EE663
Image Processing
Lossless Compression

Huffman coding
Arithmetic coding
Dictionary techniques
Predictive coding
Bit-plane encoding

Dr. Samir H. Abdul-Jauwad
Electrical Engineering Department
King Fahd University of Petroleum & Minerals

Lossless Compression Techniques

Lossless Compression Techniques

- Run Length Encoding
- Shannon–Fano Coding
- Huffman Coding
- Tunstall Coding
- Arithmetic Coding
- Lempel-Ziv Coding
- Lossless Predictive Coding
- Bit–plane Encoding

Run Length Encoding (RLE)

- A run is a subsequence of the same symbol value
- RLE replaces each maximal run by the length of the run followed by the repeating symbol of the run

  0000100111100011111111000

  4,0;1,1;2,0;4,1;3,0;6,1;3,0

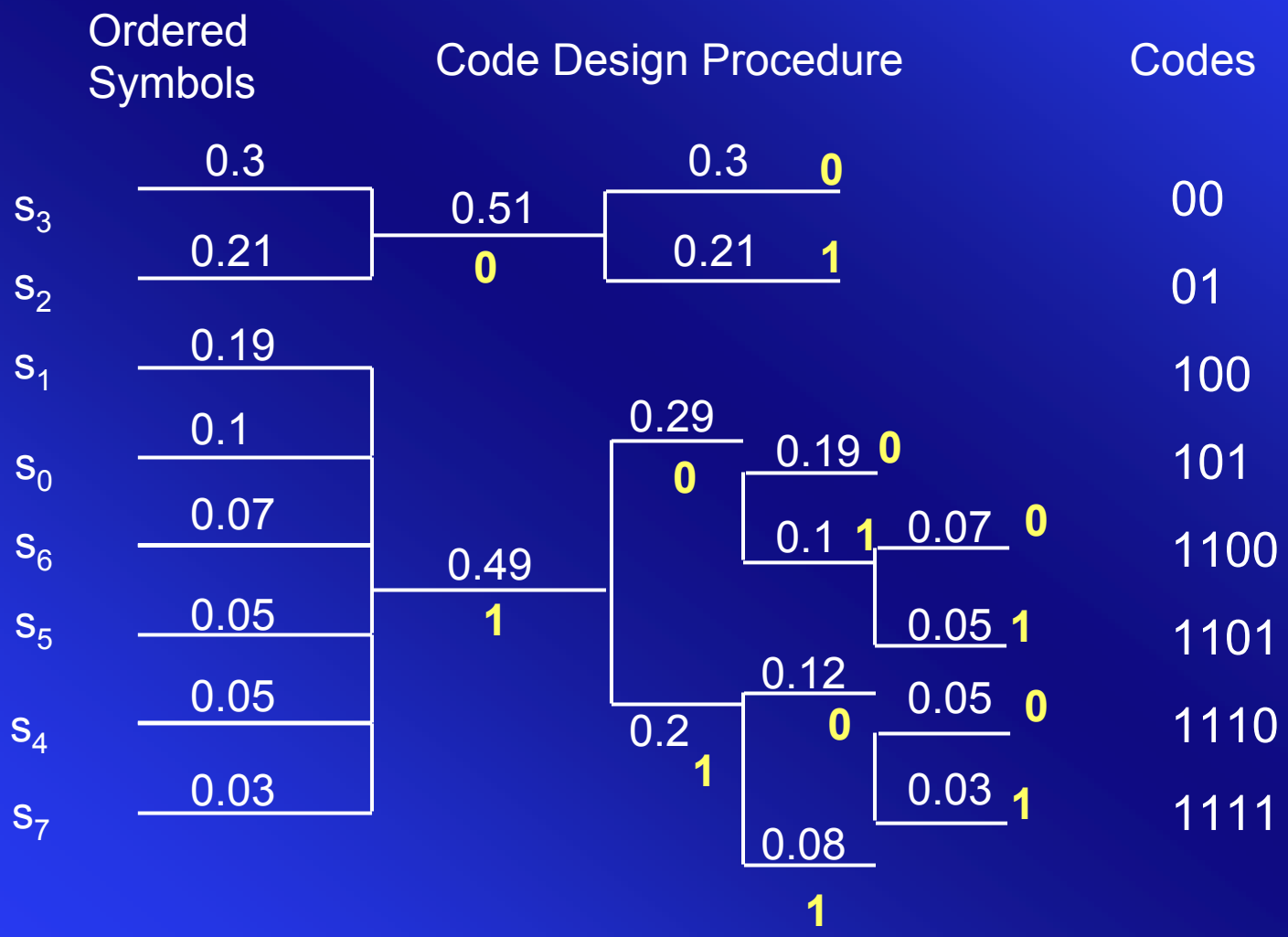
  0 & 1 can be removed if the alphabet is binary

