More Bits and Bytes
Huffman Coding
Encoding Text: How is it done?
ASCII, UTF, Huffman algorithm
## ASCII

<table>
<thead>
<tr>
<th>ASCII</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0000</th>
<th>N</th>
<th>S</th>
<th>S</th>
<th>E</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>H</th>
<th>L</th>
<th>Y</th>
<th>F</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>0010</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0100</th>
<th>@</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
</tr>
<tr>
<td>0110</td>
<td>~</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>0111</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1000</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>P</td>
<td>S</td>
<td>C</td>
<td>M</td>
<td>S</td>
<td>E</td>
<td>P</td>
<td>O</td>
<td>Q</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>i</td>
<td>c</td>
<td>£</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
</tr>
<tr>
<td>1011</td>
<td>°</td>
<td>±</td>
<td>²</td>
<td>³</td>
<td>µ</td>
<td>µ</td>
<td>µ</td>
<td>µ</td>
<td>µ</td>
<td>µ</td>
<td>µ</td>
<td>µ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1100</th>
<th>À</th>
<th>Á</th>
<th>Â</th>
<th>À</th>
<th>Ä</th>
<th>À</th>
<th>È</th>
<th>È</th>
<th>È</th>
<th>È</th>
<th>È</th>
<th>È</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>Ð</td>
<td>Ñ</td>
<td>Ì</td>
<td>Ò</td>
<td>Ó</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1110</th>
<th>à</th>
<th>á</th>
<th>ã</th>
<th>à</th>
<th>æ</th>
<th>ç</th>
<th>è</th>
<th>è</th>
<th>è</th>
<th>è</th>
<th>è</th>
<th>è</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
</tr>
</tbody>
</table>
UTF-8: All the alphabets in the world

- Uniform Transformation Format: a variable-width encoding that can represent every character in the Unicode Character set
- 1,112,064 of them!!!
- UTF-8 is the dominant character encoding for the World-Wide Web, accounting for more than half of all Web pages.
- The Internet Engineering Task Force (IETF) requires all Internet protocols to identify the encoding used for character data
- The supported character encodings must include UTF-8.
UTF is a VARIABLE LENGTH ALPHABET CODING

- Remember ASCII can only represent 128 characters (7 bits)
- UTF encodes over one million
- Why would you want a variable length coding scheme?

<table>
<thead>
<tr>
<th>Bits</th>
<th>Last code point</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>U+007F</td>
<td>0xxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+07FF</td>
<td>110xxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>U+FFFF</td>
<td>1110xxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>U+1FFFFFF</td>
<td>11110xxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>U+3FFFFFFF</td>
<td>111110xx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>U+7FFFFFFF</td>
<td>1111110x</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
</tr>
<tr>
<td>Bits of code point</td>
<td>First code point</td>
<td>Last code point</td>
<td>Bytes in sequence</td>
<td>Byte 1</td>
<td>Byte 2</td>
<td>Byte 3</td>
<td>Byte 4</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>7</td>
<td>U+0000</td>
<td>U+007F</td>
<td>1</td>
<td>0xxxxxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+0080</td>
<td>U+07FF</td>
<td>2</td>
<td>110xxxxx</td>
<td>10xxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>U+0800</td>
<td>U+FFFF</td>
<td>3</td>
<td>1110xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>U+10000</td>
<td>U+1FFFFF</td>
<td>4</td>
<td>11110xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
</tr>
</tbody>
</table>
## UTF-8

<table>
<thead>
<tr>
<th>Bits of code point</th>
<th>First code point</th>
<th>Last code point</th>
<th>Bytes in sequence</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>U+0000</td>
<td>U+007F</td>
<td>1</td>
<td>0xxxxxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+0080</td>
<td>U+07FF</td>
<td>2</td>
<td>110xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>U+0800</td>
<td>U+FFFF</td>
<td>3</td>
<td>1110xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>U+10000</td>
<td>U+1FFFFF</td>
<td>4</td>
<td>11110xxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
</tr>
</tbody>
</table>

What is the first Unicode value represented by this sequence?

