BASIC COMPRESSION TECHNIQUES

N. C. State University

CSC557 ♦ Multimedia Computing and Networking

Fall 2001

Lectures # 04

Some Definitions and Tradeoffs

- Why needed if storage is cheap?
- Compression / Encoding vs. Decompression / Decoding
- Compression ratio
  - "before_size" : "after_size"
  - larger is better (greater reduction in size)
- Lossy vs. lossless compression
  - "loss" = sacrifice in quality
- Software vs. hardware implementation
  - advantages of software decompression
- Compression time vs. compression quality

Quality vs. Amount of Compression

- Reduced quality is often OK for multimedia
- Example compression (same image, multiple formats)
  - .bmp (1153 KB)
  - .jpg max quality (168 KB)
  - .gif (196 KB)
  - .jpg low quality (63 KB)
- "Reasonable" compression ratios with acceptable quality
  - Audio: 4:1
  - Images: 10:1→20:1
  - Video: 50:1

Some Lossless Compression Techniques

❶ Huffman (Entropy) Coding

❷ Run-Length Encoding

❸ Prediction-Based Coding

❹ LZW (Dictionary) Coding

Entropy and Information Theory

- Entropy = Uncertainty, Surprise, or Information in a "message"
- Symbols and symbol probabilities
  - based on past measurements, or predictions
  - q is # of symbols in the alphabet
  - i^{th} symbol occurs with probability p_i
- Entropy per symbol = $\sum_{i=1}^{q} p_i * \log_2(1/p_i)$
  - this is a lower bound on the average # of bits needed per symbol in the alphabet

Example

- Assume symbol probabilities = .5, .3, .1 and .1
- Entropy =

  .5*log(1/.5) + .3*log(1/.3) + .1*log(1/.1) + .1*log(1/.1) = 1.67
- 1.67 is a lower bound on the average number of bits needed per symbol

❶ Huffman (Entropy) Data Encoding

- Construct a binary encoding for each symbol, and then substitute this encoding for the symbol wherever it occurs in the "message"
  - How construct this "code"?
- Encodings of symbols may be of different (varying) lengths in bits!
  - Main idea: frequently-occurring symbols require few bits, infrequently-occurring symbols require more bits
  - What problems are there with variable-length codes?
- A necessary constraint on the encoding: no symbol's encoding may be a prefix of any other symbol's encoding
  - why?

Huffman Code Construction: The Forward Phase

1. Order the symbols by probability of occurrence (greatest to least)

2. Merge the two least probable symbols into a single "symbol group"
   - Compute probability of the symbol group = sum of the probabilities of the merged symbols