  $\Rightarrow$ 4,1,2,4,3,6,3
- The RLE sequence can then be coded using Huffman Coding

Shannon-Fano Coding

1. Sort set of symbols S in order of non-increasing probabilities.
2. Divide S into two parts S_1, S_2 such that each part has approximately equal probability
3. Assign ‘0’(‘1’) to symbols of $S_1(S_2)$
4. Continue (2) and (3) on each of the parts until each part contains only one symbol

An Example of Shannon-Fano Coding



Huffman Coding

- Statistical Coding
- Popular Technique
- Builds a Code Tree
- Many Variations
  - Adaptive Huffman Coding
  - Minimum Variance Huffman Coding

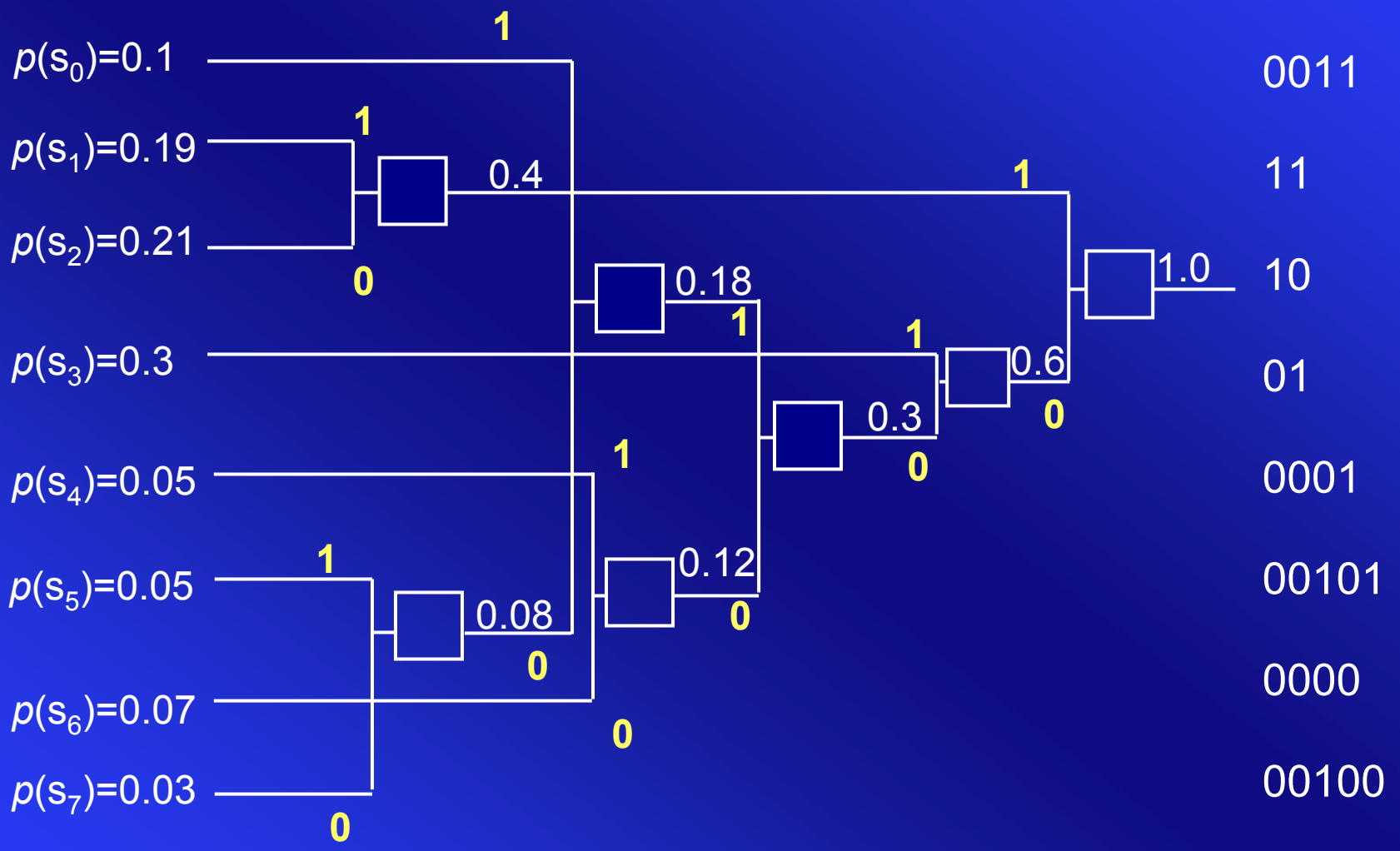
Building Huffman Tree

- Make a list L of all symbols
- Create a node for each symbol
- Find the two smallest probability symbols s_j, s_k
- Create a 'new' node as the parent of s_j,s_k with probability $p_j + p_k$
- Remove s_j,s_k from L
- Repeat the above three steps until L is not empty
- Label left edges with '0' and right edges with '1'

Huffman Encoding

Code symbol s_i with the binary sequence obtained using the labels on the path from the root to the node s_i

