6.004 Computation Structures
Spring 2009

Welcome to 6.004!

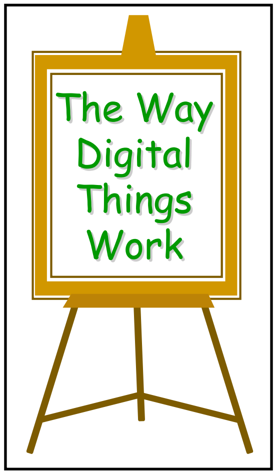


Figure by MIT OpenCourseWare.

I thought this course was called "Computation Structures"

Handouts: Lecture Slides, Calendar, *Info sheet*

Course Mechanics

Unlike other big courses, you'll have

- NO evening quizzes
- NO final exam
- NO weekly graded problem sets

Instead, you'll face

- Repository of tutorial problems (with answers)
- FIVE quizzes, based on these problems (in Friday sections)
- EIGHT labs + on-line lab questions + Design Contest (all labs and olqs must be completed to pass!)
- ALGORITHMIC assignment of your grade!

How do you build systems with >1G components?

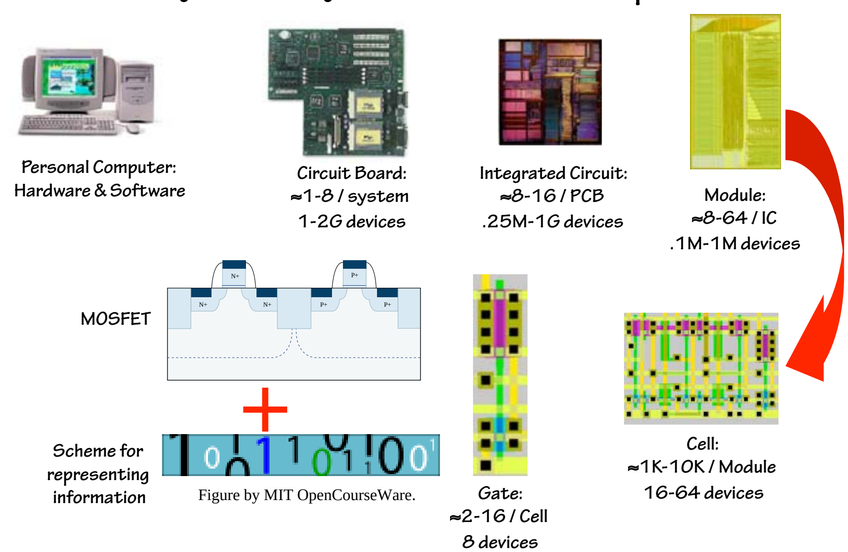


Figure by MIT OpenCourseWare.

What do we see?

- Structure
  - hierarchical design:
    - limited complexity at each level
    - reusable building blocks
- Interfaces
  - Key elements of system engineering; typically outlive the technologies they interface
  - Isolate technologies, allow evolution
  - Major abstraction mechanism
- What makes a good system design?
  - "Bang for the buck": minimal mechanism, maximal function
  - reliable in a wide range of environments
  - accommodates future technical improvements

Our plan of attack…

- Understand how things work, *bottom-up*
- Encapsulate our understanding using appropriate abstractions
- Study organizational principles: abstractions, interfaces, APIs.

- Roll up our sleeves and design at each level of hierarchy
- Learn engineering tricks
  - history
  - systematic approaches
  - algorithms
  - diagnose, fix, and avoid bugs

First up: INFORMATION

Whirlwind, MIT Lincoln Labs
http://www.chick.net/wizards/whirlwind.html

If we want to design devices to manipulate, communicate and store information then we need to quantify information so we can get a handle on the engineering issues. Goal:

Two photographs removed due to copyright restrictions.
Please see http://www.chick.net/wizards/images/whirlwind.jpg and http://www.chick.net/wizards/images/wwtea.gif.

good implementations

- Easy-to-use
- Efficient
- Reliable
- Secure
- …

- Low-level physical representations
- High-level symbols and sequences of symbols

What is "Information"?

information, *n.* Knowledge communicated or received *concerning a particular fact or circumstance.*

Tell me something new…

Information resolves uncertainty.
Information is simply that which cannot be predicted.

