CS 106X

Lecture 28: Conclusion

Friday, March 17, 2017

Programming Abstractions (Accelerated)
Winter 2017
Stanford University
Computer Science Department

Lecturer: Chris Gregg

Today's Topics

- Logistics
  - Final Exam Monday. See website for details and practice exam.
  - Any last minute concerns: please email Chris

- Finishing up Bloom Filters
- Where we have been
- Where you are going

Back to Bloom Filters

A bloom filter is a space efficient, probabilistic data structure that is used to tell whether a member is in a set.

Bloom filters are a bit odd because they can *definitely* tell you whether an element is *not* in the set, but can only say whether the element is *possibly* in the set.

Bloom Filters

In other words: “false positives” are possible, but “false negatives” are not.
(A *false positive* would say that the element is in the set when it isn’t, and a *false negative* would say that the element is not in the set when it is.

Bloom Filters

The idea is that we have a "bit array." We will model a bit array with a regular array, but you can compress a bit array by up to 32x because there are 8 bits in a byte, and there are 4 bytes to a 32-bit number (thus, 32x!) (although Bloom Filters themselves need more space per element than 1 bit).

Bloom Filters

a bit array:

1	**0**	**1**	**1**	**0**	**1**	**1**	**1**

Bloom Filters

Bloom Filters: start with an empty bit array (all zeros), and *k* hash functions.

$k1 = (13 - (x \% 13))\% 7$
$k2 = (3 + 5x) \% 7$, etc.

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

Bloom Filters

Bloom Filters: start with an empty bit array (all zeros), and *k* hash functions.

The hash functions should be independent, and the optimal amount is calculable based on the number of items you are hashing, and the length of your table (see Wikipedia for details).

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

Bloom Filters

Values then get hashed by all k hashes, and the bit in the hashed position is set to 1 in each case.

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

Bloom Filter Example

Insert 129: x=129, k1=1, k2=4

k1 = (13 - (x % 13))% 7
k2 = (3 + 5x) % 7

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

k1 == 1, so we change bit 1 to a 1
k2 == 4, so we change bit 4 to a 1

Bloom Filters

Insert 479: x=479, k1=2, k2=4

k1 = (13 - (x % 13))% 7

k2 = (3 + 5x) % 7

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

k1 == 2, so we change bit 2 to a 1

k2 == 4, so we would change bit 4 to a 1, but it is already a 1.

Bloom Filters

To check if 129 is in the table, just hash again and check the bits.

k1=1, k2=4: probably in the table!

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

$k1 = (13 - (x \% 13)) \% 7$, $k2 = (3 + 5x) \% 7$, etc.

Bloom Filters

To check if 123 is in the table, hash and check the bits. k1=0, k2=2: *cannot* be in table because the 0 bit is still 0.

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

k1 = (13 - (x % 13))% 7, k2 = (3 + 5x) % 7, etc.

Bloom Filters

To check if 402 is in the table, hash and check the bits. k1=1, k2=4:

Probably in the table (but isn't! False positive!).

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

Online example: http://billmill.org/bloomfilter-tutorial/

k1 = (13 - (x % 13))% 7, k2 = (3 + 5x) % 7, etc.

Bloom Filters: Probability of a False Positive

What is the probability that we have a false positive?

If *m* is the number of bits in the array, then the probability that a bit is not set to 1 during a hash insertion is

$$1 - \frac{1}{m}$$

Bloom Filters: Probability of a False Positive

If k is the number of hash functions, the probability that the bit is not set to 1 by any hash function is

$$\left(1 - \frac{1}{m}\right)^{k}$$

Bloom Filters: Probability of a False Positive

If we have inserted n elements, the probability that a certain bit is still 0 is

$$\left(1 - \frac{1}{m}\right)^{kn}$$

Bloom Filters: Probability of a False Positive

To get the probability that a bit is 1 is just 1- the answer on the previous slide:

$$1 - \left(1 - \frac{1}{m}\right)^{kn}$$

Bloom Filters: Probability of a False Positive

Now test membership of an element that is not in the set. Each of the k array positions computed by the hash functions is 1 with a probability as above. The probability of all of them being 1, (false positive):

$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$$

Bloom Filters: Probability of a False Positive

For our previous example, m=8, n=2, k=2, so:

$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k$$ = 0.17, or 17% of the time we will get a false positive.

Bloom Filters: Why?

Why would we want a structure that can produce false positives?

Example 1: Google Chrome used to use a local Bloom Filter to check for malicious URLs — if there is a hit, a stronger check is performed.

Example 2: The Akamai web server keeps track of web requests, and stores the requests in a bloom filter. Only when the request is sent a second time is the whole page cached -- this saves lots of cache space.

Bloom Filters: Why?

There is one more negative issue with a Bloom Filter: you can't delete! If you delete, you might delete another inserted value, as well! You could keep a second bloom filter of removals, but then you could get false positives in that filter…

