CSEP 521 Course Introduction

- CSEP 521, Applied Algorithms
  - Monday’s, 6:30-9:20 pm
  - CSE 305 and Microsoft Building 99

- Instructor
  - Richard Anderson, anderson@cs.washington.edu
  - Office hours:
    - CSE 582
    - Monday, 4:00-5:00 pm or by appointment

- Teaching Assistant
  - Tanvir Aumi, tanvir@cs.washington.edu
  - Office hours:
    - TBD
Announcements

• It’s on the web.
• Homework due at start of class on Mondays
  – HW 1, Due January 14, 2013
  – It’s on the web

http://www.cs.washington.edu/education/courses/csep521/13wi/
Text book

- Algorithm Design
- Jon Kleinberg, Eva Tardos

- Read Chapters 1 & 2

- Expected coverage:
  - Chapter 1 through 7
Recorded lectures

• This is a distance course, so lectures are recorded and will be available online for later viewing.

• However, low attendance in the distance PMP course is a concern.
  – Various draconian measures are under discussion.

• We will make lectures available.
  – Please attend class, and participate.
  – Participation may be a component of the class grade.
Lecture schedule

• Monday holidays:
  – Monday, January 21, MLK
  – Monday, February 18, President’s day

• Make up lectures will be scheduled, which will be recorded for offline viewing
  – Hopefully, some students will attend, so there is a studio audience
  – First makeup lecture:
    • Thursday, January 17, 5:00-6:30 pm

• Additional makeup lectures to accommodate RJA’s travel schedule
Course Mechanics

• Homework
  – Due Mondays
  – Textbook problems and programming exercises
    • Choice of language
    • Expectation that Algorithmic Code is original
  – Target: 1 week turnaround on grading
  – Late Policy: Two assignments may be turned in up to one week late

• Exams (In class, tentative)
  – Midterm, Monday, Feb 11 (60 minutes)
  – Final, Monday, March 18, 6:30-8:20 pm

• Approximate grade weighting
  – HW: 50, MT: 15, Final: 35
All of Computer Science is the Study of Algorithms
How to study algorithms

• Zoology
• Mine is faster than yours is
• Algorithmic ideas
  – Where algorithms apply
  – What makes an algorithm work
  – Algorithmic thinking
Introductory Problem: Stable Matching

• Setting:
  – Assign TAs to Instructors
  – Avoid having TAs and Instructors wanting changes
    • E.g., Prof A. would rather have student X than her current TA, and student X would rather work for Prof A. than his current instructor.
Formal notions

- Perfect matching
- Ranked preference lists
- Stability

\[ m_1 w_1 \]
\[ m_2 w_2 \]
Example (1 of 3)

\[ m_1 : w_1 \ w_2 \]
\[ m_2 : w_2 \ w_1 \]
\[ w_1 : m_1 \ m_2 \]
\[ w_2 : m_2 \ m_1 \]
Example (2 of 3)

\[ m_1 : w_1 \quad w_2 \]
\[ m_2 : w_1 \quad w_2 \]
\[ w_1 : m_1 \quad m_2 \]
\[ w_2 : m_1 \quad m_2 \]
Example (3 of 3)

\[m_1 : w_1 \ w_2\]
\[m_2 : w_2 \ w_1\]
\[w_1 : m_2 \ m_1\]
\[w_2 : m_1 \ m_2\]
Formal Problem

• Input
  – Preference lists for $m_1, m_2, \ldots, m_n$
  – Preference lists for $w_1, w_2, \ldots, w_n$

• Output
  – Perfect matching $M$ satisfying stability property:

  \[ \text{If } (m', w') \in M \text{ and } (m'', w'') \in M \text{ then} \\
  \quad (m' \text{ prefers } w' \text{ to } w'') \text{ or } (w'' \text{ prefers } m'' \text{ to } m') \]
Idea for an Algorithm

m proposes to w
  If w is unmatched, w accepts
  If w is matched to m₂
    If w prefers m to m₂ w accepts m, dumping m₂
    If w prefers m₂ to m, w rejects m

Unmatched m proposes to the highest w on its preference list that it has not already proposed to
Algorithm

Initially all m in M and w in W are free
While there is a free m
    w highest on m’s list that m has not proposed to
    if w is free, then match (m, w)
else
    suppose (m_2, w) is matched
    if w prefers m to m_2
       unmatch (m_2, w)
       match (m, w)
Example

\[
\begin{align*}
  m_1 &: w_1 \ w_2 \ w_3 \\
  m_2 &: w_1 \ w_3 \ w_2 \\
  m_3 &: w_1 \ w_2 \ w_3 \\
  w_1 &: m_2 \ m_3 \ m_1 \\
  w_2 &: m_3 \ m_1 \ m_2 \\
  w_3 &: m_3 \ m_1 \ m_2
\end{align*}
\]
Does this work?