An Example of Huffman Coding



HUFFMAN ENCODE (in: Σ, Π)

- begin
  - Create a node labeled $s_i, 1 \le i \le n$
  - F ← { s_i | $1 \le i \le n$ }
  - for i = 1 to n-1 do
  - Find the two smallest probability symbols $s_j, s_k \in F$
  - Create a new node with label si` as the parent of s_j, s_k
  - with probability $p_j + p_k$

HUFFMAN ENCODE (in: Σ, Π)

- $F \leftarrow (F - \{s_j, s_k\} \cup \{s_i'\}$
- endfor
- Label left edges of tree with ‘0’ and right edges with ‘1’
- for i = 1 to n do
  - code s_i with the binary sequence labeling the path from the root to the node s_i
- endfor

- end

Arithmetic Coding

- Non-block code
- Codeword assigned to entire sequence of symbols.
- Unique Code for each Input Sequence.
- Well suited to:
  - Small Size Alphabets
    - E.g. Binary sources
  - Alphabets with Highly Skewed Probabilities

Arithmetic Coding Principle

Given:

- Alphabet S is finite
- All possible sequences of length ‘m’ is finite
- All possible sequences: $S^0 \cup S^1 \cup S^2$ ... is countably infinite
- Number of real numbers in the internal [0,1) is uncountably infinite

Implication:

