Announcements
Encoding Strings
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers and bytes

A byte is 8 bits and can encode any integer 0–255.

<table>
<thead>
<tr>
<th>bytes</th>
<th>integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
</tr>
</tbody>
</table>

Variable-length encoding: integers vary in the number of bytes required to encode them.

In Python: `string` length is measured in characters, `bytes` length in bytes.

(Demo)
Fixed-Length Encodings
**A First Attempt**

- Let’s use an encoding

<table>
<thead>
<tr>
<th>Letter</th>
<th>Binary</th>
<th>Letter</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>o</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>q</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>r</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>s</td>
<td>1</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>t</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>u</td>
<td>0</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>v</td>
<td>1</td>
</tr>
<tr>
<td>j</td>
<td>1</td>
<td>w</td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>0</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
<td>y</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>z</td>
<td>0</td>
</tr>
</tbody>
</table>
Decoding

• An encoding without a deterministic decoding procedure is not very useful

• How many bits do we need to encode each letter uniquely?
  • lowercase alphabet

• 5 bits
## A Second Attempt

- Let’s try another encoding

<table>
<thead>
<tr>
<th>Letter</th>
<th>Binary</th>
<th>Letter</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00000</td>
<td>n</td>
<td>01101</td>
</tr>
<tr>
<td>b</td>
<td>00001</td>
<td>o</td>
<td>01110</td>
</tr>
<tr>
<td>c</td>
<td>00010</td>
<td>p</td>
<td>01111</td>
</tr>
<tr>
<td>d</td>
<td>00011</td>
<td>q</td>
<td>10000</td>
</tr>
<tr>
<td>e</td>
<td>00100</td>
<td>r</td>
<td>10001</td>
</tr>
<tr>
<td>f</td>
<td>00101</td>
<td>s</td>
<td>10010</td>
</tr>
<tr>
<td>g</td>
<td>00110</td>
<td>t</td>
<td>10011</td>
</tr>
<tr>
<td>h</td>
<td>00111</td>
<td>u</td>
<td>10100</td>
</tr>
<tr>
<td>i</td>
<td>01000</td>
<td>v</td>
<td>10101</td>
</tr>
<tr>
<td>j</td>
<td>01001</td>
<td>w</td>
<td>10110</td>
</tr>
<tr>
<td>k</td>
<td>01010</td>
<td>x</td>
<td>10111</td>
</tr>
<tr>
<td>l</td>
<td>01011</td>
<td>y</td>
<td>11000</td>
</tr>
<tr>
<td>m</td>
<td>01100</td>
<td>z</td>
<td>11001</td>
</tr>
</tbody>
</table>
Analysis

Pros

• Encoding was easy

• Decoding was deterministic

Cons

• Takes more space...

• What restriction did we place that’s unnecessary?

• Fixed length
Variable-Length Encodings
**Variable Length Encoding**

- Encoding Candidate 1: A: 1, B: 01, C: 10, D: 11, E: 100, F: 101, ...
  - What does 0111 encode?

- Encoding Candidate 2: A: 00, B: 01, C: 100, D: 101, E: 1100, F: 1101, ...
  - What does 0100101 encode? How about 1011001101001001100?

- Deterministic decoding from left to right is possible if the encoding of one character is **never** a proper prefix of the decoding of another character.
Deterministic Codes Have a Tree Structure

\[
\begin{array}{c}
0 \\
1 \\
C
\end{array}
\begin{array}{c}
A \\
B \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Letter</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
Huffman Encoding

Let’s pretend we want to come up with the optimal encoding:

• AAAAAAAAAABBBCCCCCCDDDDDDDDDD

• A appears 10 times
• B appears 5 times
• C appears 7 times
• D appears 9 times
Huffman Encoding

- Start with the two smallest frequencies
  - A appears 10 times, B appears 5 times, C appears 7 times, D appears 9 times
Huffman Encoding

• Continue...

  • A appears 10 times, B & C appear a combined 12 times, D appears 9 times
Huffman Encoding

- And finally…

```
A 0 1
  B
D
```

```
A 0 1
  B
D
```

```
  0 1
  B  C
    0 1
    A  D
```

```
  0 1
  B  C
    0 1
    A  D
```

```
Huffman Encoding

• Another example...
  
  • AAAAAAAAAABCCD
  
  • A appears 10 times
  
  • B appears 1 time
  
  • C appears 2 times
  
  • D appears 1 time
Huffman Encoding

- Start with the two smallest frequencies
  - A appears 10 times, B appears 1 time, C appears 2 times, D appears 1 time
Huffman Encoding

• Start with the two smallest frequencies
  • A appears 10 times, B & D appear a combined 2 times, C appears 2 times
Huffman Encoding

- And finally...