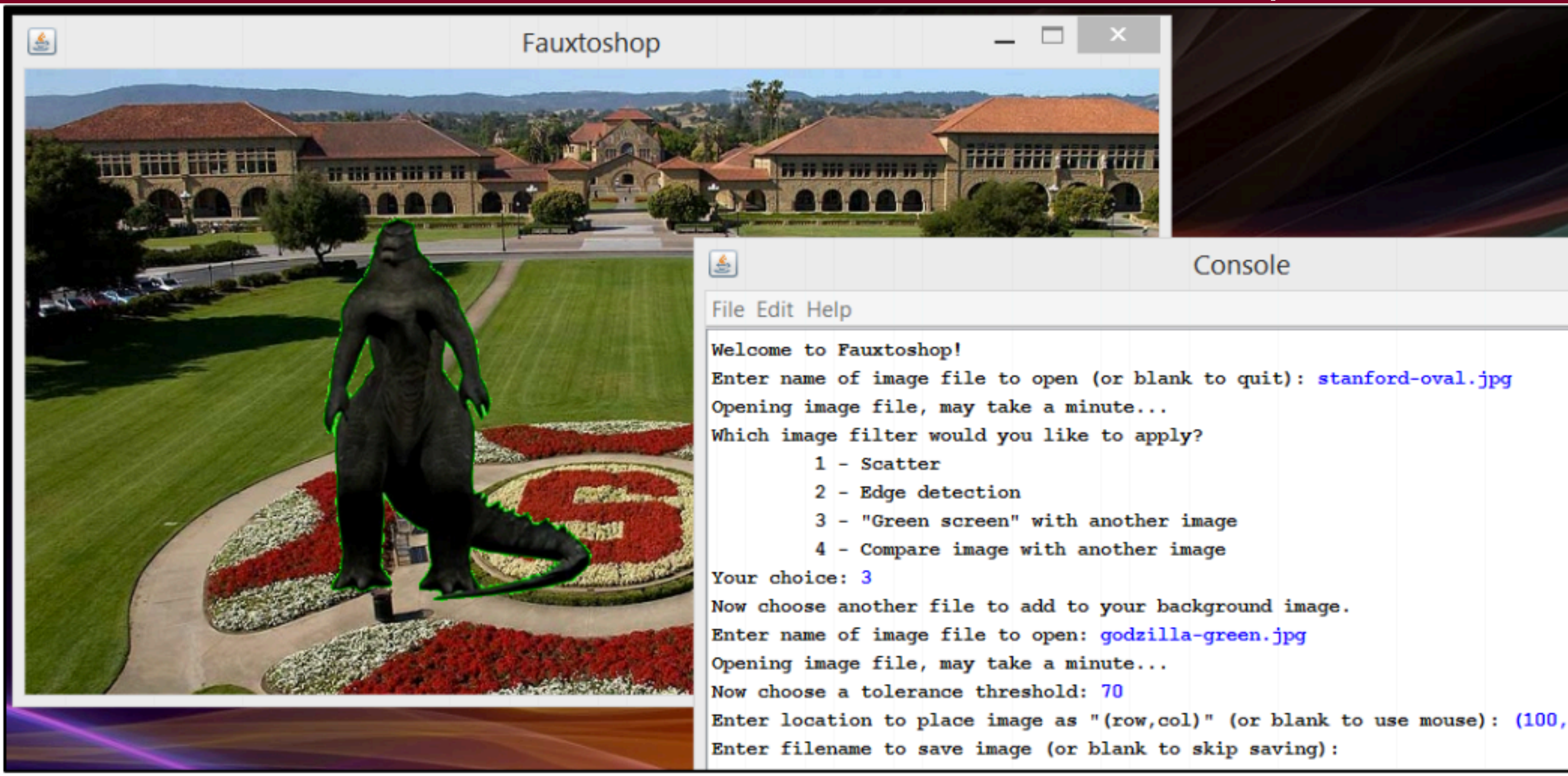
Bloom Filters: Why?

You have to perform *k* hashing functions for an element, and then either flip bits, or read bits. Therefore, they perform in O(k) time, which is independent of the number of elements in the structure. Additionally, because the hashes are independent, they can be parallelized, which gives drastically better performance with multiple processors.

Where We Have Been

CS 106X

Where We Have Been: Fauxtoshop



Where We Have Been: ADTs

MEET_ME_AT_SEVEN_PM_IN_GATES

plaintext

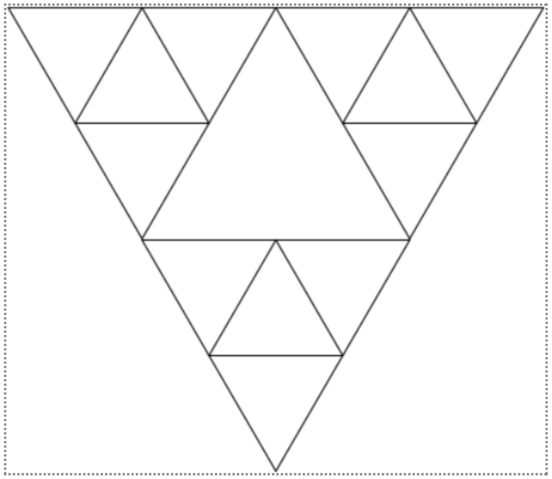
key: COMPSCI

MEET_ME
_AT_SEV
EN_PM_I
N_GATES

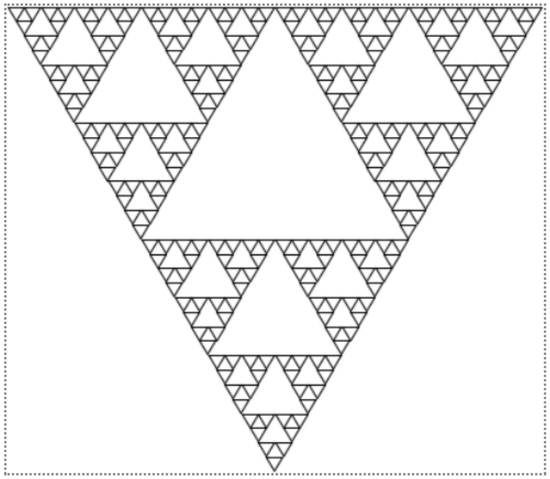
```
Welcome to CS 106X Random Writer ('N-Grams')!
This program generates random text based on a document.
Give me an input file and an 'N' value for groups of
words, and I will generate random text for you.

Input file name: hamlet.txt
Value of N: 3

# of random words to generate (0 to quit): 40
... chapel. Ham. Do not believe his tenders, as you go to this fellow.
Whose grave's this, sirrah? Clown. Mine, sir. [Sings] O, a pit of clay
for to the King that's dead. Mar. Thou art a scholar; speak to it. ...

# of random words to generate (0 to quit): 20
... a foul disease, To keep itself from noyance; but much more handsome
than fine. One speech in't I chiefly lov'd. ...

# of random words to generate (0 to quit): 0
Exiting.
```

```
Welcome to CS 106X Word Ladder!
Give me two English words, and I will
into the second by changing one lette

Dictionary file name: dictionary.txt

Word 1 (or Enter to quit): code
Word 2 (or Enter to quit): data
A ladder from data back to code:
data date cate cade code

Word 1 (or Enter to quit):
Have a nice day.
```