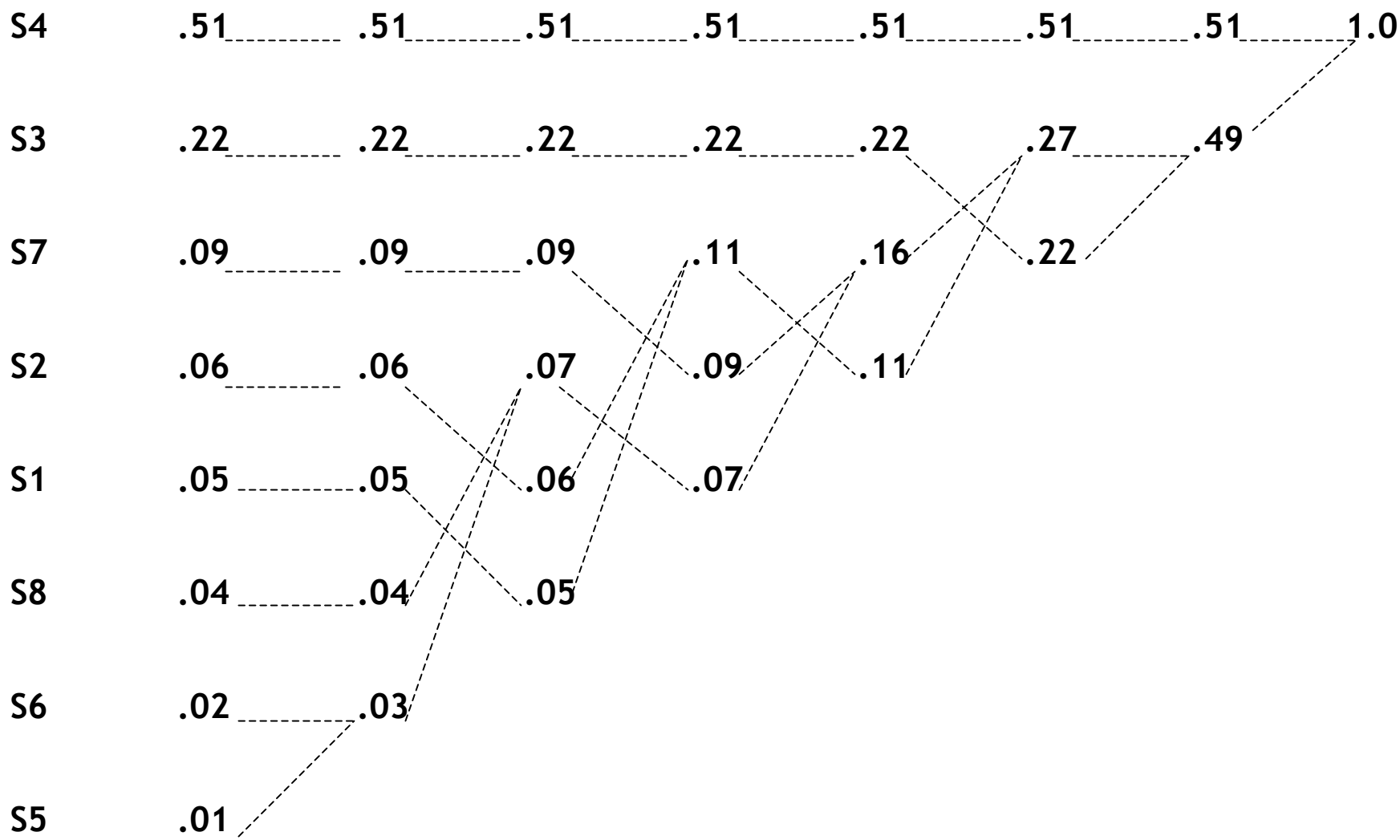
3. Continue this process (steps #1 and #2) until there is only one (grouped) symbol left
   - The probability of this grouped symbol = 1 (why?)

Example (Forward Phase of Code Construction)

Symbol	S1	S2	S3	S4	S5	S6	S7	S8
Probability	.05	.06	.22	.51	.01	.02	.09	.04
$Log_2(1/P_i)$ = desired # of bits	4.3	4.0	2.2	.97	6.6	5.6	3.5	4.6

- Goal: encode S4 with fewest bits, S5 with greatest # of bits, etc.

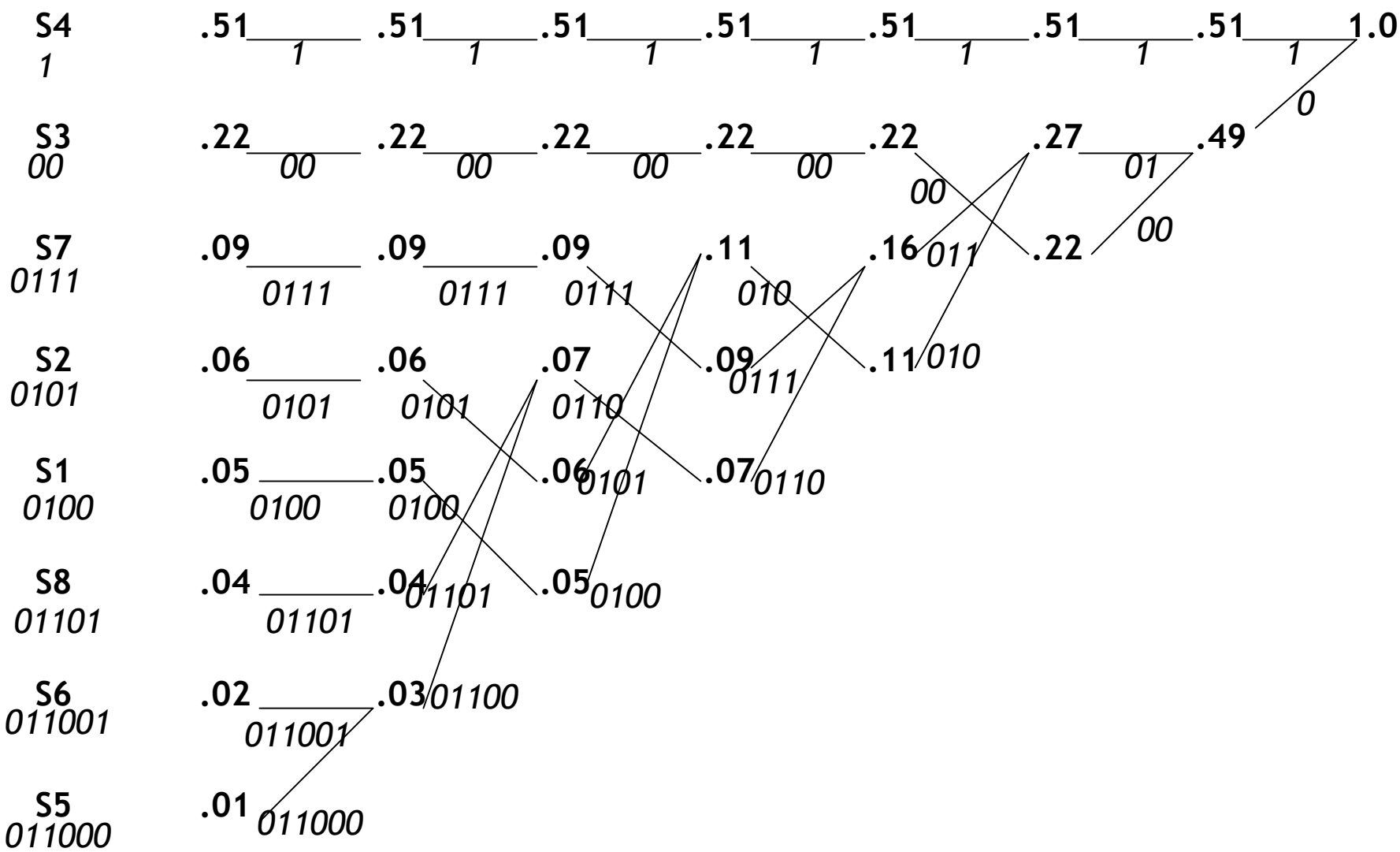
Example (Forward Phase)



