CSE 202
Algorithm basics II

Fan Chung Graham
UC San Diego

An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.
What is an algorithm?
“A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.” — webster.com

“An algorithm is a finite, definite, effective procedure, with some input and some output.”

— Donald Knuth
Euclidean algorithm:

Find the largest common factor between 36 and 123.

\[
\begin{array}{c|c}
36 & 3 \\
36 & 123 \\
108 & 15 \hline
15 & 36 \\
30 & 6 \hline
6 & 15 \\
12 & 3 \hline
3 & 2 \hline
3 \hline
6 & 6 \\
& 0 \\
\end{array}
\]

\[
123 = 3(36)+15 \\
36 = 2(15)+6 \\
15 = 2(6)+3 \\
6 = 2(3)
\]
Euclidean algorithm:

Find the largest common factor between \(a\) and \(b\).

Algorithm #1:

```plaintext
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    return a
```

Algorithm #2:

```plaintext
function gcd(a, b)
    while a ≠ b
        if a > b
            a := a - b
        else
            b := b - a
    return a
```
Algorithm analysis:

- ✔ Termination?
- ✔ Correctness?
- ✔ Efficiency?

Emphasizes critical thinking, problem-solving
Algorithm Analysis

- Worst case running time. Obtain bound on largest possible running time of algorithm on input of a given size $N$.
  - Generally captures efficiency in practice.
  - Draconian view, but hard to find effective alternative.

- Average case running time. Obtain bound on running time of algorithm on random input as a function of input size $N$.
  - Hard (or impossible) to accurately model real instances by random distributions.
  - Algorithm tuned for a certain distribution may perform poorly on other inputs.
Polynomial-Time

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor $C$.

There exists constants $c > 0$ and $d > 0$ such that on every input of size $N$, its running time is bounded by $cN^d$ steps.

A step. a single assembly-language instruction, one line of a programming language like C...

What happens if the input size increases from $N$ to $2N$?

Def. An algorithm is poly-time if the above scaling property holds.
Worst-Case Polynomial-Time

Def. An algorithm is efficient if its running time is polynomial.

Justification: It really works in practice!
- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.
- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used
Asymptotic Order of Growth

- We try to express that an algorithm's worst case running time is at most proportional to some function $f(n)$.
- Function $f(n)$ becomes a bound on the running time of the algorithm.

- Pseudo-code style.
  - counting the number of pseudo-code steps.
  - step. Assigning a value to a variable, looking up an entry in an array, following a pointer, a basic arithmetic operation...

"On any input size $n$, the algorithm runs for at most $1.62n^2 + 3.5n + 8$ steps."

Do we need such precise bound?
<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1^n</td>
<td>1 µsec</td>
<td>2 µsec</td>
<td>3 µsec</td>
<td>4 µsec</td>
<td>5 µsec</td>
<td>10 µsec</td>
<td>100 µsec</td>
</tr>
<tr>
<td>n^2</td>
<td>0.1 µsec</td>
<td>0.4 µsec</td>
<td>0.9 µsec</td>
<td>1.6 µsec</td>
<td>2.5 µsec</td>
<td>10 µsec</td>
<td>1 msec</td>
</tr>
<tr>
<td>n^3</td>
<td>1 µsec</td>
<td>8 µsec</td>
<td>27 µsec</td>
<td>64 µsec</td>
<td>0.13 msec</td>
<td>1 msec</td>
<td>1 sec</td>
</tr>
<tr>
<td>2^n</td>
<td>1 µsec</td>
<td>1 msec</td>
<td>1.1 sec</td>
<td>18.3 min</td>
<td>2.1 yr</td>
<td>2.4 x 10^3 cent</td>
<td>----</td>
</tr>
<tr>
<td>n!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
<table>
<thead>
<tr>
<th>size complexity</th>
<th>100n</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>1 µsec</td>
<td>2 µsec</td>
<td>3 µsec</td>
<td>4 µsec</td>
<td>5 µsec</td>
<td>10 µsec</td>
<td>100 µsec</td>
<td></td>
</tr>
<tr>
<td>n^2</td>
<td>.1 µsec</td>
<td>.4 µsec</td>
<td>.9 µsec</td>
<td>1.6 µsec</td>
<td>2.5 µsec</td>
<td>10 µsec</td>
<td>1 msec</td>
<td></td>
</tr>
<tr>
<td>n^3</td>
<td>1 µsec</td>
<td>8 µsec</td>
<td>27 µsec</td>
<td>64 µsec</td>
<td>.13 msec</td>
<td>1 msec</td>
<td>1 sec</td>
<td></td>
</tr>
<tr>
<td>2^n</td>
<td>1 µsec</td>
<td>1 msec</td>
<td>1.1 sec</td>
<td>18.3 min</td>
<td>2.1 yr</td>
<td>2.4 x 10^3 cent</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>n!</td>
<td>3.6 msec</td>
<td>1.8 yr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
</tr>
<tr>
<td>(n^2)</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
</tr>
<tr>
<td>(n^3)</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
</tr>
<tr>
<td>(2^n)</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
</tr>
<tr>
<td>(n!)</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
<td>(\mu\text{sec})</td>
</tr>
</tbody>
</table>

