Lossless Compression

--- ref. Major from ch.2 of textbook1

- **Definition:** Compression methods for which the original uncompressed data set can be recovered exactly from the compressed stream
- Usage: digital medical data , bitonal image, artwork design, ...

Preliminaries

Generic model: given an input set of symbols, a modeler generates an estimate of the probability distribution of the input symbols. This probability model is then used to map symbols into codewords.

Entropy coding: the combination of the modeling and the symbol-to-codeword mapping functions is usually referred to as "Entropy coding"

Key idea: use short codewords for symbols that occur with high probability and long codewords for symbols that occur with low probability

Entropy decoding: decompression process

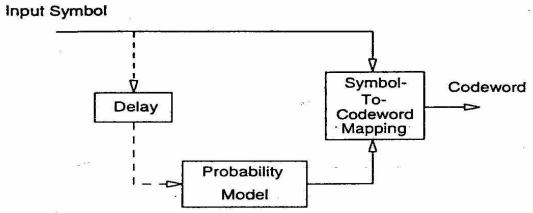

Figure 2.1 A generic model for lossless compression.

Message-to-Symbol Partitioning

- **Typical image**: from 256*256 to 64000*64000 pixels
- **Instance and symbol**: the whole image can be looked as one instance with n x n symbols. Ex. 256x256 image --> 64000 symbols.
- **Distribution model:** it's difficult to provide probability model for such long symbols. Revision --> use one pixel as message, ex. 8 bits/pel will have 256 symbols and then estimate the distribution.
- **Partitioning:** partition the data into blocks, better to match with the hardware units 8-, 16-, 32-, or 64-bit, where each block may be composed of several input units. The more higher units, the higher compression ratio and higher coding complexity.

Differential Coding

- **Key idea:** to make the distribution more amenable
- **Coding theory:** the ideal symbol-to-codeword mapping function will produce a codeword requiring $\boldsymbol{log_2(1/p_i)}$ bits.
- **Algorithm:** instead of the pixels of the image, $x_1, x_2, \ldots, x_n$ to $y_i = x_i - x_{i-1}$, where $i=1,2,\ldots,N$, and $x_0=0$.
- **Prediction residual:** *y_i is called the prediction residual of x_i*

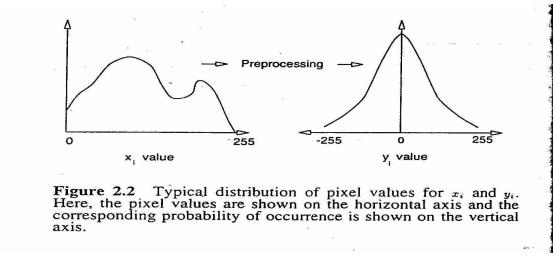

Figure 2.2 Typical distribution of pixel values for x_i and y_i. Here, the pixel values are shown on the horizontal axis and the corresponding probability of occurrence is shown on the vertical axis.

a) Uniform distribution
b) Skewed probability distribution

Huffman Encoding

1952 D.A. Huffman

Algorithm:

1. Order the symbols according to their probabilities.
   ---the frequency of occurrence of each symbol must be known a priori.
2. Apply a contraction process to the two symbols with the smallest probabilities
3. Repeat the previous step until the final set has only one member.

