Text Compression: Examples

Encodings

ASCII: 8 bits/character

Unicode: 16 bits/character

Symbol	ASCII	Fixed length	Var. length
a	01100001	000	000
b	01100010	001	11
c	01100011	010	01
d	01100100	011	001
e	01100101	100	10

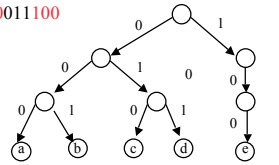
"abcde" in the different formats

ASCII:

0110000101100010011000110110010 0…

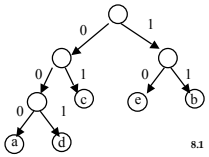
Fixed:

000001010011100



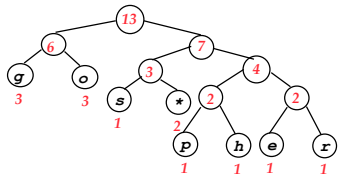
Var:

000110100110



Huffman coding: *go go gophers*

	ASCII		3 bits	Huffman
g	103	*1100111*	000	00
o	111	*1101111*	001	01
p	112	*1110000*	010	1100
h	104	*1101000*	011	1101
e	101	*1100101*	100	1110
r	114	*1110010*	101	1111
s	115	*1110011*	110	101
sp.	32	*1000000*	111	101



- Encoding uses tree:
  - 0 left/1 right
  - How many bits? 37!!
  - Savings? Worth it?

Huffman Coding

- D.A Huffman in early 1950's
- Before compressing data, analyze the input stream
- Represent data using variable length codes
- Variable length codes though *Prefix* codes
  - Each letter is assigned a codeword
  - Codeword is for a given letter is produced by traversing the Huffman tree
  - Property: *No codeword produced is the prefix of another*
  - Letters appearing frequently have short codewords, while those that appear rarely have longer ones
- Huffman coding is optimal *per-character* coding method

Building a Huffman tree

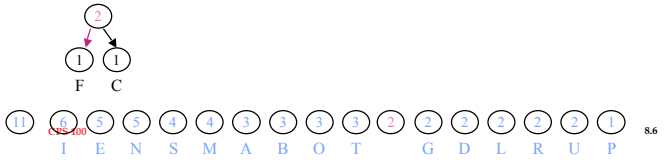
- Begin with a forest of single-node trees (leaves)
  - Each node/tree/leaf is weighted with character count
  - Node stores two values: character and count
  - There are *n* nodes in forest, *n* is size of alphabet?
- Repeat until there is only one node left: root of tree
  - Remove two minimally weighted trees from forest
  - Create new tree with minimal trees as children,
    - New tree root's weight: sum of children (character ignored)
- Does this process terminate? How do we get minimal trees?
  - Remove minimal trees, hummm……

Building a tree

"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"

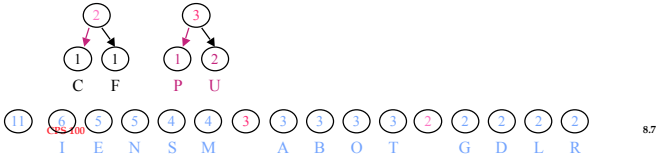
Building a tree

"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"



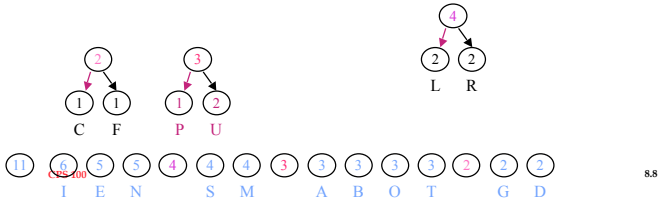
Building a tree

"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"



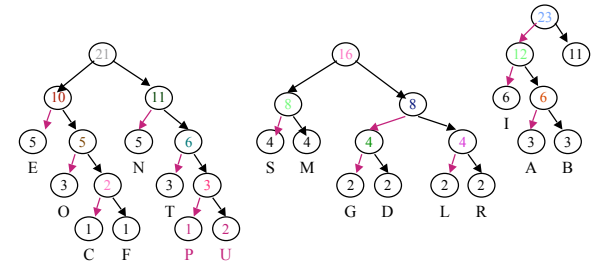
Building a tree

"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"



Building a tree

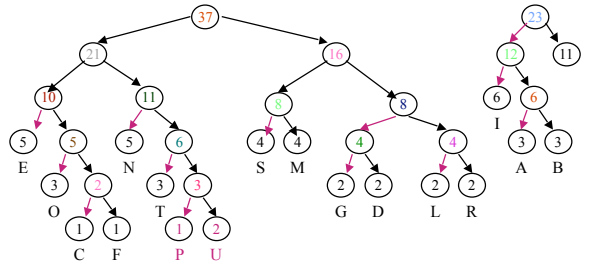
"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"



