# David A. Huffman



David Huffman is best known for the invention of Huffman code, a highly important compression scheme for lossless variable length encoding. It was the result of a term paper he wrote while a graduate student at the Massachusetts Institute of Technology (MIT)…



From: Wikipedia

# Huffman coding algorithm



1. Take the two least probable symbols in the alphabet
   (longest code words, equal length, differing in last digit)
2. Combine these two symbols into a single symbol
3. Repeat

# Example: Huffman coding



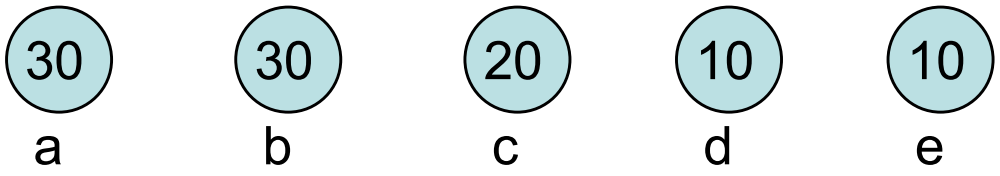
| S | Freq |
|---|---|
| a | 30 |
| b | 30 |
| c | 20 |
| d | 10 |
| e | 10 |

# Example



| S | Freq | Hu |
|---|---|---|
| a | 30 | |
| b | 30 | |
| c | 20 | |
| d | 10 | |
| e | 10 | |



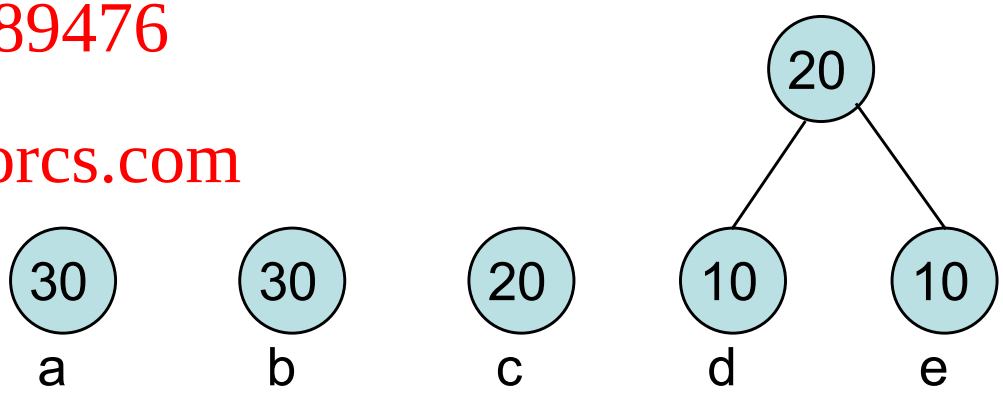


# Example



| S | Freq | Hu |
|---|---|---|
| a | 30 | |
| b | 30 | |
| c | 20 | |
| d | 10 | |
| e | 10 | |



