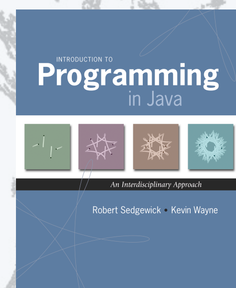
Computer Science 126

General Computer Science

Fall 2014

Robert Sedgewick

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E



http://introcs.cs.princeton.edu

Prologue: A Simple Machine

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E

Prologue: A Simple Machine

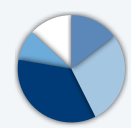
- **Brief introduction**
- Secure communication with a one-time pad
- Linear feedback shift registers
- Implications

CS.0.A.Prologue.Introduction

Who are you?

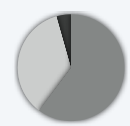
[data from 2011-12]

Intended major



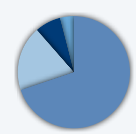
- Social Sciences
- other Science/Math
- other Engineering
- Humanities
- CS

Programming experience



- none
- some
- lots

Class

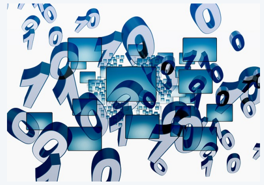


- 1st year
- Sophomore
- Junior
- Senior

Over 60% of all Princeton students take COS 126

What is this course about?

A broad introduction to computer science.



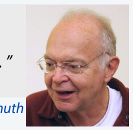
Goals
- Demystify computer systems.
- Empower you to exploit available technology.
- Build awareness of substantial intellectual underpinnings.

Topics
- Programming in Java.
- Design and architecture of computers.
- Theory of computation.
- Applications in science and engineering.

and art, music, finance, and many other fields.

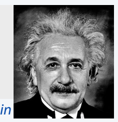
"Science is everything we understand well enough to explain to a computer."

— Don Knuth



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

— Albert Einstein



The basics

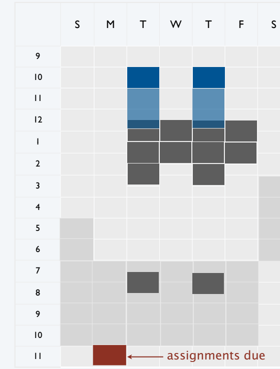
Lectures. [Sedgewick]

RS office hours. ← everyone needs to meet me!

Precepts. [Gabai, Ginsburg and team]
- Tips on assignments / worked examples
- Questions on lecture material.
- Informal and interactive.

Friend 016/017 lab. [undergraduate assistants]
- Help with systems/debugging.
- No help with course material.

Piazza. [online discussion]
- Best chance of quick response to a question.
- Post to class or private post to staff.



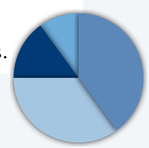
See www.princeton.edu/~cos126 for full current details and office hours.

Grades

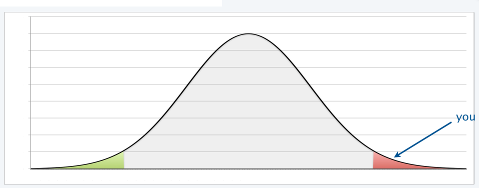
are based on achievement.

Opportunities for us to determine your level of achievement:
- 9 programming assignments.
- 2 written exams (in class, 10/9 and 12/11).
- 2 programming exams (evenings, 10/23 and 12/8).
- Final programming project (due Dean's date – 1).
- Extra credit / staff discretion. Adjust borderline cases.

participation helps
frequent absence hurts



We do not grade on a "curve".

