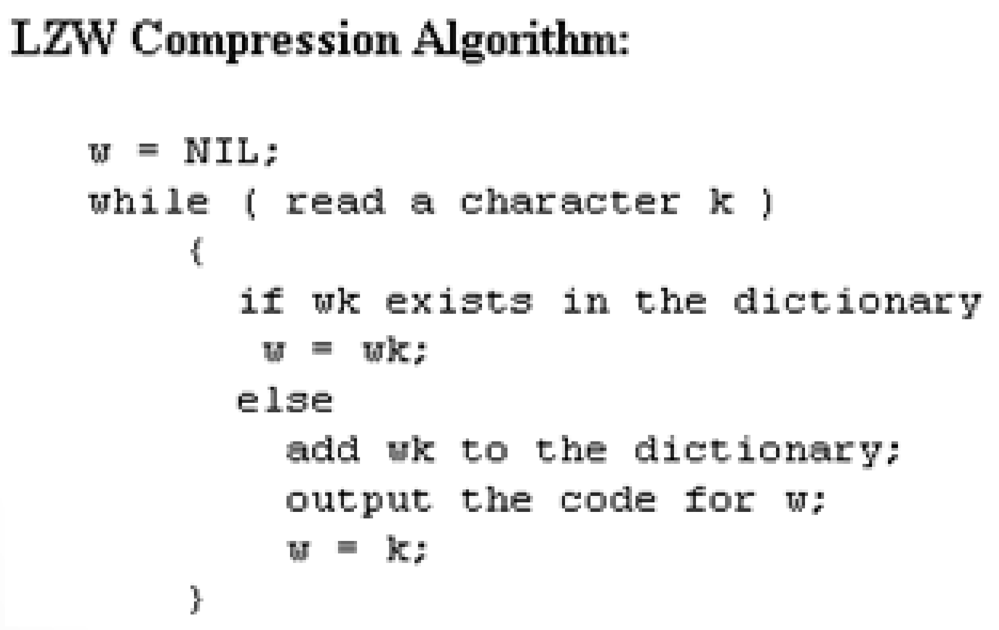
After a node switch (A was incremented twice)

- **Note:** Code for a particular symbol changes during the adaptive coding process.
- Maintain Huffman coding property after swapping.

Lempel-Ziv-Welch Algorithm

- **Motivation:** Suppose we want to encode the Webster's English dictionary which contains about 159,000 entries. Why not just transmit each word as an 18 bit number? (2^{18}=262144)
- **Problems:** (a) Too many bits, (b) everyone needs a dictionary, (c) only works for English text.
- Solution: Find a way to build the dictionary adaptively.
- Original methods due to Ziv and Lempel in 1977 and 1978. Terry Welch improved the scheme in 1984 (called LZW compression).
- The LZW algorithm is a very common compression technique. This algorithm is used in GIF.