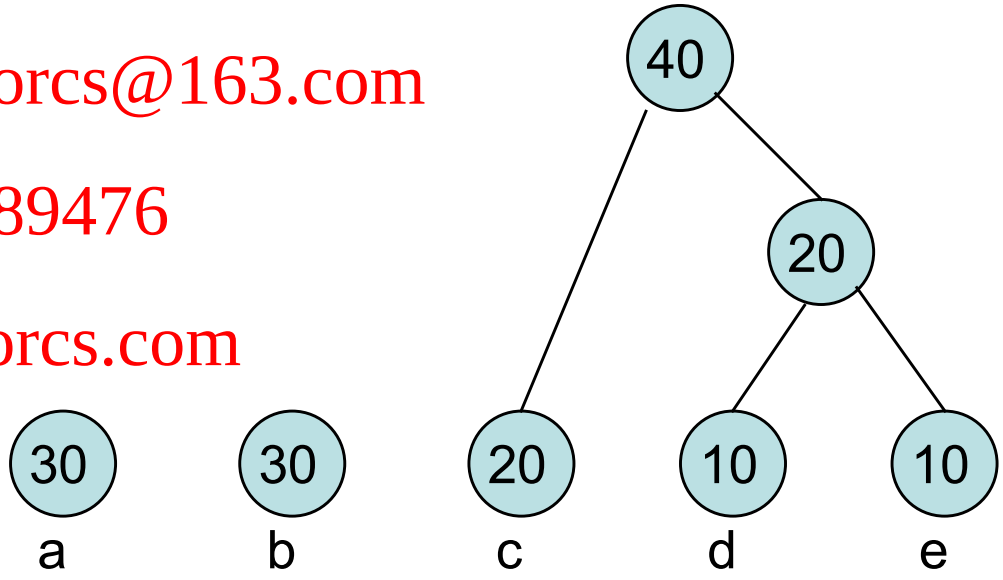


# Example



| S | Freq | Hu |
|---|---|---|
| a | 30 | |
| b | 30 | |
| c | 20 | |
| d | 10 | |
| e | 10 | |



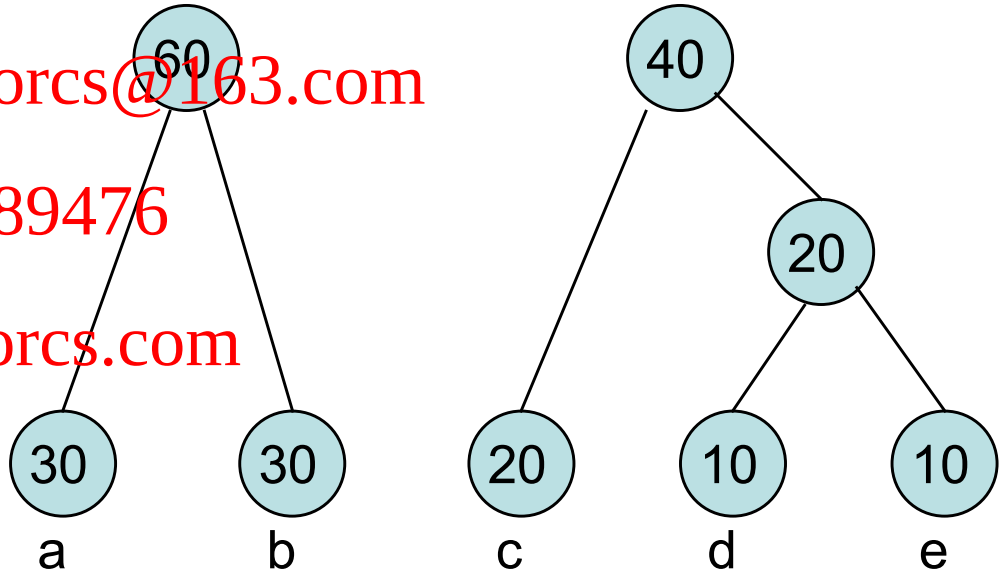


# Example



| S | Freq | Hu |
|---|---|---|
| a | 30 | |
| b | 30 | |
| c | 20 | |
| d | 10 | |
| e | 10 | |