• Does it terminate?
• Is the result a stable matching?

• Begin by identifying invariants and measures of progress
  – m’s proposals get worse (have higher m-rank)
  – Once w is matched, w stays matched
  – w’s partners get better (have lower w-rank)
Claim: The algorithm stops in at most $n^2$ steps
When the algorithms halts, every $w$ is matched.

Why?

Hence, the algorithm finds a perfect matching.
The resulting matching is stable

Suppose

\[(m_1, w_1) \in M, (m_2, w_2) \in M\]

\[m_1 \text{ prefers } w_2 \text{ to } w_1\]

How could this happen?
Result

- Simple, $O(n^2)$ algorithm to compute a stable matching
- Corollary
  - A stable matching always exists
A closer look

Stable matchings are not necessarily fair

m_1: w_1 w_2 w_3
m_2: w_2 w_3 w_1
m_3: w_3 w_1 w_2

w_1: m_2 m_3 m_1
w_2: m_3 m_1 m_2
w_3: m_1 m_2 m_3

How many stable matchings can you find?
Algorithm under specified

- Many different ways of picking m’s to propose
- Surprising result
  - All orderings of picking free m’s give the same result
- Proving this type of result
  - Reordering argument
  - Prove algorithm is computing something more specific
    - Show property of the solution – so it computes a specific stable matching
Proposal Algorithm finds the **best possible** solution for \( M \)

Formalize the notion of best possible solution:

\[(m, w) \text{ is valid if } (m, w) \text{ is in some stable matching}\]

\[\text{best}(m): \text{the highest ranked } w \text{ for } m \text{ such that } (m, w) \text{ is valid}\]

\[S^* = \{(m, \text{best}(m))\}\]

Every execution of the proposal algorithm computes \( S^* \)
Proof

See the text book – pages 9 – 12

Related result: Proposal algorithm is the worst case for W
Algorithm is the M-optimal algorithm
Proposal algorithms where w’s propose is W-Optimal
Best choices for one side may be bad for the other

Design a configuration for problem of size 4:

**M proposal algorithm:**
All m’s get first choice, all w’s get last choice

**W proposal algorithm:**
All w’s get first choice, all m’s get last choice
But there is a stable second choice

Design a configuration for problem of size 4:

M proposal algorithm:
All m’s get first choice, all w’s get last choice

W proposal algorithm:
All w’s get first choice, all m’s get last choice

There is a stable matching where everyone gets their second choice
Suppose there are $n$ m’s, and $n$ w’s

• What is the minimum possible M-rank?

• What is the maximum possible M-rank?

• Suppose each m is matched with a random w, what is the expected M-rank?
Random Preferences

Suppose that the preferences are completely random

\[ m_1: w_8 \ w_3 \ w_1 \ w_5 \ w_9 \ w_2 \ w_4 \ w_6 \ w_7 \ w_{10} \]
\[ m_2: w_7 \ w_{10} \ w_1 \ w_9 \ w_3 \ w_4 \ w_8 \ w_2 \ w_5 \ w_6 \]
\[ \ldots \]
\[ w_1: m_1 \ m_4 \ m_9 \ m_5 \ m_{10} \ m_3 \ m_2 \ m_6 \ m_8 \ m_7 \]
\[ w_2: m_5 \ m_8 \ m_1 \ m_3 \ m_2 \ m_7 \ m_9 \ m_{10} \ m_4 \ m_6 \]
\[ \ldots \]

If there are \( n \) m’s and \( n \) w’s, what is the expected value of the M-rank and the W-rank when the proposal algorithm computes a stable matching?
What is the run time of the Stable Matching Algorithm?

