Welcome to 6.004!

L1: Basics of Information
- Course overview
- Quantifying information
- Fixed-length encodings
- Variable-length encodings
- Error detection and correction

Today’s handouts:
- Lecture slides
- Course calendar

I thought this course was called “Computation Structures”

New, experimental 6.004:
6.004 Course Clickables

https://6004.mit.edu/

- Announcements, corrections, etc.
- On-line copies of all handouts
- MITx course & tutorial problems for each lecture
- Current status, access to assignments
- Course information, staff, policies, etc.
- On-line Q&A

Where to find machines and help:

Combination Lock:___________
6.004: from atoms to Amazon

- Parallelism & communication
- Virtual Memory
- Operating System
- Interpretation & Compilation
- Data and Control structures
- Programmable architectures
- FSMs + Datapaths
- Digital Circuits
- Devices
- Materials
- Atoms

Cloud
Virtual Machines
Programming languages
Instruction set + memory
Bits, Logic gates
Lumped component model
Insulator, conductor, semiconductor
The Power of Engineering Abstractions

Good abstractions allow us to reason about behavior while shielding us from the details of the implementation.

Corollary: implementation technologies can evolve while preserving the engineering investment at higher levels.

Leads to hierarchical design:
- Limited complexity at each level ⇒ shorten design time, easier to verify
- Reusable building blocks

Cloud
Virtual Machines
Programming languages
Instruction set + memory
Bits, Logic gates
Lumped component model
Insulator, conductor, semiconductor
What is “Information”? 

Information, n. Data communicated or received that resolves uncertainty about a particular fact or circumstance.

Example: you receive some data about a card drawn at random from a 52-card deck. Which of the following data conveys the most information? The least?

- A. The card is a heart
- B. The card is not the Ace of spades
- C. The card is a face card (J, Q, K)
- D. The card is the “suicide king”
Quantifying Information  
(Claude Shannon, 1948)

Given discrete random variable X
- N possible values: $x_1, x_2, \ldots, x_N$
- Associated probabilities: $p_1, p_2, \ldots, p_N$

Information received when learning that choice was $x_i$:

$$I(x_i) = \log_2 \left( \frac{1}{p_i} \right)$$

$1/p_i$ is proportional to the uncertainty of choice $x_i$. 

Information is measured in bits (binary digits) = number of 0/1’s required to encode choice(s)
Information Conveyed by Data

Even when data doesn’t resolve all the uncertainty

\[ I(\text{data}) = \log_2 \left( \frac{1}{p_{\text{data}}} \right) \]

e.g., \( I(\text{heart}) = \log_2 \left( \frac{1}{13/52} \right) = 2 \) bits

Common case: Suppose you’re faced with \( N \) equally probable choices, and you receive data that narrows it down to \( M \) choices. The probability that data would be sent is \( M \cdot (1/N) \) so the amount of information you have received is

\[ I(\text{data}) = \log_2 \left( \frac{1}{M \cdot (1/N)} \right) = \log_2 \left( \frac{N}{M} \right) \text{ bits} \]
Example: Information Content

Examples:

- Information in one coin flip:
  \[ N= 2 \quad M= 1 \quad \text{Info content}= \log_2(2/1) = 1 \text{ bit} \]

- Card drawn from fresh deck is a heart:
  \[ N= 52 \quad M= 13 \quad \text{Info content}= \log_2(52/13) = 2 \text{ bits} \]

- Roll of 2 dice:
  \[ N= 36 \quad M= 1 \quad \text{Info content}= \log_2(36/1) = 5.17 \]

\[ .17 \text{ bits ???} \]
Probability & Information Content

<table>
<thead>
<tr>
<th>data</th>
<th>(p_{\text{data}})</th>
<th>(\log_2(1/p_{\text{data}}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>a heart</td>
<td>(13/52)</td>
<td>2 bits</td>
</tr>
<tr>
<td>not the Ace of spades</td>
<td>(51/52)</td>
<td>0.028 bits</td>
</tr>
<tr>
<td>a face card (J, Q, K)</td>
<td>(12/52)</td>
<td>2.115 bits</td>
</tr>
<tr>
<td>the “suicide king”</td>
<td>(1/52)</td>
<td>5.7 bits</td>
</tr>
</tbody>
</table>

