Text Compression: Examples

Encodings
ASCII: 8 bits/character
Unicode: 16 bits/character

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>ASCII</th>
<th>Fixed length</th>
<th>Var. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>01100001</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>01100010</td>
<td>001</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>01100011</td>
<td>010</td>
<td>01</td>
</tr>
<tr>
<td>d</td>
<td>01100100</td>
<td>011</td>
<td>001</td>
</tr>
<tr>
<td>e</td>
<td>01100101</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>
```

“abcde” in the different formats

ASCII:
0110000101100100110001101100100...

Fixed:
000001010011100

Var:
000110100110

Encodings
ASCII: 8 bits/character
Unicode: 16 bits/character
Huffman coding: *go go gophers*

<table>
<thead>
<tr>
<th>ASCII</th>
<th>3 bits</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>g 103</td>
<td>1100111</td>
<td>000 00</td>
</tr>
<tr>
<td>o 111</td>
<td>1101111</td>
<td>001 01</td>
</tr>
<tr>
<td>p 112</td>
<td>1110000</td>
<td>010 1100</td>
</tr>
<tr>
<td>h 104</td>
<td>1101000</td>
<td>011 1101</td>
</tr>
<tr>
<td>e 101</td>
<td>1100101</td>
<td>100 1110</td>
</tr>
<tr>
<td>r 114</td>
<td>1110010</td>
<td>101 1111</td>
</tr>
<tr>
<td>s 115</td>
<td>1110011</td>
<td>110 101</td>
</tr>
<tr>
<td>sp. 32</td>
<td>1000000</td>
<td>111 101</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 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”
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”
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

01100000100001001101

CPS 100
Decoding a message

1100000100001001101

C P U
2 1
E N F
2 3
O T P
1 1 2
G D L R
2 2 2 2
A B I
3 3 6 6
23 12 11 6 6
Decoding a message

100000100001001101

CPS 100
Decoding a message

0000100001001101

CPS 100
Decoding a message

0000100001001101

CPS 100
Decoding a message

GOOD
Decoding a message

GOOD
Huffman coding: *go go gophers*

<table>
<thead>
<tr>
<th>ASCII</th>
<th>3 bits</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>103</td>
<td>1100111</td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>1101111</td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>1110000</td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>1101000</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>1100101</td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>1110010</td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>1110011</td>
</tr>
<tr>
<td>sp.</td>
<td>32</td>
<td>1000000</td>
</tr>
</tbody>
</table>

- **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” ⇔ “10101101001000101001110011100000”
Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  
  - E.g. "A SIMPLE" ⇔ “1010110100100010100111001100000"
Huffman Tree 2

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

```
A SIMPLE
1010110100
```

```
1000101001110011100000
```

```
10101101001000101001110011100000
```
**Huffman Tree 2**

- “A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS”
  - E.g. “A SIMPLE” ⇔ “1010110100100010100111001110000”
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” ⇔ “101011010010001010011100111000000”
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