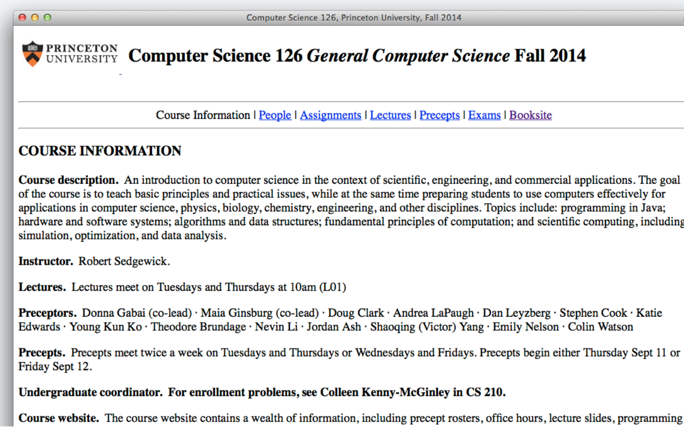


Due dates

	Su	Mo	Tu	We	Th	Fr	Sa
SEP		1	2	3	4	5	6
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30				
OCT				1	2	3	4
	5	6	7	8	9	10	11
	12	13	14	15	16	17	18
	19	20	21	22	23	24	25
	26	27	28	29	30	31	
NOV							1
	2	3	4	5	6	7	8
	9	10	11	12	13	14	15
	16	17	18	19	20	21	22
	23	24	25	26	27	28	29
	30						
DEC		1	2	3	4	5	6
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30	31			
JAN					1	2	3
	4	5	6	7	8	9	10
	11	12	13	14	15	16	17
	18	19	20	21	22	23	24
	25	26	27	28	29	30	31

Course website

http://www.princeton.edu/~cos126 ← bookmark this page!



Computer Science 126, Princeton University, Fall 2014

PRINCETON UNIVERSITY **Computer Science 126 *General Computer Science* Fall 2014**

Course Information | People | Assignments | Lectures | Precepts | Exams | Booksite

COURSE INFORMATION

Course description. An introduction to computer science in the context of scientific, engineering, and commercial applications. The goal of the course is to teach basic principles and practical issues, while at the same time preparing students to use computers effectively for applications in computer science, physics, biology, chemistry, engineering, and other disciplines. Topics include: programming in Java; hardware and software systems; algorithms and data structures; fundamental principles of computation; and scientific computing, including simulation, optimization, and data analysis.

Instructor. Robert Sedgewick.

Lectures. Lectures meet on Tuesdays and Thursdays at 10am (L01)

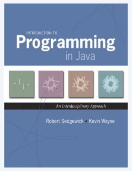
Preceptors. Donna Gabai (co-lead) · Maia Ginsburg (co-lead) · Doug Clark · Andrea LaPaugh · Dan Leyzberg · Stephen Cook · Katie Edwards · Young Kun Ko · Theodore Brundage · Nevin Li · Jordan Ash · Shaoqing (Victor) Yang · Emily Nelson · Colin Watson

Precepts. Precepts meet twice a week on Tuesdays and Thursdays or Wednesdays and Fridays. Precepts begin either Thursday Sept 11 or Friday Sept 12.

Undergraduate coordinator. For enrollment problems, see Colleen Kenny-McGinley in CS 210.

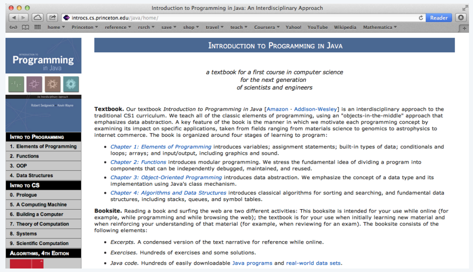
Course website. The course website contains a wealth of information, including precept rosters, office hours, lecture slides, programming

Textbook and Booksite



Textbook.
- Full introduction to course material.
- Developed for this course.
- For use while learning and studying.

Booksite.
- Summary of content.
- Code, exercises, examples.
- Supplementary material.
- NOT the textbook.
- (also not the course web page).
- For use while online.

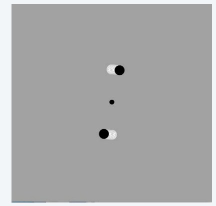


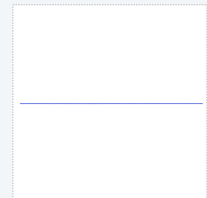
`http://introcs.cs.princeton.edu` ← bookmark this page, too!

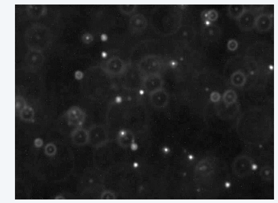
Programming assignments

are an essential part of the experience in learning CS.

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


N-body simulation


pluck a guitar string


estimate Avogadro's number

What's Ahead?

Coming events
- Lecture 2. Basic programming concepts.
- Precept 1. Meets today/tomorrow.
- Not registered? Go to any precept now; officially register ASAP.
- Change precepts? Use SCORE. ← see Colleen Kenny-McGinley in CS 210 if the only precept you can attend is closed

→ Assignment 0 due Monday 11:59PM ←

Things to do before attempting assignment
- Read Sections 1.1 and 1.2 in textbook.
- Read assignment carefully.
- Install `introcs` software as per instructions.
- Do a few exercises.
- Lots of help available, don't be bashful.

