Text Compression

**Input**
- Input: String $S$
- Output: String $S'$
  - Shorter
  - $S$ can be reconstructed from $S'$
# 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: 01100001011000101100110001101100100…
Fixed: 0000010100111100

Var: 00011010011110
```

![Binary tree representation of character codes](image)

---

CompSci 100
Huffman coding: *go go gophers*

ASCII(7bits) 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 an *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, hmmm......
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”
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”
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”
Encoding

1. Count occurrence of all occurring characters $O(N)$
2. Build priority queue $O(A)$
3. Build Huffman tree $O(A \log A)$
4. Create Table of codes from tree $O(A \log A)$
5. Write Huffman tree and coded data to file $O(N)$
Properties of Huffman coding

- Want to minimize weighted path length $L(T)$ of tree $T$
  \[ L(T) = \sum_{i \in \text{Leaf}(T)} d_i w_i \]
  - $w_i$ is the weight or count of each codeword $i$
  - $d_i$ is the leaf corresponding to codeword $i$

- How do we calculate character (codeword) frequencies?

- Huffman coding creates pretty full bushy trees?
  - When would it produce a “bad” tree?

- How do we produce coded compressed data from input efficiently?
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

0110000010001001101

Diagram of a binary tree with nodes labeled from 10 to 60, and leaves labeled with letters (e.g., E, N, O, C, F, G, D, L, R).
Decoding a message

1100000100001001101

CompSci 100
Decoding a message

100000100001001101
Decoding a message

00000100001001101

Compressing a message

60

37

21

10

5

E

3

O

C

1

F

1

P

1

U

5

5

11

11

N

3

T

1

C

G

1

G

1

D

1

L

1

R

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

0000100001001101

G

CompSci 100

21.31
Decoding a message

000100001001101

CompSci 100 21.32
Decoding a message

00100001001101
Decoding a message

0100001001101

G
Decoding a message

100001001101

G
Decoding a message

00001001101

GO

CompSci 100
Decoding a message

0001001101

GO
Decoding a message

001001101

GO
Decoding a message

01001101

GO
Decoding a message

1001101

GO
Decoding a message

001101

GOO
Decoding a message

01101

GOO
Decoding a message

1101

GOO
Decoding a message
Decoding a message

01

G O O
Decoding a message

GOOD
Decoding a message

0110000010001001101

GOOD
Decoding

1. Read in tree data \( O(\quad) \)

2. Decode bit string with tree \( O(\quad) \)
Huffman coding: go go gophers

ASCII 3 bits Huffman

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII</th>
<th>Huffman</th>
<th>3 bits</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>103</td>
<td>1100111</td>
<td>000</td>
<td>??</td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>1101111</td>
<td>001</td>
<td>??</td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>1110000</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>1101000</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>1100101</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>1110010</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>1110011</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>sp.</td>
<td>32</td>
<td>1000000</td>
<td>111</td>
<td></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” ⇔ “1010110100100101001110011100000"

```
   37
  /   \
 /     \|
21      16
|       |
10      8
  |      |
5       4
  |      |
5       4
  |      |
11      4
  |      |
6       4
  |      |
11      4
  |      |
6       4
  |      |
13      4
  |      |
23      4
  |      |
3       3
  |      |
3       3
  |      |
I       3
  |      |
3       3
  |      |
A       2
  |      |
A       B
```

CompSci 100 21.51
“A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS”

E.g. “A SIMPLE” ⇔ “1010110100100101001110011100000”
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" ⇔ "10101101001001001110011001100000"

```
10
  E
  5
  5
    O
    1
    C
    1
  10
  11
    N
    5
    6
      S
      6
      4
      4
    16
      M
      4
    8
      G
      2
    8
      D
      2
    8
      L
      2
    8
      R
      2
    8
      12
      I
      6
      6
        A
        3
      11
      11
```

CompSci 100
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” $\Leftrightarrow “1010110100100101001110011100000”$
Huffman Tree 2

- “A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS”

  - E.g. “A SIMPLE” $\Leftrightarrow$ “1010110100100101001110011100000”
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