Computer Science 50 Week 11 Wednesday: December 3, 2008
Fall 2008 Andrew Sellergren, Anjuli Kannan
Scribe Notes

Contents

Computer Science 50 Week 11 Wednesday: December 3, 2008
Fall 2008 Andrew Sellergren, Anjuli Kannan
Scribe Notes

1 Announcements (0:00–3:00)

- 1 handout
- 4 guests
- Check out the CS 50 Store!
- Seminars have begun! Feel free to attend one (or more than one) even if you didn't RSVP. Attend them all and win a prize![1]
- Quiz 1 Review will be held this Friday, 3:30–5:00 p.m., location TBA on the course website. There will also be last-minute OHs with David from 9:00 p.m. to midnight next Tuesday.
- Last Lunch with David this Friday, 1:15 p.m. E-mail `rsvp@cs50.net` to join!

2 CS 51 (3:00–33:00)

- The difference between *programming* and *computer science* is perspective. From the right perspective, many problems that look hard become easy.
- CS 51 teaches you to write code that is a work of art. Learn how to replace 10 pages of bad code with 1 line of good code.
- New algorithms have emerged in the last 15 years that are transforming the field of computer science. At semester's end, you'll have a set of tools that will allow you to solve computationally enormous tasks.
- The goal is simplicity! Efficiency means fast, simple, readable, extensible code. Sometimes efficiency is a matter of changing the problem to one that is easier to solve.
- The language that we will use in CS 51 is Scheme, a dialect of Lisp. The particular language, of course, is not as important as the underlying concepts. Still, choosing the right language is important: think of trying to do computation with Roman numerals–the tools are simply the wrong ones for the job!
- There's an intimate connection between proofs and code in Scheme. Often, you can write the proof and extract your code from it, or vice versa.
  - proof by induction ==> code by recursion
  - proof by cases ==> pattern matching
  - proof by lemma ==> functional application

[1] The prize of new knowledge, that is.

- There's usual an obvious way to a computational problem. For example, in the case of the spellchecker we implemented, the obvious solution might be an array on which you perform linear search.

- What happens when the obvious solution is way too slow? Perhaps not on the order of 10 minutes, but on the order of the lifetime of the universe.

- What kind of problems will you be exposed to in CS 51? First, let's talk about two versions of the same problem:

  – Conversation Problem: Each person in a network of friends has a conversation with each friend. Can we find a way for good news to spread?

  – Notification Problem: Each person hears the news once through a friend. Can we find a way for good news to spread?

- These two problems look very similar. How do we solve them? One way would be *exhaustive search*—that is, consider every single possibility. In a small network of friends, this might be feasible, but as the network gets larger and larger, this approach will take way too long.

- Another obvious algorithm would be to pick a friend who hasn't heard the news and either notify them or have a conversation with them. We'll call this the greedy algorithm.

- One of these problems can be solved in $O(n)$ using a variation of the greedy algorithm. The other can't be solved except with exhaustive search. Which is which?

- Turns out that the Notification Problem is exceedingly difficult, despite appearing deceptively as easy as the Conversation Problem.

- Let's look at another problem: the world's toughest job. J. Allard, Corporate Vice President of Microsoft, has the job of making the Zune popular. It has two unfortunate features: it can't use iTunes and it's made by Microsoft.

- Let's say that J has a limited advertising budget, but he can scrape together 25 Zunes to be given free to facebook users. Who should they be given to? This problem is actually very similar to the problem of immunization and social networking, which is an area of active research.

- One solution to this problem is exhaustive search, but it just might take the lifetime of the universe.

- Another solution is the greedy algorithm. We'll give the first Zune to Barack Obama, who has the most friends in our network. We'll give the second Zune to Hillary Clinton, who has the second-most friends in our network. And so on.

- However, Barack and Hillary's friends overlap, so this isn't the best solution either.
- Let's assess exhaustive search. If we assume that facebook has 10 million users (in fact, it has many more) and we can test 10^{100} groups per second, we would never finish. The age of the universe is less than 10^{18} seconds and there are over 10^{149} groups to test.
- And yet, exhaustive search has yet to be beaten in solving this problem!
- If we consider a clever algorithm by which we give a Zune to Barack and disregard him and all his friends (because they're already *infected*), then we give a Zune to whoever remaining has the most friends and disregard them, and continue until the Zune budget is exhausted, we will do pretty well with this problem. If exhaustive search finds a way to reach 1 million people with 25 Zunes, this algorithm will, by comparison, reach over 632,120 people. Not bad.
- Some of the other problems you'll be exposed to in CS 51
  - the Netflix/Walmart problem: how to determine what's popular?
  - the Legal Gibberish problem: how to determine the source?
  - the Data Partitioning problem: how to appropriately group it?