`http://introcs.cs.princeton.edu/assignments.php`

END OF ADMINISTRATIVE STUFF

COMPUTER SCIENCE
SEDGEWICK / WAYNE

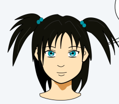
1. Prologue: A Simple Machine

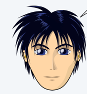
- Brief introduction
- **Secure communication with a one-time pad**
- Linear feedback shift registers
- Implications

CS.1.B.Prologue.OneTimePad

Sending a secret message with a cryptographic key

Alice wants to send a secret message to Bob.
- Sometime in the past, they exchange a cryptographic key.
- Alice uses the key to encrypt the message.
- Bob uses the *same* key to decrypt the message.





encrypted message is "in the clear" (anyone can read it)

Critical point: Without the key, Eve cannot understand the message.

Q. How does the system work?

Encrypt/decrypt methods

Goal. Design a method to encrypt and decrypt data.

S	E	N	D	M	O	N	E	Y

↓ encrypt

g	X	7	6	W	3	v	7	K

↓ decrypt

S	E	N	D	M	O	N	E	Y

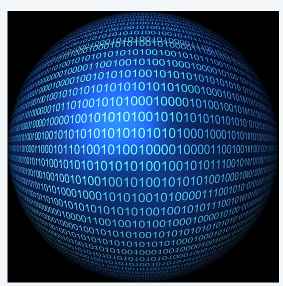
Example 1. Enigma encryption machine [German code, WWII]
- Broken by Turing bombe (one of the first uses of a computer).
- Broken code helped win Battle of Atlantic by providing U-boat locations.

Example 2. One-time pad [details to follow]

Example 3. Linear feedback shift register [later this lecture]

A digital world

A *bit* is a basic unit of information.
- Two possible values (0 or 1).
- Easy to represent in the physical world (*on* or *off*).



In modern computing and communications systems, we represent *everything* as a sequence of bits.

- Text [details to follow in this lecture]
- Numbers
- Sound [details to follow in this course]
- Pictures [details to follow in this course]
- ...
- Programs [profound implications, stay tuned].

0 1 0 0 0 1 0 1

$01000101_2 = 69_{10}$

Bottom line. If we can send and receive bits, we can send and receive *anything*.

Encoding text as a sequence of bits

Base64 encoding of character strings
- A simple method for representing text.
- 64 different symbols allowed: A-Z, a-z, 0-9, +, /.
- 6 bits to represent each symbol.
- ASCII and Unicode methods used on your computer are similar.

	bits	symbols
Base64	6	64
ASCII	8	256
Unicode	16	65,536+

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

Example:

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

SENDMONEY → 010010000100001101000011001100001110001101000100011000

One-Time Pads

What is a one-time pad?
- A *cryptographic key* known only to the sender and receiver.
- Good choice: A *random* sequence of bits (stay tuned).
- Security depends on each sequence being used only once.

y T 2 5 a 5 i / S

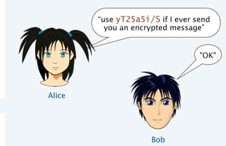
110010|010011|110110|111001|011010|111001|100010|111111|010010 → y T 2 5 a 5 i / S

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

Note: Any sequence of bits can be decoded into a sequence of characters. ← more convenient than bits for initial exchange

Encryption with a one-time pad

Preparation
- Create a "random" sequence of bits (a one-time pad).
- Send one-time pad to intended recipient through a secure channel.



Encryption
- Encode text as a sequence of *N* bits.
- Use the first *N* bits of the pad. ← important point: need to have as many bits in the pad as there are in the message.
- Compute a new sequence of *N* bits from the message and the pad.
- Decode result to get a sequence of characters.

Result: A ciphertext (encrypted message).

message	S E N D M O N E Y	→	010010000100001101000011001100001110001101000100011000
one-time pad	y T 2 5 a 5 i / S	→	110010010011110110111001011010111001100010111111010010
ciphertext	g X 7 6 W 3 v 7 K	←	100000010111111011111010010110110111101111111011001010

a simple machine

A (very) simple machine for encryption

To compute a cyphertext from a message and a one-time pad
- Encode message and pad in binary.
- Each cyphertext bit is the *bitwise exclusive or* of corresponding bits in message and pad.