Initially all m in M and w in W are free
While there is a free m
    w highest on m’s list that m has not proposed to
if w is free, then match (m, w)
else
    suppose (m₂, w) is matched
    if w prefers m to m₂
        unmatch (m₂, w)
    match (m, w)

Executed at most $n^2$ times
O(1) time per iteration

- Find free m
- Find next available w
- If w is matched, determine $m_2$
- Test if w prefers m to $m_2$
- Update matching
What does it mean for an algorithm to be efficient?
Key ideas

• Formalizing real world problem
  – Model: graph and preference lists
  – Mechanism: stability condition
• Specification of algorithm with a natural operation
  – Proposal
• Establishing termination of process through invariants and progress measure
• Under specification of algorithm
• Establishing uniqueness of solution
Five Problems
Theory of Algorithms

• What is expertise?
• How do experts differ from novices?
Introduction of five problems

• Show the types of problems we will be considering in the class
• Examples of important types of problems
• Similar looking problems with very different characteristics
• Problems
  – Scheduling
  – Weighted Scheduling
  – Bipartite Matching
  – Maximum Independent Set
  – Competitive Facility Location
What is a problem?

• Instance
• Solution
• Constraints on solution
• Measure of value
Problem: Scheduling

• Suppose that you own a banquet hall
• You have a series of requests for use of the hall: 
  \((s_1, f_1), (s_2, f_2), \ldots\)

  _______    _______    ___
  ___    _______    _______    ___
  ____    _______    _______    ___
  ___    ___    _______    __________

• Find a set of requests as large as possible with no overlap
What is the largest solution?

____  ____  ______

____  ____  ______  ____

____  ____  ____  ______

____  ____  _____  ____  ______

____  _____  ____  ______  ____

____  ___  ______  ____  ______

____  ____  ______  ____  ____  ____

____  ___  ____  ______  ____  ____

____  ____  ____  ______  ____  ____
Greedy Algorithm

• Test elements one at a time if they can be members of the solution
• If an element is not ruled out by earlier choices, add it to the solution
• Many possible choices for ordering (length, start time, end time)
• For this problem, considering the jobs by increasing end time works
Suppose we add values?

- \((s_i, f_i, v_i)\), start time, finish time, payment
- Maximize value of elements in the solution

\[
\begin{array}{ccc}
5 & 2 & 1 \\
1 & 2 & 4 & 1 \\
3 & 1 & 6 \\
\end{array}
\]
Greedy Algorithms

• Earliest finish time

• Maximum value

• Give counter examples to show these algorithms don’t find the maximum value solution
Dynamic Programming

- Requests $R_1$, $R_2$, $R_3$, . . .
- Assume requests are in increasing order of finish time ($f_1 < f_2 < f_3$ . . .)
- $\text{Opt}_i$ is the maximum value solution of $\{R_1, R_2, \ldots, R_i\}$ containing $R_i$
- $\text{Opt}_i = \max\{ j \mid f_j < s_i \}[\text{Opt}_j + v_i]$
Matching

- Given a bipartite graph $G=(U,V,E)$, find a subset of the edges $M$ of maximum size with no common endpoints.

- Application:
  - $U$: Professors
  - $V$: Courses
  - $(u,v)$ in $E$ if Prof. $u$ can teach course $v$
Find a maximum matching
Augmenting Path Algorithm
Reduction to network flow

• More general problem
• Send flow from source to sink
• Flow subject to capacities at edges
• Flow conserved at vertices
• Can solve matching as a flow problem
Maximum Independent Set

• Given an undirected graph $G=(V,E)$, find a set $I$ of vertices such that there are no edges between vertices of $I$
• Find a set $I$ as large as possible
Find a Maximum Independent Set
Verification: Prove the graph has an independent set of size 10.
Key characteristic

• Hard to find a solution
• Easy to verify a solution once you have one
• Other problems like this
  – Hamiltonian circuit
  – Clique
  – Subset sum
  – Graph coloring
NP-Completeness

• Theory of Hard Problems
• A large number of problems are known to be equivalent
• Very elegant theory
Are there even harder problems?

• Simple game:
  – Players alternating selecting nodes in a graph
    • Score points associated with node
    • Remove nodes neighbors
  – When neither can move, player with most points wins
Competitive Facility Location

• Choose location for a facility
  – Value associated with placement
  – Restriction on placing facilities too close together

• Competitive
  – Different companies place facilities
    • E.g., KFC and McDonald’s
Complexity theory

• These problems are P-Space complete instead of NP-Complete
  – Appear to be much harder
  – No obvious certificate
    • G has a Maximum Independent Set of size 10
    • Player 1 wins by at least 10 points
An NP-Complete problem from Digital Public Health

- ASHAs use Pico projectors to show health videos to Mothers’ groups
- Limited number of Pico projectors, so ASHAs must travel to where the Pico projector is stored
- Identify storage locations for $k$ Pico projectors to minimize the maximum distance an ASHA must travel
Summary

• Scheduling
• Weighted Scheduling
• Bipartite Matching
• Maximum Independent Set
• Competitive Scheduling