Lempel-Ziv-Welch Algorithm



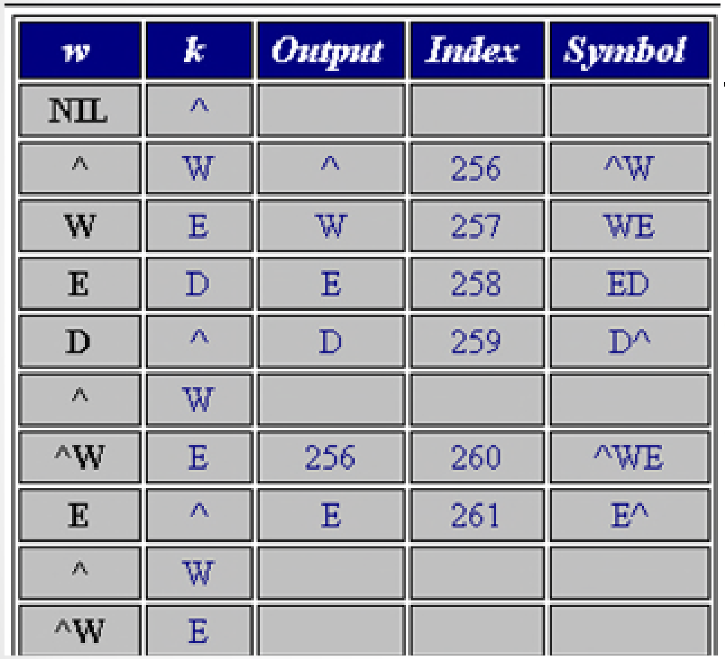
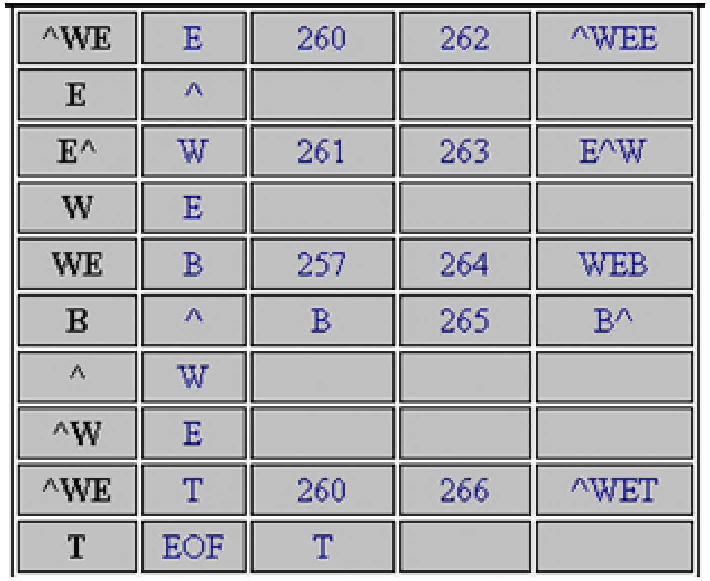
LZW Compression Algorithm:

```
w = NIL;
while ( read a character k )
    {
      if wk exists in the dictionary
       w = wk;
      else
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
```

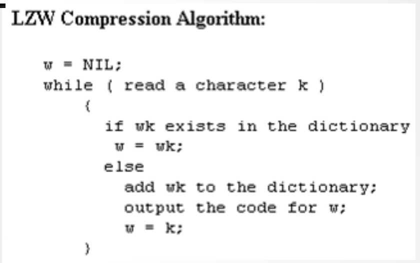
- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.

Lempel-Ziv-Welch Algorithm

- **Example:** Input string is "^WED^WE^WEE^WEB^WET"