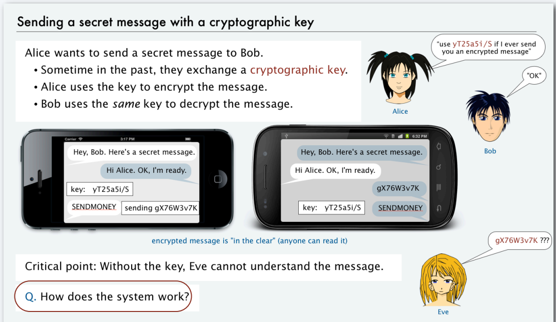
Def. The bitwise exclusive or of two bits is 1 if they differ, 0 if they are the same.

		S E N D M O N E Y
S E N D M O N E Y	message	010010|000100|001101|000011|001100|001110|001101|000100|011000
y T 2 5 a 5 i / S	one-time pad	110010|010011|110110|111001|011010|111001|100010|111111|010010
		y T 2 5 a 5 i / S
		XOR
g X 7 6 W 3 v 7 K	cyphertext	100000|010111|111011|111010|010110|110111|101111|111011|001010
		g X 7 6 W 3 v 7 K

Self-assessment on bitwise XOR encryption

Q. Encrypt the message E A S Y with the pad 0 1 2 3.

Decryption with a one-time pad



A. Alice's device uses a "bitwise exclusive or" machine to encrypt the message.

Q. What kind of machine does Bob's device use to *decrypt* the message?

A. The same one (!!)

A (very) simple machine for encryption and decryption

To compute a *message* from a *cyphertext* and a one-time pad
- Use binary encoding of cyphertext and pad.
- Each message bit is the *bitwise exclusive or* of corresponding bits in cyphertext and pad.

← 1 if they differ; 0 if they are the same

		g	X	7	6	W	3	v	7	K
g X 7 6 W 3 v 7 K	cyphertext	100000	010111	111011	111010	010110	110111	101111	111011	001010
y T 2 5 a 5 i / S	one-time pad	110010	010011	110110	111001	011010	111001	100010	111111	010010
		y	T	2	5	a	5	i	/	S

XOR ↓

S E N D M O N E Y	message (!)	010010	000100	001101	000011	001100	001110	001101	000100	011000
		S	E	N	D	M	O	N	E	Y

Why does it work?

		S	E	N	D	M	O	N	E	Y
message	S E N D M O N E Y	010010	000100	001101	000011	001100	001110	001101	000100	011000
one-time pad	y T 2 5 a 5 i / S	110010	010011	110110	111001	011010	111001	100010	111111	010010
ciphertext	g X 7 6 W 3 v 7 K	100000	010111	111011	111010	010110	110111	101111	111011	001010
one-time pad	y T 2 5 a 5 i / S	110010	010011	110110	111001	011010	111001	100010	111111	010010
message	S E N D M O N E Y	010010	000100	001101	000011	001100	001110	001101	000100	011000
		S	E	N	D	M	O	N	E	Y

XOR (message, one-time pad) → ciphertext; XOR (ciphertext, one-time pad) → message

Crucial property: Decrypted message is the same as the original message.
Let m be a bit of the message and k be the corresponding bit of the one-time pad.
To prove: $(m \wedge k) \wedge k = m$ ← Notation: $m \wedge k$ is equivalent to XOR(m, k)

Approach 1: Truth tables

m	k	$m \wedge k$	$(m \wedge k) \wedge k$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

✓

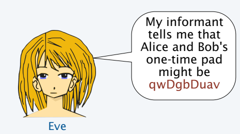
Approach 2: Boolean algebra

$$(k \wedge k) = 0$$
$$m \wedge 0 = m$$
$$(m \wedge k) \wedge k = m \wedge (k \wedge k) = m \wedge 0 = m$$

✓

Decryption with the wrong pad

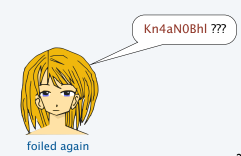
Eve *cannot* read a message without knowing the pad.



		g	X	7	6	W	3	v	7	K
ciphertext	g X 7 6 W 3 v 7 K	100000	010111	111011	111010	010110	110111	101111	111011	001010
wrong pad	q w D g b D u a v	101010	110000	000011	100000	011011	000011	101110	011010	101111
gibberish	K n 4 a N 0 B h l	001010	100111	111000	011010	001101	110100	000001	100001	100101
		K	n	4	a	N	0	B	h	l

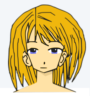
XOR (ciphertext, wrong pad) → gibberish

One-time pad is provably secure [Shannon, 1940s]
- IF each pad is used only once,
- AND the pad bits are random,
- THEN Eve cannot distinguish cyphertext from random bits.

