CS 312
Algorithm Design

Dan Sheldon
sheldon@cs.umass.edu
dsheldon@mtholyoke.edu

http://people.cs.umass.edu/~sheldon/teaching/cs312/

Office	Clapp 200
Mon	4:15-5:15
Tues	10-11
Thurs	By appointment

Today

- Introductions
- Logistics
- What is algorithm design?
  - An example: Stable Matching

What is Algorithm Design?

- How do you write a computer program to solve a complex problem?
    - Routing packets on the Internet
    - Computing similarity between DNA sequences
    - Scheduling final exams at a university

What is Algorithm Design?

- DNA sequence similarity
  - Input: two n-bit strings (AGGCTACC, CAGGCTAC)
  - Output: number between 0 and 1
  - ???

- Even if the objective is clear, we are often not ready to start coding right away!

What is Algorithm Design?

- Formulate the problem precisely
- Design an algorithm
- Prove the algorithm is correct
- Analyze the algorithm's runtime

An Example:
Stable Matching Problem

Goal. Given a set of preference among colleges and applications, design a self-reinforcing admissions process

What is self-reinforcing? Easier to describe when something is not self-reinforcing

College c prefers student s to admitted student

Student s prefers college c to admitted college

College c and student s are an unstable pair (s should transfer)

Stable assignment: assignment with no unstable pairs

Stable Matching Problem

- Goal. Given a set of preferences among colleges and high school students, design an admissions process with these properties:

- Perfect matching: everyone is matched one-to-one.
  - Each college gets exactly one student.
  - Each student gets exactly one college.

- Stability::no incentive to deviate from matching
  - In matching M, pair (c,s) is an unstable pair if college c and student s prefer each other to current partners.
  - Unstable pair (c,s) could each improve by switching. Chaos!

- Stable matching: perfect matching with no unstable pairs

Question 1

- Can we always find a stable matching?

Stable Roommate Problem

- Goal. Given 2n students, find a "suitable" matching.
- Students rank each other.

	Preferences		
Alice	Bob	Carol	Doofus
Bob	Carol	Alice	Doofus
Carol	Alice	Bob	Doofus
Doofus	Alice	Bob	Carol

Is there a stable matching?

More Questions

- If the sets being matched are disjoint, as in the college-student problem, is there always a stable matching?
- Is the stable matching unique?
- Can we find a stable matching efficiently?

Thoughts on Solving the Problem

- Initially, no colleges and students are matched.
- Pick an arbitrary college and have it admit its favorite student. Are we guaranteed that pair will be part of a stable matching?
  - Should a student accept her first offer?
  - If not, what should the student do?
- When are we done? Do we need to consider all combinations???

Propose-and-Reject (Gale-Shapley) Algorithm

```
Initialize each college and student to be free.
while (some college is free and hasn't made
offers to every student) {
   Choose such a college c
   s = 1st student on c′s list to whom c has not
     made offer
   if (s is free)
      assign c and s to be engaged
   else if (s prefers c to current college c′)
      assign c and s to be engaged, and c′ to be
      free
   else
      s rejects c
}
```

Questions about the Gale-Shapley Algorithm

- Does the algorithm terminate?
- Is the matching perfect, that is, is it one-to-one?
- Is the matching stable?

Proof by Contradiction (Review)

- Goal: prove that A is true

1. Assume A is false.
2. Reason to a contradiction with some other known fact
3. Conclude that A must therefore be true.

What is Algorithm Design?

- Formulate the problem precisely*
- Design an algorithm
- Prove the algorithm is correct
- Analyze the algorithm's runtime

*Gale-Shapley algorithm is actually used to match residents to hospitals

An Iterative Process

- Usually don't get it right the first time
- May be no correct answer
    - Stable roommate problem
- May be no correct efficient answer
    - NP-completeness

Course Goals

- Learn to apply this process (by practice!)
- Learn specific algorithm design techniques
  - Greedy, Divide-and-Conquer, Dynamic Programming, Network Flows
- Prove no exact efficient solution is possible
  - Intractability and NP-completness