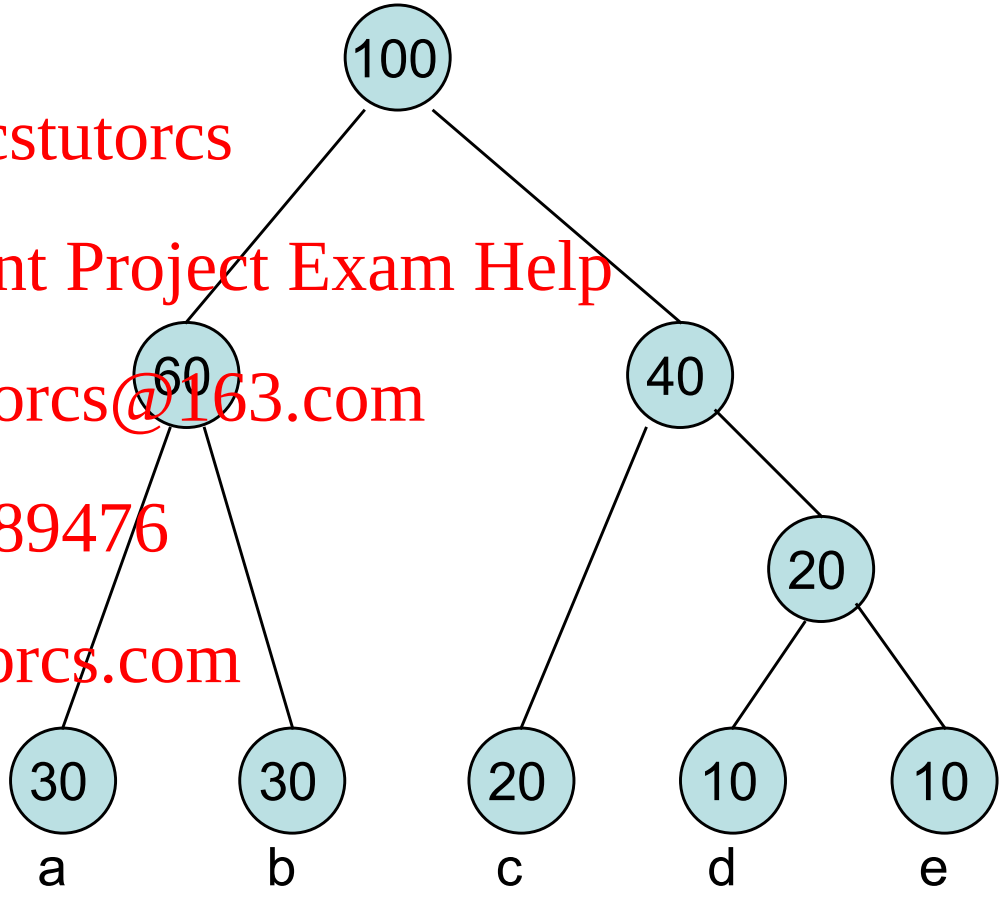


# Example



| S | Freq | Hu |
|---|---|---|
| a | 30 | |
| b | 30 | |
| c | 20 | |
| d | 10 | |
| e | 10 | |



