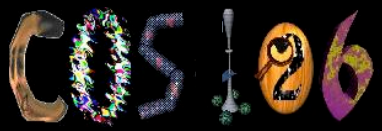


General Computer Science
Princeton University
Spring 2008

Robert Sedgewick

Overview

What is COS 126?

- Broad, but technical, intro to CS.
  No prerequisites, intended for novices.

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.

The Basics

Lectures.

- Tuesdays and Thursdays, Frist 302.
- Same lecture at 10 and 11am.

Precepts.

- Tue+Thu or Wed+Fri.
- Tips on assignments, worked examples, clarify lecture material.

Friend 016/017 lab. [Undergrad lab assistants]

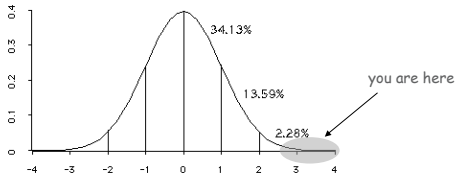
- Weekdays 7-11pm, some weekend hours.
- Full schedule on Web.

For full details: See `www.princeton.edu/~cos126`

Grades

Course grades. No preset curve or quota.

8 programming assignments. 40%.
2 exams. 50%. ← can drop lowest one
Final programming project. 10%.
Extra credit and staff discretion. Adjust borderline cases.
← participation helps, frequent absences hurts



Course Materials

Course website. [www.princeton.edu/~cos126]
- Submit assignments, check grades.
- Programming assignments.
- Lecture notes.

print slides before lecture; annotate during lecture

skim before lecture; read thoroughly afterwards

Required readings. Sedgewick and Wayne. *Intro to Programming in Java: An Interdisciplinary Approach.* [Labyrinth Books]

Recommended readings. Harel. *What computers can't do.* [Labyrinth]

Programming Assignments

Desiderata.
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.

Examples.
- N-body simulation.
- Pluck a guitar string.
- DNA sequence alignment.
- Estimate Avogadro's number.

} you solve scientific problems from scratch!

Due. Mondays 11:00pm 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.
Precept 2. Meets Thu/Fri.

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

see Donna O'Leary in CS 410 if the only precept you can attend is closed

Assignment 0. Due Monday, 11:00pm.
- Read Sections 1.1 and 1.2 in textbook.
- Install Java programming environment + a few exercises.
- Lots of help available, don't be bashful.

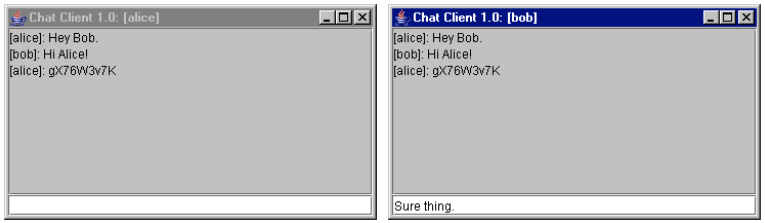
END OF ADMINISTRATIVE STUFF

0. Prologue: A Simple Machine

Secure Chat

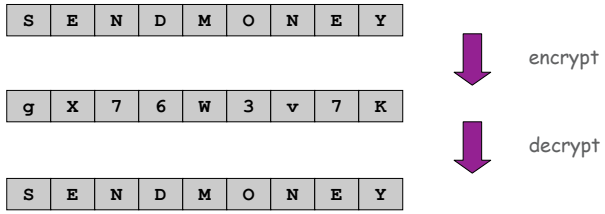
Alice wants to send a secret message to Bob?

- Can you read the secret message `gX76W3v7K` ?
- But Bob can. How?



Encryption Machine

Goal. Design a machine to encrypt and decrypt data.



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. ← 0 or 1

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

File formats. txt, pdf, doc, ppt, jpeg, mp3, divx, java, exe, …

computer with a lens

computer with earbuds

computer with a radio

A Digital World

Data is a sequence of bits. ← 0 or 1

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

Base64 encoding. Use 6 bits to represent each alphanumeric symbol.