3 CS 61 (33:00–55:00)

- A few administrative details:
  - Fall 2009: Tuesday/Thursday 2:30–4:00
  - Prerequisites: CS 50 (or C programming experience)
  - Can be used for CS concentration breadth requirement("middle digit")
  - Can be use for CS secondary area requirement
  - You can, and should, take both CS 51 and CS 61 at the same time!
- How do computers really work? In CS 61, we'll get "under the hood" and learn how processors work, how memory works, how compilers work and, in general, what affects the performance of programs running on your computer.
- For example, most processors have many megabytes of cache memory on the processor chip itself. Without this cache, the computer would be 100 to 1000 times slower than it is. How can we write code that makes good use of this cache memory?
- CS 61 is meant to fill the gap between the *concepts* and the *reality* of programming. This gap becomes more profound as you move to higher-level languages. In some ways, this is a good thing—it makes programming

easier. However, without a good understanding of what's actually going on inside the machine, you will be unable to get the best performance possible from programs written in higher-level languages.

- Matt promises that you will *really* understand pointers by the time you're done with CS 61!
- How many times have you seen the Blue Screen of Death on your Windows machine? With CS 61 under your belt, you'll have a fighting chance of understanding its message!
- What exactly is a segmentation fault and the core file it dumps? The core file is an image of the memory of the system at the time it crashed.
- You've already learned that you can examine the core file with GDB. Let's say you've been staring at the C source code for hours and you've been unable to find the problem. Why not dig deeper? Type `disass` to print out all the machine instructions that were running inside your program at the time it crashed. It will even point you to the single line that caused the crash. In this case, the instruction was `movb $0x42,(%eax)`. This instruction says "move the hex value 42 into the memory location pointed to by the eax register." What does that mean exactly?
- Assembly code is like Latin: although being able to write it is not very useful, being able to read it is extremely useful.
- Let's look at a practical example of this: say that there was a password-protected program in Matt's home directory that, when executed, starts running a shell as user mdw. The trick is that a wrong guess will immediately be reported, so we have to get it right on the first try!
- With the shell command `objdump`, we can disassemble the executable. After taking CS 61, we might pick out the line `mov 0x8049814,%eax`. So we're moving something at this memory location into the eax register. Perhaps it's the stored password?
- If we examine the contents of the memory at that address, we find *another* memory address which contains the string `takecs61`. This is the password!
- The first assignment in CS 61 is a binary bomb which contains 7 password-protected levels. If you guess a password wrong, you lose a quarter point on the assignment—this is merely to prevent you from brute-forcing it.
- Become a h4xx0r! Ken Thompson, the co-inventor of UNIX, won the Turing Award, the highest award in computer science, in 1983. In his lecture, he made a stunning admission: he had hacked the C compiler over twenty years ago.

- In the early days of UNIX, Thompson wanted to be able to login to any UNIX system in the world, so he enabled the login program to accept his magic passoword.
- However, the source code for the login program was widely distributed, so anyone would have been to recognize it. The solution? He hacked the C compiler to recognize when it was compiling the login program and insert the backdoor code at compile time.
- But even the C compiler's source code was available, so someone could have recognized the backdoor code there. So he hacked the C compiler to recognize when it was compiling itself and insert the code in itself which would insert the backdoor code in the login program. Whew. Basically, he pwned everyone.
- There are 5 labs in CS 61. You can work with a partner on them! The midterm and final are take-home. Check out the course website here.

4 CS 171 (55:00–77:00)

- How do we convey information through visual representations?
- Visualization is not a new concept. William Playfair invented the line graph and pie chart in 1786 and 1801, respectively.
- Turns out as humans, our brains are geared very well toward comprehending information in a visual way.
- What has been the biggest change in visualization? Interaction! Check out the NY Times Budget Shortfall Map, Yahoo Finance, and Map of the Market for examples of visualization with interaction.
- Why do we need visualization? One reason is to communicate. For example, see this map of the California wildfires. Another reason is to answer questions. For example, if you wanted to know where the closest shelter was for a given wildfire, you might use someone's mashup on Google Maps.
- What are some of the problems with visualization? If we take a look at a classic representation of 2004 election results with states colored red or blue, we might be misled into believing that it was a landslide. However, this representation doesn't give us any idea of population, so we have no conception of how many people voted for each candidate.
- Another type of visualization is a word tree, such as this one which enables us to see the phrases which Alberto Gonzalez repeated during his deposition.
- Visualization allows us to uncover patterns. Matthew Ericson of the NY Times used a graph of batting averages of Hank Aaron, Babe Ruth, and

Barry Bonds to show that Barry Bonds had an uncharacteristic spike in performance late in his career which may have been associated with steroid use.

- Check out John McCain's concession speech and Obama's acceptance speech in the form of wordles, another type of visualization which displays word frequency.
- The goals of CS 171 are as follows:
  - Principles of effective visualizations
  - Overview of visualization applications
  - Implementing interactive visualizations
  - Take your programming a step further
- We'll be using mainly Java and its Processing framework, but also Python, which we'll use for web scraping.
- CS 171 will be presented in three acts, just like Italian opera:
  - Act I: Foundations
  - Act II: Visualizations
  - Act III: Guest Lectures
- Check out the course website here for more details on the syllabus and also examples of final projects from previous years.