Week 3: More Processing, Bits and Bytes. Blinky, Logic Gates, Digital Representations, Huffman Coding

Announcements

- 3 homeworks this week:
  - 2 processing homeworks, one easy (Thurs), one harder (Tues).
  - Privacy essay due on Saturday at 5 PM.
  - Thursday Blinky Stars (only 2 points but start this week because otherwise you might not get Making Blinky Work (the other 8 points) to work before next Tuesday.
  - Questions should be posted on Piazza. Check before posting whether it was already answered. Not a huge volume on Piazza so easy to check. Respect the TAs.
  - Questions about grading: ask Gabby.

Next Processing Homework: DUE THURS!

- Write your own code from scratch, not just change existing code.
- But lots of hints and instructions in the homework PDF.

Robot Homework. Due before class today.

More Announcements

- Guest Lecture Thursday: Lifelogging & Social Media
- Prof. Whittaker’s Home Page
- Testing on invited talks:
  - No homework on them but possible clicker questions or midterm questions that demonstrate
    • That you were there
    • That you were paying attention
    • That you understood the presentation
    • Come prepared to ask questions!

Step 2. We take each “square” region in the Blinky picture shown as covering 10x10 pixels on the screen – so it’s four pixels wide, of course. The easiest way to draw the ghost is to draw vertical strips each of 10 pixels wide. So, we begin at some arbitrary position on the canvas, say, 50, 70, and begin drawing: the spot corresponds to the yellow arrow marked in the figure. Recall that this position is 50 pixels across and 70 pixels down from the upper left corner. Then we use the rectangle drawing function

```python
rect 50, 70, 10, 90: //From left, bar 1
So, the next bar must be at 60 pixels over (the last one was at 50 pixels over, and the bar we drew was 10 wide), and it must be 40 pixels down (the last one was 70 down, but the second bar needs to start 30 pixels higher up, right?). It will be 10 wide, of course, and how long? It’s longer than the last one; it was already 90 long – by an additional 30 at the top and 10 at the bottom. So the new one must be 130 pixels total. See the figure.
```

After typing in the command, check the drawing to make sure it is
A General Idea

- Digital Information: Detecting the presence or absence of a phenomenon at a specific place and time: PandA
- Phenomena: light, magnetism, charge, mass, color, current, ...
- Detecting depends on phenomenon – but the result must be discrete: it was detected or not; there is no option for "sorta there"

McCulloch & Pitts Neuron

- Input = (W1 * X1) + (W2 * X2) + ... + (Wn * Xn)
- Output = If Input > Θ Threshold then 1, otherwise 0

Fundamental units of computers: Logic Gates

Just like the neurons, but built out of transistors. Let’s look at what these do. THEIR BEHAVIOR IS PRECISELY DEFINED.

Truth Table for And (using True and False)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P and Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Truth Table for And (using 0 and 1)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P and Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth Table for OR (using 0 and 1)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P OR Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Can get complex! Cascades of these things.

Truth Table for Not And (using 0 and 1)

- Not And = NAND

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P AND Q</th>
<th>NOT (P AND Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(NOT P) OR (NOT Q) vs. NOT (P AND Q)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>NOT P</th>
<th>NOT Q</th>
<th>P AND Q</th>
<th>NOT (P AND Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- NOT (P ∧ Q) = NOT P ∨ NOT Q
- This is DeMorgan’s Law of Boolean Algebra
- Boolean Algebra is “binary algebra”
- We showed the Law just with truth tables!
Fundamental units of computers: Logic Gates

Exclusive-OR == XOR

Consider two propositions, either of which may be true or false

Exclusive-or is the relationship between them when JUST ONE OF THEM is true.

It EXCLUDES the case when both are true, so exclusive-or of the two is...

False when both are true or both are false, and true in the other two cases.

Truth Table for XOR

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P XOR Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What would you ever want XOR for anyway?

- [http://en.wikipedia.org/wiki/Adder_(electronics)]
- Binary Addition!

What is binary addition? Let's clicker this logic out!
First: let’s talk about binary numbers

How do we represent the number 2?

Clickers

A. 01
B. 11
C. 10
D. 110

Clickers: Which one represents different numbers?