23 21 16

Building a tree

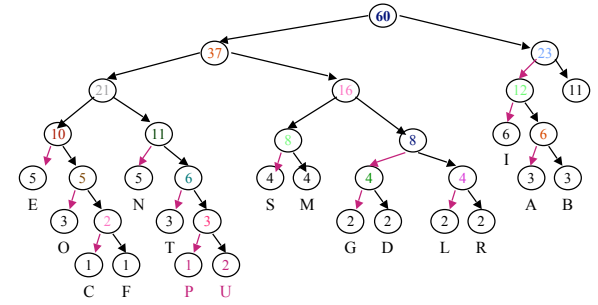
"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"



37 23

Building a tree

"A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"



60

Mary Shaw

- **Software engineering and software architecture**
  - **Tools for constructing large software systems**
  - **Development is a small piece of total cost, maintenance is larger, depends on well-designed and developed techniques**
- **Interested in computer science, programming, curricula, and canoeing, health-care costs**
- **ACM Fellow, Alan Perlis Professor of Compsci at CMU**

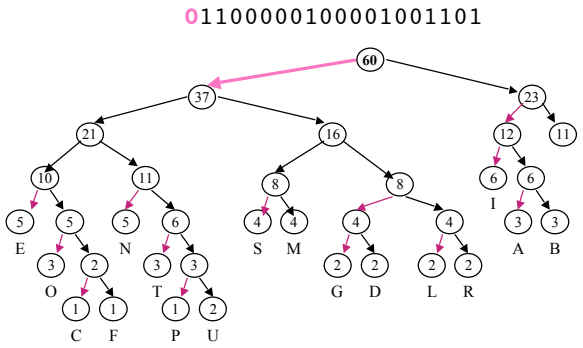
Huffman Complexities

- How do we measure? Size of input file, size of alphabet
  - Which is typically bigger?
- Accumulating character counts: _____
  - How can we do this in O(1) time, though not really
- Building the heap/priority queue from counts ____
  - Initializing heap guaranteed
- Building Huffman tree ____
  - Why?
- Create table of encodings from tree ____
  - Why?
- Write tree and compressed file _____

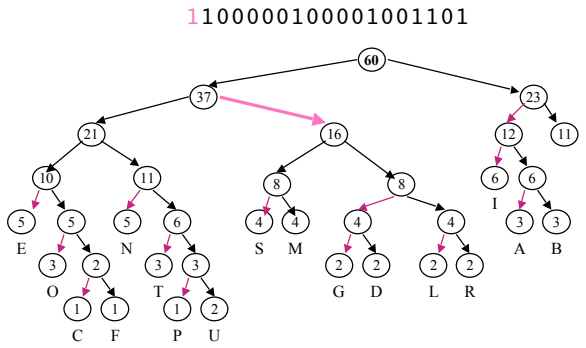
Writing code out to file

- How do we go from characters to encodings?
  - Build Huffman tree
  - Root-to-leaf path generates encoding
- Need way of writing bits out to file
  - Platform dependent?
  - Complicated to write bits and read in same ordering
- See BitInputStream and BitOutputStream classes
  - Depend on each other, bit ordering preserved
- How do we know bits come from compressed file?
  - Store a *magic* number