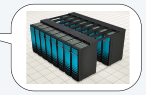


Eve's problem with one-time pads

Eve has a computer. Why not try all possibilities?



Eve



Problem
- 54 bits, so there are 2^{54} possible pad values.
- Suppose Eve could check a million values per second.
- It would still take 570+ years to check all possibilities.

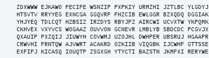
Much worse problem
- There are also 2^{54} possible messages.
- If Eve were to check all the pads, she'd see all the *messages*.
- No way to distinguish the real one from any other.

pad value	*message?*
AAAAAAAAA	gX76W3v7K
AAAAAAAAB	gX76W3v7L
AAAAAAAAC	gX76W3v7I
...	
qwDgbDuav	Kn4aNOBhl
...	
tTtpWk+1E	NEWTATTOO
...	
yT25a5i/S	SENDMONEY
...	
////////+	fo7FpIQE0
/////////	fo7FpIQE1

One-time pad is provably secure.

Goods and bads of one-time pads

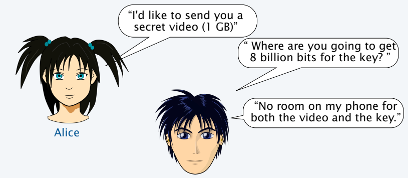
Goods.
- Very simple encryption method.
- Decrypt with the same method.
- Provably unbreakable if bits are truly random.
- Widely used in practice.



a one-time pad

cold war hotline

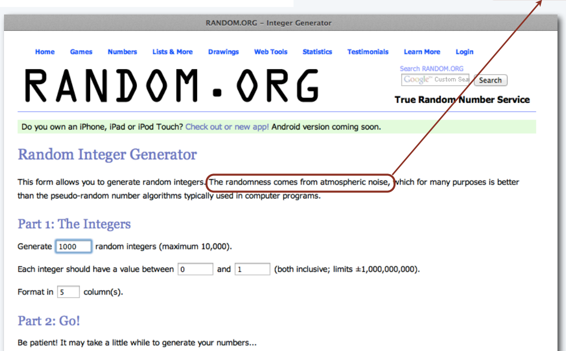
Bads.
- Easily breakable if seed is re-used.
- ➡ Truly random bits are very hard to come by.
- Need separate secure channel to distribute key.
- ➡ Pad must be as long as the message.



Random bits are not so easy to find

You might look on the internet.

The randomness comes from atmospheric noise



... if you trust the internet.

Next: Creating a (long) sequence of "pseudo-random" bits from a (short) key.

COMPUTER SCIENCE
SEDGEWICK / WAYNE

Prologue: A Simple Machine

- Brief introduction
- Secure communication with a one-time pad
- **Linear feedback shift registers**
- Implications

CS.1.C.Prologue.LFSR

A pseudo-random number generator

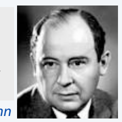
is a *deterministic* machine that produces a long sequence of *pseudo random* bits.

Examples

Enigma.
Linear feedback shift register (next).
Blum-Blum-Shub generator.
...
[an early application of computing]
[research still ongoing]

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

– John von Neumann



A pseudo-random number generator

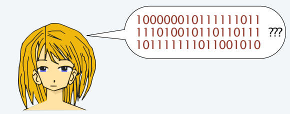
is a *deterministic* machine that produces a long sequence of *pseudo random* bits.

Deterministic: Given the current state of the machine, we know the next bit.

An absolute requirement: Alice and Bob need the same sequence.

Random: We never know the next bit.

Pseudo-random: The sequence of bits *appears to be* random.



Appears to be random??

- A profound and elusive concept.
- For this lecture: "Has enough properties of a random sequence that Eve can't tell the difference".

Ex. 1: No long repeats
Ex. 2: About the same number of 0s and 1s
Ex. 3: About the same number of 00s, 01s, 10s, and 11s.
...

Which of these sequences appear to be random?

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	✗	
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	✗	but # of 0s and 1s are about equal
0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 0 0	✗	but # of 00s 01s 10s and 11s are about equal
0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0	✗	S E N D M O N E Y
1 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0	✓	key for Alice and Bob
1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 1 0	✓	ciphertext for SENDMONEY
1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0	✓	generated by coin flips
1 0 0 0 1 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 1 0 1 0	✗	typed arbitrarily (no long seqs of 0s or 1s)

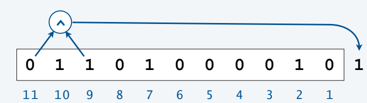
Note: Any one of them *could be* random!

