What is Algorithm Design?

How do you write a computer program to solve a complex problem?

- Computing similarity between DNA sequences
- Routing packets on the Internet
- Scheduling final exams at a college
- Assign medical residents to hospitals
- Find all occurrences of a phrase in a large collection of documents
- Finding the smallest number of coffee shops that can be built in the US such that everyone is within 20 minutes of a coffee shop.

COVID-19

- Welcome back!
- Please be courteous and use common sense
  - follow current public health guidelines
  - don’t come to class if credible risk of infecting others
  - In Fall 22, UMass is a “mask-welcome” campus, and “masks strongly encouraged for first couple weeks”
DNA sequence similarity

- **Input:** two n-bit strings $s_1$ and $s_2$
  - $s_1 = \text{AGGCTACC}$
  - $s_2 = \text{CAGGCTAC}$
- **Output:** minimum number of insertions/deletions to transform $s_1$ into $s_2$
- **Algorithm:** ????
  - Even if the objective is precisely defined, we are often not ready to start coding right away!

What is Algorithm Design?

- **Step 1:** Formulate the problem precisely
- **Step 2:** Design an algorithm
- **Step 3:** Prove the algorithm is correct
- **Step 4:** Analyze its running time

**Important:** this is an iterative process, e.g., sometimes you’ll even want to redesign the algorithm to make it easier to prove that it is correct.

Course Goals

- Learn how to apply the algorithm design process... by practice!
- Learn specific algorithm design techniques
  - Greedy
  - Divide-and-conquer
  - Dynamic Programming
  - Network Flows
- Learn to communicate precisely about algorithms
  - Proofs, reading, writing, discussion
- Prove when no exact efficient algorithm is possible
  - Intractability and NP-completeness

Prerequisites: CS 187 and 250

- Familiarity with:
  - data structures (lists, stacks, queues, ...)
  - mathematical objects (sets, lists, relations, partial orders)
  - recursion: many algorithm design patterns based on recursion
  - proofs: correctness of algorithms. contradiction, induction, ...
Course Information

Course websites:
- people.cs.umass.edu/~sheldon/teaching/cs311/
- umass.moonami.com
- piazza.com
- gradescope.com

Announcements: Check UMass email / Piazza regularly for course announcements.

A Week in the Life of CS 311

Tue  Lecture (iClicker)
Wed Challenge problems (due 11:59pm) OR midterm exam 7–9pm
Thu  Lecture (iClicker)
Fri  Discussion section in morning (worksheets), weekly homework due 11:59pm

Weekly Homework (Gradescope Online Assignments)

- due most Fridays. HW 1 posted, due 9/10
- focused on specific learning goal mastery (see detailed learning goals on course page)
- midterms will look similar
- Collaboration: OK to ask for help on how to solve a problem, but do them on your own. Copying, sharing, or viewing any solutions that are not your own is a violation of course policy (and you won’t learn what you need to know for midterms).
Challenge Problems

**Problem 1.** Stable Matching: Running Time. In class, we saw that the Propose-and-reject algorithm terminates in at most n^2 iterations, where n is the number of students and men.

**Grade Criteria**

<table>
<thead>
<tr>
<th>Grade</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+</td>
<td>Complete at least 14 challenge problems with ✓ or better; including at least 7 with ✓+</td>
</tr>
<tr>
<td>A</td>
<td>Complete at least 12 challenge problems with ✓ or better; including at least 6 with ✓+</td>
</tr>
<tr>
<td>B</td>
<td>Complete at least 8 challenge problems with ✓ or better; including at least 4 with ✓+</td>
</tr>
<tr>
<td>C</td>
<td>Complete at least 6 challenge problems with ✓</td>
</tr>
<tr>
<td>D</td>
<td>Attempt at least 6 challenge problems and complete at least 3 with a ✓</td>
</tr>
</tbody>
</table>

- Solutions typed or written neatly and uploaded to Gradescope as high-quality PDF
- Usually involve designing an algorithm and proving it correct
- Choose which problems to submit (at least one per assignment)
- Graded as one of x, ✓−, ✓, or ✓+ using rubric on course web page.
  - ✓ and ✓+ indicate mastery (fairly high standards)
  - contribute to grade as follows

- Collaboration OK (e.g. discuss problem, generate ideas, work on whiteboard), but read/attemt on your own first. The written solution must be your own. Looking at written solutions that are not your own (other students, web) is considered cheating. There will be formal action if cheating is suspected. List collaborators and any printed or online sources at the top of each assignment.

- Don’t need to complete every problem, so focus on high-quality solutions to ones you can solve
- No benefit to guessing, vague answers
Grading Breakdown

- Participation (10%): discussion section, in-class iClicker questions
- Homework (10%): ≈8–10 weekly assignments
- Challenge problems (25%): 6 assignments, roughly bi-weekly
- Challenge problem self-assessments (5%): Review solutions and post self-assessment of your challenge problems solutions after due date
- Midterms 1, 2, 3 (10% each): each covers about one quarter of the course
- Final (20%): covers all course materials

Late Policies

- Homework and challenge problems: Submit via Gradescope by 11:59pm on due date.
- Late: no credit (after short grace period)
- Each student is allowed to submit three assignments up to 24 hours late without penalty
- At most one late day per assignment (solutions posted after 24 hours)

Collaboration and Academic Honesty

- Homework and challenge problems: see above
- Discussions: Groups for the discussion section exercises will be assigned randomly at the start of each session. You must complete the discussion session exercise with your assigned group.
- Exams: Closed book and no electronics.
- Formal action will be pursued for suspected cheating. Penalty may be an F in course. If in doubt whether something is allowed, ask!