Shannon’s definition for information content lines up nicely with my intuition: I get more information when the data resolves more uncertainty about the randomly selected card.
Entropy

In information theory, the entropy $H(X)$ is the average amount of information contained in each piece of data received about the value of $X$:

$$H(X) = E(I(X)) = \sum_{i=1}^{N} p_i \cdot \log_2 \left( \frac{1}{p_i} \right)$$

**Example:** $X=\{A, B, C, D\}$

<table>
<thead>
<tr>
<th>choice$_i$</th>
<th>$p_i$</th>
<th>$\log_2(1/p_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>“A”</td>
<td>1/3</td>
<td>1.58 bits</td>
</tr>
<tr>
<td>“B”</td>
<td>1/2</td>
<td>1 bit</td>
</tr>
<tr>
<td>“C”</td>
<td>1/12</td>
<td>3.58 bits</td>
</tr>
<tr>
<td>“D”</td>
<td>1/12</td>
<td>3.58 bits</td>
</tr>
</tbody>
</table>

$H(X) = (1/3)(1.58) + (1/2)(1) + 2(1/12)(3.58) = 1.626$ bits
Meaning of Entropy

Suppose we have a data sequence describing the values of the random variable X.

Average number of bits used to transmit choice

- Urk, this isn’t good 😞
- This is perfect!
- This is okay, just inefficient
An **encoding** is an *unambiguous* mapping between bit strings and the set of possible data.

<table>
<thead>
<tr>
<th>Encoding for each symbol</th>
<th>Encoding for “ABBA”</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**ABBA? ABC? ADA?**
Encodings as Binary Trees

It’s helpful to represent an unambiguous encoding as a binary tree with the symbols to be encoded as the leaves. The labels on the path from the root to the leaf give the encoding for that leaf.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Binary tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → 0</td>
<td></td>
</tr>
<tr>
<td>A → 11</td>
<td></td>
</tr>
<tr>
<td>C → 100</td>
<td></td>
</tr>
<tr>
<td>D → 101</td>
<td></td>
</tr>
</tbody>
</table>

01111

BAA
Fixed-length Encodings

If all choices are equally likely (or we have no reason to expect otherwise), then a fixed-length code is often used. Such a code will use at least enough bits to represent the information content.

All leaves have the same depth!

Note that the entropy for $N$ equally-probable symbols is

$$\sum_{i=1}^{N} \left( \frac{1}{N} \right) \log_2 \left( \frac{1}{1/N} \right) = \log_2(N)$$

Examples:

- 4-bit binary-coded decimal (BCD) digits $\log_2(10) = 3.322$
- 7-bit ASCII for printing characters $\log_2(94) = 6.555$
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number encoded in this fashion is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

\[ \begin{array}{cccccccccccc}
2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
\end{array} \]

\[ V = 0*2^{11} + 1*2^{10} + 1*2^9 + ... \]
\[ = 1024 + 512 + 256 + 128 + 64 + 16 \]
\[ = 2000 \]

Smallest number: 0  Largest number: \( 2^{N-1} \)
Hexademical Notation

Long strings of binary digits are tedious and error-prone to transcribe, so we usually use a higher-radix notation, choosing the radix so that it’s simple to recover the original bits string.

A popular choice is transcribe numbers in base-16, called hexadecimal, where each group of 4 adjacent bits are represented as a single hexadecimal digit.