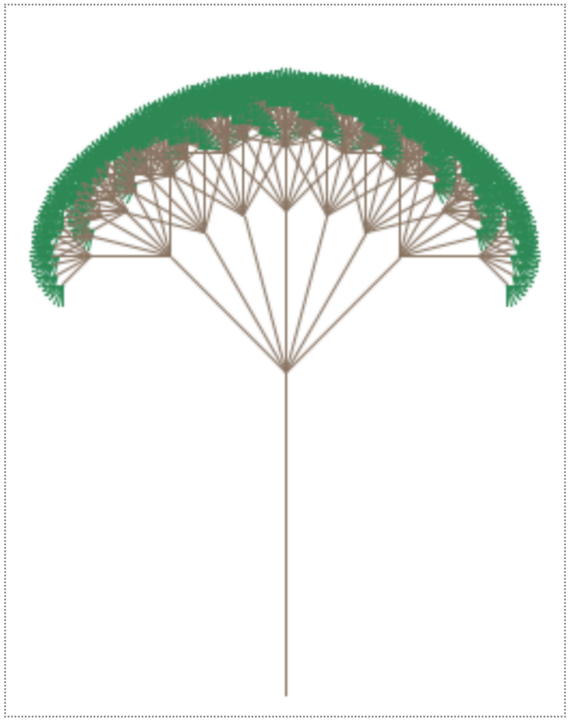
Where We Have Been: Fractals



Order-3

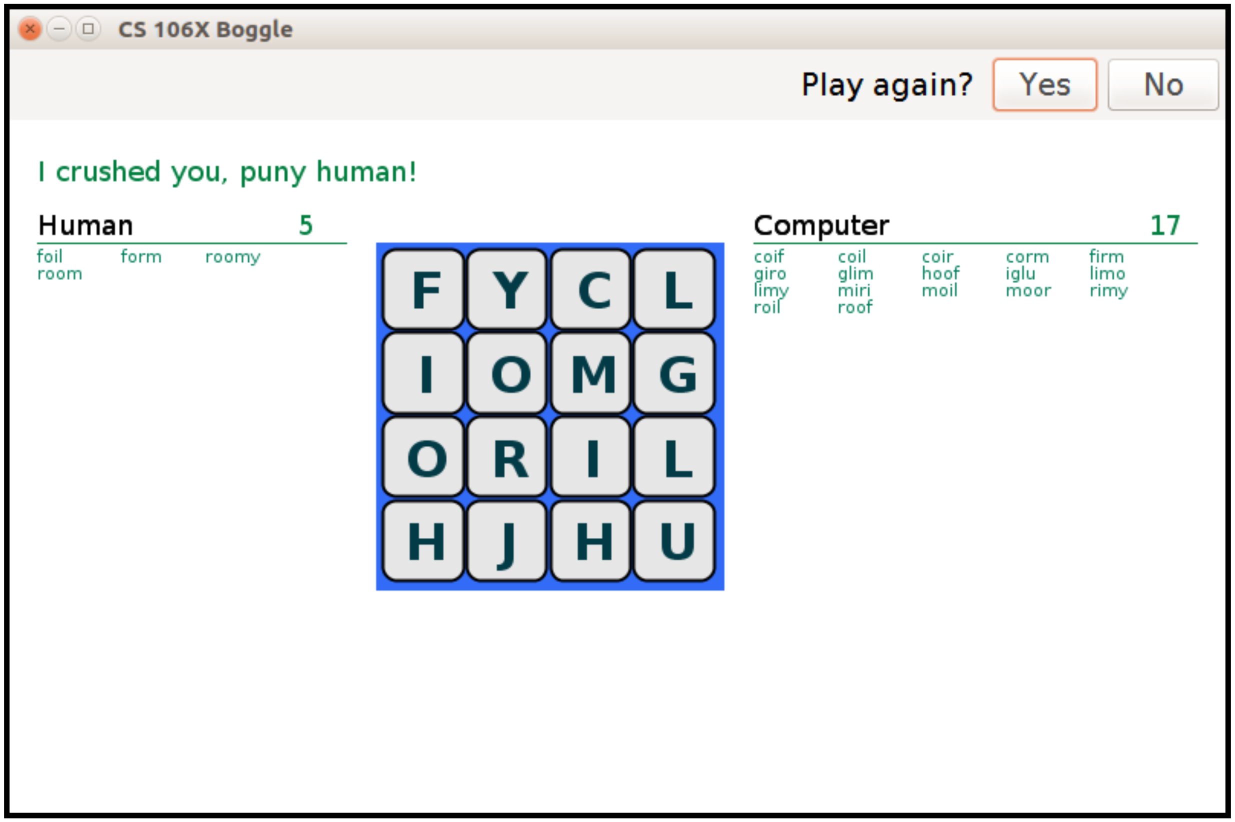


... Order-6



Order-5 tree fractal

Where We Have Been: Backtracking

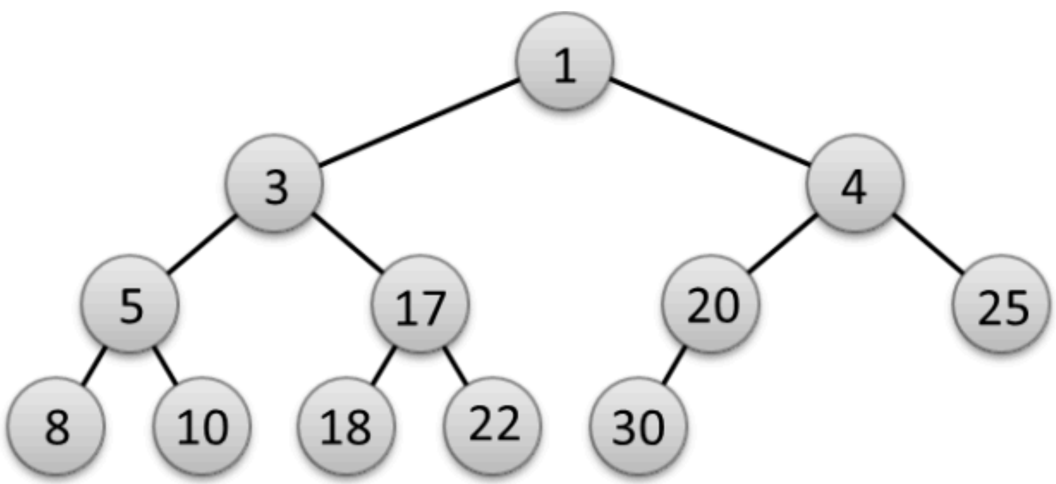


Where We Have Been: Linked Lists and Heaps

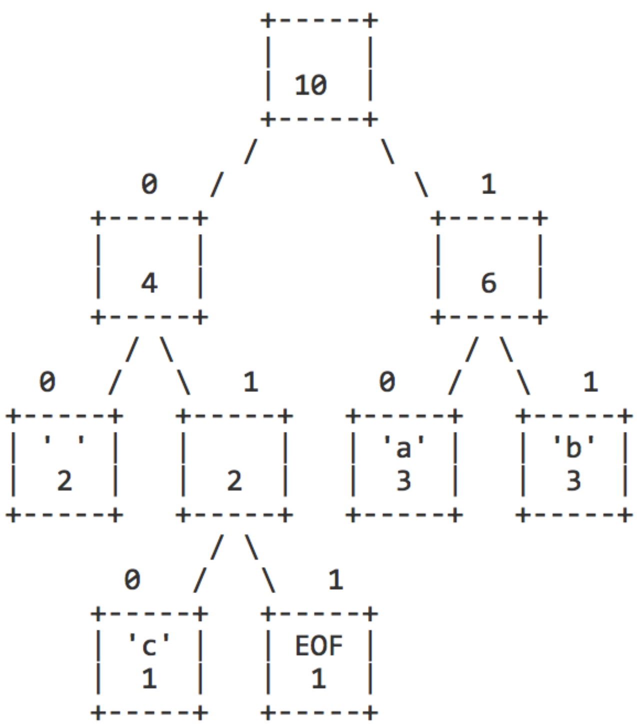
zero nodes	`front /`

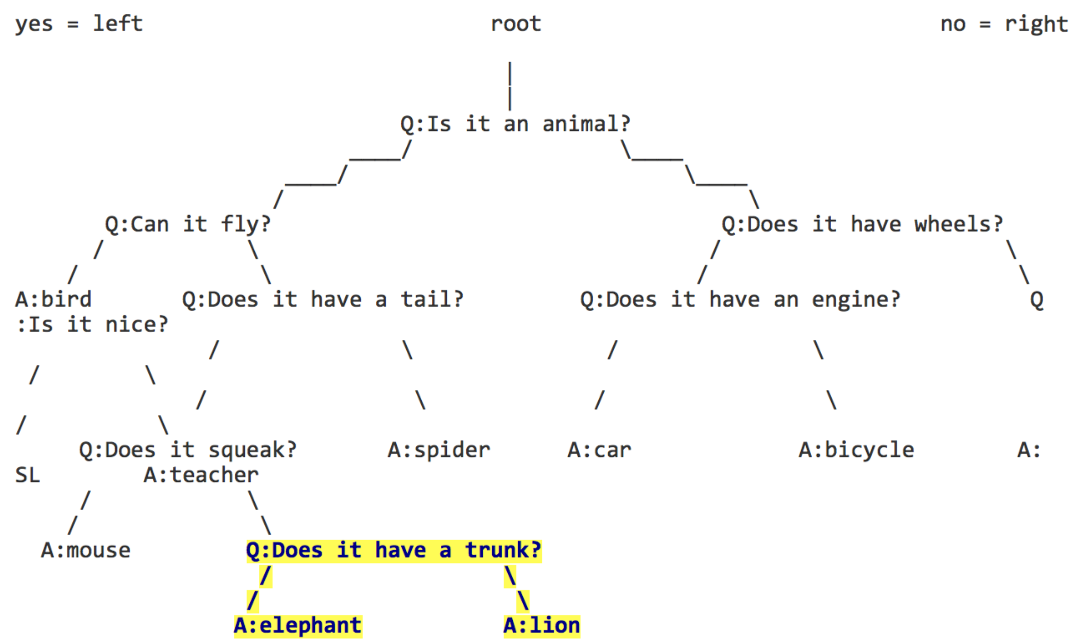
one node	`        +---+---+`
