More Bits and Bytes
Huffman Coding

Encoding Text: How is it done?
ASCII, UTF, Huffman algorithm

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?

Next Processing Homework: DUE THURS!

CMPS10: Introduction to Computer Science
Homework 7: Making Blinky Work

Goal: In this assignment you will extend a program that you developed earlier, making it “work” with mouse clicks and mouse motion. This will require that you do four things:
- Make the program Active
- Make Blinky move right (you’ve done “move right” before) and left
- Add a mousePressed( ) block
- Control Blinky using the position of the mouse

Wrap Up
Blinky is now animated and he can move left or right according to where the mouse is located. If looks in a standard direction. Also, when the mouse is clicked, he can change color.

Turn In

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) where used, how many bits would you need for each character? (Hint: $2^{10} = 1024$)

A. 10  
B. 17  
C. 21  
D. 32  
E. 40

Huffman Coding:
http://en.wikipedia.org/wiki/Huffman_coding

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:  
  ESARINTULOMDPFCBVHGJQZYXKW

  9 12
Compare Two Orderings

- 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:
  
  ESARINTULOMDPCFBVHGJQZYXKW

- Asking in Alphabetical Order:
  
  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: 100110100 ⇒ ?*

  *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 ⇒ can only store ½ of data
    - Example
      - 000, 001, 010, 011, 100, 101, 110, 111 ⇒ 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 => 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

Huffman Code Example

- “dog cat 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 Encoding</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Huffman Encoding</td>
<td>110</td>
<td>10</td>
<td>0</td>
<td>111</td>
</tr>
</tbody>
</table>

- **Expected size**
  - Original ⇒ 1/8*2 + 1/4*2 + 1/2*2 + 1/8*2 = 2 bits / symbol
  - Huffman ⇒ 1/8*3 + 1/4*2 + 1/2*1 + 1/8*3 = 1.75 bits / symbol

Huffman Code Algorithm: Data Structures

- Binary (Huffman) tree
  - Represents Huffman code
  - Edge ⇒ code (0 or 1)
  - Leaf ⇒ symbol
  - Path to leaf ⇒ 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 Example

- **Symbol** | **Dog** | **Cat** | **Bird** | **Fish**
- **Frequency** | 1/8 | 1/4 | 1/2 | 1/8 |
- **Original Encoding** | 00 | 01 | 10 | 11 |
- **Huffman Encoding** | 110 | 10 | 0 | 111 |

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Dog</th>
<th>Cat</th>
<th>Bird</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Encoding</td>
<td>2 bits</td>
<td>2 bits</td>
<td>2 bits</td>
<td>2 bits</td>
</tr>
<tr>
<td>Huffman Encoding</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 – Creating Tree

- **Algorithm (Recipe)**
  - Place each symbol in leaf
    - Weight of leaf = symbol frequency
  - Select two trees L and R (initially leafs)
    - Such that L, R have lowest frequencies in 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

Huffman Tree Construction 2: can first re-order by frequency

Huffman Tree Construction 3

Huffman Tree Construction 4

Huffman Tree Construction 5

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>01</td>
</tr>
<tr>
<td>I</td>
<td>00</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>111</td>
</tr>
<tr>
<td>H</td>
<td>100</td>
</tr>
</tbody>
</table>

E = 01
I = 00
C = 10
A = 111
H = 110
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 1

Huffman Decoding 2

Huffman Decoding 3

Huffman Decoding 4
Huffman Code Properties

- **Prefix code**
  - No code is a prefix of another code
  - Example
    - Huffman("dog") → 01
    - Huffman("cat") → 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

- **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

- R = 1
- N = 1
- E = 2
- B = 2
- T = 3
- O = 4

Huffman Tree: TO BE OR NOT TO BE

- R = 1
- N = 1
- E = 2
- B = 2
- T = 3
- O = 4

Huffman Tree: TO BE OR NOT TO BE

- R = 1
- N = 1
- E = 2
- B = 2
- T = 3
- O = 4

Huffman Tree: TO BE OR NOT TO BE

- R = 1
- N = 1
- E = 2
- B = 2
- T = 3
- O = 4

Huffman Tree: TO BE OR NOT TO BE

- R = 1
- N = 1
- E = 2
- B = 2
- T = 3
- O = 4
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

32 bits
Huffman Code Properties

- Prefix code
  - No code is a prefix of another code
  - Example
    - Huffman("dog") \(\rightarrow 01\)
    - 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

No code is prefix of another

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1110</td>
</tr>
<tr>
<td>R</td>
<td>1111</td>
</tr>
<tr>
<td>E</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
</tr>
<tr>
<td>O</td>
<td>10</td>
</tr>
<tr>
<td>T</td>
<td>00</td>
</tr>
</tbody>
</table>

DECODING: Your turn

\[11110011000 = ?\]

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

Encoding Information: There’s more!

- Bits and bytes encode the information, but that’s not all
- Tags encode format and some structure in word processors
- Tags encode format and some structure in HTML
- Tags are one form of meta-data
- Meta-data is information about information

- We will return to this when we talk about HTML and the WEB later.