Binary	Char	Binary	Char	Binary	Char	Binary	Char	Binary	Char	Binary	Char
000000	A	001011	L	010110	W	100001	h	101100	s	110111	3
000001	B	001100	M	010111	X	100010	i	101101	t	111000	4
000010	C	001101	N	011000	Y	100011	j	101110	u	111001	5
000011	D	001110	O	011001	Z	100100	k	101111	v	111010	6
000100	E	001111	P	011010	a	100101	l	110000	w	111011	7
000101	F	010000	Q	011011	b	100110	m	110001	x	111100	8
000110	G	010001	R	011100	c	100111	n	110010	y	111101	9
000111	H	010010	S	011101	d	101000	o	110011	z	111110	+
001000	I	010011	T	011110	e	101001	p	110100	0	111111	/
001001	J	010100	U	011111	f	101010	q	110101	1		
001010	K	010101	V	100000	g	101011	r	110110	2		

One-Time Pad Encryption

Encryption.

- Convert text message to N bits.

0 or 1

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
...	...	...
Y	24	011000
...	...	...

S	E	N	D	M	O	N	E	Y	
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64

message

One-Time Pad Encryption

Encryption.

- Convert text message to N bits.
- Generate N random bits (one-time pad).

S	E	N	D	M	O	N	E	Y	
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits

message

One-Time Pad Encryption

Encryption.

- Convert text message to N bits.
- Generate N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

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

XOR Truth Table

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

S	E	N	D	M	O	N	E	Y	
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR

message

0 ^ 1 = 1

One-Time Pad Encryption

Encryption.

- Convert text message to N bits.
- Generate N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
...	...	...
X	23	010111
...	...	...

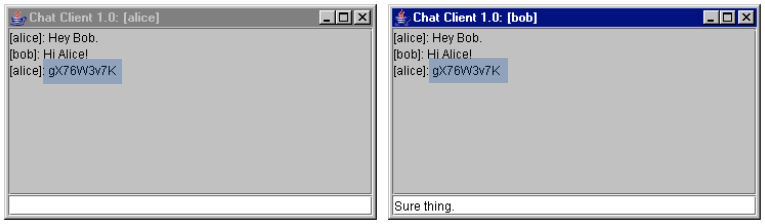
S	E	N	D	M	O	N	E	Y	
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR
g	X	7	6	W	3	v	7	K	encrypted

message

Secure Chat

Alice wants to send a secret message to Bob?

- Can you read the secret message `gX76W3v7K` ?
- But Bob can. How?



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

char	dec	binary
A	0	000000
B	1	000001
...	...	...
Y	24	011000
...	...	...

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	random bits
010010	000100	001101	000011	001100	001110	001101	000100	011000	XOR
S	**E**	**N**	**D**	**M**	**O**	**N**	**E**	**Y**	message

Why Does It Work?

Crucial property. Decrypted message = original message.

Notation	Meaning
a	original message bit
b	one-time pad bit
^	XOR operator
a ^ b	encrypted message bit
(a ^ b) ^ b	decrypted message bit

Why is crucial property true?

- Use properties of XOR.
- (a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a

associativity of ^ | always 0 | identity

XOR Truth Table

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

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.

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101000	011100	110101	101111	010010	111001	100101	101010	001010	wrong bits
001000	001011	001110	010101	000100	001110	001010	010001	000000	XOR
I	**L**	**O**	**V**	**E**	**O**	**K**	**R**	**A**	wrong message

Goods and Bads of One-Time Pads

Good.

- Very simple encryption/decryption processes.
- Provably unbreakable if pad is truly random. [Shannon, 1940s]

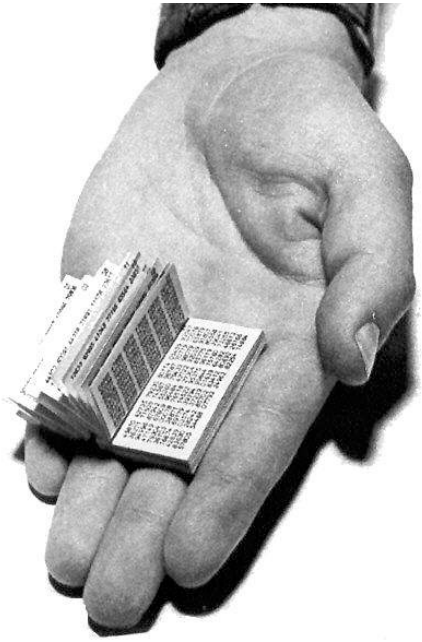
Bad.