	`front --> | ? | / |`
	`        +---+---+`

N nodes	`        +---+---+     +---+---+         +---+---+    +---+---+`
	`front --> | ? |   | --> | ? |   |--> ... | ? |   |--> | ? | / |`
	`        +---+---+     +---+---+         +---+---+    +---+---+`

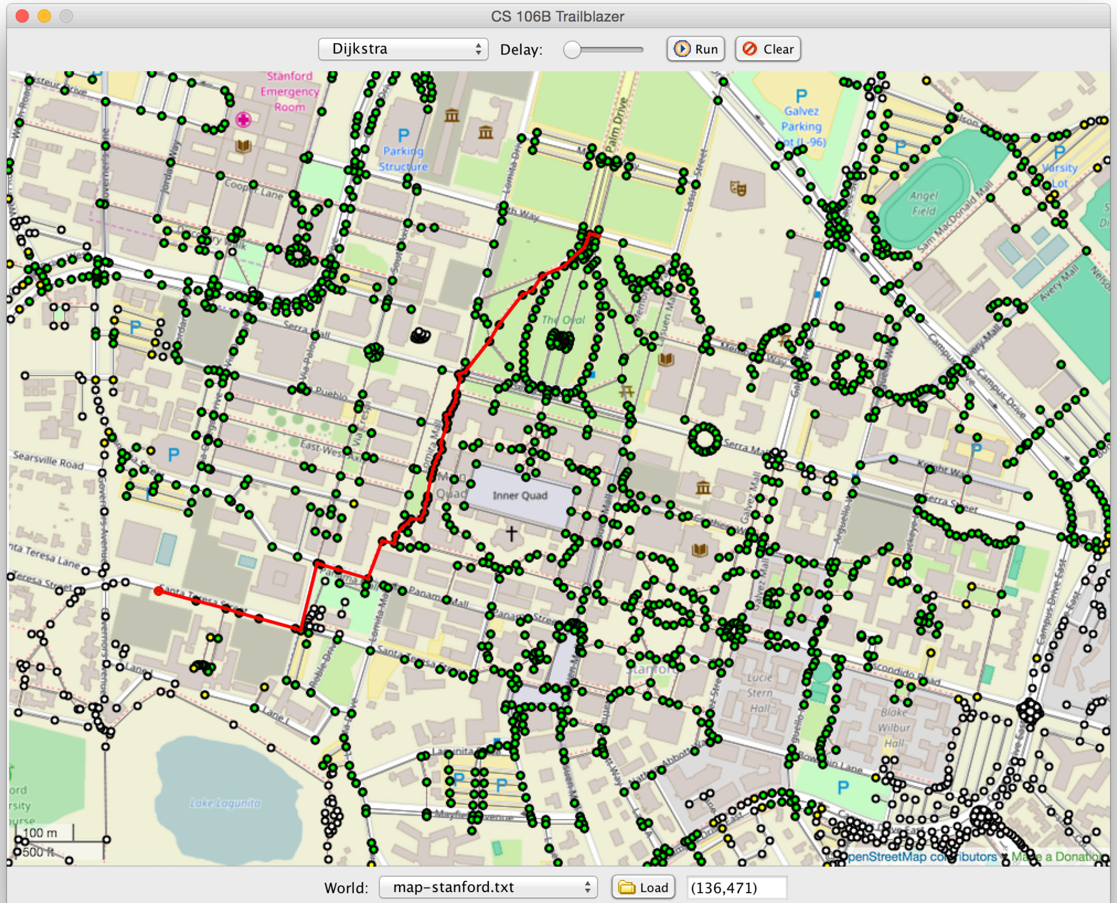


Where We Have Been: Binary Trees





Where We Have Been: Graphs



Where We Have Been: Sorting

So many ways to sort things!

