Outline

1. Variable-Length Codes
2. Prefix Codes
3. Huffman Codes
4. Creating a Huffman Code
5. Assignment
Variable-Length Codes

Definition (Variable-Length Code)

A variable-length code is a code in which the codewords do not all have the same length.

- Morse code is an example of a variable-length code.
  - A = · −
  - B = − · · ·
  - C = − · − ·
  - D = − · ·
  - E = ·
Prefix Codes

Definition (Prefix Property)

A code has the **prefix property** if no codeword is a prefix of another codeword.

- Morse code does not have the prefix property.
- For example,
  - D (−···) is a prefix of B (−····).
  - E (·) is a prefix of A (·−).
As a consequence, Morse code really depends on three types of signal:
- Short (dot).
- Long (dash).
- Space.

The “spaces” are needed to distinguish.
- BED = — · · · ■ · ■ — · ·
- DEED = — · · ■ · ■ · ■ — · ·

Thus, Morse code is not truly a binary code.
Had Morse made a fixed-length code, the spaces would not be needed.

For example:

- A = · − · · ·
- B = − · · · −
- C = − · − · ·
- D = − · · − −
- E = · − − · −

What is the benefit of variable-length codes?
Morse employed the principle that the more common letters should have shorter codewords.

One one hand
- E = ·
- T = −

On the other hand
- Q = − − · −
- X = − · · −
Fixed vs. Variable-Length Codes

The benefit is that messages, on the average, are shorter because the majority of the codewords are the shorter ones.
A Huffman code is a variable-length prefix code that uses the same principle as Morse code, but applies it more carefully.
Example

For the message

ATTACK AT CRACK OF DAWN!!

the Huffman code would be

<table>
<thead>
<tr>
<th>Char</th>
<th>Codeword</th>
<th>Char</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>000</td>
<td>R</td>
<td>01010</td>
</tr>
<tr>
<td>N</td>
<td>00100</td>
<td>W</td>
<td>01011</td>
</tr>
<tr>
<td>O</td>
<td>00101</td>
<td>C</td>
<td>011</td>
</tr>
<tr>
<td>D</td>
<td>00110</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>F</td>
<td>00111</td>
<td>T</td>
<td>110</td>
</tr>
<tr>
<td>!</td>
<td>0100</td>
<td>K</td>
<td>111</td>
</tr>
</tbody>
</table>
Example

- The message would be encoded as
  1011011010011111000101100000110101010
  01111100001010011100001101001011001
  0001000100

- Even though there are no spaces, this message can be decoded unambiguously.
Furthermore, this encoded message is as short as possible for any variable-length code.

It’s average length per character is

$$\frac{84 \text{ bits}}{25 \text{ chars}} = 3.36 \text{ bits/char.}$$
Creating a Huffman Code

- To create a Huffman code,
- Find the relative frequencies of all characters in the message.

<table>
<thead>
<tr>
<th>Char</th>
<th>Freq</th>
<th>Char</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.20</td>
<td>O</td>
<td>0.04</td>
</tr>
<tr>
<td>C</td>
<td>0.12</td>
<td>R</td>
<td>0.04</td>
</tr>
<tr>
<td>D</td>
<td>0.04</td>
<td>T</td>
<td>0.12</td>
</tr>
<tr>
<td>F</td>
<td>0.04</td>
<td>W</td>
<td>0.04</td>
</tr>
<tr>
<td>K</td>
<td>0.08</td>
<td>!</td>
<td>0.08</td>
</tr>
<tr>
<td>N</td>
<td>0.04</td>
<td>⊥</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Creating a Huffman Code

For each symbol, create a tree with a single node holding that symbol and the relative frequency.

```
A: 0.20
C: 0.12
D: 0.04
F: 0.04
K: 0.08
N: 0.04
O: 0.04
R: 0.04
T: 0.12
W: 0.04
!: 0.08
": 0.16
```
Put the trees in a list, sorted by relative frequency.
Creating a Huffman Code

While there are at least two trees remaining,

- Combine the two trees of minimum “weight.”
- Make one the left subtree and the other the right subtree of a new tree.
- The weight of the new tree is the combined weight of the two minimal trees.
- Remove the two minimal trees and add the new tree in its proper place in the list.
Creating a Huffman Code
Creating a Huffman Code

Variable-Length Codes

Prefix Codes

Huffman Codes

Creating a Huffman Code

Assignment
Creating a Huffman Code
Creating a Huffman Code

Huffman Codes
Robb T. Koether
Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code
Creating a Huffman Code
Creating a Huffman Code
Creating a Huffman Code

Huffman Codes
Robb T. Koether

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code

0.20 A
0.16 C
0.08 D
0.04 E
0.16 F
0.08 G
0.04 H
0.04 I
0.04 J
0.04 K
0.04 L
0.04 M
0.04 N
0.04 O
0.04 P
0.04 Q
0.04 R
0.04 S
0.04 T
0.04 U
0.04 V
0.04 W
0.04 X
0.04 Y
0.04 Z
Creating a Huffman Code