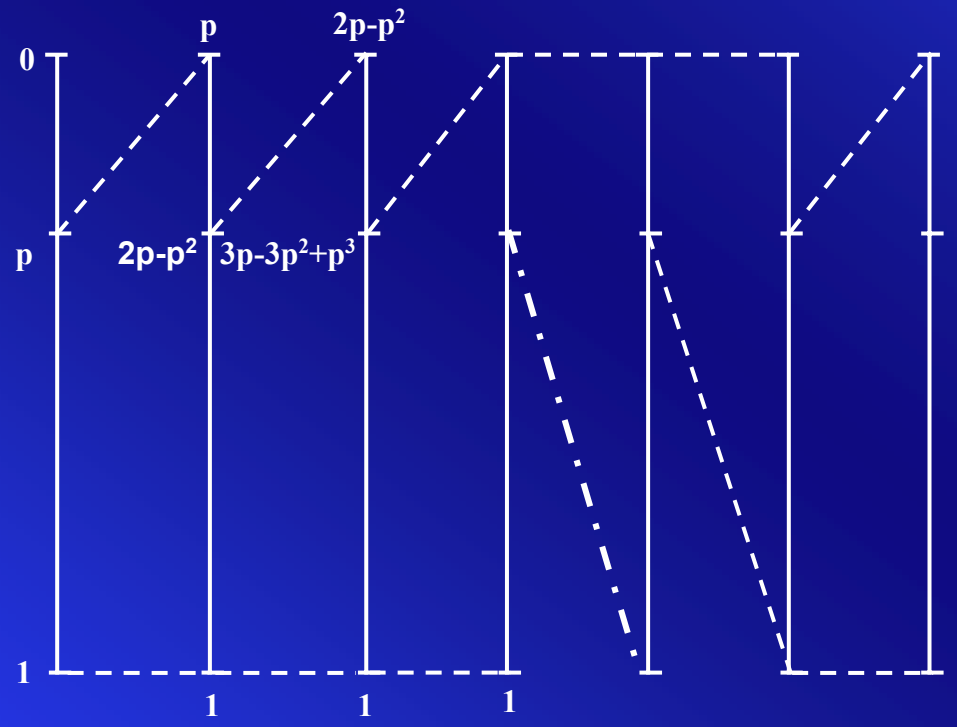
- It is possible to assign a unique code (tag) for any given sequence of symbols

Arithmetic Coding on Binary Alphabet: Procedure

- Alphabet S={s_1,s_2}; Probabilities P={p,q}
- Divide Interval [0,1] in the ratio p,q
  - Intervals correspond to Symbols
- For Successive Symbols in Input do:
  - Pick Sub-interval Corresponding to symbol
  - Divide the Sub-interval in the ratio p,q
- Pick any Point (Usually Middle) in the final Sub-Interval.
- Binary Encode this point which is the Tag (code) for the Input Sequence

Arithmetic Coding on Binary Alphabet: Example

- S = {0,1}; P = { p,q }; q = 1-p
  (Say 0: black ; 1:white and q>>p) (Typical of printed text; fax)
- Example Sequence: 1 1 1 0 0 1 1 1 1 0 1 1



Question

- Give the procedure for decoding a tag (Arithmetic Decoding for Binary Alphabet)

Break

Arithmetic Coding on Non-Binary Alphabet Preliminaries

- Given: Alphabet S and Probabilities P
- Define a Random Variable X such that $X(s_i)=i$
- Probability Density Function for X: $P(X=i) = P(s_i)$
- Cumulative Density Function (CDF):

