Welcome!
Algorithm Design & Analysis

Course Focus

• Effective fundamental design paradigms
• Tools & techniques for analysis of resource use
• Ability to accurately gauge problem difficulty
• Strategies for “intractable” problems
• Enhanced problem-solving skills
Who’s Who

Instructor

• Bill Lenhart (wlenhart@williams.edu)
  • Office: TPL 304 (x2371)
  • Office Hours: M/T/W/Th: 2:00-3:50 pm

TAs

• Quan Do, Javier Esparza, Minwoo Kang, Jamie Kasulis, Lester Lee, Emily Zheng
  • TA Hours: T/W/Th: 7:00-11:00pm
    • Location (Currently): T/Th: SSL030A; W: TCL 206
What’s What

The Text

- *Algorithm Design* by Kleinberg & Tardos (1st Ed.)

Problem Sets, Exams, Projects

- Weekly Problem Sets
  - Assigned on Friday and submitted by noon (in class) the following Friday
  - Except for 2/15 (Winter Carnival): Submit to my CS ‘cubby’ by 5:00 pm
  - Collaboration is fine but must be cited
  - Problem Set 0 is out now; due next Friday, Feb. 8
  - Problem Sets must be typeset using LaTeX

- Mid-Term (4/5 - 4/8) and Final (24-hour) Exams
  - Take-home; collaboration prohibited

The Course Website

- https://www.cs.williams.edu/~lenhart/cs256/index.html
The Fine Print

Expectations

• Read all information on the course website (by Monday noon!)

• Come to class (reasonably) prepared
  • Read in advance to get general idea

• Contribute
  • Attend and participate
  • Help make the course a great experience for all

• Assignments
  • Start early, finish on time

• Adhere to the Honor Code

• Have fun & learn (a lot)
An Illustrative Example

Matching Problems

• Assigning first year students to advisors
• Pairing job candidates with employers
• The “Algorithm of Happiness”
  • The National Resident Matching Program

Fundamental Problem

• Find a matching that protects against opportunistic swapping
Hospital Preference Lists

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td>C</td>
</tr>
<tr>
<td>W</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>X</td>
<td>B</td>
<td>E</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>Y</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>Z</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>E</td>
<td>C</td>
</tr>
</tbody>
</table>

Matching

- V - D
- W - A
- X - E
- Y - C
- Z - B

Student Preference Lists

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Z</td>
<td>V</td>
<td>W</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>W</td>
<td>Y</td>
<td>V</td>
<td>Z</td>
</tr>
<tr>
<td>C</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>D</td>
<td>V</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
<td>W</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>W</td>
<td>Z</td>
<td>X</td>
<td>V</td>
</tr>
</tbody>
</table>

Instability

- W - A
- Z - B
State the Problem

Two groups: Students & Hospitals

- Each student has ranked desired hospitals
- Each hospital has ranked desired students

Goal

- Match as many students as possible, avoiding instabilities
  - Instability: s is not matched to h, but s & h both prefer each other to their matches
Hospital Preference Lists

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td>C</td>
</tr>
<tr>
<td>W</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>X</td>
<td>B</td>
<td>E</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>Y</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>Z</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>E</td>
<td>C</td>
</tr>
</tbody>
</table>

Partial Matching

Student Preference Lists

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Z</td>
<td>V</td>
<td>W</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>W</td>
<td>Y</td>
<td>V</td>
<td>Z</td>
</tr>
<tr>
<td>C</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>D</td>
<td>V</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
<td>W</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>W</td>
<td>Z</td>
<td>X</td>
<td>V</td>
</tr>
</tbody>
</table>
Find the Simplest Interesting Version

Distracting Complications

- Students/Hospitals have incomplete rankings
- Hospital might want several students
- Might not be enough posts for all students

The Original: Stable Marriage Problem

- \( n \) men & \( n \) women
- every man and woman submits a complete ranking
- We’ll use a set \( X \) of \( n \) hospitals and \( Y \) of \( n \) students

Goal

- Find a stable perfect matching
  - Perfect: Every student/hospital is matched
Restate the Problem

Instability in matching $M$

- Unmatched pair $\{h, s\}$, with $\{h, s'\}$, $\{s, h'\}$ in $M$ where $h$ prefers $s$ to $s'$ and $s$ prefers $h$ to $h'$

Stable Matching: No instabilities

- For every unmatched $\{h, s\}$,
  - Either $h$ prefers its match to $s$,
  - Or $s$ prefers its match to $h$

Does such a matching always exist?

Can we find it? What if $n = 50,000$?