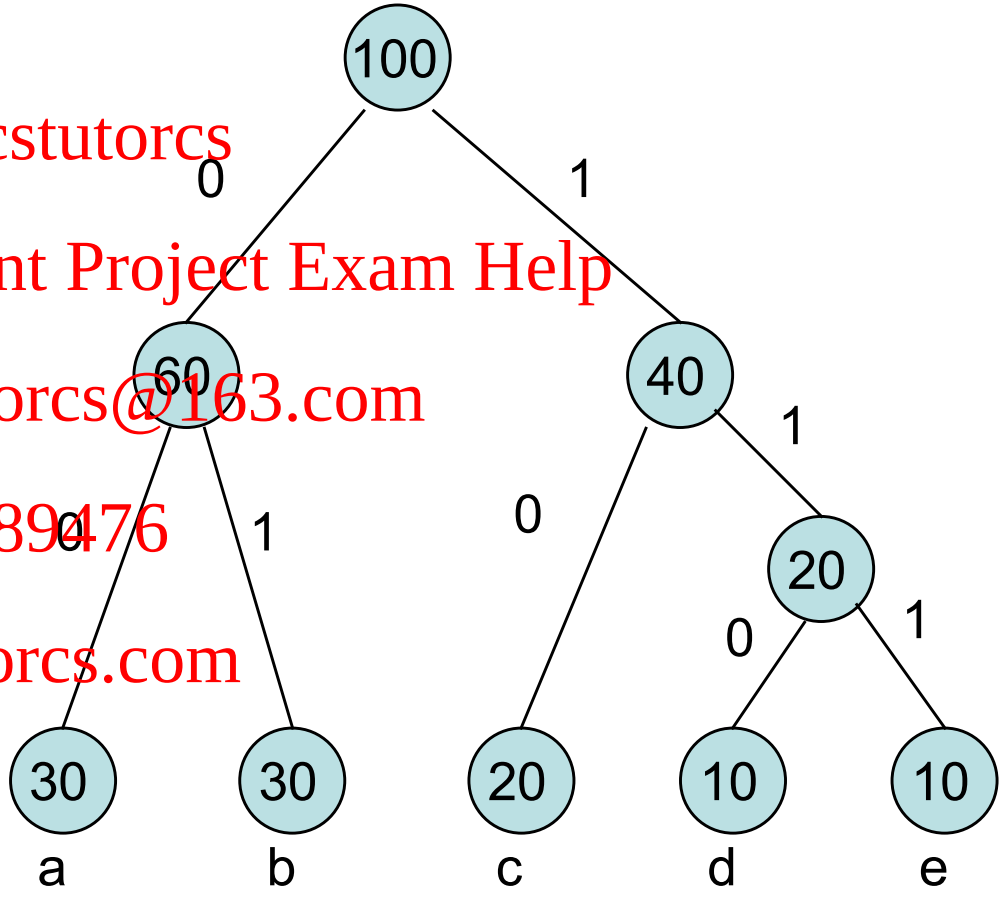


# Example



| S | Freq | Huff |
|---|---|---|
| a | 30 | 00 |
| b | 30 | 01 |
| c | 20 | 10 |
| d | 10 | 110 |
| e | 10 | 111 |





# Average length L



= ( 30*2 + 30*3 + 10*2 + 10*3 + 10*3 ) / 100

= 220 / 100

= 2.2

# Average length L



= ( 30*2 + 30*2 + 10*3 + 10*3 ) / 100

= 220 / 100

= 2.2



Better than using fixed length 3 bits for 5 symbols.

# Entropy



H = -0.3 * log 0.3 + -0.3 * log 0.3 + -0.2 * log 0.2 + -0.1 * log 0.1 + -0.1 * log 0.1

= -0.3*(-1.737) + -0.3*(-1.737) + -0.2 * (-2.322) + -0.1 * (-3.322) + -0.1 * (-3.322)



= 0.3 log 10/3 + 0.3 log 10/3 + 0.2 log 5 + 0.1 log 10 + 0.1 log 10

= 0.3*1.737 + 0.3*1.737 + 0.2* 2.322 + 0.1*3.322 + 0.1*3.322

= 2.17

# Another example



- S={a, b, c, d} with {4, 2, 1, 1}
- $H = 4/8*\log_2 2 + 2/8*\log_2 4 + 1/8*\log_2 8 + 1/8*\log_2 8$



- H = 1/2 + 1/2 + 3/8 + 3/8 = 1.75





- a => 0 b => 10 c => 110 d => 111
- Message: {abcdabaa} => {0 10 110 111 0 10 0 0}





- Average length L = 14 bits / 8 chars = 1.75
- If equal probability, i.e. fixed length, need $\log_2 4 = 2$ bits

# Huffman coding



| S | Freq | Huffman |
|---|---|---|
| a | 30 | |
| b | 30 | |
| c | 20 | |
| d | 10 | |
| e | 10 | |