Decoding a message

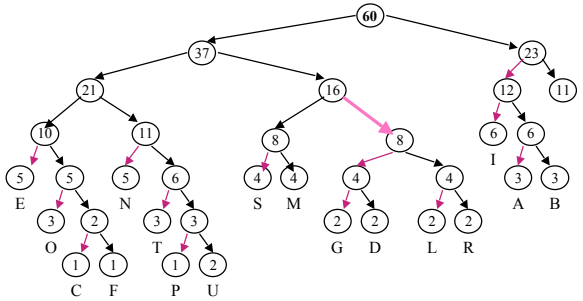


Decoding a message



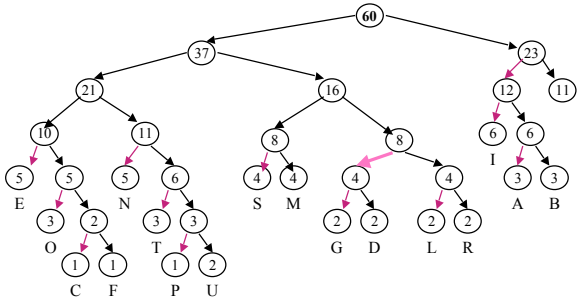
Decoding a message

1000001000010011101



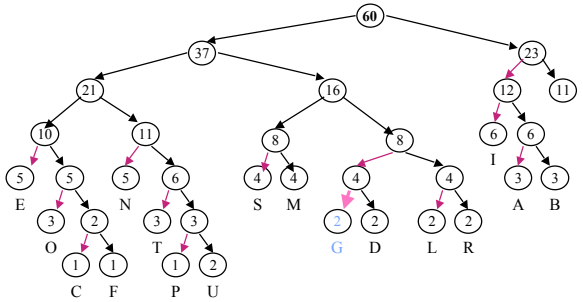
Decoding a message

0000100001001101



Decoding a message

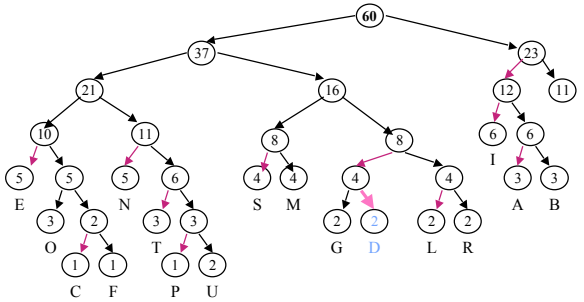
0000100001001101



G

Decoding a message

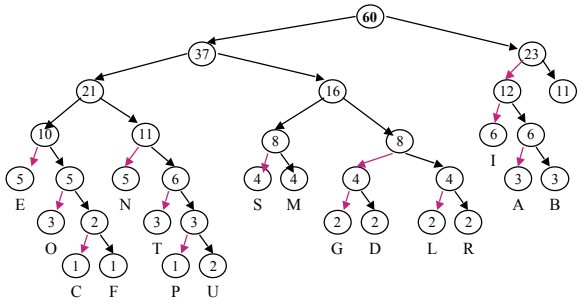
1



GOOD

Decoding a message

01100000100001001101

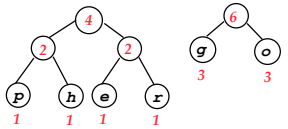


GOOD

Huffman coding: *go go gophers*

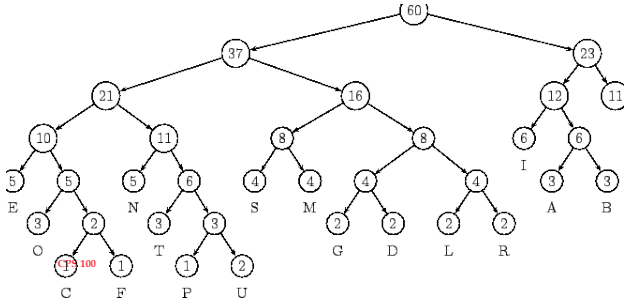
	ASCII	3 bits	Huffman
g	103 *1100111*	000	??
o	111 *1101111*	001	??
p	112 *1110000*	010	
h	104 *1101000*	011	
e	101 *1100101*	100	
r	114 *1110010*	101	
s	115 *1110011*	110	
sp.	32 *1000000*	111	

- choose two smallest weights
  - combine nodes + weights
  - Repeat
  - Priority queue?
- Encoding uses tree:
  - 0 left/1 right
  - How many bits?



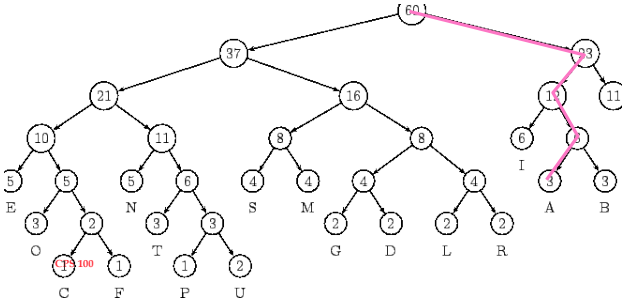
Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "10101101001000101001110011100000"



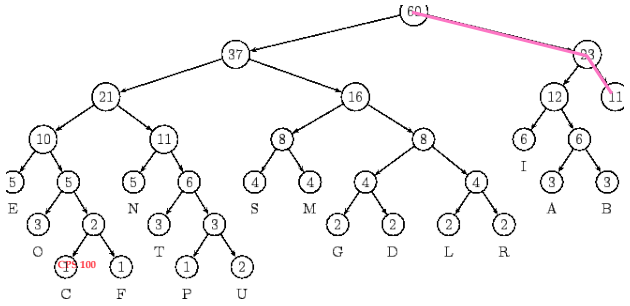
Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "10101101001000101001110011100000"