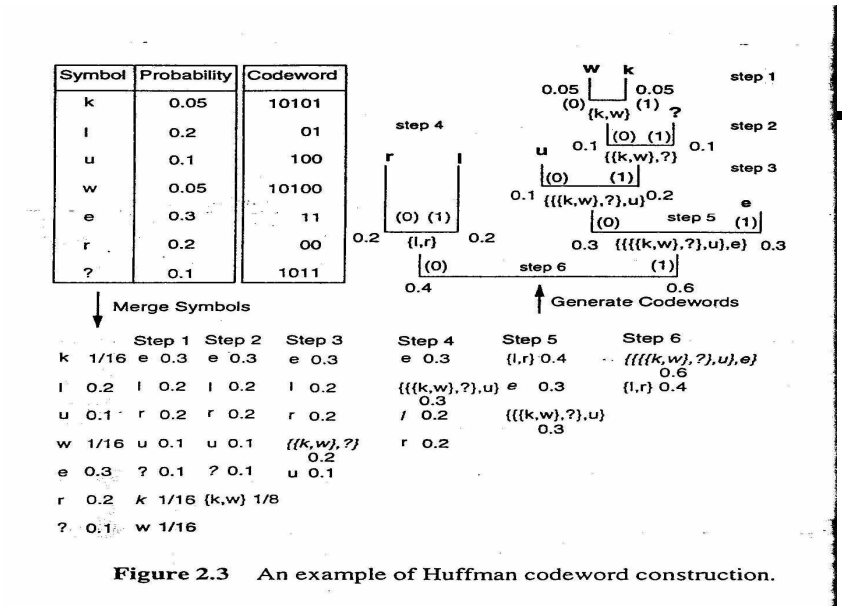

Figure 2.3 An example of Huffman codeword construction.

Huffman Encoding

- Average codeword length:
  - *defined as:* $l_{avg} = l_i * p_i$, where l_i is the codeword length for the codeword correponding to symbol s_i.
  - is a measure of the compression ratio, ex Cr= 3 bits/ 2.6 bits =1.15
- The modeling and the symbol-to-codeword mapping functions are combined
- The codewords are determined by traversing through the "binary tree"

Properties of Huffman Codes

- **Entropy of source S** : $H(S) = \sum_i p_i * \log(1/p_i)$
- **Information theory:** if the symbols are distinct, then the average number of bits needed to encode them is always bounded by their entropy
- As for Huffman coding , $H(S) \le l_{ave} < H(S)+1$
  - Or $H(S) \le l_{ave} < p + 0.086$
  - where p is the probability of the most frequently occurring symbol
  - the equality is achieved when all symbol probabilities are inverse powers of two, that means
  - the lossin performance can be viewed as a result of the quantization effect on the symbol probabilities.
- Bottom-up method v.s. top-down method, $O(n\log_2 N)$
- Problem?? --> fixed-length input symbols ae mapped into the variable-length codewords

Huffman Decoding

----Bit-Serial Decoding

- **Concept:** reconstruct the binary coding tree to the decoder from the symbol-to-codeword table
- **Algorithm:**
  - 1. Read the input compressed stream bit by bit and traverse the tree until a leaf is reached
  - 2. Discard each used input bit
  - 3. When the leaf node is reached, out the symbol at the leaf node.
    - This completes the decoding for this symbol
- The above scheme has a *fixed input bit rate* but a *variable output symbol rate*

Huffman Decoding

---- Lookup-Table-Based Decoding

- **Lookup table:** construct from the symbol-to-codeword mapping table
- **Two-tuple representation**: (s_i, l_i) , symbol and length
- The address of table has L bits, with the first l_i bits representing the original codeword for s_i
- **Algorithm:**
  - 1. Read L bits from the compressed input bit stream into a buffer
  - 2. Check it in the lookup table ,and thus decoding one symbol, say, s_k
  - 3. Discard the first l_k bits from the buffer and append l_k bits from the input to the buffer has again L bits.
  - 4. Repeat steps 2 and 3 until all the symbols have been decoded
- The above scheme has a *variable input bit rate* but a *fixed output symbol rate*
- *Adv: fast and simple*
- *Disadv: L is the longest codeword, means the possible space constraints*

Huffman Codes With Constrained length
---the constraint longest length

- **Problem:** yield possibly when some of the symbol probabilities are extremelt small. They will require a large number of bits.
- **Solution:** shorten and hierarchy
- **Algorithm:**
  - Partition symbol set S into two sets S_1 ,S_2 ,using $p_i=1/2^L$ as the boundary , where L is the max. codeword length expected
  - Create a special symbol Q such that its frequncy of occurrence is the sum of all p_i in S_2
  - Augment S_1 by Q to form a new W . The new set has the occurrence frequencies corresponding to symbols in S_1 and the special symbol Q.
  - Reconstruct the Huffman tree for W and S_2