We learned:

- Insertion sort
- Selection Sort
- Merge Sort
- Quicksort
- Radix Sort (on the exam...)

Other sorts:

- Shell Sort
- Heap Sort
- Tim Sort
- Bubble Sort

Trying to achieve O(n log n)
(but there are exceptions for certain types of data!)

Where We Have Been: C++

For many of you, a new language!

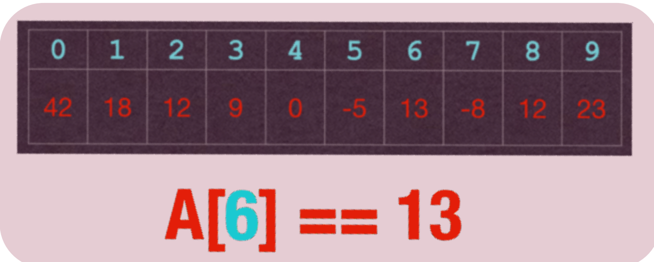
Highlights:

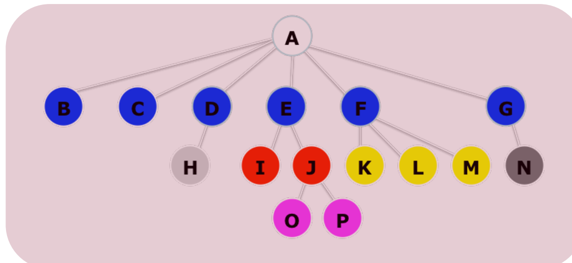
- Object oriented language with classes
- Fast (except our wonky graphics...)
- Extremely robust (too much sometimes)
- Widely used in industry and for making games

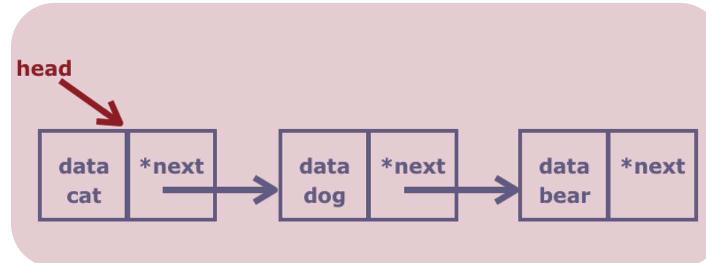
Differences you probably saw from other languages:

- Mutable strings
- Input / Output streams
- Operator overloading
- Pointers
- Memory Management: new, delete
- Inheritance and Polymorphism

The Importance of Data Structures







Why Data Structures are Important

One reason we care about data structures is, quite simply, time. Let's say we have a program that does the following (and times the results):

- Creates four "list-like" containers for data.
- Adds 100,000 elements to each container – specifically, the even integers between 0 and 198,998 (sound familiar?).
- Searches for 100,000 elements (all integers 0-100,000)
- Attempts to delete 100,000 elements (integers from 0-100,000)

What are the results?

The Importance of Data Structures

Structure	Overall(s)
Unsorted Vector	
Linked List	
Hash Table	
Binary Tree	
Sorted Vector	

The Importance of Data Structures

Results:

Structure	Overall(s)
Unsorted Vector	15.057
Linked List	92.202
Hash Table	0.145
Binary Tree	0.164
Sorted Vector	1.563

A factor of 103x

A factor of 636x!

Overall, the Hash Table "won" — but (as we shall see!) while this is generally a *great* data structure, there are trade-offs to using it.

Processor: 2.8GHz Intel Core i7 (Macbook Pro)
Compiler: clang++

Note: In general, for this test, we used optimized library data structures (from the "standard template library") where appropriate. The Stanford libraries are not optimized.

Full Results

Structure	Overall(s)	Insert(s)	Search(s)	Delete(s)
Unsorted Vector	15.057	0.007	10.307	4.740
Linked List	92.202	0.025	46.436	45.729
Hash Table	0.145	0.135	0.002	0.008
Binary Tree	0.164	0.133	0.010	0.0208
Sorted Vector	1.563	0.024	0.006	1.534

Why are there such discrepancies??

Bottom line:

- **Some structures carry more *information* simply because of their design.**
- **Manipulating structures takes time**

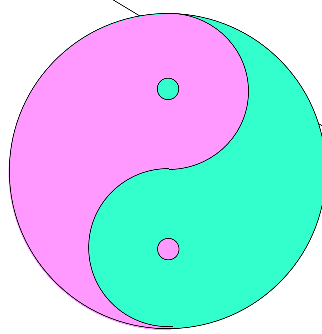
Where to from here?