Linear feedback shift register

Terminology

- Bit: 0 or 1.
- Cell: storage element that holds one bit.
- Register: sequence of cells.
- Seed: initial sequence of bits.
- Feedback: Compute XOR of two bits and put result at right.
- Shift register: when clock ticks, bits propagate one position to left.

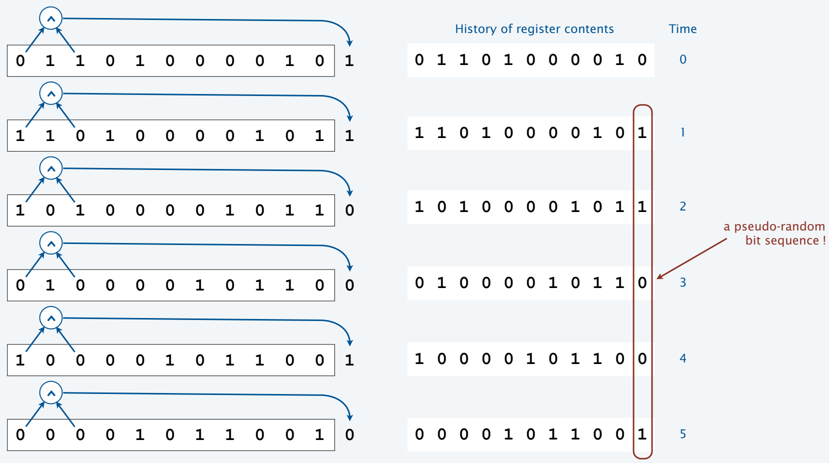
An [11, 9] LFSR



More terminology

- Tap: Bit positions used for XOR (one must be leftmost). ← Numbered from right, starting at 1.
- [*N*, *k*] LFSR: *N*-bit register with taps at *N* and *k*. ← Not all values of k give desired effect (stay tuned).

Linear feedback shift register simulation



A random bit sequence?

Q. Is this a random sequence?

No long repeats.
997 0s, 1003 1s.
256 00s, 254 01s, 256 10s, 257 11s.
...

one-time pad in our example

11001001001111011011100101101011100110001011111101001000010011010010111100110010011111110111000000101
0110001000011101010011010000111100100110011101111111010100000100001000101001010100011000001011110001
001001101011011110001101001101110011110101111001000100111010101110100000101001000100011010101010111000
0000101100000100111000101110110100101011001100001111111001100000111111000110000110111100111010011110
1001110010011101110111010101010100000000000100000000010100000001000100001010101001000000011010000011100
1000110111010111010100010100001010001001000101011010100001100001001111001011100111001011110111001001
0101110110000101011100100001011101001001010011011000111101110110010101011110000000100110000101111001
0010001110110101101011000110001110111101101010010110000110011100111111011110000101001100100011111101
0110000100011100101011011100001101011001110001111101101100010110111010011010100111100001110011001101
1111111010000000010010000010110100010011001010111111000010000110010100111110001110001101101101110110
1101010110110000011011100011101011011010001101100101110111100101010011100000111011000110101110111000
1010101101000000110010000111110100110001001111101011100010001011010101001100000011111000011000110011
1101111110010100001110001001101101011110110001001011101011001010001111000101100110100111111001110000
1111011001100101111111100100000011101000011010010011100110111011111010101000100000010101000010000010
0101000101100010100111010001110100101101001100110011111111111000000000110000000111100000110011000111
11111011000000010111000010010110010110011110011111001111000111100110110011111011111000101000110100010
1110010100101110001100101101111100110100011111001011000111001110110111101011010010001100110101111111
00010000001101010001110000101101100100110111101111010010100100110001101111101110100010101001010000011
0001000111101010110010000011110100011001001011111011001000101111010100100100001101101001110110011101
0111110100010001001010101011000000000111000000110110000111011100110101011111000001000110001010111010

A. No. It is the output of an [11, 9] LFSR with seed 01101000010!

It is *pseudo-random* (at least to some observers).

