Data Compression: Huffman Coding

10.1.2 in Weiss (p.389)

Why compress files?

- Resources are limited
  - Long-term storage (disk space)
  - Internet transfers (network bandwidth)
  - Fast memory access (cache)
- Because we can

Is compression possible?

- Most data contains redundancies
  - E.g. Human-readable text
  - Not all combinations are equally likely.
  - In English, some letter pairs (“qu”, “th”, etc.) appear more frequently than others.
- The essential information content is much less
  - Information theory developed by Shannon in 1950s
  - If you have $n$ equally likely symbols, how many bits do you need to represent them?

What can be compressed?

- Which of the following would we require in pristine shape? (lossless)
  - C++ source file
  - Binary executable
  - Photograph of your thumb
  - Video of a monkey eating a banana
  - MP3 ringtones
  - E-mail
Data Compression

- **Lossless** compression $X = X'$
- **Lossy** compression $X \neq X'$
- **Compression Ratio** $|X|/|Y|$
  - Where $|X|$ is the # of bits in $X$.

---

Lossy Compression

- Ideal for signals with more data than humans can process (high-fidelity).
- Most audio and video information can be removed without being noticed.

**Standards:**
- JPEG (Joint Photographic Experts Group)
- MPEG (Motion Picture Experts Group)
- MP3 (MPEG-1, Layer 3)

---

Lossless Compression

- No data is lost.
- Information is low-fidelity to begin with.

**Standards:**
- Gzip, Unix compress, zip, GIF

Another technique is **run-length encoding (RLE)**, part of several compression techniques (BMP, PCX, TIFF, PDF)

A run of characters is replaced by the **number of characters** of that type and a **single character**:

```
RTAAAAAADEEEE  RT*6AD*4E  
```

---

Lossless Compression of text

**ASCII** = fixed 8 bits per character

**Example:** “hello there”
  - 11 characters * 8 bits = 88 bits
  - Can we encode this message using fewer bits?

  **We could look **JUST** at the message, there are only 6 possible characters + one space. = 7 things**
  - **needs 3 bits.**

  **Encode:** aabddcaa = could do as 16 bits (each character = 2 bits each)
  - **Huffman can do as 14 bits**
**Huffman Coding**

- Uses *frequencies* of symbols in a string to build a **prefix code**.
- **Prefix Code** – no code in our encoding is a prefix of another code.

<table>
<thead>
<tr>
<th>Letter</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>100</td>
</tr>
<tr>
<td>c</td>
<td>101</td>
</tr>
<tr>
<td>d</td>
<td>11</td>
</tr>
</tbody>
</table>

**Note:** codes are variable length – (0 to 3 bits per character)

**Huffman Tree**

- All symbols at leaves
- Edges labeled with 0-1
- Why does this guarantee prefix code?

**Decoding a Prefix Code**

Loop

- start at root of tree
  
  loop
    
  if bit read = 1 then take 1-child
  
  else, take 0-child
  
  until node is a leaf
  
  Report character found!
  
Until end of the message

**Decode:** 11100010100110

<table>
<thead>
<tr>
<th>Letter</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>100</td>
</tr>
<tr>
<td>c</td>
<td>101</td>
</tr>
<tr>
<td>d</td>
<td>11</td>
</tr>
</tbody>
</table>

**Why did we need the code to be a prefix code?**

- 8 characters: 
  - 8*8 bits = 64 bits in ASCII
  - 8*2 bits = 16 bits (if used 2 bits each)
  - 14 bits = Huffman (uses frequency)
Cost of a Huffman Tree

Cost of a Huffman Tree containing n symbols is the expected length of a codeword.

\[ C(T) = p_1 \times r_1 + p_2 \times r_2 + p_3 \times r_3 + \ldots + p_n \times r_n \]

For previous example = (.50 \times 1) + (.125 \times 3) + (.125 \times 3) + (.25 \times 2)

Where:
- \( p_i \) = the probability that a symbol occurs
- \( r_i \) = the length of the path from the root to the node

Constructing a Huffman Tree

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>.50</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>.125</td>
<td>100</td>
</tr>
<tr>
<td>c</td>
<td>.125</td>
<td>101</td>
</tr>
<tr>
<td>d</td>
<td>.25</td>
<td>11</td>
</tr>
</tbody>
</table>

Huffman Tree Construction

Part the First
- Given a symbol-frequency table:
  - Start with a forest of one-node trees
  - One for each symbol
  - Associate a frequency with each tree

Huffman Tree Construction

Part the Second
- While there is more than one tree
  - Pick the two trees with smallest frequency
  - Combine them into one tree
  - And add their frequencies
Huffman Tree Construction
Part the Third

- Pick arbitrary 0-1 labellings for the edges
  - More than one Huffman tree is possible
  - How to get from one Huffman tree to another?

Digression:
Why “anti-compress” files?

- Error-correcting codes
  - By adding redundancies into data instead of removing it, we can make it robust to noise.
  - Noise on our communication channel will corrupt this redundancy.
    - CD/DVD optical storage
    - Hard disk magnetic storage
    - WiFi
    - Ethernet / CDMA
  - Examples: checksums, phonetic alphabet