
Taking A Bite Out of
Your Bytes

Why We Need Data Compression

- Lots of Data and Information Needs to be Stored
  - Reduce Storage Requirements
- Lots of Data and Information Needs to be Transmitted
  - Less Bandwidth Needed
- Can be Slow and Expensive to Store and Transmit these Amounts of Data

Compression

- Used on Data, Text, Voice & Images
- Removes Redundancy or Irrelevancy
- Uses Representation or Approximation Techniques

Why We have a Space Issue

- Block Codes or Fixed Length Codes
  - All Characters are Allocated the Same Amount of Space
    - ASCII
    - EBCDIC

Block Codes (Fixed Length)

0	0	0	0	=	A
0	0	0	1	=	B
0	0	1	0	=	C
0	0	1	1	=	D
0	1	0	0	=	E
.					.
.					.
.					.
1	1	1	1	=	P

Note: 16 Distinct Characters Require at Least 4 Bits to Code Properly

Logical Compression

- Results from Alphanumeric, Numeric, or Binary Representation of Data in Shorthand Notation
- Results from Elimination of Redundant Fields in Database
- Example

Day	Month	Year
Second	April	2008
2	4	8

Null Suppression

- Data Stream Scanned for Repeated Blanks or Nulls
  - Two characters are used to replace string of Nulls
    - First Shows existence, the second shows the length (> 2)
    - Example:

ABCƀƀƀƀƀƀCDDEE

Gets Encoded ABC#6CDDEE

Run Length Coding

- Used when Character Repeating is Greater than 3 and Occurs Often
- Utilizes Special Character, say $, Indicating Compression Follows, Followed by Repeating Data and then the Character Count
- Example: ABBBBBBCCDDDDDDDD

  Becomes A$B6CC$D8

Half-Byte Packing

- Takes Advantage of Structure of Certain Characters in Data Set
- Compression occurs when portion of Bit Representation Repeats
- Example: EBCDIC Numerals-Prefix = 1111

11110000 = 0	11110100 = 4	11110111 = 7
11110001 = 1	11110101 = 5	11111000 = 8
11110010 = 2	11110110 = 6	11111001 = 9
11110011 = 3		

Half-Byte Packing Continued

- Encoding is Preformed Using a Special Character, say $, indicating half-byte packing; followed by the repeating prefix; followed by 4 bits indicating the number < 16 packed; followed by the suffix of each character.

Example: Original data string is 3 1 2 2 4 5

Becomes

$ 1111 0110 0011 0001 0010 0010 0100 0101

EBCDIC Prefix

Repeats 6 Times

Statistical Data Compression

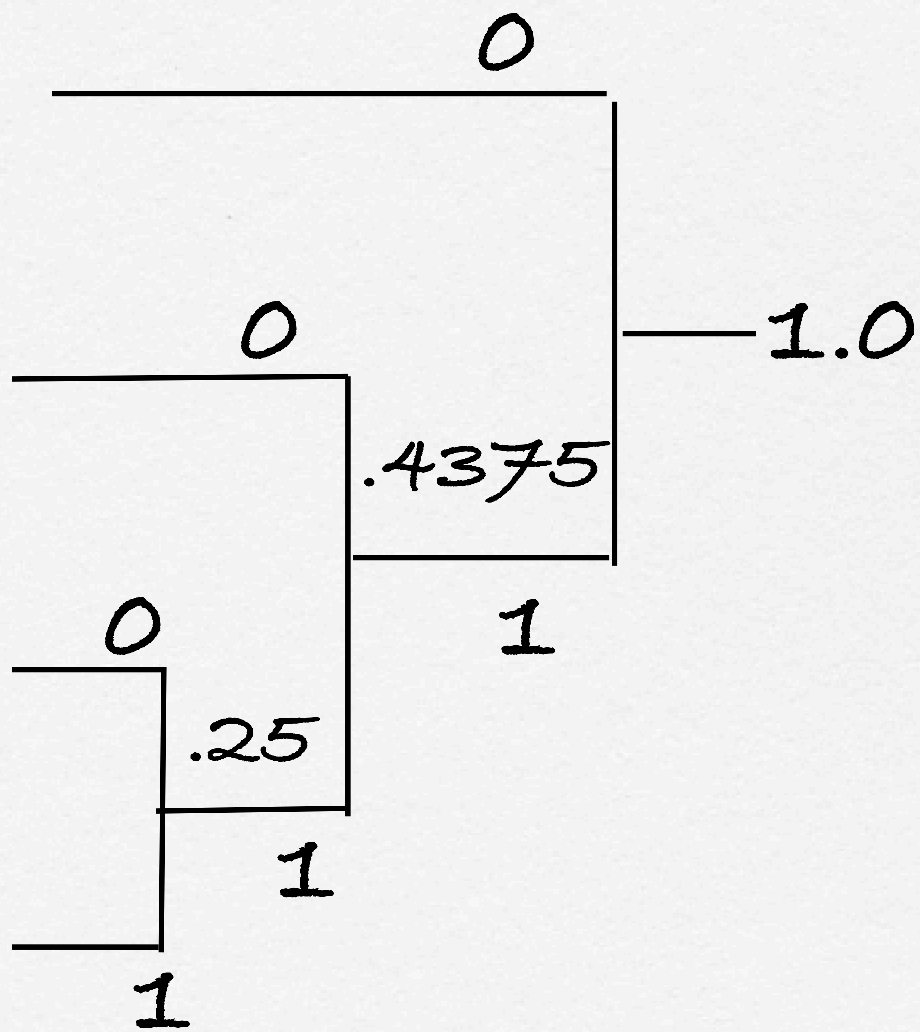
- Each Symbol is assigned a code based on the probability of its occurrence.
  - More probable symbols get shorter codes
  - For Example, in the English Language the letter 'E' is the most widely used letter. Hence it should get a shorter code than, say, 'Z'

Huffman Coding

- Unlike ASCII or EBCDIC, which are block codes, Huffman Coding is a Variable Length Coding Scheme
- Results in the shortest average code length
- Obeys a prefix property - No short code group is duplicated as the beginning of a longer code group
  - This implies that if one character is represented by 100 then 1001 or 1000 cannot be used as another code.

Implementing The Huffman Code

Character	Probability	Code
E	.5625	0
I	.1875	10
A	.1875	110
O	.0625	111



Average Number of Bits Per Symbol

- Lets Compute the Average Number of Bits per symbol:

1*0.5625 + 2*0.1875 + 3*0.1875 + 3*0.0.625 = 1.63 Average Bits per symbol

For a Block Code we would have needed at Least 2 bits

Hence we Have Data Compression

Number of Bits Required

The Number of Bits Required to encode a letter using the Huffman Coding Scheme is Determined by the following formula:

$$b=f(-\log_2 P)$$

where P = The probability of occurrence of the letter

f(x)=The Integer Ceiling function

The Probability of E is 0.5625 and

$$-\log_2 (.5625) = 0.83$$

$$f(0.83) = 1$$

An Example

0/10/0/10/111 E I E I O

Notice, it ONLY took 9 Bits to Code

This String

A Block Code Would have taken a minimum

of 10 (Ten)

So we did save (OK not much)

However, statistically this string is rare

The 'E' would usually be used more often