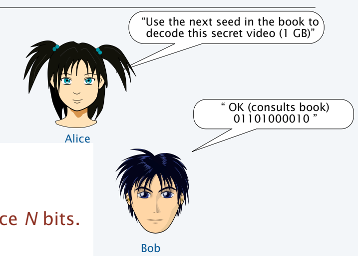
Self-assessment on LFSRs

Q. Give first 10 steps of [5,4] LFSR with initial fill 00001.

Encryption/decryption with an LFSR

Preparation

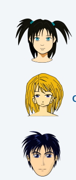
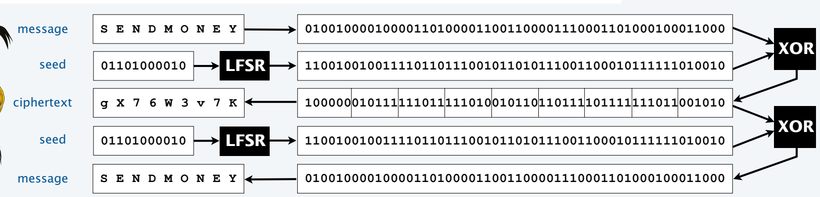
- Alice creates a book of "random" (short) seeds.
- Alice sends the book to Bob through a secure channel.



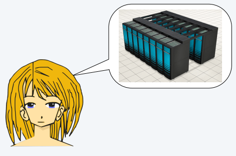
Encryption/decryption

- Alice sends Bob a description of which seed to use.
- They use the specified seed to initialize an LFSR and produce N bits.

[and proceed in the same way as for one-time pads]

Eve's opportunity with LFSR encryption



Eve has computers. 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.

Good news (for Eve): This approach can work.

- Ex: 11-bit register implies 2047 possibilities.
- Extremely likely that only *one* of those is not gibberish.
- After this course, *you* could write a program to check whether any of the 2047 messages have words in the dictionary.

Bad news (for Eve): It is easy for Alice and Bob to use a much longer LFSR.

Key properties of LFSRs

Property 1.

- Don't use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

Key properties of LFSRs

Property 1.

- Don't use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

Property 2. Bitstream must eventually cycle.

- $2^N - 1$ nonzero fills in an N-bit register.
- Future output completely determined by current fill.

Ex. [4,3] LFSR

0	0	1	0	0	0
0	1	0	0	1	1
1	0	0	1	1	2
0	0	1	1	0	3
0	1	1	0	1	4
1	1	0	1	0	5
1	0	1	0	1	6
0	1	0	1	1	7
1	0	1	1	1	8
0	1	1	1	1	9
1	1	1	1	0	10
1	1	1	0	0	11
1	1	0	0	0	12
1	0	0	0	1	13
0	0	0	1	0	14
0	0	1	0		15

Key properties of LFSRs

Property 1.

- Don't use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

Property 2. Bitstream must eventually cycle.

- $2^N - 1$ nonzero fills in an N-bit register.
- Future output completely determined by current fill.

Property 3. Cycle length in an N-bit register is *at most* $2^N - 1$.

- Could be smaller; cycle length depends on tap positions.
- Need theory of finite groups to know good tap positions.

Ex. [4,2] LFSR

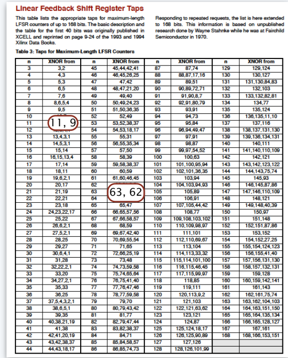
0	0	1	0	1	0
0	1	0	1	1	1
1	0	1	1	1	2
0	1	1	1	1	3
1	1	1	1	0	4
1	1	1	0	0	5
1	1	0	0	0	6
1	0	0	0	1	7
0	0	0	1	0	8
0	0	1	0		

Key properties of LFSRs

Property 1.
- Don't use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

Property 2. Bitstream must eventually cycle.
- $2^N - 1$ nonzero fills in an N-bit register.
- Future output completely determined by current fill.

Property 3. Cycle length in an N-bit register is at most $2^N - 1$.
- Could be smaller; cycle length depends on tap positions.
- Need theory of finite groups to know good tap positions.



Linear Feedback Shift Register Taps

This table lists the appropriate taps for maximum-length LFSR counters of up to 168 bits. The basic description and the table for the first 40 bits was originally published in XCELL and reprinted on page 9-24 of the 1993 and 1994 Xilinx Data Books.

Responding to repeated requests, the list is here extended to 168 bits. This information is based on unpublished research done by Wayne Stahnke while he was at Fairchild Semiconductor in 1970.

Table 3: Taps for Maximum-Length LFSR Counters

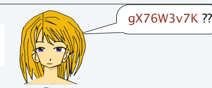
n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	[illegible]	52	52,49	94	94,73	136	136,135,11,10
11	11, 9	53	53,52,38,37	95	95,84	137	137,116
12	[illegible]	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	[illegible]	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63, 62	105	105,89	147	147,146,110,109
22	22,21	64	[illegible]	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

XILINX manual, 1990s

Bottom line.
- [11, 9] register generates 2047 bits before repeating.
- [63, 62] register generates 2^{63} -1 bits before repeating. ← Definitely preferable: small cost, huge payoff.