If so, how?
Design an Algorithm


- Iterative process in which
  - A hospital offers a student a post
  - A student can reject current matched hospital in favor of a preferred one
- Stop when everyone is matched (hopefully!)
- Claim: Stable matching has been produced
Gale & Shapley: Propose - Reject Algorithm (1962)

Initialize each entity to be free (unmatched).

while (some hospital is free and hasn't offered a post to every student) {
    Choose such a hospital h
    s = h’s most preferred student to whom h hasn’t yet offered a post

    if (s is free)
        assign h and s to be matched  // h & s no longer free
    else if (s prefers h to its current match h')
        assign h and s to be matched, and h' to be free
    else
        s rejects h  // h remains free
}

Questions

• Does it halt?
• Does it match everyone?
• Is the matching stable?
Analyzing the Algorithm: Correctness

Look for Helpful Features of Algorithm

• Every iteration eliminates a potential offer
  • So algorithm at least halts [! Find a progress measure !]
• Once matched, a student stays matched
• Students always ‘trade up’
  • If s breaks match with h in favor of h’ then s prefers h’ to h
• Every student gets matched
  • So the matching is complete (perfect matching)
The Matching is Stable

• Suppose not: Then final matching has an instability \{h,s\}
  • h is matched to s’ but h prefers s over s’
  • s is matched to h’ but s prefers h over h’
  • So h offered a post to s before offering one to s’
    • So either s broke the match to h at some point, or s already had a match h’’ that s preferred over h
  • But students always trade up, so final match of s (h’) must be preferred by s to h—>Contradiction!
Analyzing the Algorithm: Performance

Establish Running Time and Space Requirements

• Space: $2n$ lists of length $n$: $O(n^2)$
• Time: Count a representative set of operations
  • Select a free hospital $h$
  • Find most preferred $s$ not yet offered a post by $h$
  • Find $s$’s ranking of a given hospital
  • Make a matched hospital free again
  • Add to & delete from set of matched couples
Wait! What’s the Input?

- Assume each entity provides complete ordered list (most to least preferred) of choices

Identify Efficient Data Structures for Operations

- Assume students are named 1..n, and hospitals are named 1..n
- Free hospital: get(), put() : Queue: $O(1)$ for both
- Of students not yet offered a post by h, find most preferred
  - $h$ provided a sorted list: $O(1)$
Identify Efficient Data Structures for Operations

- Find s’s ranking of a given hospital
  - For each s, an array, indexed by hospital, of hospitals as ranked by s (inverted index)
    - \( s(h) = i \) if \( h \) has rank \( i \) in s’s list
    - Build this from s’s ordered list of hospitals: \( O(n) \) per list
- Add to & delete from set of matched pairs:
  - Array, \( \text{Matched}(s) = h \) currently matched to \( s \) (or ‘free’): Creation time \( O(n) \); update time \( O(1) \)
Analyzing the Algorithm: Performance IV

Initialize each entity to be free.
while (some h is free and hasn't offered a post to each s) {
    Choose such an h
    s = most preferred student of h not yet offered a post by h
    if (s is free)
        assign h and s to be matched  // h & s no longer free
    else if (s prefers h to current match h')
        assign h and s to be matched, and h' to be free
    else
        s rejects h  // h remains free

Overall Running Time

• Preprocessing Time + Loop Time
  • Preprocessing: O(n²) (for inverted lists) + O(n) : (for Matched())
  • Loop: O(n²) * O(1) (all steps take constant time)
Analyzing the Algorithm: Performance V

• How efficient is P-R precisely?
  • Measure in terms of input size
    • For \( n \) hospitals and \( n \) students, input size is \( K = n^2 \)
    • P-R takes time proportional to \( K \)
    • So P-R is a linear-time algorithm—and it’s greedy
      • Is it optimal?
      • May depend on the model of computation...

____________________________

**Principle:** Measure complexity relative to size of instance
What else can we learn from the P-R algorithm

- Seems to favor the proposers over rejectors
- Empirical Observation: Seems to produce same matching regardless of choice of selection of free hospital
Best Feasible Partner

Let $I$ be an instance of the stable marriage problem

- An instance of a problem is any single valid input
- $s \in Y$ is a feasible partner for $h \in X$, if $\{h,s\}$ is part of some stable matching of $I$.

- $\{h,s\}$ is then called a feasible pair
- For $h \in X$, let $\text{best}(h)$ denote the most preferred feasible partner of $h$

- What can we say about $S^* = \{\{h,\text{best}(h)\} : h \in X\}$?
The Surprising Set $S^*$

It turns out that:

- $S^*$ is a matching
- $S^*$ is stable
- $S^*$ is optimal from the proposers’ perspective
- $S^*$ is the unique output of P-R regardless of next free hospital selection method!
The Proof: Overview

• Assume that P-R does not produce $S^*$.
  • That is, for at least some $h$, $h$ is **not** matched with $best(h)$

• Derive contradiction: Show there exists a perfect matching $S'$ such that
  • $S'$ is stable, and
  • $S'$ is *not* stable

• Key Ideas:
  • Every $h$ will, at some point during P-R algorithm, offer a post to $best(h)$
  • If $h$ is matched to $s$ in P-R algorithm, then $h$ is rejected by all $s'$ that $h$ prefers to $s$
  • Rejection has two flavors: $s$ already has better match or $s$ prefers new hospital to current match
**The Proof II**

Assume that P-R does not produce $S^*$.

