Overview

What is COS 126? Broad, but technical, introduction to computer science.

Goals.
- Demystify computer systems.
- Empower you to exploit available technology.
- Build awareness of substantial intellectual underpinnings.

Topics.
- Programming in Java.
- Machine architecture.
- Theory of computation.
- Applications to science, engineering, and commercial computing.

"Computers are incredibly fast, accurate, and stupid; humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination." – Albert Einstein

The Basics

Lectures. [Sedgewick]

RS office hours. everyone needs to meet me!

Precepts. [Gabai · Moretti · August · Berkiten · Ghosh · Hug · Kiefer · Lee · Suleimenov · Wang]
- Tips on assignments / worked examples
- Questions on lecture material.
- Informal and interactive.

Friend 016/017(?) lab. [Ugrad assistants]
- Help with systems/debugging.
- No help with course material.

Reading period. No lectures; precepts T and W.

Grades

Course grades. No preset curve or quota.

9 programming assignments. 40%.
2 exams (in class, 10/13-14 and 12/15-16). 50%.
Final programming project (due Dean's date - 1). 10%.
Extra credit / staff discretion. Adjust borderline cases.

Due dates

See www.princeton.edu/~cos126 for full details and preceptor office hours.
Course Materials

Course website. [www.princeton.edu/~cos126]
- Submit assignments
- Programming assignments.
- Lecture slides.

Course text.
Sedgewick and Wayne.
Intro to Programming in Java: An Interdisciplinary Approach.

Recommended reading (lectures 18-20).
Harel. What computers can’t do.

Programming Assignments

Desiderata.
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!

Due. Mondays 11pm via Web submission.

Computing equipment.
- Your laptop. [OS X, Windows, Linux, iPhone, ... ]
- OIT desktop. [Friend 016 and 017(?) labs]

What’s Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.

Not registered? Go to any precept now; officially register ASAP.
Change precepts? Use SCORE.

Assignment 0. [www.princeton.edu/~cos126/assignments.php]
- Due Monday 11PM.
- Read Sections 1.1 and 1.2 in textbook.
- Install Java programming environment + a few exercises.
- Lots of help available, don’t be bashful.
0. Prologue: A Simple Machine

**Secure Chat with a One-Time Pad**

Alice wants to send a secret message to Bob
- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.

```plaintext
Encrypt SENDMONEY with yT25a5i/S
```

**Key point:** Without the pad, Eve cannot understand the message.

---

**Encryption Machine**

**Goal.** Design a machine to encrypt and decrypt data.

```
SENDMONEY
```

encrypt

```
gX76W3v7K
```

decrypt

```
SENDMONEY
```

**Enigma encryption machine.**
- "Unbreakable" German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.

---

**A Digital World**

**Data is a sequence of bits.** [bit = 0 or 1]--- can use decimal digits, letters, or some other system, but bits are more easily encoded physically ("on-off", "up-down", "hot-cold", ...)

- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, ...

---

Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com

Image courtesy of David August
A Digital World

Data is a sequence of bits. [bit = 0 or 1]
  • Text.
  • Programs, executables.
  • Documents, pictures, sounds, movies, ...

Ex. Base64 encoding of text.
  • Simple method for representing A-Z, a-z, 0-9, +, /
  • 6 bits to represent each symbol (64 symbols)

<table>
<thead>
<tr>
<th>binary</th>
<th>char</th>
<th>dec</th>
<th>Base64 Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>000001</td>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>001111</td>
<td>S</td>
<td>12</td>
<td>001100</td>
</tr>
</tbody>
</table>

One-Time Pad Encryption

Encryption.
  • Convert text message to N bits.

<table>
<thead>
<tr>
<th>message</th>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>S E N D M O N E Y</td>
<td>010010 001000 001101 001100 001100 001101 000100 011000</td>
</tr>
</tbody>
</table>

Secure Chat with a One-Time Pad

First challenge: Create a one-time pad.

Good choice: A random sequence of bits (stay tuned).