<table>
<thead>
<tr>
<th>Hexadecimal - base 16</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 - 0</td>
<td>0000 - 0</td>
</tr>
<tr>
<td>0001 - 1</td>
<td>0001 - 1</td>
</tr>
<tr>
<td>0010 - 2</td>
<td>0010 - A</td>
</tr>
<tr>
<td>0011 - 3</td>
<td>0011 - B</td>
</tr>
<tr>
<td>0100 - 4</td>
<td>0100 - C</td>
</tr>
<tr>
<td>0101 - 5</td>
<td>0101 - D</td>
</tr>
<tr>
<td>0110 - 6</td>
<td>0110 - E</td>
</tr>
<tr>
<td>0111 - 7</td>
<td>0111 - F</td>
</tr>
</tbody>
</table>

\[ \begin{array}{cccccccccccc}
2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0
\end{array} \]

\[ 0b011111010000 = 0x7D0 \]
Encoding Signed Integers

We use a signed magnitude representation for decimal numbers, encoding the sign of the number (using “+” and “-”) separately from its magnitude (using decimal digits).

We could adopt that approach for binary representations:

```
1 1 1 1 1 1 1 0 1 0 0 0 0
```

“0” for “+”
“1” for “-”

Range: \(- (2^{N-1} - 1)\) to \(2^{N-1} - 1\)

-2000

But: two representations for 0 (+0, -0) and we’d need different circuitry for addition and subtraction
Two’s Complement Encoding

In a two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

- Negative numbers have “1” in the high-order bit
- Most negative number: 10...0000 \(-2^{N-1}\)
- Most positive number: 01...1111 \(+2^{N-1} - 1\)
- If all bits are 1: 11...1111 \(-1\)
- If all bits are 0: 00...0000 0

Range: \(-2^{N-1}\) to \(2^{N-1} - 1\)
More Two’s Complement

• Let’s see what happens when we add the N-bit values for -1 and 1, keeping an N-bit answer:

\[
\begin{array}{c}
11\ldots1111 \\
+00\ldots0001 \\
\hline
00000000
\end{array}
\]

Just use ordinary binary addition, even when one or both of the operands are negative. 2’s complement is perfect for N-bit arithmetic!

• To compute B-A, we’ll just use addition and compute B+(-A). But how do we figure out the representation for -A?

\[
A+(-A) = 0 = 1 + -1
\]

\[
-A = (-1 - A) + 1
\]

\[
= \sim A + 1
\]

To negate a two’s complement value: bitwise complement and add 1.
Variable-length Encodings

We’d like our encodings to use bits efficiently:

GOAL: When encoding data we’d like to match the length of the encoding to the information content of the data.

On a practical level this means:

• Higher probability $\rightarrow$ __________ encodings
• Lower probability $\rightarrow$ __________ encodings

Such encodings are termed variable-length encodings.
### Example

<table>
<thead>
<tr>
<th>choice_i</th>
<th>p_i</th>
<th>encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>“A”</td>
<td>1/3</td>
<td>11</td>
</tr>
<tr>
<td>“B”</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>“C”</td>
<td>1/12</td>
<td>100</td>
</tr>
<tr>
<td>“D”</td>
<td>1/12</td>
<td>101</td>
</tr>
</tbody>
</table>

Expected length of this encoding:

\[
(2)(1/3) + (1)(1/2) + (3)(1/12)(2) = 1.667 \text{ bits}
\]

Expected length for 1000 symbols:

- With fixed-length, 2 bits/symbol = \(\underline{2000}\) bits
- With variable-length code = \(\underline{1667}\) bits
- Lower bound (entropy) = \(\underline{1626}\) bits
Huffman’s Algorithm

Given a set of symbols and their probabilities, constructs an optimal variable-length encoding.

Huffman’s Algorithm:
- Build subtree using 2 symbols with lowest $p_i$
- At each step choose two symbols/subtrees with lowest $p_i$, combine to form new subtree
- Result: optimal tree built from the bottom-up

Example:

$$A = \frac{1}{3}, B = \frac{1}{2}, C = \frac{1}{12}, D = \frac{1}{12}$$

```
C  D
1/12  1/12
```

$$B$$

```
0 1
1/2
```

$$A$$

```
0 1
1/6 1/3
```

$$1/2$$

```
0 1
```

6.004 Computation Structures
Can We Do Better?

Huffman’s Algorithm constructed an optimal encoding... does that mean we can’t do better?