CS Core

CS 106X
Programming Abstractions (Accelerated)

CS 107
Computer Organization and Systems

CS 110
Principles of Computer Systems

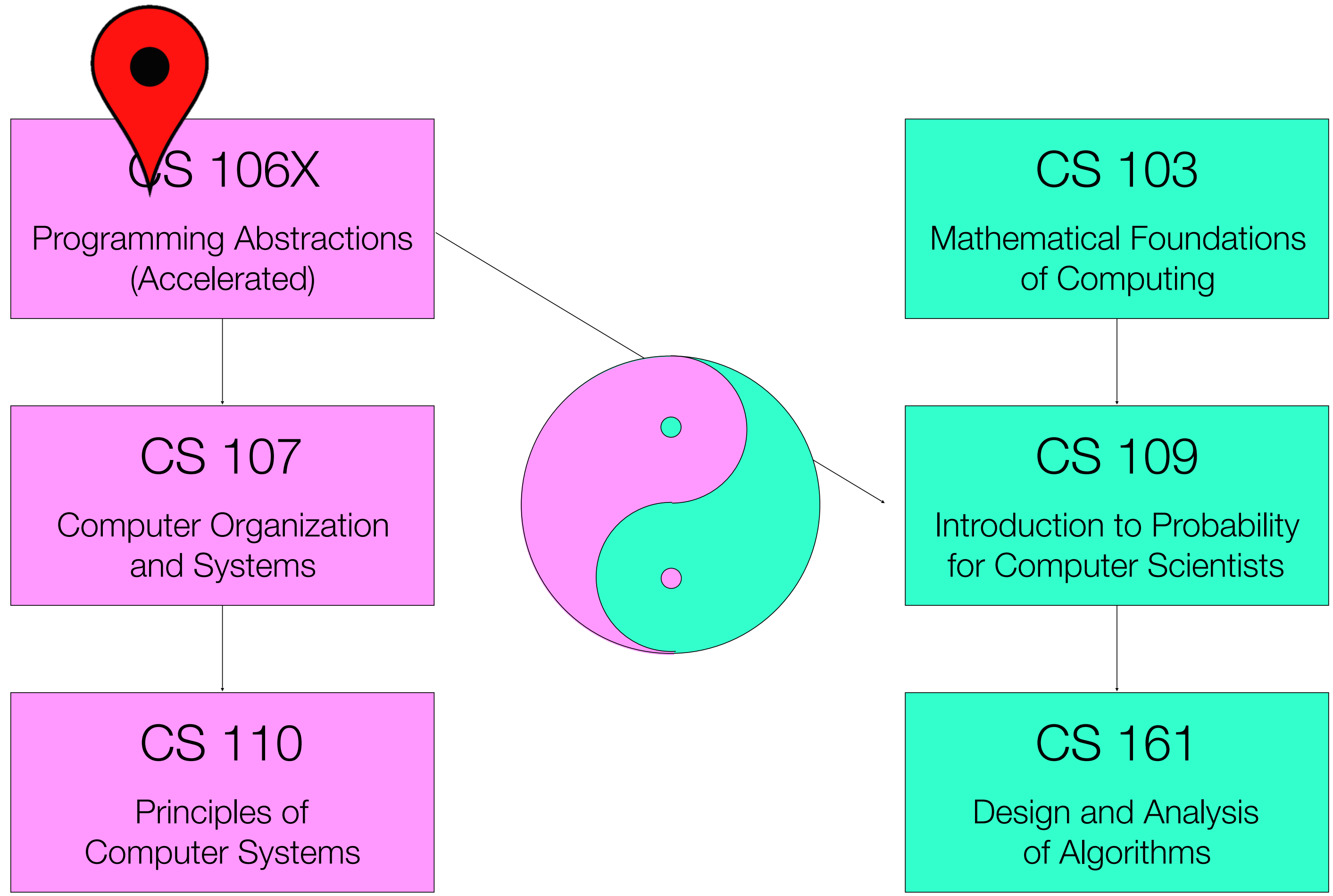


CS 103
Mathematical Foundations of Computing

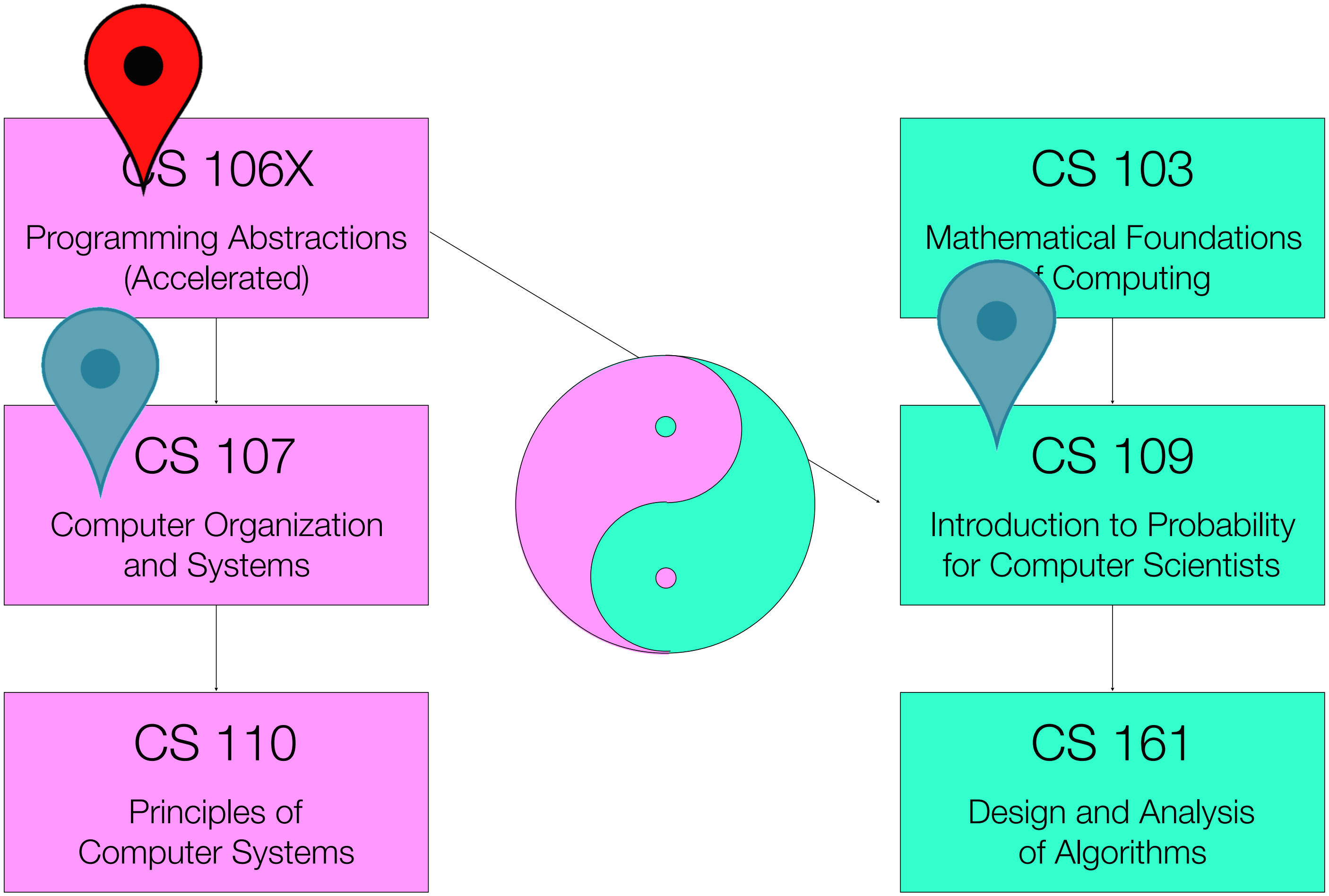
CS 109
Introduction to Probability for Computer Scientists