Note: any sequence of bits can be encoded as characters

<table>
<thead>
<tr>
<th>one-time pad</th>
<th>encoded as characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>y T 2 5 a 5 i / S</td>
<td>110010 010011 110110 111001 011010 110010 111011 010110</td>
</tr>
</tbody>
</table>

One-Time Pad Encryption

Encryption.
  • Convert text message to N bits.
  • Use N random bits as one-time pad.

<table>
<thead>
<tr>
<th>message base64</th>
<th>one-time pad</th>
</tr>
</thead>
<tbody>
<tr>
<td>S E N D M O N E Y</td>
<td>010010 001000 001101 001100 001100 001101 000100 011000</td>
</tr>
<tr>
<td>y T 2 5 a 5 i / S</td>
<td>110010 010011 110110 111001 011010 110010 111011 010110</td>
</tr>
</tbody>
</table>
One-Time Pad Encryption

Encryption.
• Convert text message to N bits.
• Use N random bits as one-time pad.
• Take bitwise XOR of two bitstrings.

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>000100</td>
<td>011000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111011</td>
<td>010010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>011111</td>
<td>111011</td>
<td>110110</td>
<td>010111</td>
<td>101111</td>
<td>001100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XOR Truth Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

sum corresponding pair of bits: 1 if sum is odd, 0 if even

Secure Chat with a One-Time Pad

Alice wants to send a secret message to Bob
• Sometime in the past, they exchange a one-time pad.
• Alice uses the pad to encrypt the message.

Secure Chat 1.0 [alice]
[alice]: Hey, Bob
[bob]: Hi, Alice!
[alice]: SENDMONEY

Encrypt SENDMONEY with yT25a5i/S

Key point: Without the pad, Eve cannot understand the message.
But how can Bob understand the message?

Secure Chat 1.0 [bob]
[alice]: Hey, Bob
[bob]: Hi, Alice!
[alice]: gX76W3v7K

Secure Chat 1.0 [alice]
[alice]: Hey, Bob
[bob]: Hi, Alice!
[alice]: SENDMONEY

Encrypt SENDMONEY with yT25a5i/S

Key point: Without the pad, Eve cannot understand the message.

One-Time Pad Encryption

Encryption.
• Convert text message to N bits.
• Use N random bits as one-time pad.
• Take bitwise XOR of two bitstrings.

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>000100</td>
<td>011000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111011</td>
<td>010010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>011111</td>
<td>111011</td>
<td>110110</td>
<td>010111</td>
<td>101111</td>
<td>001100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XOR

g x 7 6 w 3 v 7 K

Secure Chat with a One-Time Pad

Alice wants to send a secret message to Bob
• Sometime in the past, they exchange a one-time pad.
• Alice uses the pad to encrypt the message.
• Bob uses the same pad to decrypt the message.

Secure Chat 1.0 [alice]
[alice]: Hey, Bob
[bob]: Hi, Alice!
[alice]: SENDMONEY

Encrypt SENDMONEY with yT25a5i/S

Secure Chat 1.0 [bob]
[alice]: Hey, Bob
[bob]: Hi, Alice!
[alice]: gX76W3v7K

SENDMONEY

Secure Chat 1.0 [alice]
[alice]: Hey, Bob
[bob]: Hi, Alice!
[alice]: SENDMONEY

Encrypt SENDMONEY with yT25a5i/S

Key point: Without the pad, Eve cannot understand the message.
One-Time Pad Decryption

Decryption.
• Convert encrypted message to binary.

\[ \begin{array}{cccccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K \\ 100000 & 010111 & 111011 & 111010 & 010110 & 101111 & 101111 & 101111 & 001010 \\
110010 & 010011 & 111010 & 111010 & 010110 & 111011 & 101111 & 110111 & 100010 \\
y & T & 2 & 5 & a & 5 & i & / & S \\
\end{array} \]

Encrypted: \[ gX76W3v7K \]
Base64: \[ 100000 010111 111011 111010 010110 101111 101111 101111 001010 \]
One-time pad: 100000 010111 111011 111010 010110 111011 101111 110111 100010

One-Time Pad Decryption

Decryption.
• Convert encrypted message to binary.
• Use same N random bits (one-time pad).