w	k	Output	Index	Symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

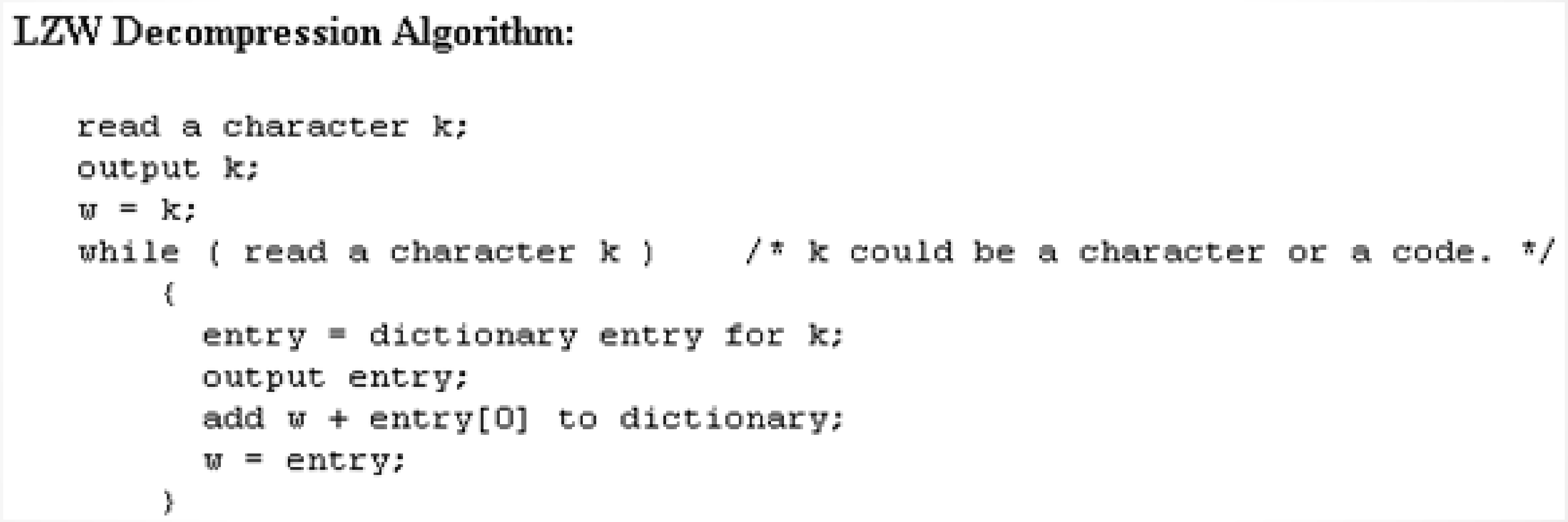


LZW Compression Algorithm:

```
w = NIL;
while ( read a character k )
    {
      if wk exists in the dictionary
        w = wk;
      else
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
```

- Dictionary contains Index and Symbol.
- A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits.

Lempel-Ziv-Welch Algorithm

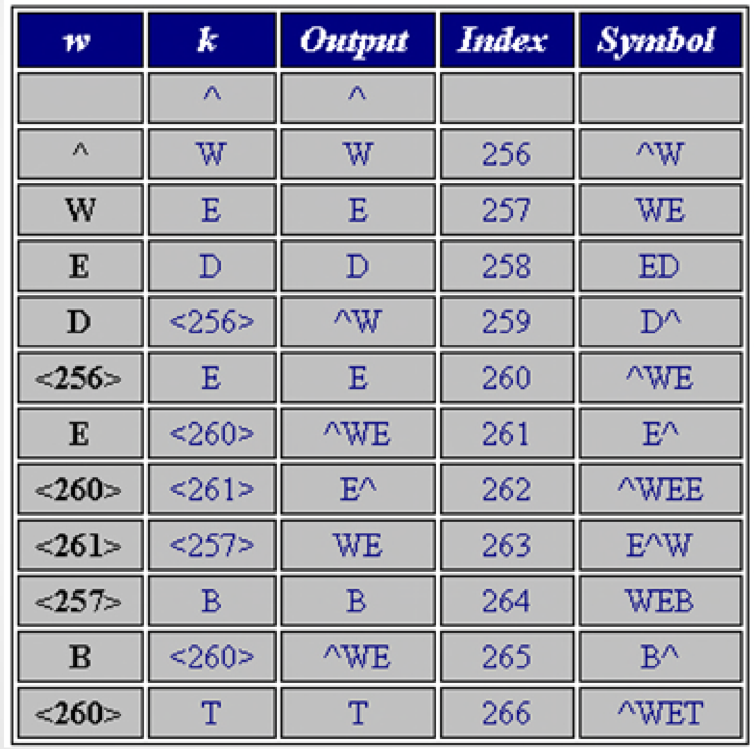


LZW Decompression Algorithm:

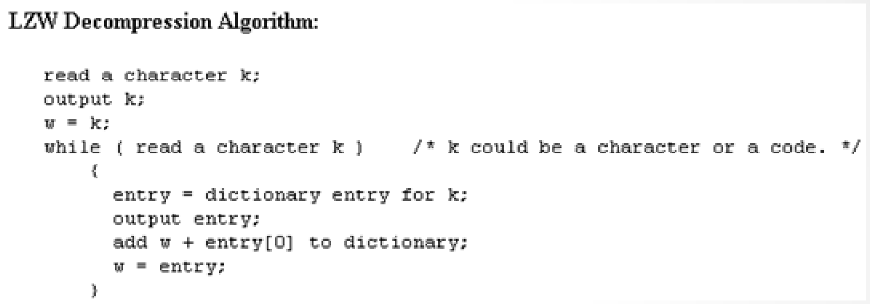
```
read a character k;
output k;
w = k;
while ( read a character k )    /* k could be a character or a code. */
    {
      entry = dictionary entry for k;
      output entry;
      add w + entry[0] to dictionary;
      w = entry;
    }
```

Lempel-Ziv-Welch Algorithm

- **Example:** Input string is "^WED<256>E<260><261><257>B<260>T".