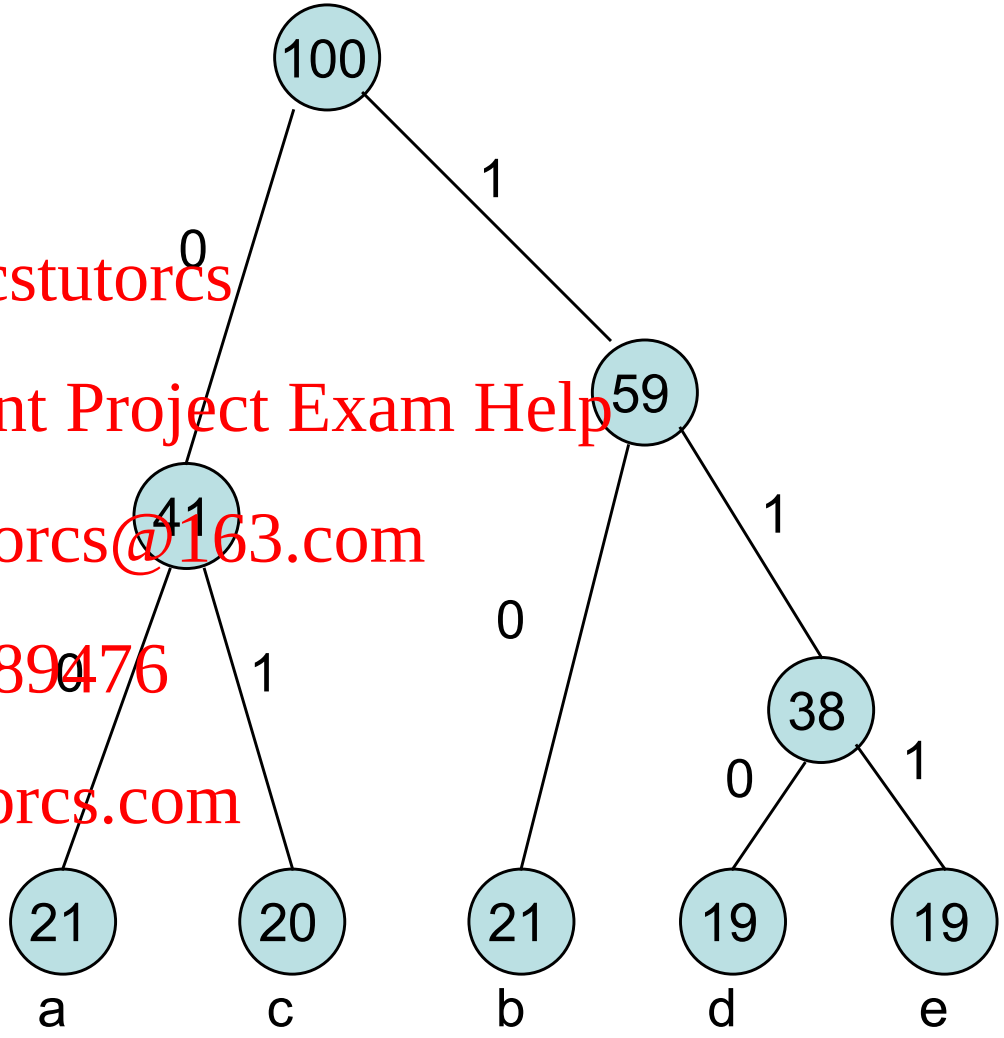
Total: 100

# Huffman coding



| S | Freq | Huff |
|---|---|---|
| a | 21 | 00 |
| b | 21 | 10 |
| c | 20 | 01 |
| d | 19 | 110 |
| e | 19 | 111 |





# Huffman optimal?



H = 0.21 log 100/21 + 0.21 log 100/21 + 0.2 log 5 + 0.19 log [illegible] + 0.19 log 100/19

= 0.21*2.252 + 0.21*2.252 + 0.2* 2.322 + 0.19*2.396 + 0.19*2.396

= 2.32

L = (21*2 + 21*2 + 20*2 + 19*3 + 19*3)/100

= 2.38

# Huffman coding



| S | Freq | Huffman |
|---|---|---|
| a | 30 | 100000 |
| b | 30 | 0 |
| c | 20 | 2 |
| d | 10 | 1 |
| e | 10 | 1 |