"one time" means one time only

- 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.

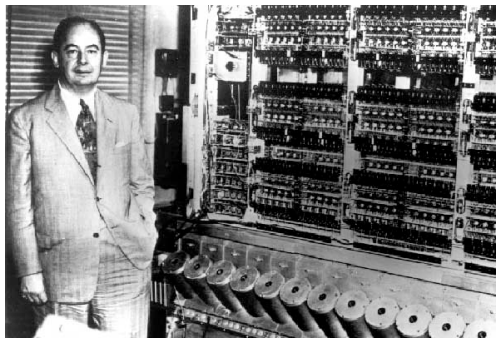
impractical for Web commerce

A Russian One-Time Pad



Random Numbers

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



Jon von Neumann (left), ENIAC (right)

Pseudo-Random Bit Generator

Practical middle-ground.

- Let's make a pseudo-random bit generator gadget.
- Alice and Bob each get identical small gadgets.

instead of identical large one-time pads

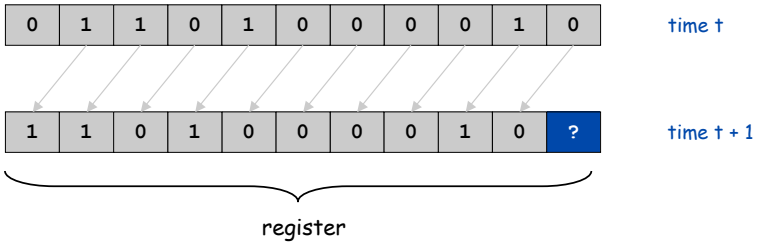
How to make small gadget that produces "random" numbers.

- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- ...

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.

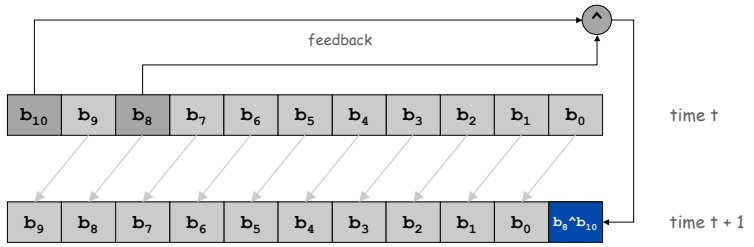


Linear Feedback Shift Register

{8, 10} linear feedback shift register.

- 11 bit shift register.
- New output bit 0 is XOR of previous bits 8 and 10.
- Output bit = bit 0.

LFSR demo



Random Numbers

Q. Are these 2000 numbers random? If not, what is the pattern?

```
11001001001111011011100101101011100110001011111101001000010011010010111100110010011111
10111000001010110001000011101010011010000111100100110011101111111010100001000010001010
01010100011000001011110001001001101101101111000110100110111001111010111100100001001110101
01110100000101001000100011010101011100000001011000001001110001011101101001010110011000
11111110011000001111110001100001101111001110100111010011100100111011101010101001000
00000001000000001010000001000100001010101001000000011010000011100100011011101011010100
01010000101000100100010101101010000110000100111100101110011100101111011001001010111011
00001010111001000010111010010010011011000111101110110010101011110000001001100001101111
10010010001110110101101011000110001110111101101010010110000110011100111111011110000101
01100100011111101011000010001110010101101110000110101100111000111110110100010110111010
01101010011110000111001100101111111110100000001001000001011010001001100101011111100001
00001100101001111100011000110110110111011011010101101100000110111000111010110110100011
01100101111011111001010100111000001110110001101011101110001010101101000000110010000111110
10011000100111110101110001000101101010100110000001111100001100011001111011111100101000
11100010011010101111011000100101110101100101000111100010110011010011111100111000011110
11001100101111111100100000011101000010011001011101111111010100100010000001010100000
10000010010100010110001010011101000111010010110100110011001111111111100000000110000000
11110000011001100011111111011000000101110000100101100101100111100111110011100111100011110011
01100111101111100010100011010001011100101001011100011001011011111001101000111100101100
00111001110110111010110100100011001101011111100010000011010100011100001011011001001100
11110111101001010010011000110111101101000101010010100000110001000111101010110010000001
11101000110010010111101100100010111010100100100001101101001110110001110100111101011111010001000
01001010101011000000001110000001101100001110111001101010111110000010001100010101111010
```