Huffman Code Contruction: The Backwards Phase

- Split each symbol group with binary encoding *x* into two symbols
  - assign more probable symbol the encoding *x*1, less probable symbol the encoding *x*0
- Continue this process repeatedly until there are no symbol groups left

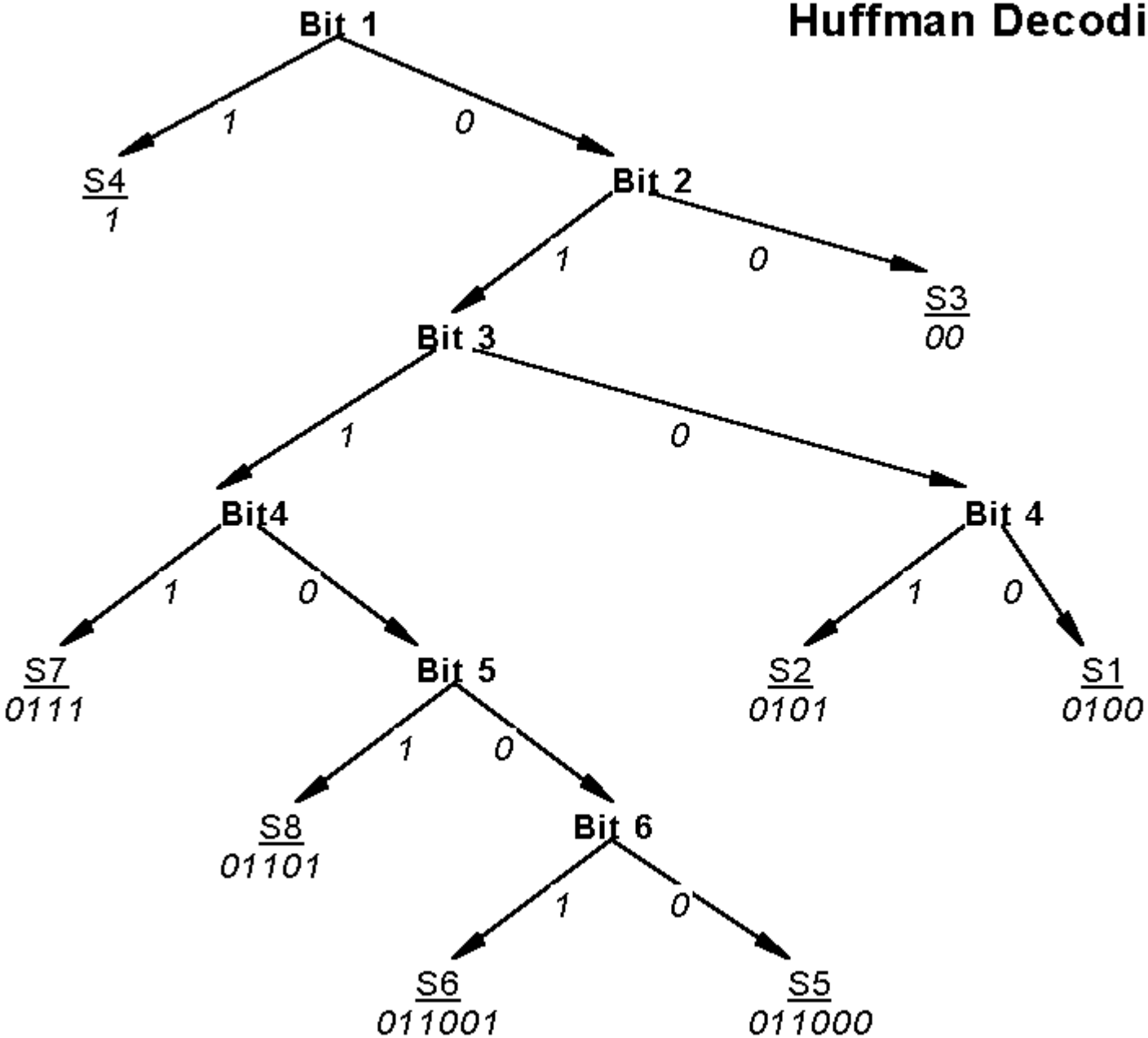
Example (Backwards Phase)