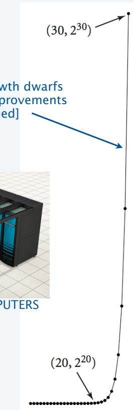
Eve's problem with LFSR encryption

Without the seed, Eve cannot read the message.



Eve

Eve has computers. 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.



Exponential growth dwarfs technological improvements [stay tuned]


NOT ENOUGH COMPUTERS

Bad news (for Eve): There are still way too many possibilities.
- Ex: 63-bit register implies $2^{63} - 1$ possibilities.
- If Eve could check 1 million seeds per second, it would take her 2923 centuries to try them all!

Bad news (for Alice and Bob): LFSR output is *not* random.

experts have cracked LFSRs

Goods and bads of LFSRs

Goods.
- Very simple encryption method.
- Decrypt with the same method.
- Scalable: 20 cells for 1 million bits; 30 cells for 1 billion bits.
- Widely used in practice. [Example: military cryptosystems.]

a commercially available LFSR

Bads.
- Easily breakable if seed is re-used.
- Still need secure key distribution.
- Experts can crack LFSR encryption.

Example.
- CSS encryption widely used for DVDs.
- Widely available DeCSS breaks it!

```
/*     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
                            ;}}
```

DeCSS DVD decryption code

COMPUTER SCIENCE
SEDGEWICK / WAYNE

Prologue: A Simple Machine

- Brief introduction
- Secure communication with a one-time pad
- Linear feedback shift registers
- **Implications**

CS.1.D.Prologue.Implications

LFSRs and general-purpose computers

LFSR

computer

component	LFSR	computer
control	start, stop, load	same
clock		same
memory	12 bits	billions of bits
input	12 bits	bit sequence
computation	shift, XOR	+ − * / ...
output	pseudo-random bit sequence	any computable bit sequence

Important similarities.
- Both are built from simple components.
- Both scale to handle huge problems.
- Both require careful study to use effectively.

Critical differences: Operations, input. ← but the simplest computers differ only slightly from LFSRs!
- General purpose computer can simulate *any* abstract machine.
- All general purpose computers have equivalent power (!) [stay tuned].

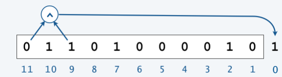
A Profound Idea

Programming. We can write a Java program to simulate the operation of any abstract machine.
- Basis for theoretical understanding of computation.
- Basis for bootstrapping real machines into existence.

Stay tuned (we cover these sorts of issues in this course).

YOU will be writing code like this within a few weeks. →

```
public class LFSR
{
   public static void main(String[] args)
   {
      int[] a = { 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0 };
      for (int t = 0; t < 2000; t++)
      {
         a[0] = (a[11] ^ a[9]);
         System.out.print(a[0]);
         for (int i = 11; i > 0; i--)
            a[i] = a[i-1];
      }
      System.out.println();
   }
}
```




```
% java LFSR
110010010011110110111001011010111001100011
011111101001000010011010010111100110010011
111110111000001010110001000011101010011010
000011110010011001110111111101010000010000
100010100101010001100000101111000100100110
1011011110001101001101110011110 1...
```

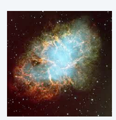
Note: You will write and apply an LFSR simulator in Assignment 5.

Profound questions

Q. What is a random number?

LFSRs *do not* produce random numbers.
- They are *deterministic*. ← von Neumann's "state of sin": we *know* that "deterministic" is incompatible with "random"
- It is not obvious how to distinguish the bits LFSRs produce from random,
- BUT experts have figured out how to do so.

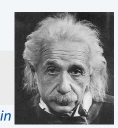
Q. Are random processes found in nature?
- Motion of cosmic rays or subatomic particles?
- Mutations in DNA?



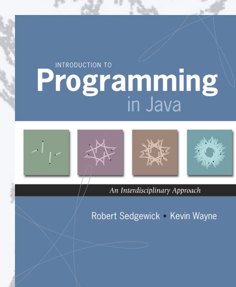
Q. Is the natural world a (not-so-simple) deterministic machine??

"God does not play dice."

– Albert Einstein



COMPUTER SCIENCE
SEDGEWICK / WAYNE



http://introcs.cs.princeton.edu

1. Prologue: A Simple Machine