A. No. This is output of an 11 bit LFSR!

The Science Behind A LFSR

Q. Are the bits really random?
A. No! Real machines are deterministic.

Q. Will bit pattern repeat itself?
A. Yes, after $2^{11} - 1 = 2047$ steps.

Q. What if I need more bits?
A. Scalable: 20 cells for 1 million bits, 30 for 1 billion.

Q. Will the machine work equally well if we XOR bits 4 and 10?
A. No! Need to understand theory of finite groups.

Q. How many cells do I need to guarantee a certain level of security?
A. Subject of active research.

Other LFSR Applications

What else can we do with a LFSR?

- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

```
/*    efdtt.c     Author:  Charles M. Hannum <root@ihack.net>              */
/*    Usage is:  cat title-key scrambled.vob | efdtt >clear.vob            */

#define m(i)(x[i]^s[i+84])<<

                    unsigned char x[5]      ,y,s[2048];main(
                    n){for( read(0,x,5     );read(0,s ,n=2048
                           ); write(1   ,s,n)          )if(s
                   [y=s      [13]%8+20]  /16%4   ==1      ){int
                   i=m(      1)17  ^256 +m(0)    8,k      =m(2)
                   0,j=      m(4)   17^ m(3)    9^k*     2-k%8
                   ^8,a     =0,c    =26;for    (s[y]    -=16;
                  --c;j  *=2)a=     a*2^i&     1,i=i /2^j&1
                  <<24;for(j=        127;      ++j<n;c=c>
                                      y)
                                      c

                           +=y=i^i/8^i>>4^i>>12,
                     i=i>>8^y<<17,a^=a>>14,y=a^a*8^a<<6,a=a
                  >>8^y<<9,k=s[j],k          ="7Wo~'G_\216"[k
                   &7]+2^"cr3sfw6v;*k+>/n."[k>>4]*2^k*257/
                        8,s[j]=k^(k&k*2&34)*6^c+~y
                                    ;})
```

http://www.cs.cmu.edu/~dst/DeCSS/Gallery

LFSR and "General Purpose Computer"

Important properties.

- Built from simple components.
- Scales to handle huge problems.
- Requires a deep understanding to use effectively.

Basic Component	LFSR	Computer
control	start, stop, load	same
clock	regular pulse	2.8 GHz pulse
memory	11 bits	1 GB
input	seed	sequence of bits
computation	shift, XOR	logic, arithmetic, …
output	pseudo-random bits	Sequence of bits

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.

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]

Ex. This program will make sense to you in two weeks.

```
public class LFSR {
   public static void main(String[] args) {
      boolean[] a = { false, true, false, false, false, false, true, false, true, true, false };
      int TAP = 8;                          // tap position
      int T = Integer.parseInt(args[0]);    // number of steps

      // simulate T steps of the LFSR
      for (int t = 0; t < T; t++) {
         boolean next = (a[N-1] ^ a[TAP]);      // compute next bit
         for (int i = a.length - 1; i > 0; i--)
            a[i] = a[i-1];                      // shift one position
         a[0] = next;                           // put next bit on right end
         if (next) System.out.print("1");
         else      System.out.print("0");
      }
      System.out.println();
   }
}
```

```
% java LFSR 2000
11001001001111011011100101101
01110011000101111110100100001
00110100101111001100100111...
```