The less predictable a message is, the more information it conveys!

Quantifying Information

(Claude Shannon, 1948)

Suppose you're faced with N equally probable choices, and I give you a fact that narrows it down to M choices. Then I've given you

$\log_2(N/M)$ *bits* of information

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

Examples:

- information in one coin flip: $\log_2(2/1) = 1$ bit
- roll of 2 dice: $\log_2(36/1) = 5.2$ bits
- outcome of a Red Sox game: 1 bit

  (well, actually, are both outcomes equally probable?)

Encoding

- Encoding describes the process of *assigning representations to information*
- Choosing an appropriate and efficient encoding is a real engineering challenge
- Impacts design at many levels
  - Mechanism (devices, # of components used)
  - Efficiency (bits used)
  - Reliability (noise)
  - Security (encryption)

Next lecture: encoding a *bit.*

*What about **longer** messages?*

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.

ex. Decimal digits 10 = {0,1,2,3,4,5,6,7,8,9}

4-bit BCD (binary coded decimal)

$$\log_2(10) = 3.322 < 4\,bits$$

ex. ~86 English characters =

{A-Z (26), a-z (26), 0-9 (10), punctuation (11), math (9), financial (4)}

7-bit ASCII (American Standard Code for Information Interchange)

$$\log_2(86) = 6.426 < 7\,bits$$

Encoding numbers

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:

$$v = \sum_{i=0}^{n-1} 2^i b_i$$

$2^{11}\,2^{10}\,2^9\,2^8\,2^7\,2^6\,2^5\,2^4\,2^3\,2^2\,2^1\,2^0$

0 1 1 1 1 1 0 1 0 0 0 0 $= 2000_{10}$

7 d 0

Oftentimes we will find it convenient to cluster groups of bits together for a more compact notation. Two popular groupings are clusters of 3 bits and 4 bits.

03720

Octal - base 8

000 - 0	100 - 4
001 - 1	101 - 5
010 - 2	110 - 6
011 - 3	111 - 7

0x7d0

Hexadecimal - base 16

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f

Signed integers: 2's complement

N bits

-2^{N-1}	2^{N-2}	• • •	• • •	• • •	2^3	2^2	2^1	2^0

"sign bit"

Range: -2^{N-1} to $2^{N-1}-1$

"decimal" point

8-bit 2's complement example:

$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$

If we use a two's complement representation for signed integers, the same binary addition mod 2^n procedure will work for adding positive and negative numbers (don't need separate subtraction rules). The same procedure will also handle unsigned numbers!

By moving the implicit location of "decimal" point, we can represent fractions too:

$1101.0110 = -2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = -8 + 4 + 1 + 0.25 + 0.125 = -2.625$

When choices aren't equally probable

When the choices have different probabilities (p_i), you get more information when learning of a unlikely choice than when learning of a likely choice

Information from choice$_i$ = $\log_2(1/p_i)$ bits

Average information from a choice = $\Sigma p_i \log_2(1/p_i)$

Example

choice$_i$	*p_i*	*$\log_2(1/p_i)$*
"A"	1/3	1.58 bits
"B"	1/2	1 bit
"C"	1/12	3.58 bits
"D"	1/12	3.58 bits

Average information
= (.333)(1.58) + (.5)(1)
+ (2)(.083)(3.58)
= 1.626 bits

Can we find an encoding where transmitting 1000 choices is close to 1626 bits on the average? Using two bits for each choice = 2000 bits

Variable-length encodings

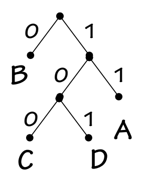
(David Huffman, MIT 1950)