Huffman Codes
Robb T. Koether

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code
Creating a Huffman Code
Creating a Huffman Code

Huffman Codes
Robb T. Koether

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code
Creating a Huffman Code
Creating a Huffman Code
Creating a Huffman Code

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code

Huffman Codes
Robb T. Koether

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code

- Variable-Length Codes
- Prefix Codes
- Huffman Codes
- Creating a Huffman Code
- Assignment
Creating a Huffman Code

Huffman Codes
Robb T. Koether
Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code

Huffman Codes
Robb T. Koether

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment
Creating a Huffman Code

Huffman Codes
Robb T. Koether

Variable-Length Codes
Prefix Codes
Huffman Codes
Creating a Huffman Code
Assignment

Creating a Huffman Code

1.00
0.04 0.04
0.08
0.16
0.32
0.60
0.04 0.04
0.08 0.08
0.16
0.04 0.04
0.08
0.20
0.40
0.12 0.08
0.20
0.16 0.12
0.28
N O D F
' '
R W
C
A
T K

Diagram:

1.00

0.60

0.32

0.16

0.16

0.08
0.08

0.04 0.04 0.04 0.04
N O D F

0.08

0.08

0.08
0.08

0.12 0.08 0.12 0.08
C A T K

0.20

0.40
Creating a Huffman Code

- Label every left branch 0.
- Label every right branch 1.
Creating a Huffman Code
Creating a Huffman Code

Example
The Huffman code turns out to be

<table>
<thead>
<tr>
<th>Char</th>
<th>Codeword</th>
<th>Char</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>🎃</td>
<td>000</td>
<td>R</td>
<td>01010</td>
</tr>
<tr>
<td>N</td>
<td>00100</td>
<td>W</td>
<td>01011</td>
</tr>
<tr>
<td>O</td>
<td>00101</td>
<td>C</td>
<td>011</td>
</tr>
<tr>
<td>D</td>
<td>00110</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>F</td>
<td>00111</td>
<td>T</td>
<td>110</td>
</tr>
<tr>
<td>!</td>
<td>0100</td>
<td>K</td>
<td>111</td>
</tr>
</tbody>
</table>
Creating a Huffman Code

- For each character, follow the path from root to leaf for that character.
- The sequence of 0’s and 1’s is the codeword for that character.
- For example,
  - A = 10
  - C = 011
  - W = 01011
Why must the resulting code must have the prefix property?
Average Number of Bits per Character

- We found the average number of bits per character to be 3.36.
- We may also compute this average theoretically from the code and the distribution of characters.
### Average Number of Bits per Character

<table>
<thead>
<tr>
<th>Char</th>
<th>Code</th>
<th>Length</th>
<th>Rel Freq</th>
<th>Prod</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>2</td>
<td>0.20</td>
<td>0.40</td>
</tr>
<tr>
<td>C</td>
<td>011</td>
<td>3</td>
<td>0.12</td>
<td>0.36</td>
</tr>
<tr>
<td>D</td>
<td>00110</td>
<td>5</td>
<td>0.04</td>
<td>0.20</td>
</tr>
<tr>
<td>F</td>
<td>00111</td>
<td>5</td>
<td>0.04</td>
<td>0.20</td>
</tr>
<tr>
<td>K</td>
<td>111</td>
<td>3</td>
<td>0.08</td>
<td>0.24</td>
</tr>
<tr>
<td>N</td>
<td>00100</td>
<td>5</td>
<td>0.04</td>
<td>0.20</td>
</tr>
<tr>
<td>O</td>
<td>00101</td>
<td>5</td>
<td>0.04</td>
<td>0.20</td>
</tr>
<tr>
<td>R</td>
<td>01010</td>
<td>5</td>
<td>0.04</td>
<td>0.20</td>
</tr>
<tr>
<td>T</td>
<td>110</td>
<td>3</td>
<td>0.12</td>
<td>0.36</td>
</tr>
<tr>
<td>W</td>
<td>01011</td>
<td>5</td>
<td>0.04</td>
<td>0.20</td>
</tr>
<tr>
<td>!</td>
<td>0100</td>
<td>4</td>
<td>0.08</td>
<td>0.32</td>
</tr>
<tr>
<td>□</td>
<td>000</td>
<td>3</td>
<td>0.16</td>
<td>0.48</td>
</tr>
</tbody>
</table>
Average Number of Bits per Character

- Add up the last column to get the average:

\[
\text{Average} = 3.36.
\]
Encoding and Decoding

- To encode the message, write the code as a table and use table lookup.
- To decode the message, write the code as a binary tree and move left or right according the bits.
Homework

- Read Section 15.1, pages 832 - 839.