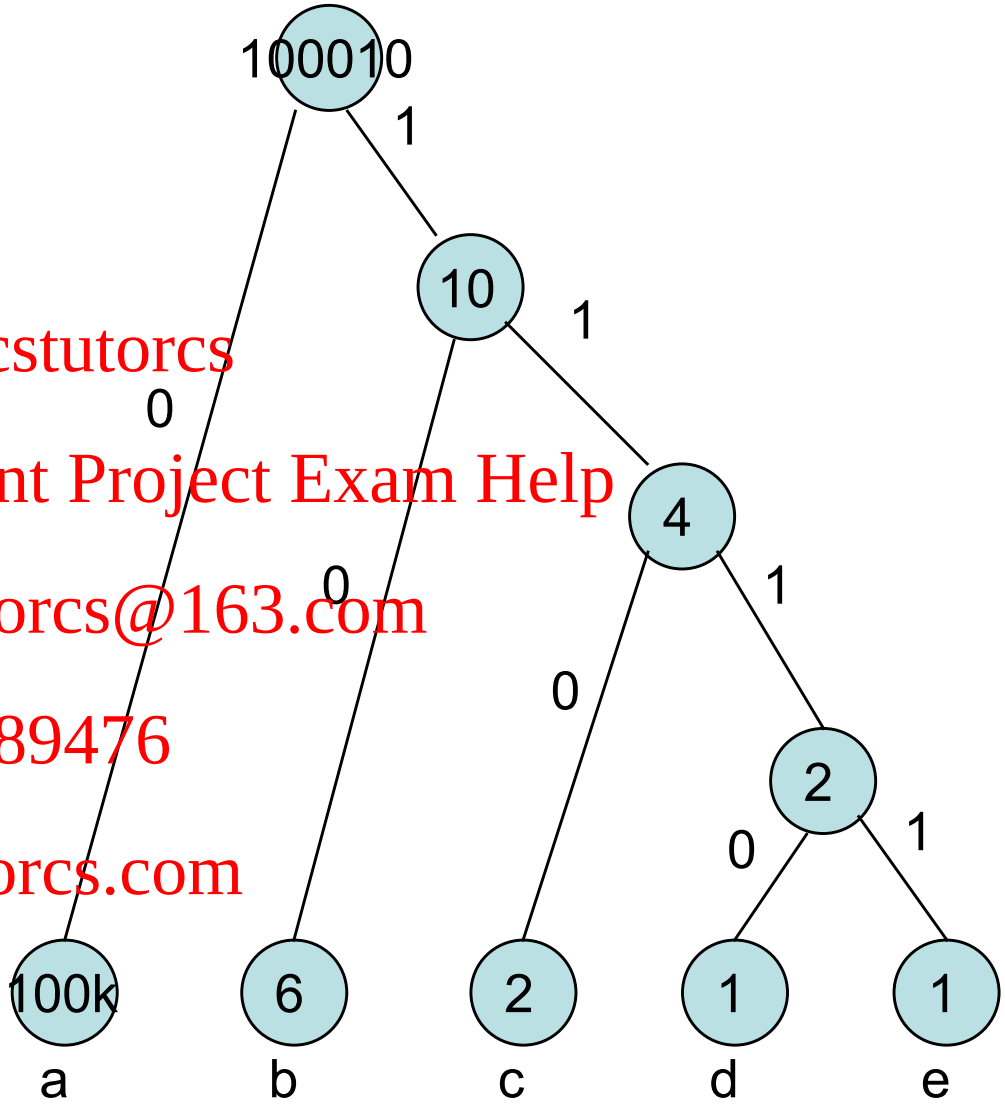
Total: 100010

# Huffman coding



| S | Freq | Huf |
|---|---|---|
| a | 100000 | 0 |
| b | 6 | 10 |
| c | 2 | 110 |
| d | 1 | 1110 |
| e | 1 | 1111 |





# Huffman optimal?



H = 0.9999 lo[illegible]1 + 0.00006 log 16668.333
+ … + 1/10( g 100010
≈ 0.00



L = (100000*1 + …)/100010
≈ 1

# Problems of Huffman coding



- Huffman codes have an integral # of bits.
  - E.g., log (3) = 1.5 while Huffman may need 2 bits
- Noticeable non-optimality when prob of a symbol is high.

=> Arithmetic coding

# Arithmetic coding



Message to encode:

BILL GATES

| 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 |



Example extracted from February, 1991 issue of Dr. Dobb's Journal

# Arithmetic coding



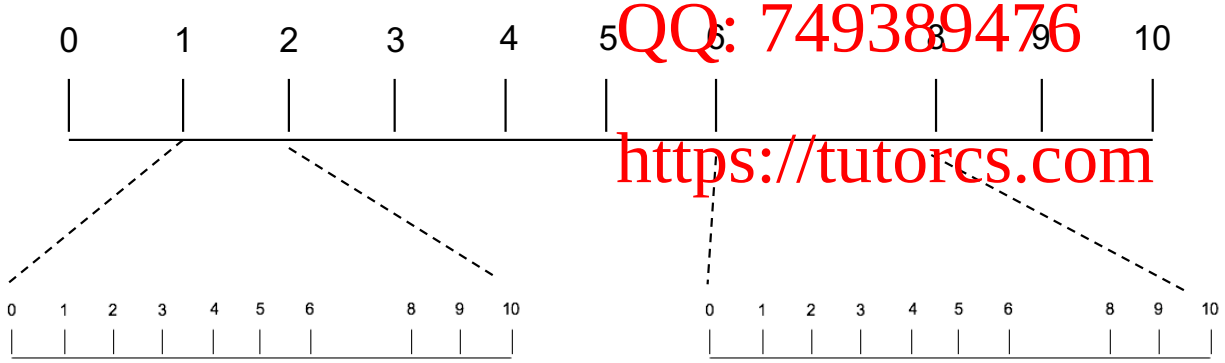
| Character | Probability | Range |
|---|---|---|
| SPACE | 1/10 | 0.00 - 0.10 |
| A | 1/10 | 0.10 - 0.20 |
| B | 1/10 | 0.20 - 0.30 |
| E | 1/10 | 0.30 - 0.40 |
| G | 1/10 | 0.40 - 0.50 |
| I | 1/10 | 0.50 - 0.60 |
| L | 2/10 | 0.60 - 0.80 |
| S | 1/10 | 0.80 - 0.90 |
| T | 1/10 | 0.90 - 1.00 |