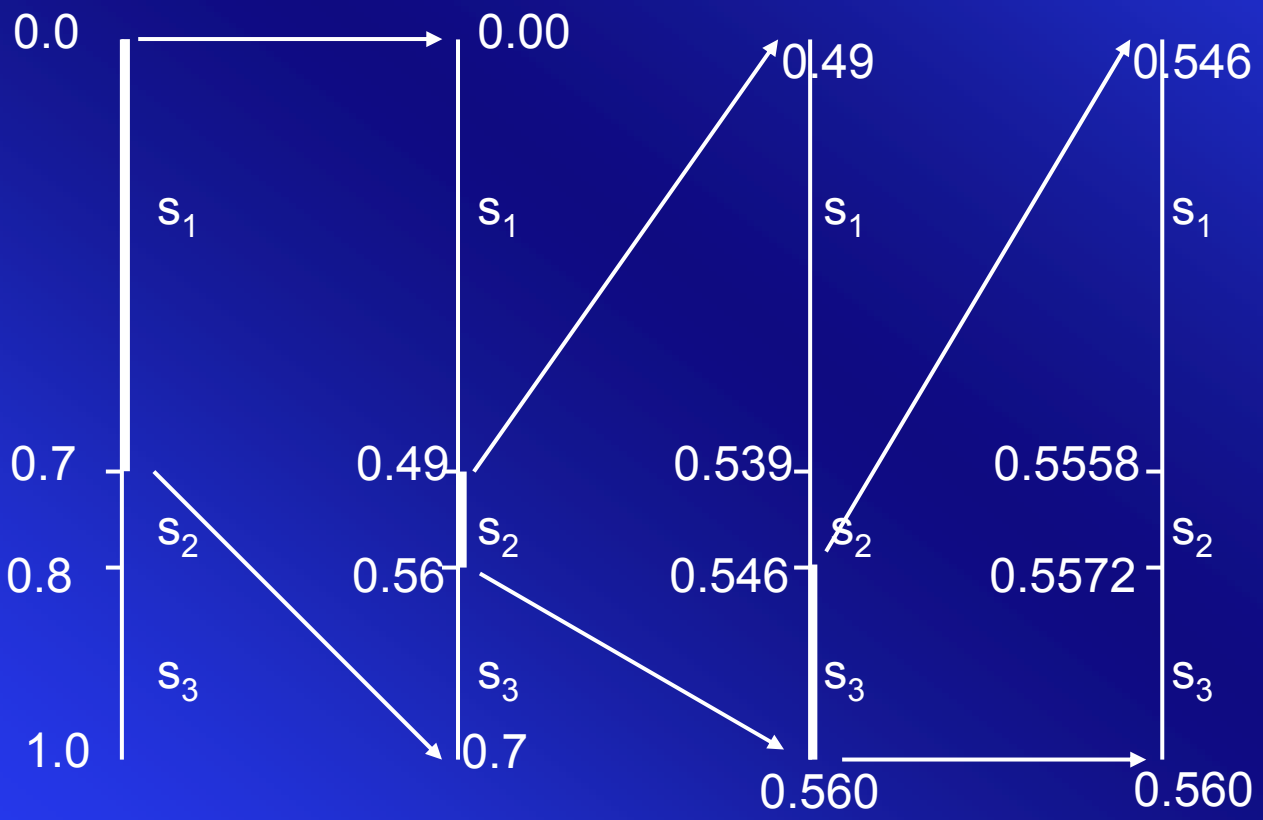
$$F_X(i) = \sum_{k=1}^{i} P(X = k)$$

Arithmetic Coding on a Non Binary alphabet: Procedure

- Alphabet $S=\{s_1,s_2\ldots s_N\}$ Probabilities $P=\{p_1,p_2\ldots p_N\}$
- Divide Interval [0,1] in proportions given by
  $F_x(i),1\leq i \leq N$
  - Intervals Correspond to Symbols
- For Successive Symbols in Input do:
  - Pick Sub-Interval Corresponding to Symbol
  - Divide Sub-Interval in proportions given by $F_x(i)$
- Pick any point (Usually Middle) in the final Sub-Interval.
- Binary Encode this point which is the Tag (code) for the Input Sequence

Arith. Coding on Non-Binary Alphabet: Example

- S={a,b,c} P={0.7,0.1,0.2}
- $F_X(1) = 0.7$, $F_X(2) = 0.8$, $F_X(3) = 1.0$
- CDF is used to partition the interval into sub-intervals of the form: $[F_X(j-1), F_X(j)]$ where $X(s_i)=j$



Decoding Arithmetic Code

- Given: S,P, Tag, m=Length of Sequence
- Start with Interval of [0,1]
- Repeat the following three steps ‘m’ times
  - Divide Sub-Interval in proportions given by $F_x(i)$
  - Determine the Sub-Interval containing the Tag
  - Output the Symbol corresponding to the Sub-Interval

Arithmetic Decoding: Example

S = { a,b,c}

P = { 0.7,0.1,0.2 }

Tag = 0.7665

L = 0;R = 1;

t* = 0.7665

J = 2 since

t* ε [$F_X(1),F_X(2)$]

L' ← 0 ; R' ← 1;

L ← 0.7; R ← 0.8

s_m^k ← b

t* = (0.7665 – 0.7)/ (0.1) = 0.665

j = 1 since t* ε [$F_X(0),F_X(1)$]

L' ← 0.7 ; R' ← 0.8 ;

L ← 0.7 + (0.1) 0 = 0.7 ;

R ← 0.7 + (0.1) 0.7 = 0.77 ;

s_m^k ← ba

t* = (0.7665 – 0.7)/ (0.07) = 0.95

j = 3 since t* ε [$F_X(2),F_X(3)$]

L' ← 0.7 ; R' ← 0.77 ;

L ← 0.7 + (0.007) 0.8 = 0.756 ;

R ← 0.7 + (0.07) 1 = 0.77 ;

s_m^k ← bac

… etc.