CS 161
Design and Analysis of Algorithms

CS Core

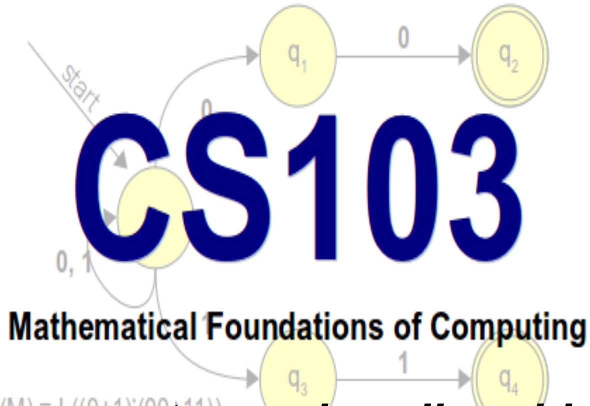

CS 106X
Programming Abstractions (Accelerated)
CS 107
Computer Organization and Systems
CS 110
Principles of Computer Systems
CS 103
Mathematical Foundations of Computing
CS 109
Introduction to Probability for Computer Scientists
CS 161
Design and Analysis of Algorithms

CS Core



CS 106X
Programming Abstractions
(Accelerated)
CS 107
Computer Organization
and Systems
CS 110
Principles of
Computer Systems
CS 103
Mathematical Foundations
f Computing
CS 109
Introduction to Probability
for Computer Scientists
CS 161
Design and Analysis
of Algorithms

CS 103



Can computers solve all problems?

Spoiler alert: no!

Why are some problems harder than others?

We can do find in an unsorted array in O(N), and we can sort an unsorted array in O(NlogN). Is sorting just inherently a harder problem, or are there better O(N) sorting algorithm yet to be discovered?

How can we be certain about this?

CS107 (kind of like CS106C)

How do we encode text, numbers, programs, etc. using just 0s and 1s?

Where does memory come from? How is it managed?

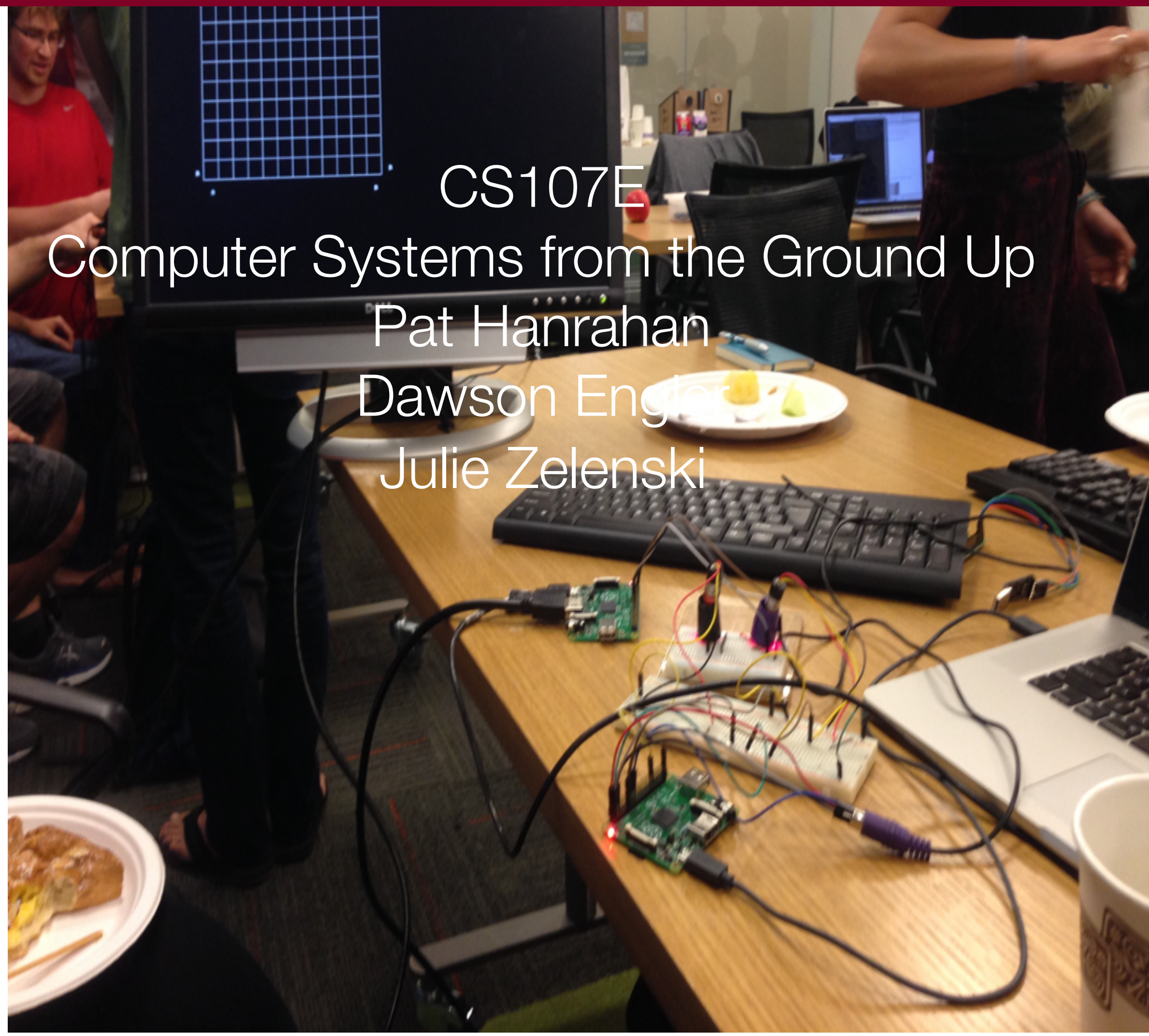
How do compilers, debuggers, etc. work?