# Arithmetic coding



| Character | Probability | |
|---|---|---|
| SPACE | 1/10 | 0 |
| A | 1/10 | 0 |
| 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 algorithm



Set low to 0.0
Set high to 1.0
While there are still input symbols do
    get an input symbol
    code_range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
End of While
output low or a number within the range

# Arithmetic coding



| New Charact | w value | High Value |
|---|---|---|
| | 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 |

# Example



Consider the s  L as new char:

code_range = 0.258 – 0.256 = 0.002

high = 0.256 + 0.002*0.8 = 0.2576

low = 0.256 + 0.002*0.6 = 0.2572

# Decoding algorithm



get encoded number

Do

find symbol whose range straddles the encoded number

output the symbol

range = symbol high value - symbol low value

subtract symbol low value from encoded number

divide encoded number by range

until no more symbols

# Arithmetic coding



| Encoded Number | Symbol | Low | High | Range |
|---|---|---|---|---|
| 0.2572167752 | | 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 | I | 0.6 | 0.8 | 0.2 |
| 0.041938 | SPACE | 0.0 | 0.1 | 0.1 |
| 0.41938 | U | 0.4 | 0.5 | 0.1 |
| 0.1938 | A | 0.2 | 0.3 | 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 | | | | |

# Example



At the first L, encoded number is 0.72167752.

output the first L



range = 0.8 – 0.6 = 0.2



encoded number = (0.72167752 – 0.6) / 0.2

= 0.6083876

# Advantage of arithmetic coding



Assume: A 90% END 10%

To encode: AAAAAAA

| New Character | Low value | High Value |
|---|---|---|
| | 0.0 | 1.0 |
| A | 0.0 | 0.9 |
| A | 0.0 | 0.81 |
| A | 0.0 | 0.729 |
| A | 0.0 | 0.6561 |
| A | 0.0 | 0.59049 |
| A | 0.0 | 0.531441 |
| A | 0.0 | 0.4782969 |
| END | 0.43046721 | 0.4782969 |

# Advantage of arithmetic coding



Assume: A 90% END 10

To encode: AAAAAAA

| New Character | Low value | High Value |
|---|---|---|
| | 0.0 | 1.0 |
| A | 0.0 | 0.9 |
| A | 0.0 | 0.81 |
| A | 0.0 | 0.729 |
| A | 0.0 | 0.6561 |
| A | 0.0 | 0.59049 |
| A | 0.0 | 0.531441 |
| A | 0.0 | 0.4782969 |
| END | 0.43046721 | 0.4782969 |



e.g., 0.45

# Patents on AC



- Bzip2 and JPEG use Huffman as AC protected by patents
- PackJPG using AC shows 25% of size saving

# Some AC patents (expiring)



U.S. Patent 4,122,440 — (IB[illegible]arch 77, Granted 24 October 78 (Now expired)

U.S. Patent 4,286,256 — (IB[illegible]25 August 81 (Now expired)

U.S. Patent 4,467,317 — (IB[illegible]21 August 84 (Now expired)

U.S. Patent 4,652,856 — (IBM) Granted 4 February 86 (Now expired)

U.S. Patent 4,891,643 — (IBM) Filed 18 September 86, granted 2 January 90 (Now expired)

U.S. Patent 4,905,297 — (IBM) Filed 18 November 88, granted 27 February 90 (Now expired)

U.S. Patent 4,933,883 — (IBM) Filed 3 May 88, granted 12 June 90 (Now expired)

U.S. Patent 4,935,882 — (IBM) Filed 20 July 88, granted 19 June 90 (Now expired)

U.S. Patent 4,989,000 — Filed 19 June 89, granted 29 January 91 (Now expired)

U.S. Patent 5,099,440 — (IBM) Filed 5 January 90, granted 24 March 92 (Now expired)

U.S. Patent 5,272,478 — (Ricoh) Filed 17 August 92, granted 21 December 93 (Now expired)