Text Compression: Examples

"abcde" in the different formats

ASCII:
01100001 01100010 01100011 01100100 ...

Fixed:
000001 01100011

Var.
000110100110

Huffman coding: go go gophers

ASCII: 3 bits

<table>
<thead>
<tr>
<th>Symbol</th>
<th>ASCII</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>103</td>
<td>000</td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>001</td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>010</td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>011</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>100</td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>101</td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>110</td>
</tr>
<tr>
<td>sp.</td>
<td>32</td>
<td>111</td>
</tr>
</tbody>
</table>

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 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”
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

Encoded message: 01100000100001001101

Decoding process:

```
0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1
```

Decoding a message

Encoded message: 1100000100001001101

Decoding process:

```
1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1
```
Decoding a message

100000100001001101

Decoding a message

00000100001001101

Decoding a message

0000100001001101

Decoding a message

1
Decoding a message

Huffman coding:

- **ASCII:** 3 bits
- **Huffman:**
  - g: 103 → 000
  - o: 111 → 001
  - p: 112 → 010
  - h: 104 → 011
  - e: 101 → 100
  - r: 114 → 101
  - s: 115 → 110
  - sp.: 32 → 111

- **“go gophers”**
  - 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" ⇔ "1010101001001010010010100100101001001110011100000"

Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  > E.g. "A SIMPLE" ⇔ "101011010100010100100111100110000000"

Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  > E.g. "A SIMPLE" ⇔ "10101101010001010011100111000000"

Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  > E.g. "A SIMPLE" ⇔ "10101110100100100111001110000000"
Huffman Tree 2
- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. "A SIM" ⇔ "101101001001011001100000"

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