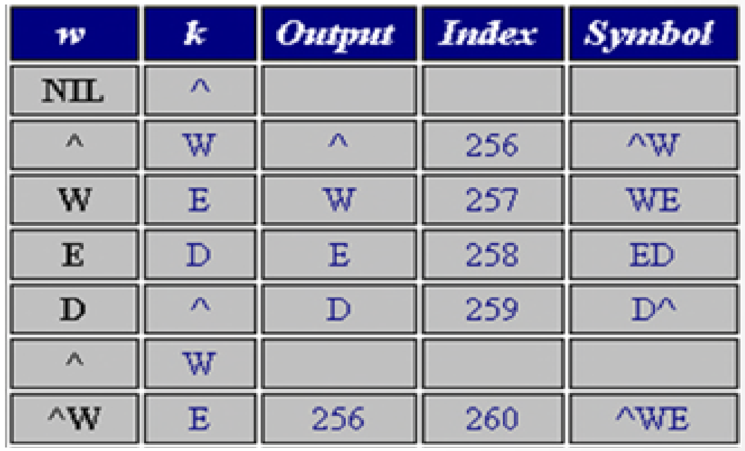


w	k	Output	Index	Symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET



LZW Decompression Algorithm:

```
read a character k;
output k;
w = k;
while ( read a character k )    /* k could be a character or a code. */
     {
       entry = dictionary entry for k;
       output entry;
       add w + entry[0] to dictionary;
       w = entry;
     }
```



w	k	Output	Index	Symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE

- Lempel-Ziv-Welch is a dictionary-based compression method. It maps a variable number of symbols to a fixed length code.

Arithmetic Coding

- Arithmetic coding don’t use the idea of replacing an input symbol with a specific codeword. Instead, it takes input symbols and replaces it with one floating point output number.
- The longer and the more complex the message, the more bits are needed in the output number.
- The output from arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them.

Arithmetic Coding

- For example, if I was going to encode the random message "BILL GATES", I would have a probability distribution that looks like this:

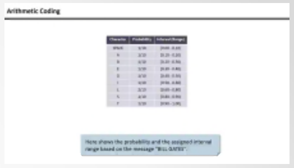
Character	Probability
SPACE	1/10
A	1/10
B	1/10
E	1/10
G	1/10
I	1/10
L	2/10
S	1/10
T	1/10

Arithmetic Coding

- As the character probabilities are known, the individual symbols need to be assigned a range. It doesn't matter which characters are assigned which segment of the range, as long as it is done in the same manner by encoder and decoder.

Character	Probability	Interval (Range)
SPACE	1/10	[0.00 - 0.10)
A	1/10	[0.10 - 0.20)
B	1/10	[0.20 - 0.30)
E	1/10	[0.30 - 0.40)
G	1/10	[0.40 - 0.50)
I	1/10	[0.50 - 0.60)
L	2/10	[0.60 - 0.80)
S	1/10	[0.80 - 0.90)
T	1/10	[0.90 - 1.00)

Arithmetic Coding



Character	Probability	Interval (Range)
SPACE	1/10	[0.00 - 0.10)
A	1/10	[0.10 - 0.20)
B	1/10	[0.20 - 0.30)
E	1/10	[0.30 - 0.40)
G	1/10	[0.40 - 0.50)
I	1/10	[0.50 - 0.60)
L	2/10	[0.60 - 0.80)
S	1/10	[0.80 - 0.90)
T	1/10	[0.90 - 1.00)

New Character	Low value	High Value
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

- The final low value, 0.2572167752 representing the message "BILL GATES" using this encoding scheme.

Arithmetic Coding

Decoded Number	Output Symbol	Low	High	Interval
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	SPACE	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.1	0.2	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

New Decoded Number=(Decoded Number-Low)/Interval

Character	Probability	Interval (Range)
SPACE	1/10	[0.00 - 0.10)
A	1/10	[0.10 - 0.20)
B	1/10	[0.20 - 0.30)
E	1/10	[0.30 - 0.40)
G	1/10	[0.40 - 0.50)
I	1/10	[0.50 - 0.60)
L	2/10	[0.60 - 0.80)
S	1/10	[0.80 - 0.90)
T	1/10	[0.90 - 1.00)