ARITHMETIC ENCODE (in: s_m^k,π;out: tag)

- **begin**
  - *Compute $F_X(i)$;*
  - *$L \leftarrow 0$; $R \leftarrow 1$;*
  - **for** *l* = 1 **to** *m* **do**
    - *$j \leftarrow i_l$;*
    - *$L' \leftarrow L$; $R' \leftarrow R$;*

ARITHMETIC ENCODE (in: *smk*,π;out tag)

- $L \leftarrow L' + (R' - L') F_X(j-1)$;
- $R \leftarrow L' + (R' - L') F_X(j)$;

– **endfor**

– *tag* ← *a suitable point* ∈ [*L*, *R*)

- **end**

Compute F_X

- **begin**
  - $F_X(0) \leftarrow 0$;
  - **for** $i = 1$ **to** n **do**
    - $F_X \leftarrow F_X(i - 1) + \Pi(i)$;
  - **endfor**
- **end**

ARITHMETIC DECODE (in: *tag*,π;out: s_m^k)

- **begin**
  - $s_m^k \leftarrow \emptyset$
  - COMPUTE_F_X ;
  - *L* ← *0; R*←*1 ;*
  - **for** *I* = 1 **to** *m* **do**
    - *t** ← (*tag - L)/(R - L);*
    - *Find j* ($1 \le i \le n$) *s.t. t** ∈ [F_X(*j* - 1) , F_X(*j*));

ARITHMETIC DECODE (in: *tag*,π;out: s_m^k)

- $L' \leftarrow L$; $R' \leftarrow R$;
- $L \leftarrow L' + (R' - L') F_X(j - 1)$;
- $R \leftarrow L' + (R' - L') F_X(j)$;
- $s_m^k \leftarrow s_m^k \, o \, s_j$

– **endfor**

- **end**

Question

- What is the Complexity of Arithmetic Coding?
- What is the Complexity of Arithmetic Decoding?

  (Asymptotic Complexity)

Dictionary Technique

- Build a list of commonly occurring patterns : Dictionary D
- A pattern occurring in the input is searched in D

  - **if** found, use its index in D in the encoding

  - **else** code it with some (less efficient) method

Example

L = {a,b,.....z} P={‘ ‘, ‘,’ , ‘.’ , ‘?’, ‘;’ , ‘!’}

$S = L \bigcup P$ Alphabet size = 32

- Consider 4-character words
  - first 3 characters $\in$ L
  - last character $\in$ P
- If each character is equally likely,
  - $\Rightarrow$ 5 bits/character
  - $\Rightarrow$ 20 bits/word

Example (Contd.)

- Size of Dictionary, D = 256
- Put 256 most likely words in D
- **If** word (pattern) $\in$ D **then**
  - transmit/store 0<8-bit index> 9 bits $\Rightarrow$
- **Else** transmit/store 1<20-bit encoding> $\Rightarrow$ 21 bits
- p = probability of word $\in$ D
- Average number of bits/word,

  $R = 9p + 21(1-p) = 21 - 12p$
- For R to be <20, p should be $\geq \frac{1}{12} = 0.084$

Dictionary Coding Variants

- Generic: Lempel – Ziv Coding
- Variants
  - LZ77, LZ78, LZW
- Applications:
  - UNIX compress
  - Gzip
  - GIF
  - V.42 (Compression over modems)

LZ77: Outline of Encoding

- Process portion of input under a sliding window.
- Sliding window has two parts:
  - Search Buffer (B1): Contains portion of recently encoded sequence.
  - Look-ahead buffer (B2) contains next portion of the sequence to be encoded.
- Move a Search pointer (from right to left) over search buffer B1 to look for the longest match for the symbols in the Look-ahead buffer B2.
- If match is found, encode it with < o, l, c >

o: Offset = distance of the pointer in B1 from the start of B2

l : Length of match,

c: Codeword of symbol in B2 that follows the match

LZ77: Encoding Example

Input: …….c a b r a c a d a b r a r r a r r a d….