\[ \begin{array}{cccccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K \\
100000 & 010111 & 111011 & 111010 & 010110 & 101111 & 101111 & 101111 & 001010 \\
110010 & 010011 & 111010 & 111010 & 010110 & 111011 & 101111 & 110111 & 100010 \\
y & T & 2 & 5 & a & 5 & i & / & S \\
\end{array} \]

Encrypted: \[ gX76W3v7K \]
Base64: \[ 100000 010111 111011 111010 010110 101111 101111 101111 001010 \]
One-time pad: 100000 010111 111011 111010 010110 111011 101111 110111 100010
XOR Truth Table
\[ \begin{array}{ccc}
x & y & x \oplus y \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array} \]

\[ 1 \oplus 1 = 0 \]
One-Time Pad Decryption

Decryption:
• Convert encrypted message to binary.
• Use same N random bits (one-time pad).
• Take bitwise XOR of two bitstrings.
• Convert back into text.

---

Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>W</td>
<td>12</td>
<td>001100</td>
</tr>
</tbody>
</table>

---

Why Does It Work?

Crucial property.
Decrypted message = original message.

Why is crucial property true?
• Use properties of XOR.
  • \((a \oplus b) \oplus b = a \oplus (b \oplus b) = a \oplus 0 = a\)

---

One-Time Pad Decryption (with the wrong pad)

Decryption:
• Convert encrypted message to binary.

---

XOR Truth Table

\[
\begin{array}{c|c|c}
    x & y & x \oplus y \\
    \hline
    0 & 0 & 0 \\
    0 & 1 & 1 \\
    1 & 0 & 1 \\
    1 & 1 & 0 \\
\end{array}
\]
One-Time Pad Decryption (with the wrong pad)

Decryption.
• Convert encrypted message to binary.
• Use wrong N bits (bogus one-time pad).
• Take bitwise XOR of two bitstrings.
• Convert back into text: Oops.

Decrypted:

\[
\begin{array}{cccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K \\
100000 & 011111 & 111010 & 011101 & 011110 & 111011 & 111011 & 011010 \\
101000 & 011010 & 111011 & 101111 & 101010 & 101010 & 011010 \\
\end{array}
\]

Eve's Problem (one-time pads)

Key point: Without the pad, Eve cannot understand the message.

But Eve has a computer. Why not try all possible pads?

One problem: it might take a long time [stay tuned].

Worse problem: she would see all possible messages!

• 54 bits
• \(2^{54}\) possible messages, all different.
• \(2^{54}\) possible encoded messages, all different.
• No way for Eve to distinguish real message from any other message.

One-time pad is "provably secure".

Wrong message [usually gibberish]
**Goods and Bads of One-Time Pads**

*Good.*
- Easily computed by hand.
- Very simple encryption/decryption processes.
- Provably unbreakable if bits are truly random. [Shannon, 1940s]

*Bad.*
- Easily breakable if pad is re-used.
- Pad must be as long as the message.
- Truly random bits are very hard to come by.
- Pad must be distributed securely.

*one time* means one time only

impractical for Web commerce

a Russian one-time pad

**Pseudo-Random Bit Generator**

*Practical middle-ground.*
- Make a "random" bit generator gadget.
- Alice and Bob each get identical small gadgets
  [same gadget works for both]
- Alice and Bob also each get identical books of small seeds.

"one seed 92 to decrypt the contents of this DVD"

instead of identical large one-time pads

**Shift Register**

*Shift register terminology.*
- Bit: 0 or 1.
- Cell: storage element that holds one bit.
- Register: sequence of cells.
- Seed: initial sequence of bits.
- Shift register: when clock ticks, bits propagate one position to left.

0 1 1 0 1 0 0 0 0 1 0
time t

0 1 1 0 1 0 0 0 0 1 0
time \( t+1 \)

register

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

— Jon von Neumann (left)
— ENIAC (right)
Linear Feedback Shift Register (LFSR)

(8, 10) linear feedback shift register.
- Shift register with 11 cells.
- Bit $b_0$ is is XOR of previous bits $b_8$ and $b_{10}$.
- Pseudo-random bit = $b_0$.