Huffman Decoding Process

- Use a "decoding tree" to decode the encoded data
- If the decoding tree is not known, it may have to be transmitted with the encoded data
- Consequences of errors or mistakes in transmission or decoding?

Huffman Decoding Tree



You try it!

- Decode the message…

10101000110011101110110010110001

1 0101 00 011001 1 1 011 1 011001 011000 1

S4 S2 S3 S6 S4 S4 S7 S4 S6 S5 S4

❷ Run-Length Encoding (RLE)

- Many messages have long "runs" of a single symbol
- Idea: encode the run *length* (the number of times a symbol is repeated) instead of the actual run
- Run length is encoded in a *count* field of fixed length *k*
  - what is the optimum value for *k*?
- What constitutes a run?
  - Some minimum number of repeated symbols

Example of RLE Scheme

- A symbol is encoded in 8 bits (e.g., ASCII)
- A run consists of 2 or more repetitions of a symbol

1. A run of the same symbol *b* times is encoded as
   - first bit = '1'
   - next 7 bits = *b-2*
     - possible range of values = $2 \ldots 2^7+1$
   - next 8 bits = the symbol itself
   - What if $b > 2^7+1$?
     - Break it into multiple runs, each with $\leq 2^7+1$ repetitions of the symbol

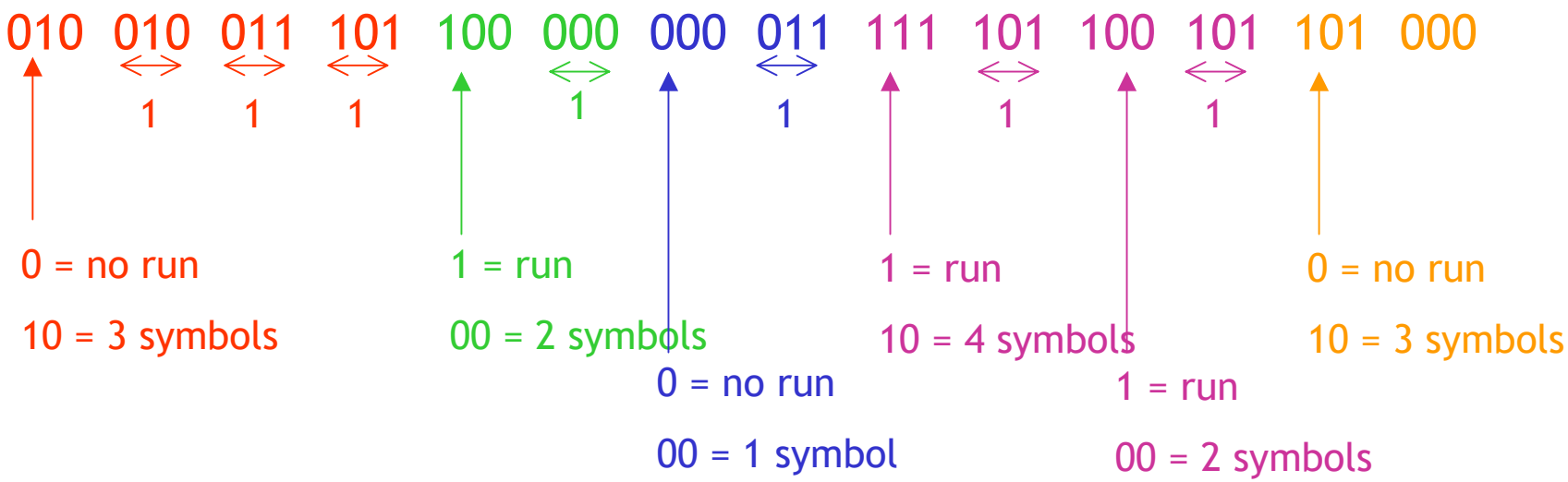
Example of RLE Scheme (cont'd)

