Today’s topics

Data Representation
- Memory organization
- Compression

Upcoming
- Systems
- Connections
- Algorithms

Reading
Computer Science, Chapter 1-3

Main Memory Cells

- Cell: A unit of main memory (typically 8 bits which is one byte)
  - Most significant bit: the bit at the left (high-order) end of the conceptual row of bits in a memory cell
  - Least significant bit: the bit at the right (low-order) end of the conceptual row of bits in a memory cell

Main Memory Addresses

- Address: A “name” that uniquely identifies one cell in the computer’s main memory
  - The names are actually numbers.
  - These numbers are assigned consecutively starting at zero.
  - Numbering the cells in this manner associates an order with the memory cells.

Figure 1.7 The organization of a byte-size memory cell

<table>
<thead>
<tr>
<th>High-order end</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>Low-order end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most significant bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Least significant bit</td>
</tr>
</tbody>
</table>
Figure 1.8 Memory cells arranged by address

Memory Terminology
- Random Access Memory (RAM): Memory in which individual cells can be easily accessed in any order
- Dynamic Memory (DRAM): RAM composed of volatile memory

Mass Storage
- On-line versus off-line
- Typically larger than main memory
- Typically less volatile than main memory
- Typically slower than main memory

Mass Storage Systems
- Magnetic Systems
  - Disk
  - Tape
- Optical Systems
  - CD
  - DVD
- Flash Drives
Figure 1.9  A magnetic disk storage system

Figure 1.10  Magnetic tape storage

Figure 1.11  CD storage

Files

- **File**: A unit of data stored in mass storage system
- **Fields and keyfields**
- **Physical record versus Logical record**
- **Buffer**: A memory area used for the temporary storage of data (usually as a step in transferring the data)
Data Compression

- Compression is a high-profile application
  - .zip, .mp3, .jpg, .gif, .gz, ...
  - What property of MP3 was a significant factor in what made Napster work (why did Napster fundamentally fail?)

- Why do we care?
  - Secondary storage capacity doubles every year
  - Disk space fills up quickly on every computer system
  - More data to compress than ever before

More on Compression

- What’s the difference between compression techniques?
  - .mp3 files and .zip files?
  - .gif and .jpg?
  - Lossless and lossy

- Is it possible to compress (lossless) every file? Why?
- Lossy methods
  - Good for pictures, video, and audio (JPEG, MPEG, etc.)
- Lossless methods
  - Run-length encoding, Huffman, LZW, ...

Text Compression

- Input: String $S$
- Output: String $S'$
  - Shorter
  - $S$ can be reconstructed from $S'$
Figure 1.13 The message “Hello.” in ASCII

```
01001000 01100101 01101100 01101100 01101110 00101110
H   e   l   l   o   .
```

Text Compression: Examples

“abcde” in the different formats

<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>0</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>1</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>

```
Huffman coding: go go gophers

<table>
<thead>
<tr>
<th>ASCII</th>
<th>3 bits</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>103 1100111</td>
<td>000 00</td>
</tr>
<tr>
<td>o</td>
<td>111 1101111</td>
<td>001 01</td>
</tr>
<tr>
<td>p</td>
<td>112 1110000</td>
<td>010 1100</td>
</tr>
<tr>
<td>h</td>
<td>104 1101000</td>
<td>011 1101</td>
</tr>
<tr>
<td>e</td>
<td>101 1100101</td>
<td>100 1110</td>
</tr>
<tr>
<td>r</td>
<td>114 1110010</td>
<td>101 1111</td>
</tr>
<tr>
<td>s</td>
<td>115 1110011</td>
<td>110 101</td>
</tr>
<tr>
<td>sp.</td>
<td>32 1000000</td>
<td>111 101</td>
</tr>
</tbody>
</table>

- 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 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, hummm......

Building a tree

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

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?
Decoding a message

01100000100001001101

Huffman Tree 2

- "A SIMPLE STRING TO BE ENCODED USING A MINIMAL NUMBER OF BITS"
  - E.g. "A SIMPLE" ⇔ "10101101001000101001110011100000"

Data Compression

<table>
<thead>
<tr>
<th>Year</th>
<th>Scheme</th>
<th>Bit/Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>1967</td>
<td>ASCII</td>
<td>7.00</td>
</tr>
<tr>
<td>1950</td>
<td>Huffman</td>
<td>4.70</td>
</tr>
<tr>
<td>1977</td>
<td>Lempel-Ziv (LZ77)</td>
<td>3.94</td>
</tr>
<tr>
<td>1984</td>
<td>Lempel-Ziv-Welch (LZW) - Unix compress</td>
<td>3.32</td>
</tr>
<tr>
<td>1987</td>
<td>(LZH) used by zip and unzip</td>
<td>3.30</td>
</tr>
<tr>
<td>1987</td>
<td>Move-to-front</td>
<td>3.24</td>
</tr>
<tr>
<td>1987</td>
<td>gzip</td>
<td>2.71</td>
</tr>
<tr>
<td>1995</td>
<td>Burrows-Wheeler</td>
<td>2.29</td>
</tr>
<tr>
<td>1997</td>
<td>BOA (statistical data compression)</td>
<td>1.99</td>
</tr>
</tbody>
</table>

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