Sliding window length = 13, length of B1 = 7, length of B2 = 6

Current Position: c a b r a c a | d a b r a r

- Look for a match for '**d**' in B1.
  - None. Encoding: <0,0,C(d)> (where C(d) is code for 'd')
  - Slide window by 1 character

    a b r a c a d | a b r a r r
- Matches for '**a**':
  - at offset 2, length 1
  - at offset 4, length 1
  - at offset 7, length 4

  Encoding : < 7,4,C(r) >

LZ77: Encoding Example (Contd.)

- Slide window by 5 characters.

  a d a b r a r | r a r r a d

- Matches for r
  - at offset 1,length 1
  - (at offset 3, length 3)
  - at offset 3,length 5 (using characters in B2 also)
  - Encoding : < 3,5,C(d) >
  - Encoded String :
    - < 0,0,C(d) > , < 7,4,C(r) >, <3,5,C(d) >

LZ77 Decoding Example

- Input: < 0,0,C(d) > , < 7,4,C(r) >,<3,5,C(d) > …

  c a b r a c a ↑

- Process < 0,0,C(d) > Output d

  c a b r a c a d

- Process < 7,4,C(r) >

  Move back 7 characters and copy 4 characters. Then append ‘r’.

  c a b r a c a d | a b r a r

  decoded characters

- Process < 3,5,C(d) >

  Move Back 3 characters and copy 5 characters. Then append ‘d’.

  c a b r a c a d a b r a r | r a r r a d

  decoded characters

Question

- Develop the pseudocode for LZ77 *encoding* as described in the previous slides
- Develop the pseudocode for LZ77 *decoding* as described in the previous slides

Predictive Coding

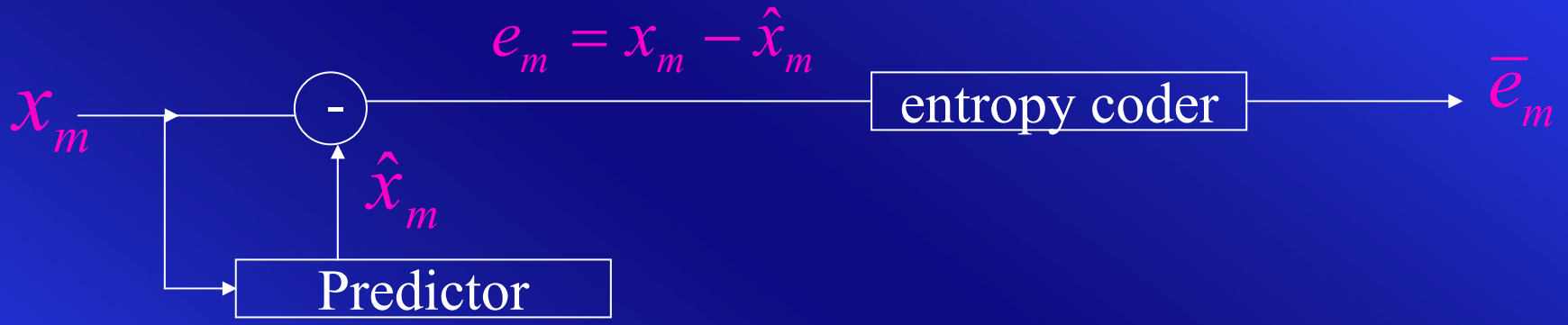
Lossless Predictive Coding

- Predictive technique
- Makes use of correlation between neighboring values
  - In typical images values of adjacent pixels are highly correlated
  - In typical audio values of adjacent samples are highly correlated (speech consists of mainly silence)
- DPCM (Differential Pulse Code Modulation): Most Common form of Lossless Predictive Coding