Stable Matching Problem

Matching applicants to medical residency programs:
- \( m \) applicants
- \( n \) slots at hospitals
- Applicants have preferences over hospitals and vice versa
- National Resident Matching Program (nrmp.org) makes matches

What is a "good" way to match applicants to programs?
- economists: matching should be stable. no incentive to switch
- Gale-Shapley algorithm
2012 Nobel Prize in Economics

Lloyd Shapley. Stable matching theory and Gale-Shapley algorithm.

Alvin Roth. Applied Gale-Shapley to matching med-school students with hospitals, students with schools, and organ donors with patients.

Problem formulation (colleges and students)

**Input:**
- $n$ colleges
- $n$ students
- preference lists

**Output:** a *stable* matching. But what does this mean?

**Matching:**
- assignment students to colleges
- set $M$ of college-student pairs, each college/student in one pair

**Instability or unstable pair** in a matching: an unmatched pair that prefer each other to their assigned matches

**Stable matching:** matching with no instabilities

**Goal:** output a stable matching
Clicker

Colleges

<table>
<thead>
<tr>
<th></th>
<th>a:</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b:</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>c:</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Students

<table>
<thead>
<tr>
<th></th>
<th>1:</th>
<th>b</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

Which pair is an instability (unstable pair) with respect to the matching \{ (a, 1), (b, 3), (c, 2) \} (marked in **bold** above)

A. (a, 2)
B. (b, 1)
C. (b, 3)
D. none of the above

Do stable matchings always exist? Are they unique?

Example 1: universal prefs

Colleges

<table>
<thead>
<tr>
<th></th>
<th>a:</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b:</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Students

<table>
<thead>
<tr>
<th></th>
<th>1:</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

▶ \( M = \{ (a, 1), (b, 2) \} \)? stable
▶ \( M = \{ (a, 2), (b, 1) \} \)? not stable

Example 2: inconsistent prefs

Colleges

<table>
<thead>
<tr>
<th></th>
<th>a:</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b:</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Students

<table>
<thead>
<tr>
<th></th>
<th>1:</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Clicker: Which matching is stable?

A. \( M = \{ (a, 1), (b, 2) \} \)
B. \( M = \{ (a, 2), (b, 1) \} \)
C. neither
D. both

▶ Answer: D, both are stable
▶ Fact: there can be multiple stable matchings
Toward an Algorithm

Let’s use a slightly bigger example to try to develop an algorithm.

<table>
<thead>
<tr>
<th>Colleges</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: 1 2 3</td>
<td>1: c a b</td>
</tr>
<tr>
<td>b: 2 1 3</td>
<td>2: a b c</td>
</tr>
<tr>
<td>c: 1 3 2</td>
<td>3: a b c</td>
</tr>
</tbody>
</table>

*Idea:* build M incrementally. What should colleges do? What should students do?

Summary

- Unmatched colleges take turns offering to students and propose in order of preference
- Students tentatively accept first offer and then “trade up” if they receive better offers

Propose-and-Reject (Gale-Shapley) Algorithm

Initially all colleges and students are free

```plaintext
while some college is free and hasn’t made offers to every student do
    Choose such a college c
    Let s be the highest ranked student to whom c has not offered
    if s is free then
        c and s become matched
    else if s is matched to c' but prefers c to c' then
        c' becomes unmatched
        c and s become matched
    else
        s rejects c and c remains free
```

Analyzing the Algorithm

*Goal:* prove that the algorithm always returns a stable matching
Observations:

- (F1) Students accept their first offer, after which they stay matched and only “upgrade” during the algorithm.
- (F2) Colleges propose to students sequentially in order of preferences.

Termination

Does the algorithm terminate?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- in each round, some college proposes to a new student in their list (by F2).
- at most \( n^2 \) proposals \( \Rightarrow \) at most \( n^2 \) rounds.

Stable Matching?

Is the final matching stable?

Proof (by contradiction)

- Suppose there is an instability \((c, s)\)
- \(c\) is matched to \(s'\) but prefers \(s\) to \(s'\)
- \(s\) is matched to \(c'\) but prefers \(c\) to \(c'\)
- Did \(c\) offer to \(s\)? Yes, by (F2), since \(c\) offered to \(s'\) who is ranked lower
- Did \(s\) accept offer from \(c\)? Maybe initially, but \(s\) must eventually reject \(c\) for another college, and, by (F1), \(s\) prefers final college \(c'\) to \(c\)
- Contradiction!
For Next Time

▶ Think about: would it be better or worse for the students if we ran the algorithm with the students proposing?
▶ Read: Chapter 1, course policies