Huffman Codes With Constrained length
---the constraint longest length

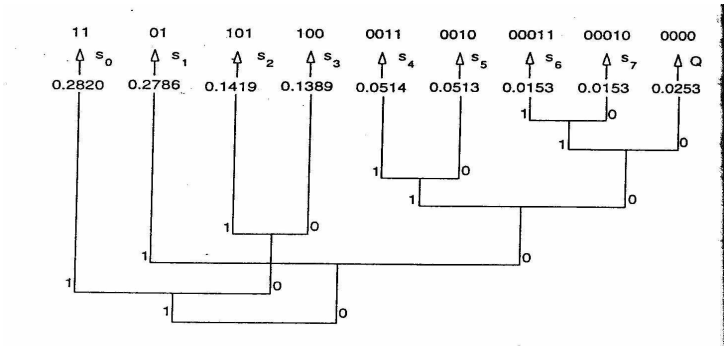

Figure 2.4 Shortened Huffman code.

Huffman Codes With Constrained length
---the constraint longest length

- For symbols in S_2, add a codeword which is its L bit binary representation, in this example, L = 7

Symbol i	p_i	l_i	Codeword	Additional
0	0.28200	2	11	
1	0.27860	2	01	
2	0.14190	3	101	
3	0.13890	3	100	
4	0.05140	4	0011	
5	0.05130	4	0010	
6	0.01530	5	00011	
7	0.01530	5	00010	
8	0.00720	11	0000	0001000
9	0.00680	11	0000	0001001
10	0.00380	11	0000	0001010
11	0.00320	11	0000	0001011
12	0.00190	11	0000	0001100
13	0.00130	11	0000	0001101
14	0.00070	11	0000	0001110
15	0.00040	11	0000	0001111

Table 2.2 Constrained-length (L = 7 bits) Huffman codewords.

@NTUEE
DSP/IC Lab

Huffman Codes With Constrained length
---the constraint longest length

- ***The decoding procedure*** is the same as the "unconstrained Huffman code" except for when "*Q*" is read.
- **Procedures:**
  - let m_k be the symbol and l_k the codelength
  - If m_k is not the special symbol *Q*, then we have correctly decoded this symbol
  - If m_k is the special symbol *Q*, additional bits from the input bit stream are need. The next l_k input bits are fetched into the buffer to represent the additional binary codeword and thus we have correctly decoded a symbol from S_2
- The resulting l_{avg} is larger than that for unconstrained- Huffman code
- The output symbol rate is not a constant
- "*Q*" is an escape key

@NTUEE
DSP/IC Lab

Constrained-Length Huffman Codes
----Ad-Hoc Design

The method does not use a prefix code

- Sort s_i, so that $p_k > p_{k+1}$
- For max. codeword length L , define $T=2^{-L}$
- For $p_i<T$, set $p_i=T$
- Design the codebook using the modifidied p_i values and the unconstrained Huffman coding method
- Unconstrained Huffman code l_{avg} < Ad-Hoc design l_{avg} < Prefix-code design l_{avg}
- With a lookup-table-based decoder ,the output decoding rate is constant

Symbol i	p_i	l	Codeword	Reordered l	Reordered Codeword
0	0.28200	2	11	2	11
1	0.27860	2	01	2	01
2	0.14190	3	101	3	101
3	0.13890	3	100	3	100
4	0.05140	4	0010	4	0010
5	0.05130	4	0001	4	0001
6	0.01530	6	001100	6	001100
7	0.01530	6	001101	6	001101
8	0.00720	7	0011110	6	000010
9	0.00680	7	0011111	6	000011
10	0.00380	7	0011100	6	000000
11	0.00320	7	0011101	6	000001
12	0.00190	6	000010	7	0011110
13	0.00130	6	000011	7	0011111
14	0.00070	6	000000	7	0011100
15	0.00040	6	000001	7	0011101
l_{avg}			2.7308		2.7141

Table 2.4 Constrained-length (L = 7 bits) Huffman codewords - Ad-hoc design.

