

Course Introduction

*August 30, 2000**Handout 2*GROUND RULES

1. Ask Questions
2. Do your homework (using your own brain!)
3. Ask Questions
4. Keep up
5. Ask Questions!!!

Advice From Students (from course evaluations):

- Do homework, redo homework and practice before exams.
- You can't "slack off."
- Do the homeworks and when studying do quiz problems, and you'll do fine.
- Go to class, lectures are informative.
- Be prepared for quizzes
- You have to keep up in the homework

GOALS: The primary goal is to develop your ability to understand, create and critique mathematical arguments.

- This teaches you to think in terms of a few general concepts that can be cleanly and generally combined
- This teaches you to communicate clearly and precisely and to argue correctly without missing cases or committing fallacies.
- This teaches you math is not just equations but more importantly ideas expressed via equations and symbols.
- Theorems are not edits that come from other people - They were discovered after trial and error or guessing, and then proved.
- This teaches you to generalize.
- This teaches you to understand implications of code you write in terms of correctness and efficiency. Then you can design better algorithms and data structures.

These skill will be valuable for many areas within computer science.

There will be heavy emphasis on learning how to convert your intuition into a concise and complete proof. What is a proof?

Proofs are a language – at first foreign, but less so as you use it more and get comfortable with them. Consider the following question: How do I get to the train station?

Which of the following answers is better?

1. Go south until you reach the corner where the Methodist church burned down 15 years ago. Then go west past the McDonalds. That McDonalds is really known for having great beanie babies. Then go past the Shell station but don't go past the library...
2. Go north 1 block and then go west 3 blocks.

Which is a better response? Why?

The second is better. The first response is not clear nor concise. As in communication proofs must be concise, clear and accurate/complete. Everyone knows that must be accurate but tend to forget about the importance of having the proofs also be clear and concise.

A secondary goal of this course is to explore topics and techniques of discrete mathematics. This provides a toolbox of mathematical techniques that will be used in later courses.

But, what's this have to with building real systems like UNIX or an ALPHA workstation?

- Making good clean definitions and theorems that are general is analogous to writing programs that are general purpose and widely useful (e.g. some UNIX utilities).
- Making a few concepts that can be composed in various ways is analogous to defining a few clean program functions that can be composed in many ways (e.g. UNIX pipes and filters).
- Same basic decomposition techniques used to create complex proofs as used to create large software systems.
- Special portions of systems can benefit from correctness checks and algorithmic work (e.g., lock synchronization, UNIX timers)

Important Contributions to CS Based on Discrete Mathematics Ideas

- reliable protocols for network routing (deliver messages in a timely manner)
- cryptography (PGP, RSA)
- checksum algorithms used on disks and CD players (convolution checksums)
- compression algorithms (UNIX compress based on Lempel-Ziv)
- compiler optimization (register allocation using graph coloring)
- fast randomized algorithms (checking primes, simulated annealing)
- relational databases
- other high-impact algorithms (Karmarkar's linear programming algorithm, fast fourier transform, fast pattern matching)
- hashing and other commonly used data structures like balanced search trees and skip lists
- good (provably) approximation algorithms for hard optimization and scheduling problems (like traveling salesman)
- plus much more

ALL INVENTED BY PEOPLE WITH DISCRETE MATHEMATICS BACKGROUND.

To explore the foundations of CS it is useful to examine its pieces: math, logic and philosophy. These existed a long time before computer science evolved as a discipline. In fact there were models of computation (e.g. the Turing machine) that were studied prior to the existence of computers!

So there's a lot of background upon which CS developed. Along with learning skills that will enable you to reason about the correctness and efficiency of new algorithms, data structures, and protocols, we'll see how to demonstrate that there are some things a computer *cannot* compute.

With that we're off and running Enjoy the ride.