To get a more efficient encoding (closer to information content) we need to encode sequences of choices, not just each choice individually. This is the approach taken by most file compression algorithms...

\[
\begin{align*}
AA &= 1/9, & AB &= 1/6, & AC &= 1/36, & AD &= 1/36 \\
BA &= 1/6, & BB &= 1/4, & BC &= 1/24, & BD &= 1/24 \\
CA &= 1/36, & CB &= 1/24, & CC &= 1/144, & CD &= 1/144 \\
DA &= 1/36, & DB &= 1/24, & DC &= 1/144, & DD &= 1/144
\end{align*}
\]

Using Huffman’s Algorithm on pairs:
Average bits/symbol = 1.646 bits

Lookup “LZW” on Wikipedia
Error Detection and Correction

Suppose we wanted to reliably transmit the result of a single coin flip:

Heads: “0”
Tails: “1”

Further suppose that during processing a single-bit error occurs, i.e., a single “0” is turned into a “1” or a “1” is turned into a “0”.

“heads”

0

“tails”

1
Hamming Distance

HAMMING DISTANCE: The number of positions in which the corresponding digits differ in two encodings of the same length.

```
0110010
```

```
0100110
```

Differs in 2 positions so Hamming distance is 2…
The Hamming distance between a valid binary code word and the same code word with a single-bit error is 1.

The problem with our simple encoding is that the two valid code words ("0" and "1") also have a Hamming distance of 1. So a single-bit error changes a valid code word into another valid code word...
Single-bit Error Detection

What we need is an encoding where a single-bit error does *not* produce another valid code word.

A parity bit can be added to any length message and is chosen to make the total number of “1” bits even (aka “even parity”). If \( \text{min HD(code words)} = 1 \), then \( \text{min HD(code words + parity)} = 2 \).
Parity check = Detect Single-bit errors

• To check for a single-bit error (actually any odd number of errors), count the number of 1s in the received message and if it’s odd, there’s been an error.

0 1 1 0 0 1 0 1 0 0 1 1 \rightarrow \text{original word with parity}
0 1 1 0 0 0 0 1 0 0 1 1 \rightarrow \text{single-bit error (detected)}
0 1 1 0 0 0 1 1 0 0 1 1 \rightarrow 2\text{-bit error (not detected)}

• One can “count” by summing the bits in the word modulo 2 (which is equivalent to XOR’ing the bits together).
Detecting Multi-bit Errors

To detect $E$ errors, we need a minimum Hamming distance of $E+1$ between code words.

With this encoding, we can detect up to two bit errors. Note that $HD(000,111) = 3$...
By increasing the Hamming distance between valid code words to 3, we guarantee that the sets of words produced by single-bit errors don’t overlap. So assuming at most one error, we can perform **error correction** since we can tell what the valid code was before the error happened.

To **correct** \( E \) errors, we need a minimum Hamming distance of \( 2E+1 \) between code words.
Summary

- Information resolves uncertainty
- Choices equally probable:
  - $N$ choices down to $M \Rightarrow \log_2(N/M)$ bits of information
  - use fixed-length encodings
  - encoding numbers: 2’s complement signed integers
- Choices not equally probable:
  - choice $i$ with probability $p_i \Rightarrow \log_2(1/p_i)$ bits of information
  - average amount of information $= H(X) = \sum p_i \log_2(1/p_i)$
  - use variable-length encodings, Huffman’s algorithm
- To detect $E$-bit errors: Hamming distance $> E$
- To correct $E$-bit errors: Hamming distance $> 2E$

Next time:
- encoding information electrically
- the digital abstraction
- combinational devices