- Consider first rejection of some $h$ by $s = \text{best}(h)$
  - *This is a common proof technique with algorithm analysis!*
- $s$ rejects $h$ for some $h'$ that $s$ prefers to $h$ [NB: 2 flavors of rejection!]
- But $\{h, s\}$ is feasible: Let $S'$ be some stable matching containing $\{h, s\}$
- In $S'$, $h'$ is matched to $s' \neq s$.
- Does $h'$ prefers $s$ to $s'$? If so, then $\{h', s\}$ is an instability in $S'$
  - This contradicts the stability of $S'$, so $h'$ prefers $s'$ to $s$....
The Proof III

So, $h'$ prefers $s'$ to $s$.

Return to the execution of the P-R algorithm on $I$

- In P-R, $h$ was the first element of $X$ to be rejected by its $\text{best}(h)$
- So $h'$ had not yet been rejected by any feasible partner
- But $s'$ is a feasible partner of $h'$, and $h'$ prefers $s'$ to $s$
- So $h'$ can’t have been rejected by $s'$
- So $h'$ can’t currently be matched to $s$ or be offering a post to $s$ at this point
- So neither flavor of rejection of $h$ by $s$ in favor of $h'$ was possible!

Contradiction: $h$ was never rejected by $\text{best}(h)$!
Result

• Propose-Reject produces $S^\ast$ regardless of free hospital selection strategy

• Thus $S^\ast$ is a matching and it is stable

• Also $S^\ast$ is clearly proposer-optimal (and, sadly, rejector-pessimal...)

Principle: Use an algorithm to prove new facts about a problem
Generalizing Our Result

P-R still works, with modifications, when

- I: Some pairings are forbidden
- II: Hospital wants several students, student wants one hospital
- III: Both I & II

Some extensions are explored in Problem Set 1
Summary: How to Solve a Problem

State the Problem

Play With the Problem

Find the Simplest Interesting Version of the Problem

Clearly restate the Simplest Version of the Problem

Design an Algorithm & Make it Precise

Identify Properties of Algorithm to Prove Correctness
  • Go back to previous step if (as is most likely) necessary

Establish Running Time and Space Requirements
  • Identify Efficient Data Structures for Operations

Reflect on implications of your solution—you may have learned something!
What Usually Happens

Poor Understanding of Problem
No Intuition About Problem
Confusion Due to Overwhelming Detail
Poor Statement of Problem
Multiple Failed Attempts to Design Algorithm
Inability to Prove Correctness
  • Often because algorithm does not actually work correctly
Trade-Offs Between Time & Space Resources Create Difficult Choices
But Wait, There’s More…. 
Five Famous Problems

- Interval Scheduling
- Weighted Interval Scheduling
- Bipartite Matching
- Independent Set
- Competitive Facility Location
Interval Scheduling

- Instance: A list of requests for a single resource
  - Request $i$ consists of a start time $s_i$ and end time $t_i$
- Goal: Find maximum-size set of non-overlapping requests
  - Efficiently solvable by a greedy algorithm

Figure 1.4 An instance of the Interval Scheduling Problem.
Weighted Interval Scheduling

- Instance: A list of requests for a single resource
  - Request \( i \) also has weight \( w_i \)
- Goal: Find maximum-weight set of non-overlapping requests
- Efficiently solvable by \textit{dynamic programming}
- Interval scheduling is a special case: \( w_i = 1 \)
Bipartite Matching

Instance: A list of requests for one of a set of resources

- Request \( i \) is a pair \( \{x_i, y_j\} : x_i \in X, y_j \in Y, X \cap Y = \emptyset \)

Goal: Find maximum-sized set of non-overlapping requests: maximum-sized matching

- Efficiently solvable by network flow techniques
- Inherently a graph-theoretic problem

Figure 1.5 A bipartite graph.
Independent Set

Instance: A list of conflicts between elements of a set $V$

- Conflict $i$ is a pair $\{u_i, v_i\}$: $u_i, v_i \in V$

Goal: Find maximum-sized conflict-free subset of $V$

- Considered to be intractable

- Inherently a graph-theoretic problem

- Interval scheduling is special case

- So is maximum matching

Figure 1.6 A graph whose largest independent set has size 4.
Competitive Facility Location

- Instance: A number $B$ and graph $G = (V,E)$ with vertex weights $w(v)$

- Two players alternately select vertices of $G$

  - All selected vertices must form a single independent set; player 1 begins

  - Player 2 wants to construct an independent set of weight at least $B$; player 1 wants to prevent this

- Goal: Determine whether player 2 can succeed

- Considered to be harder than Independent Set

- Inherently a graph-theoretic problem

Figure 1.7 An instance of the Competitive Facility Location Problem.