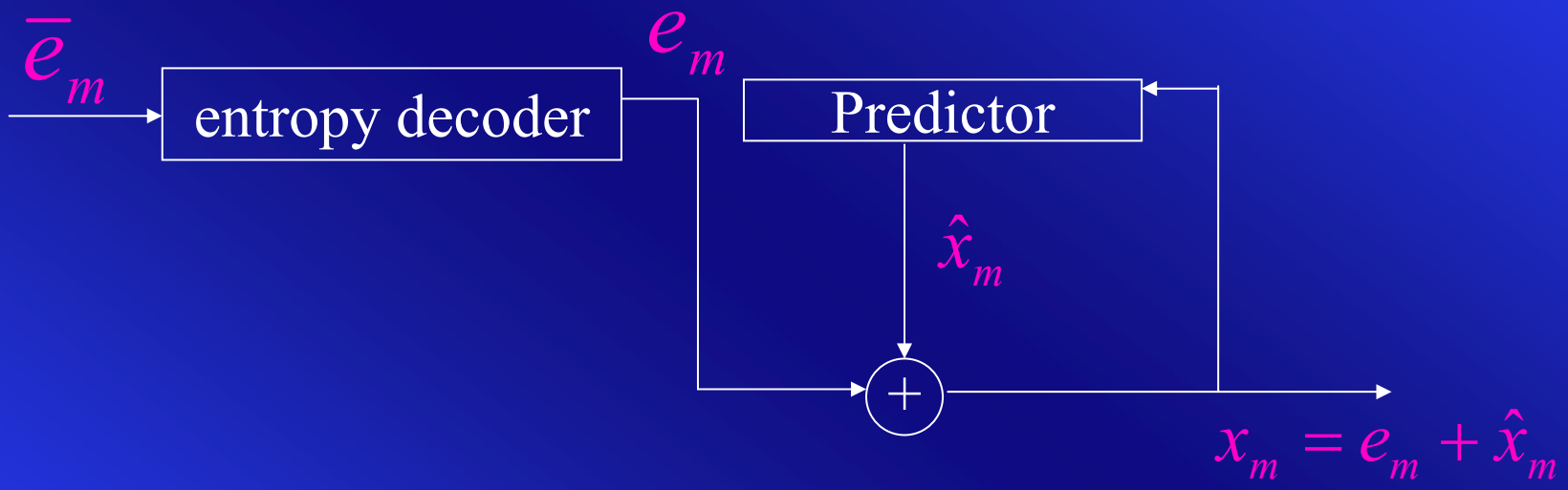
Lossless DPCM: General Procedure

- Use values of neighboring pixels to predict value of the current pixel.
  - Idea is to remove any structure that might exist.
- Find the difference between the actual and predicted values
  - Difference also called Error / Residual
- Encode the Residual

Lossless DPCM Encoder



Lossless DPCM Decoder



Bit Plane Encoding (in;I; out; code)

- Consider the Binary or Gray (or any suitable) code $c_{x,y}$ for each of pixel value $I(x,y)$of the image. Assume that the code is n bits.
- Divide I into bit planes: $T_{n-1}, T_{n-2}, \ldots T_0$, where $T_{k(x,y)}$ is the k^{th} bit of $c_{x,y}$
- Run Length Encode each T_k, $0 \le k \le n-1$,separately.
- Consider the collective alphabet of the run-length codes of all the bit planes, and derive the Huffman code.
- Huffman encode the run-length codes of each bit-plane T_k

Binary $\leftrightarrow$ Gray Conversions

$B = b_{n-1}...b_1b_0$ $\qquad\qquad$ $G=g_{n-1}...g_1g_0$

Binary $\rightarrow$Gray: $g_{n-1}=b_{n-1}$; $g_i=b_i \oplus b_{i+1}$, $\quad 0 \le i \le n-2$

Gray $\rightarrow$Binary: $b_{n-1}=g_{n-1}$; $b_i = g_i \oplus g_{i+1} \oplus ... \oplus g_{n-1}$ $\quad 0 \le i \le n-2$

$= g_i \oplus b_{i+1}$

End of lecture