Huffman Tree 2

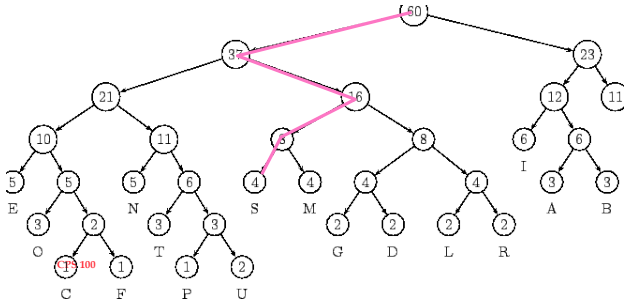
- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "1010110100100010100111001110000"



8.25

Huffman Tree 2

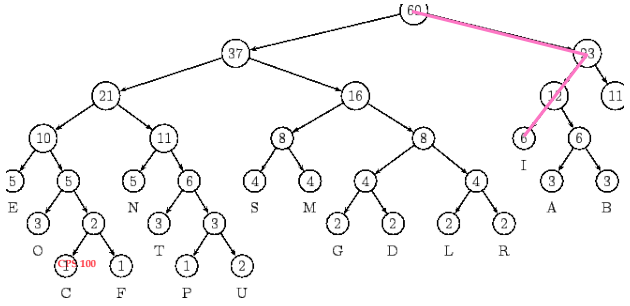
- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "1010110100100010100111001110000"



8.26

Huffman Tree 2

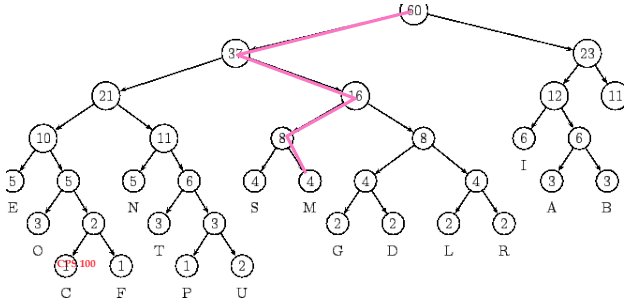
- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "1010110100100010100111001110000"



8.27

Huffman Tree 2

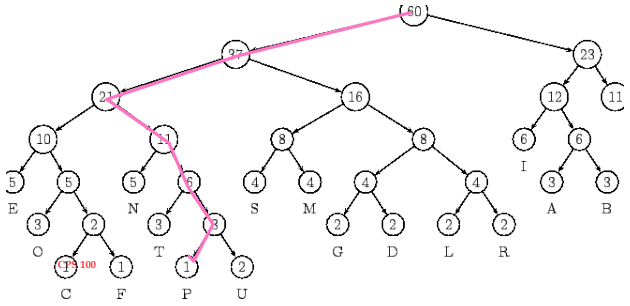
- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "1010110100100010100111001110000"



8.28

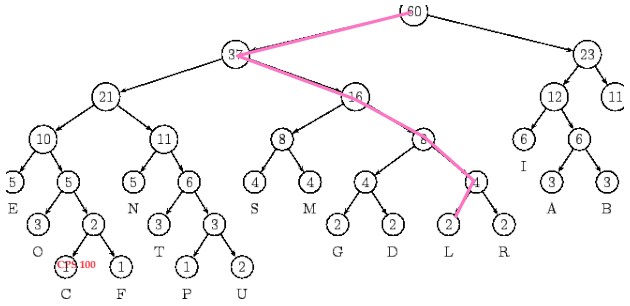
Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "10101101001000101001110011100000"



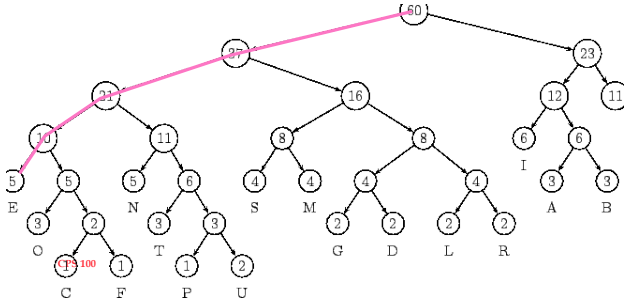
Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "10101101001000101001110011100000"



Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. " A SIMPLE" ⇔ "10101101001000101001110011100000"



Other methods

- Adaptive Huffman coding
- Lempel-Ziv algorithms
  - Build the coding table on the fly while reading document
  - Coding table changes dynamically
  - Protocol between encoder and decoder so that everyone is always using the right coding scheme
  - Works well in practice (`compress`, `gzip`, etc.)
- More complicated methods
  - Burrows-Wheeler (`bunzip2`)
  - PPM statistical methods