CS107 is *not*

- CS107 is *not* a litmus test for whether you can be a computer scientist.
  - You can be a *great* computer scientist without enjoying low-level systems programming.

- CS107 is *not* indicative of what programming is "really like."
  - CS107 does a lot of low-level programming. You don't have to do low-level programming to be a good computer scientist.

CS107E


CS107E
Computer Systems from the Ground Up
Pat Hanrahan
Dawson Engler
Julie Zelenski

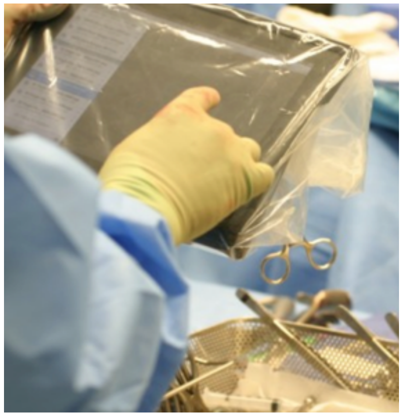
CS109

Foundations of probability

Narrative driven

Intro to
Machine Learning

Computer Science Affects Every Field

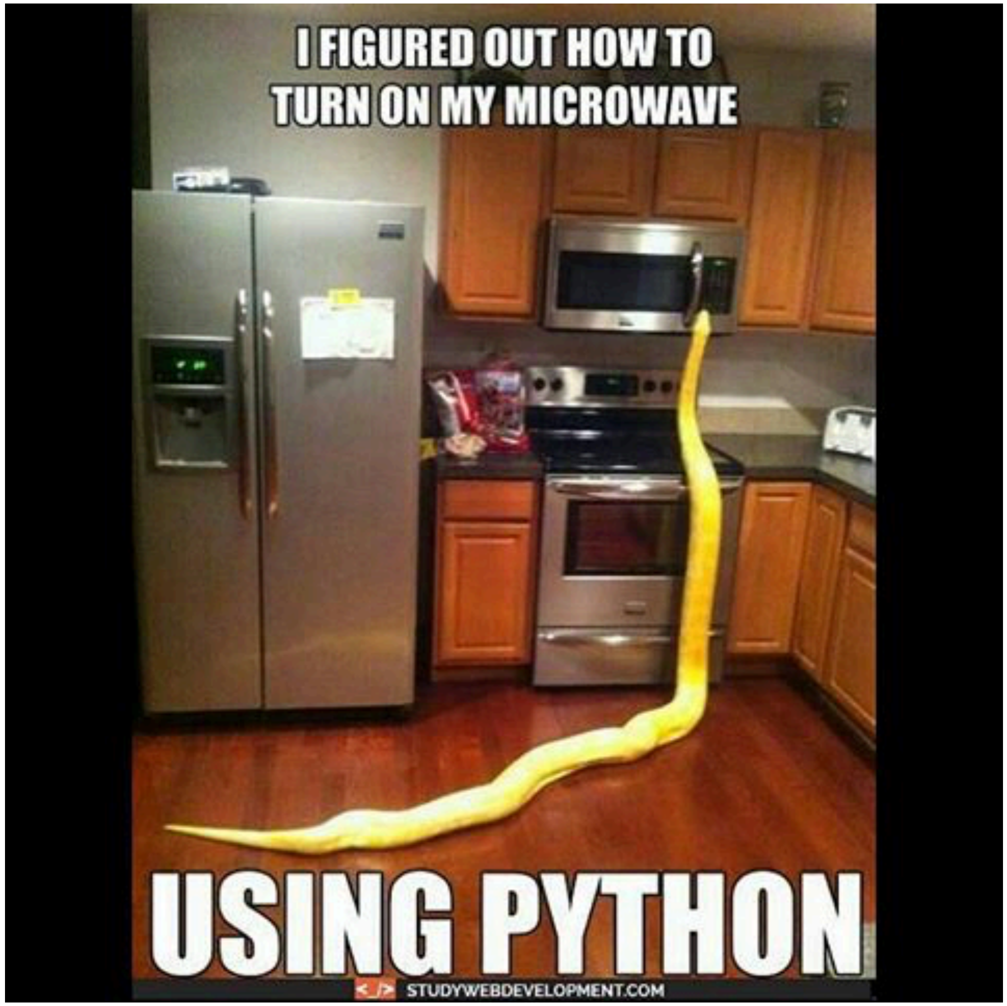


Classes Aren't Necessary!

Things to learn on your own:

- **A new language. Good candidates?**
  - Python: used *everywhere*, easy to learn, easy to write quick programs. Best online resource: https://www.reddit.com/r/Python/ (see right side-bar)
  - Haskell: a "functional" programming language. Best online resource: *Learn You a Haskell for Great Good*
- **iOS / Android Programming: Why not learn how to program your phone?**
  - Best iOS resource: https://www.raywenderlich.com
  - Good tutorials link: http://equallysimple.com/best-android-development-video-tutorials/
  - Want to code for all phones (and the web, and the desktop?) Check out React Native: https://facebook.github.io/react-native/
- **Hardware: Raspberry Pi, Arduino, FPGA: Hardware is awesome!**
  - Raspberry Pi resources: https://www.reddit.com/r/raspberry_pi/
  - Arduino Resources: https://www.reddit.com/r/arduino/
  - FPGA resources: http://www.embedded.com/design/prototyping-and-development/4006429/FPGA-programming-step-by-step
- **GPU and Multicore Programming: hard, but your code can *fly***
  - Your GPU might have hundreds of individual processors. Resources: http://gpgpu.org

Python


I FIGURED OUT HOW TO
TURN ON MY MICROWAVE
USING PYTHON
STUDYWEBDEVELOPMENT.COM

News from This Week

AARIAN MARSHALL TRANSPORTATION 03.16.17 6:15 AM

CARS NOW TALK TO OTHER CARS, IF YOU'RE INTO THAT SORT OF THING

source: https://www.wired.com/2017/03/cars-now-talk-cars-youre-sort-thing/

It is the Time and Place for CS

Thank You

Congrats (in advance)

References and Advanced Reading

- **References:**
  - Online Bloom Filter example: http://billmill.org/bloomfilter-tutorial/
  - Wikipedia Bloom Filters: https://en.wikipedia.org/wiki/Bloom_filter