2. A non-run of *b* different symbols is encoded as
   - first bit = '0'
   - next 7 bits = *b-1*
   - next *b**8 bits = the sequence of symbols themselves
     - possible range of values = $1 \ldots 2^7$
   - What if $b > 2^7$?
     - Break it into multiple non-runs, each with $\leq 2^7$ non-repeating symbols

Example of RLE Application

- (Let's use **3-bit** symbols for simplicity)
- Original (uncompressed) input (15*3 = 45 bits)

- Compressed output (42 bits, compression = 45:42)



Binary RLE

- Encoding of individual bits, not grouped into symbols
- A run consists of 2 or more repetitions of the bit 0

1. A run of *b* 0's is encoded as
   - *k* bits = *b*
     - possible range of values = 0…2^k-1
   - What if *b* > 2^{k-1}?
     - Break it into multiple runs, each with ≤ 2^k–1 repetitions of the symbol
   - For a run of 0's of length *b*= *n**(2^k-1) + *r*
     - transmit *n***k* 1's, followed by the value *r* encoded in *k* bits
   - Example: *b*=20, *k*=3 (encoding 0…7)
     - Run lengths = 2 runs of length 7, plus final run of length 6
     - *n*=2,*r*=6

Binary RLE (cont'd)

- 1's are not explicitly represented!
  - every run of zeros must have been terminated by a 1
  - If two consecutive 1's occur, encode the zero-length run of 0's between them
- Assume some convention for encoding the beginning and end of the data. Example:
  - data always starts with a run of zeros
  - data always ends with a run of zeros
  - how "escape" these conventions when not true?

Decoding of Binary RLE

- For k-bit count with value *b*
  - generate *b* 0's
  - followed by a 1, unless
    - count = 2^k-1 (why?), or
    - this is the last run of zeros of the data (why?)

Example (Binary RLE)

- Let *k* = 3 bits (value = 0 to 7)
- Uncompressed (input) data (28 bits) = 0000100000001000000000011001
- Encoded Data
  - run lengths (0's Only) = 4, 7, 10, 0, 2
  - Encoded (in decimal) as 4, 7+0, 7+3, 0, 2
  - Encoded as (22 bits) 100 111 000 111 011 000 010 1
- Compression = 27 : 22

Optimum Values for k?

Expected Run Length (in bits): $k / (1 - p^{2k-1})$

*k**p*	.5	.6	.7	.8	.9	.95	.98	.99
2	2.29	2.55	3.04	4.10	7.38	14.0	50.0	67.3
3			3.27	3.80	5.75	9.94	34.0	44.2
4				4.15	5.04	7.45	22.7	28.6
5					5.20	6.28	15.3	18.7
6						6.25	10.7	12.8
7						7.01	8.33	9.71
8							7.58	8.67
9							8.05	9.05

- p = probability that a bit will be a 0
- k = # of bits allocated for each "partial" run

(cont'd)

Efficiency of Encoding

p	optimum k of bits	expectd number of runs	expectd length	compression ratio
0.7	2	3.04	3.33	1.1
0.8	3	3.80	5.00	1.3
0.9	4	5.04	10.00	2.0
0.95	6	6.25	20.00	3.2
0.98	7	7.58	50.00	6.6
0.99	8	8.67	100.00	11.5

Sources of Information

- [Crane97] *A Simplified Approach to Image Processing*
  - Chapter 9
    - Introduction to image compression – read
    - Run-length encoding – read first half only
    - Huffman coding – read
    - Modified huffman coding – skip
    - Modified read – skip
    - LZW – read
    - Arithmetic coding – skip
    - JPEG – skip (for now)
    - Other state-of-the-art compression - skip
- [Gibson98] *Digital Compression for Multimedia*