11101010 10000011 10000111 00111111 11000011 10000000

A. 00000000001101010
B. 00000000011101010
C. 0000001010000111
D. 1010000011000111
UTF-8

<table>
<thead>
<tr>
<th>Bits of code point</th>
<th>First code point</th>
<th>Last code point</th>
<th>Bytes in sequence</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>U+0000</td>
<td>U+007F</td>
<td>1</td>
<td>0xxxxxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+0080</td>
<td>U+07FF</td>
<td>2</td>
<td>110xxxxxx</td>
<td>10xxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>U+0800</td>
<td>U+FFFFFF</td>
<td>3</td>
<td>1110xxxxxx</td>
<td>10xxxxxx</td>
<td>10xxxxxx</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>U+10000</td>
<td>U+1FFFFFF</td>
<td>4</td>
<td>11110xxxxx</td>
<td>10xxxxxx</td>
<td>10xxxxxx</td>
<td>10xxxxxx</td>
</tr>
</tbody>
</table>

How many Unicode characters are represented by this sequence? 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0

A. 1  
B. 2  
C. 3  
D. 4  
E. 5
How many bits for all of Unicode?

There are 1,112,064 different Unicode characters. If a fixed bit format (like ascii with its 7 bits) were used, how many bits would you need for each character? (Hint: $2^{10} = 1024$)

A. 10
B. 17
C. 21
D. 32
E. 40
A Curious Story…

*The Diving Bell and the Butterfly*
Jean-Dominique Bauby
Asking Yes/No Questions

- A protocol for Yes/No questions
  - One blink == Yes
  - Two blinks == No
- PandA implies that this is not the fewest number of blinks ... really?
Asking Letters

In English ETAOINSHRDLU...
Compare Two Orderings

- How many questions to encode:
  
  *Plus ça change, plus c'est la même chose?*

- Asking in Frequency Order:
  
  ESARINTULOMDPCFBVHGJQZYXKW

  9  12
How many questions to encode:

*Plus ça change, plus c'est la même chose?*

- Asking in Frequency Order:
  ESARINTULOMDPCFBVHGJQZYXKW

- Asking in Alphabetical Order:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ

12  16
Compare Two Orderings

- How many questions to encode:
  
  *Plus ça change, plus c'est la même chose?*

- Asking in Frequency Order: 247
  ESARINTULOMDPCFBVHGJQZYXKW

- Asking in Alphabetical Order: 324
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
An Algorithm

- Spelling by going through the letters is an algorithm.
- Going through the letters in frequency order is a program (also, an algorithm but with the order specified to a particular case, i.e. FR).
- The nurses didn’t look this up in a book … they invented it to make their work easier; they were thinking computationally, though they probably didn’t know it.
Coding can be used to do Compression

- **What is CODING?**
  - The conversion of one representation into another

- **What is COMPRESSION?**
  - Change the representation (digitization) in order to **reduce** size of data (number of bits needed to represent data)

- **Benefits**
  - **Reduce storage** needed
    - Consider growth of digitized data.
  - **Reduce** transmission cost / **latency** / bandwidth
  - When you have a 56K dialup modem, every savings in BITS counts, **SPEED**
    - Also consider telephone lines, texting
What makes it possible to do Compression?

- **IN OTHER WORDS: When is Coding USEFUL?**
- **When there is Redundancy**
  - Recognize repeating patterns
  - Exploit using
    - Dictionary
    - Variable length encoding
- **When Human perception is less sensitive to some information**
  - Can discard less important data
How easy is it to do it?

- Depends on data
  - Random data ⇒ hard
    - Example: 1001110100 ⇒ ?
  - Organized data ⇒ easy
    - Example: 1111111111 ⇒ 1×10

- WHAT DOES THAT MEAN?
  - There is NO universally best compression algorithm
  - It depends on how tuned the coding is to the data you have
Can you lose information with Compression?