A. 01 vs. 001
B. 11 vs. 110
C. 10 vs. 000010
D. 110 vs. 0110

Positional Notation

- Binary numbers, like decimal numbers, use place notation
  
  \[ 1101 = 1 \times 1000 + 1 \times 100 + 0 \times 10 + 1 \times 1 \]
  
  \[ = 1 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0 \]

  except that the base is 2 not 10

  \[ 1101 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \]
  
  \[ = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]

  **1101 in binary is 13 in decimal**

Binary combinations: The possibilities

- One bit
  - 0
  - 1

- Three bits
  - 000
  - 001
  - 010
  - 011

- Two Bits
  - 00
  - 01
  - 10
  - 11

- Four bits
  - 0000
  - 0001
  - 0010
  - 0011
  - 0100
  - 0101
  - 0110
  - 0111
  - 1000
  - 1001
  - 1010
  - 1011
  - 1100
  - 1101
  - 1110
  - 1111

© 2010 Larry Snyder, CSE
With 8 places how many different letters?

A. 32  
B. 64  
C. 128  
D. 256  
E. 512

With 8 places how high can you count?

A. 32  
B. 64  
C. 128  
D. 256  
E. 512

More Positional Notation

- Binary numbers, like decimal numbers, use place notation
  \[ 1101 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \]
  \[ = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]

- Clicker: 1111 in binary is WHAT in decimal?
  \[ 1111 = 1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 \]
  \[ = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]
  A. 4  
  B. 11  
  C. 15  
  D. 8

So what is binary addition? We did it a minute ago

Binary Addition

\[
\begin{array}{c|cccc|}
  & A & B & S & C \\
\hline
  0 & 0 & 0 & 1 & 0 \\
  0 & 1 & 1 & 0 & 0 \\
  1 & 0 & 0 & 1 & 0 \\
  1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

\[001101 + 010111\]
What is the answer?

001111
+110111

A. 0000110
B. 1000110
C. 1000111
D. 1000101
E. 0000101

Remember: Numbers, Letters, Colors all get represented with 0’s and 1’s. That is what digital representation means!!

Bits & Bytes

- P and A is a binary representation because it uses 2 patterns
- Bit -- it’s a contraction for “binary digit”
- a position in space/time capable of being set and detected in 2 patterns

Bytes: Standard encodings of meaning

- A byte is eight bits treated as a unit
  - Adopted by IBM in 1960s
  - A standard measure ever since
- Bytes encode the Latin alphabet using ASCII -- the American Standard Code for Information Interchange

ASCII

With 8 places how many different letters?

A. 32
B. 64
C. 128
D. 256
E. 512
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.

Why have a variable width encoding?

- Uniform Transformation Format: a variable-width encoding that can represent every character in the Unicode Character set
- 1,112,064 of them!!!
- 1 million would require how many bits?
- Hint: $2^{10} = 1024$
- $1024 \times 1024 = 2^{10} \times 2^{10} = 1,048,576$

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

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

Huffman Coding:

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

Who was David Huffman?

David A. Huffman

David Albert Huffman was a pioneer in computer science, known for Huffman coding. He was born on July 9, 1925, in Canton, Ohio. He received a B.S. in mathematics from the University of California at Santa Cruz in 1948. After serving in the U.S. Army from 1948 to 1950, he returned to the University of California at Santa Cruz, where he earned his Ph.D. in 1953. He then joined the faculty at MIT in 1953 and was promoted to professor in 1962. He remained at MIT until 1967. He then moved to California to become the founding faculty member of the Information Sciences Department at the new campus of the University of California at Santa Cruz. He retired in 1984, but remained active as an emeritus professor.
He founded the CS department here at UCSC

He taught CMPS 10. My teacher in this room!

- You are his academic grandchildren!

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

How easy is it to do it?

- Depends on data
  - Random data ⇒ hard
    - Example: 10011110100 ⇒ ?
  - 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 1/2 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

HEX 10D 10E 10F 110 111 110 111 110 111
HEX 10D 10E 10F 110 111 110 111 110 111
HEX 10D 10E 10F 110 111 110 111 110 111
Huffman Code Example

- “dog cat 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>

<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>

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 Tree Construction 1

- | Symbol | Dog | Cat | Bird | Fish |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>

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 – 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 in tree
    - Which L, R have the lowest number of occurrences in the message?
  - Create new (internal) node
    - Left child ➞ L
    - Right child ➞ R
  - New frequency = frequency( L ) + frequency( R )
  - Repeat until all nodes merged into one tree
Huffman Tree Step 2: can first re-order by frequency

Huffman Tree Construction 3

Huffman Tree Construction 4

Huffman Tree Construction 5

Huffman Coding Example

Huffman Code Algorithm Overview

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

- Input
  - ACE
- Output
  - (111)(10)(01) = 1111001

- Decoding
  - Read compressed file & binary tree
  - Use binary tree to decode file
  - Follow path from root to leaf
Huffman Decoding 7

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. Will do another example next Tuesday.
Get Blinky to move his eyes. Thurs
Do your privacy essay. Sat
Making Blinky move. Next Tuesday.