Use shorter bit sequences for high probability choices, longer sequences for less probable choices

choice$_i$	*p_i*	*encoding*
"A"	1/3	11
"B"	1/2	0
"C"	1/12	100
"D"	1/12	101

B C A B A D

010011011101



Huffman Decoding Tree

Average information
= (.333)(2)+(.5)(1)+(2)(.083)(3)
= 1.666 bits

Transmitting 1000 choices takes an average of 1666 bits… better but not optimal

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…

Data Compression

Key: *re-encoding to remove redundant information: match data rate to actual information content.*

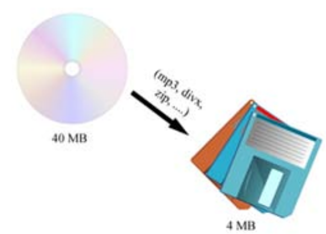


Figure by MIT OpenCourseWare.

"Outside of a dog, a book is man's best friend. Inside of a dog, its too dark to read…"

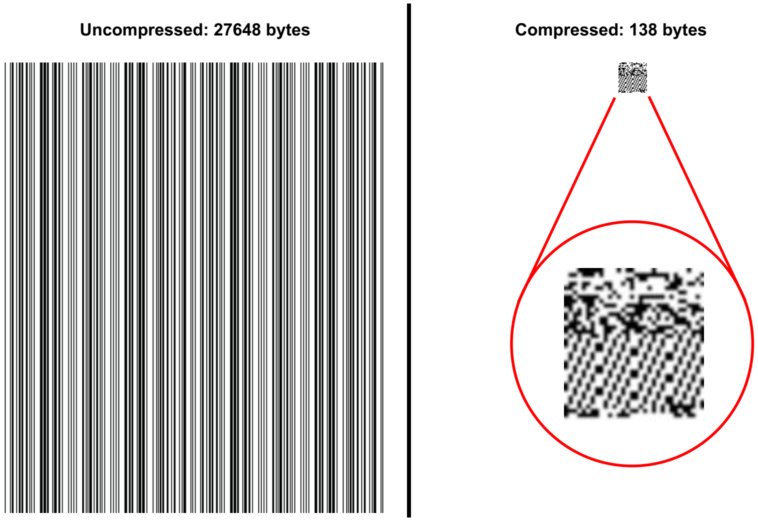
-Groucho Marx

A84b!*m9@+M(p

Ideal: No redundant info – Only unpredictable bits transmitted. *Result appears random!*

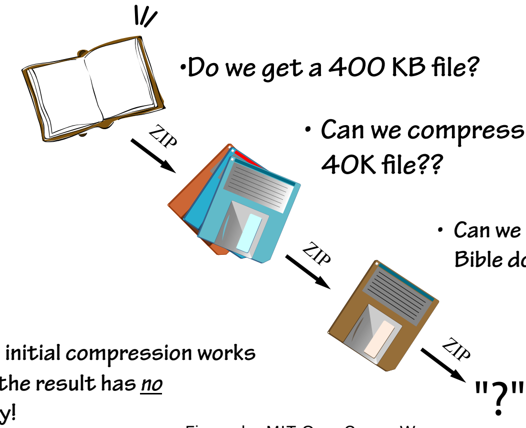
LOSSLESS: can 'uncompress', get back original.

"Able was I ere I saw Elba."*1024



Does *recompression* work?

If ZIP compression of a 40MB Bible yields a 4MB ZIP file, what happens if we compress *that?*



- Do we get a 400 KB file?
- Can we compress *that*, to get a 40K file??
- Can we compress the whole Bible down to a single bit???
- Is it a 0 or a 1????

HINT: if the initial compression works perfectly, the result has *no* redundancy!

Figure by MIT OpenCourseWare.

Is redundancy *always* bad?

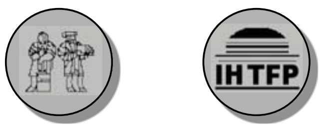
Encoding schemes that attempt to match the information content of a data stream are minimizing redundancy. They are *data compression* techniques.

However, sometimes the goal of encoding information is to *increase redundancy*, rather than remove it. Why?