- **Lossless** Compression is not guaranteed
  - Pigeonhole principle
    - Reduce size 1 bit $\Rightarrow$ can only store $\frac{1}{2}$ of data
    - Example
      - $000, 001, 010, 011, 100, 101, 110, 111 \Rightarrow 00, 01, 10, 11$
  - CONSIDER THE ALTERNATIVE
    - IF LOSSLESS COMPRESSION WERE GUARANTEED THEN
      - Compress file (reduce size by 1 bit)
      - Recompress output
      - Repeat (until we can store data with 0 bits)
  - OBVIOUS CONTRADICTION $\Rightarrow$ IT IS NOT GUARANTEED.
Huffman Code: A Lossless Compression

- Use Variable Length codes based on frequency (like UTF does)

Approach
- Exploit statistical frequency of symbols
- What do I MEAN by that? WE COUNT!!!
- HELPS when the frequency for different symbols varies widely

Principle
- Use fewer bits to represent frequent symbols
- Use more bits to represent infrequent symbols

A A A B A

A A B A

A
Huffman Code Example

- “dog cat cat bird bird bird bird fish”

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Dog</th>
<th>Cat</th>
<th>Bird</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1/8</td>
<td>1/4</td>
<td>1/2</td>
<td>1/8</td>
</tr>
<tr>
<td>Original</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Encoding</td>
<td>2 bits</td>
<td>2 bits</td>
<td>2 bits</td>
<td>2 bits</td>
</tr>
<tr>
<td>Huffman</td>
<td>110</td>
<td>10</td>
<td>0</td>
<td>111</td>
</tr>
<tr>
<td>Encoding</td>
<td>3 bits</td>
<td>2 bits</td>
<td>1 bit</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

- Expected size
  - Original ⇒ \( \frac{1}{8} \times 2 + \frac{1}{4} \times 2 + \frac{1}{2} \times 2 + \frac{1}{8} \times 2 = 2 \text{ bits / symbol} \)
  - Huffman ⇒ \( \frac{1}{8} \times 3 + \frac{1}{4} \times 2 + \frac{1}{2} \times 1 + \frac{1}{8} \times 3 = 1.75 \text{ bits / symbol} \)
Huffman Code Example

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Dog</th>
<th>Cat</th>
<th>Bird</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1/8</td>
<td>1/4</td>
<td>1/2</td>
<td>1/8</td>
</tr>
<tr>
<td>Original Encoding</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2 bits</td>
<td>2 bits</td>
<td>2 bits</td>
<td>2 bits</td>
</tr>
<tr>
<td>Huffman Encoding</td>
<td>110</td>
<td>10</td>
<td>0</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>3 bits</td>
<td>2 bits</td>
<td>1 bit</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

How many bits are saved using the above Huffman coding for the sequence Dog Cat Bird Bird Bird?

A. 0     B. 1     C. 2     D. 3     E. 4
Huffman Code Algorithm: Data Structures

- **Binary (Huffman) tree**
  - Represents Huffman code
  - Edge $\Rightarrow$ code (0 or 1)
  - Leaf $\Rightarrow$ symbol
  - Path to leaf $\Rightarrow$ encoding
  - Example
    - A = “11”, H = “10”, C = “0”
    - Good when ???
      - A, H less frequent than C in messages

- Want to efficiently build a binary tree
Huffman Code Algorithm Overview

- Order the symbols with least frequent first (will explain)
- Build a tree piece by piece…
- Encoding
  - Calculate frequency of symbols in the message, language
  - JUST COUNT AND DIVIDE BY TOTAL NUMBER OF SYMBOLS
  - Create binary tree representing “best” encoding
  - Use binary tree to encode compressed file
    - For each symbol, output path from root to leaf
    - Size of encoding = length of path
  - Save binary tree
Huffman Code – Creating Tree

Algorithm (Recipe)