Encryption.
- Convert text message to $N$ bits.
- Initialize LFSR with given seed
- Generate $N$ bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

LFSR Encryption

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>w</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

message

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>001100</td>
<td>001101</td>
<td>001111</td>
<td>001110</td>
<td>000100</td>
<td>011000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

base64

| 110010 | 010011 | 110110 | 111001 | 011010 | 110011 | 100010 | 111111 | 010010 |

LFSR bits

| 100000 | 011011 | 111110 | 111100 | 011010 | 110111 | 101111 | 111101 | 001010 |

XOR

<table>
<thead>
<tr>
<th>g</th>
<th>x</th>
<th>7</th>
<th>6</th>
<th>w</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>k</th>
</tr>
</thead>
</table>

encrypted

| 100000 | 011111 | 111110 | 111100 | 011010 | 110111 | 101111 | 111101 | 001010 |

LFSR bits

| 110010 | 010011 | 110110 | 111001 | 011010 | 110101 | 100010 | 111111 | 010010 |

XOR

| 001010 | 001010 | 001101 | 001111 | 001100 | 001110 | 001101 | 001010 | 110100 |

message

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>110010</td>
<td>011011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>110111</td>
<td>101111</td>
<td>111101</td>
<td>001010</td>
</tr>
</tbody>
</table>
Eve’s Problem (LFSR encryption/decryption)

Key point: Without the (short) seed
Eve cannot understand the (long) message.

But Eve has a computer. Why not try all possible seeds?
• Seeds are short, messages are long.
• All seeds give a tiny fraction of all messages.
• Extremely likely that all but real seed will produce gibberish.

Bad news (for Eve): There are still too many possibilities!
• Ex: 60-bit seed implies $2^{60}$ possibilities.
• If Eve could check 1 million seeds per second, it would take her 365 centuries to try them all!

Exponential growth dwarfs technological improvements [stay tuned].
• 1000 bits: $2^{1000}$ possibilities.
• Age of the universe in microseconds: $2^{70}$

Goods and Bads of LFSRs

Good.
• Easily computed with simple machine.
• Very simple encryption/decryption processes.
• Bits have many of the same properties as random bits.
• Scalable: 20 bits for 1 million bits; 30 cells for 1 billion bits.
  [ but need theory of finite groups to know where to put taps ]

Bad.
• Still need secure, independent way to distribute LFSR seed.
• The bits are not truly random.
  [ bits in our 11-bit LFSR cycle after $2^{11} - 1 = 2047$ steps ]
• Experts have cracked LFSR encryption.
  [ need more complicated machines ]

Other LFSR Applications

What else can we do with a LFSR?
• DVD encryption with CSS.
• DVD decryption with DeCSS!
• Subroutine in military cryptosystems.

Typical Exam Question (TEQ) on LFSRs

Give first 10 steps of (3, 4) LFSR with initial fill 00001.
Goal. Decrypt/encrypt 300 characters (1800 bits).

Challenge. Is it a good idea to use an 11-bit LFSR?

A. Yes, no problem.

B. No, the bits it produces are not truly random.

C. No, need a longer LFSR.

D. No, experts have cracked LFSRs

A Profound Idea

Programming. Can write a Java program to simulate the operations of any abstract machine.

• Basis for theoretical understanding of computation. [stay tuned]
• Basis for bootstrapping real machines into existence. [stay tuned]

Stay tuned. See Assignment 5.

public class LFSR
{
    private int[] seed;
    private int tap;
    private int N;

    public LFSR(String seed, int tap) {...}

    public int step() {...}

    public static void main(String[] args)
    {
        LFSR lfsr = new LFSR("01101000010", 8);
        for (int i = 0; i < 2000; i++)
        { StdOut.println(lfsr.step());
        }
    }
}

A Profound Question

Q. What is a random number?

LFSR does not produce random numbers.

• It is a very simple deterministic machine.
• Not obvious how to distinguish the bits it produces from random.
• Experts have figured out how to do so.

Q. Are random processes found in nature?

• Motion of cosmic rays or subatomic particles?
• Mutations in DNA?

Q. Is the natural world a (not-so-simple) deterministic machine?

" God does not play dice. "
− Albert Einstein