- Make the information easy to manipulate (fixed-sized encodings)
- Make the data stream resilient to noise (error detecting and correcting codes)

Error detection and correction

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



Heads: "0" Tails: "1"

This is a prototype of the "bit" coin for the new information economy. Value = 12.5¢

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

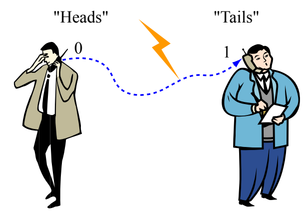


Figure by MIT OpenCourseWare.

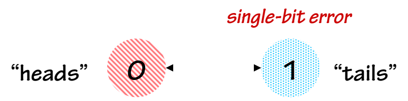
Hamming Distance

(Richard Hamming, 1950)

HAMMING DISTANCE: The number of digit positions in which the corresponding digits of two encodings of the same length are different

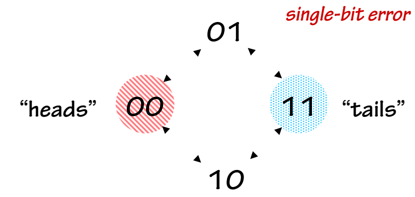
The Hamming distance between a valid binary code word and the same code word with single-bit error is 1.

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…



Error Detection

What we need is an encoding where a single-bit error doesn't produce another valid code word.

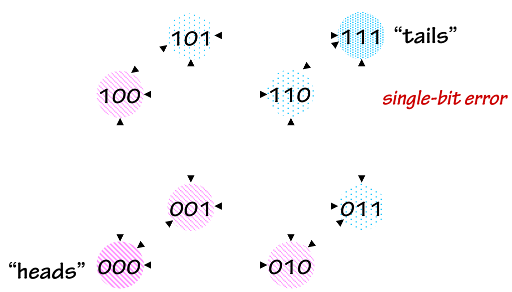


If D is the minimum Hamming distance between code words, we can detect up to (D-1)-bit errors

We can add single-bit error detection to any length code word by adding a *parity bit* chosen to guarantee the Hamming distance between any two valid code words is at least 2. In the diagram above, we're using "even parity" where the added bit is chosen to make the total number of 1's in the code word even.

Can we correct detected errors? Not yet…

Error Correction



If D is the minimum Hamming distance between code words, we can correct up to $\left\lfloor \frac{D-1}{2} \right\rfloor$ - bit errors

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 if we detect an error, we can perform *error correction* since we can tell what the valid code was before the error happened.

- Can we safely detect double-bit errors while correcting 1-bit errors?
- Do we always need to triple the number of bits?

The right choice of codes can solve hard problems

Reed-Solomon (1960)

First construct a polynomial from the data symbols to be transmitted and then send an over-sampled plot of the polynomial instead of the original symbols themselves – spread the information out so it can be recovered from a subset of the transmitted symbols.

Particularly good at correcting bursts of erasures (symbols known to be incorrect)

Used by CD, DVD, DAT, satellite broadcasts, etc.

Viterbi (1967)

A dynamic programming algorithm for finding the most likely sequence of hidden states that result in a sequence of observed events, especially in the context of hidden Markov models.

Good choice when soft-decision information is available from the demodulator.

Used by QAM modulation schemes (eg, CDMA, GSM, cable modems), disk drive electronics (PRML)

Summary

- Information resolves uncertainty
- Choices equally probable:
  - N choices down to M → $\log_2(N/M)$ bits of information
  - use fixed-length encodings
  - encoding numbers: 2's complement signed integers
- Choices not equally probable:
  - choice_i with probability p_i → $\log_2(1/p_i)$ bits of information
  - average number of bits = $\Sigma p_i \log_2(1/p_i)$
  - use variable-length encodings
- To detect D-bit errors: Hamming distance > D
- To correct D-bit errors: Hamming distance > 2D

Next time:

- encoding information electrically
- the digital abstraction
- combinational devices

Hand in Information Sheets!