Constrained-Length Huffman Codes
----The Voorhis Method

- Sort $s_1,s_2,...,s_N$ so that $p_1>p_2>...>p_N$
- Determine codeword length $l_1,l_2,...l_N$, subjected to the constraint $l_1<l_2<...<l_N<L$
- Determine the codewords
- Its l_{avg} is close to optimum
- It outperforms the ad-hoc method , but is more complex in physical relization

Arithmetic Coding

- It separates the coding from the modeling, thus allowing the ***dynamic adaption of the probability model*** without affecting the design of the coder

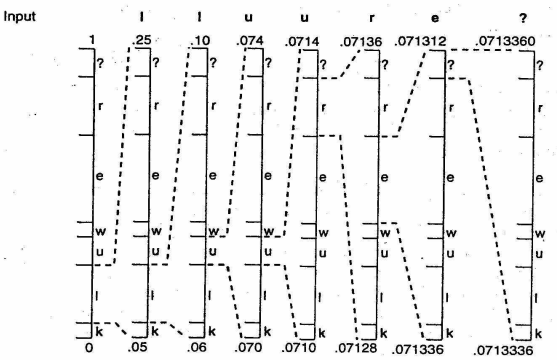

Figure 2.6 Arithmetic coding. Example of the encoding process.

s_i	p_i	Subinterval
k	0.05	[0.00,0.05)
l	0.20	[0.05,0.25)
u	0.10	[0.25,0.35)
w	0.05	[0.35,0.40)
e	0.30	[0.40,0.70)
r	0.20	[0.70,0.90)
?	0.10	[0.90,1.00)

Table 2.6 Symbol subinterval ranges.

@NTUEE DSP/IC Lab

The Decoding Process

DecodeSymbol (*value*):
Begin
$Previous_{low} = 0.$
$Previous_{high} = 1.$
$Range = Previous_{high} - Previous_{low}.$
repeat
Find i such that
$cumprob_i \leq \frac{value - Previous_{low}}{Range} < cumprob_{i-1}.$
Output symbol corresponding to i from the decoding table
Update:
$Previous_{high} = Previous_{low} + Range * cumprob_{i-1}.$
$Previous_{low} = Previous_{low} + Range * cumprob_i.$
$Range = Previous_{high} - Previous_{low}.$
until symbol decoded is ?.
End

We show a few steps of the decoding process in our example.

s_i	i	$cumprob_i$
k	7	0.00
l	6	0.05
u	5	0.25
w	4	0.35
e	3	0.40
r	2	0.70
?	1	0.90
	0	1.00

Table 2.7 Modified decoder table.

@NTUEE DSP/IC Lab

Implementatiom Issues

- Incremental output
- High-precision arithmetic
- Probability modeling

Standards for Lossless Compression
--- Facsimile Compression Standards

- Run-length coding scheme: (position, value) ==> (run, value)
- Modified Huffman (MH) Code: The image is treated as a sequence of scanlines , and a run-length description is obtained. Then the Huffman code is applied.An EOF codeword is inserted after each line
- Modified Read (MR) Code: Here, pixel values in a previous line is used as predictors. The remaining is the same as MH code

Figure 2.7 Sample scanline of a bitonal image.

JBIG Compression Standards

- It consists of a modeler and an arithmetic coder. The modeler is used to estimate the symbol probabilities.
- It has higher compression ratios than facsimile compression standards , but the software implementation is slower

Lossless JPEG Standards

- It uses differential coding followed by a Huffman coder or an arithmetic coder
- The prediction residual for pixel X is r=y-X , where y can be computed in various ways

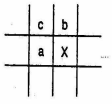

Figure 2.9 Lossless JPEG prediction kernel.

- Let a=100 ,b=191 ,c=100 ,X=180 , y=(a+b)/2=145 , r=145-180=-35 ,which belongs to category 6
- The binary number for -35 is 011100 ,and the Huffman code for 6 is 1110. Thus the overall code is 1110011100

Category	Prediction Residual
0	0
1	-1, 1
2	-3, -2, 2, 3
3	-7, .., -4, 4, .., 7
4	-15, .., -8, 8, .., 15
5	-31, .., -16, 16, .., 31
6	-63, .., -32, 32, .., 63
7	-127, .., -64, 64, .., 127
8	-255, .., -128, 128, .., 255
9	-511, .., -256, 256, .., 511
10	-1023, .., -512, 512, .., 1023
11	-2047, .., -1024, 1024, .., 2047
12	-4095, .., -2048, 2048, .., 4095
13	-8191, .., -4096, 4096, .., 8191
14	-16383, .., -8192, 8192, .., 16383
15	-32767, .., -16384, 16384, .., 32767
16	32768

Table 2.9 Prediction residual categories for lossless JPEG compression.