- Place each symbol in leaf
  - Weight of leaf = symbol frequency
- Select two trees L and R (initially leaves)
  - Such that L, R have lowest frequencies among all tree
  - Which L, R have the lowest number of occurrences in the message?
- Create new (internal) node
  - Left child \(\Rightarrow\) L
  - Right child \(\Rightarrow\) R
  - New frequency \(\Rightarrow\) \(\text{frequency}(L) + \text{frequency}(R)\)
- Repeat until all nodes merged into one tree
Huffman Tree Construction 1

A 3  C 5  E 8  H 2  I 7
Huffman Tree Step 2: can first re-order by frequency
Huffman Tree Construction 3
Huffman Tree Construction 4

A  H  C
3  2  5

10

E  I
8  7
15
Huffman Coding Example

- Huffman code
  - E = 01
  - I = 00
  - C = 10
  - A = 111
  - H = 110

- Input
  - ACE

- Output
  - (111)(10)(01) = 1111001
Huffman Code Algorithm Overview

- Decoding
  - Read compressed file & binary tree
  - Use binary tree to decode file
    - Follow path from root to leaf
Huffman Decoding I

1111001
Huffman Decoding 2

1111001
Huffman Decoding 3

1111001

A
Huffman Decoding 6

1111001
AC
Huffman Decoding 7

1111001

ACE
Huffman Code Properties

- **Prefix code**
  - No code is a *prefix* of another code
  - Example
    - \text{Huffman(“dog”) } \Rightarrow 01
    - \text{Huffman(“cat”) } \Rightarrow 011  // not legal prefix code
  - Can stop as soon as complete code found
  - No need for end-of-code marker

- **Nondeterministic**
  - Multiple Huffman coding possible for same input
  - If more than two trees with same minimal weight
Huffman Code Properties

- **Greedy algorithm**
  - Chooses best local solution at each step
  - Combines 2 trees with lowest frequency
- **Still yields overall best solution**
  - Optimal prefix code
  - Based on statistical frequency
- **Better compression possible (depends on data)**
  - Using other approaches (e.g., pattern dictionary)
Huffman Coding. Another example.
Huffman Tree Example 2. Step 1

“TO BE OR NOT TO BE”

- T = 3
- O = 4
- B = 2
- E = 2
- R = 1
- N = 1
Huffman Tree: TO BE OR NOT TO BE
Huffman Tree: TO BE OR NOT TO BE
Huffman Tree: TO BE OR NOT TO BE

R  N  E
1  1  2
2  2  4

B  T  O
2  3  4
5
Huffman Tree: TO BE OR NOT TO BE

R
1

N
1

2
E
2

O
4

8

B
2

5

T
3

4

8
Huffman Tree: TO BE OR NOT TO BE
Huffman Tree: TO BE OR NOT TO BE

- R = 1111
- N = 1110
- E = 110
- B = 01
- O = 10
- T = 00
Huffman Tree: TO BE OR NOT TO BE

R  = 1111
N  = 1110
E  = 110
B  = 01
O  = 10
T  = 00

0010.01110.101111.11
101000.0010.01110
Huffman Tree: TO BE OR NOT TO BE

N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

0010.01110.101111.11
101000.0010.01110
32 bits
How many bits would it take to store this message if every letter was represented with the same number of bits? You should first figure out how many bits it takes to represent 6 different values/letters.

A. 26
B. 32
C. 39
D. 48
E. 52

0010.01110.101111.11
101000.0010.01110

32 bits
Decoding: 2nd example “BEBE”

N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00

0111001110 = ?
No code is prefix of another

N = 1110
R = 1111
E = 110
B = 01
O = 10
T = 00
DECODING: Your turn

R = 1111
N = 1110
E = 110
B = 01
O = 10
T = 00

111110011000 = ?
DECODING: Your turn

A. ROBBER
B. REBOOT
C. ROBOT
D. ROOT
E. ROBERT

\[ 111110011000 \Rightarrow ? \]