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>(\text{µsec})</td>
<td>(\text{µsec})</td>
<td>(\text{µsec})</td>
<td>(\text{µsec})</td>
<td>(\text{µsec})</td>
<td>(\text{µsec})</td>
<td>(\text{µsec})</td>
</tr>
<tr>
<td>(n^2)</td>
<td>0.1</td>
<td>0.4</td>
<td>0.9</td>
<td>1.6</td>
<td>2.5</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>(n^3)</td>
<td>0.1</td>
<td>0.8</td>
<td>2.7</td>
<td>6.4</td>
<td>0.13</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(2^n)</td>
<td>0.1</td>
<td>1</td>
<td>1.1</td>
<td>18.3</td>
<td>2.1</td>
<td>2.4x10^3</td>
<td>----</td>
</tr>
<tr>
<td>(n!)</td>
<td>3.6</td>
<td>1.8</td>
<td>2.0x10^{15}</td>
<td>15</td>
<td>!</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
<table>
<thead>
<tr>
<th>size complexity</th>
<th>n</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>µsec</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 µsec</td>
<td>2 µsec</td>
<td>3 µsec</td>
<td>4 µsec</td>
<td>5 µsec</td>
<td>10 µsec</td>
<td>100 µsec</td>
<td></td>
</tr>
<tr>
<td>n^2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.1 µsec</td>
<td>.4 µsec</td>
<td>.9 µsec</td>
<td>1.6 µsec</td>
<td>2.5 µsec</td>
<td>10 µsec</td>
<td>1 msec</td>
<td></td>
</tr>
<tr>
<td>n^3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 µsec</td>
<td>8 µsec</td>
<td>27 µsec</td>
<td>64 µsec</td>
<td>.13 msec</td>
<td>1 msec</td>
<td>1 sec</td>
<td></td>
</tr>
<tr>
<td>2^n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 µsec</td>
<td>1 msec</td>
<td>1.1 sec</td>
<td>18.3 min</td>
<td>2.1 yr</td>
<td>2.4x10^13 cent</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>n!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.6 msec</td>
<td>1.8 yr</td>
<td>2.0x10 ^15 min</td>
<td>!</td>
<td>forget it</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
<table>
<thead>
<tr>
<th>size complexity</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>1 µsec</td>
<td>2 µsec</td>
<td>3 µsec</td>
<td>4 µsec</td>
<td>5 µsec</td>
<td>10 µsec</td>
<td>100 µsec</td>
</tr>
<tr>
<td>n^2</td>
<td>.1 µsec</td>
<td>.4 µsec</td>
<td>.9 µsec</td>
<td>1.6 µsec</td>
<td>2.5 µsec</td>
<td>10 µsec</td>
<td>1 msec</td>
</tr>
<tr>
<td>n^3</td>
<td>1 µsec</td>
<td>8 µsec</td>
<td>27 µsec</td>
<td>64 µsec</td>
<td>.13 msec</td>
<td>1 msec</td>
<td>1 sec</td>
</tr>
<tr>
<td>2^n</td>
<td>1 µsec</td>
<td>1 msec</td>
<td>1.1 sec</td>
<td>18.3 min</td>
<td>2.1 yr</td>
<td>2.4x10^3 cent</td>
<td>----</td>
</tr>
</tbody>
</table>
| n!              | 3.6 msec | 1.8 yr | 2.0x10^15 cent | ! | forget it | ? |}

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^1$</td>
<td>1 µsec</td>
<td>2 µsec</td>
<td>3 µsec</td>
<td>4 µsec</td>
<td>5 µsec</td>
<td>10 µsec</td>
<td>100 µsec</td>
</tr>
<tr>
<td>$n^2$</td>
<td>0.1 sec</td>
<td>0.4 sec</td>
<td>0.9 sec</td>
<td>1.6 µsec</td>
<td>2.5 µsec</td>
<td>10 µsec</td>
<td>1 msec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1 µsec</td>
<td>8 µsec</td>
<td>27 µsec</td>
<td>64 µsec</td>
<td>0.13 msec</td>
<td>1 msec</td>
<td>1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1 µsec</td>
<td>1 msec</td>
<td>1.1 sec</td>
<td>18.3 min</td>
<td>2.1 yr</td>
<td>2.4x10^3 cent</td>
<td>----</td>
</tr>
<tr>
<td>$n!$</td>
<td>3.6 msec</td>
<td>1.8 yr</td>
<td>2.0x10^15 cent</td>
<td></td>
<td>forget it</td>
<td>?</td>
<td>∞</td>
</tr>
</tbody>
</table>

**Polynomial vs. exponential growth**

(assuming 1,000,000,000 operations per second)
Undirected Graphs

Undirected graph. \( G = (V, E) \)

- \( V \) = nodes.
- \( E \) = edges between pairs of nodes.
- Captures pairwise relationship between objects.
- Graph size parameters: \( n = |V|, m = |E| \).

\[
V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}
\]
\[
E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 \}
\]
\[
n = 8
\]
\[
m = 11
\]
## Some Graph Applications

<table>
<thead>
<tr>
<th><strong>Graph</strong></th>
<th><strong>Nodes</strong></th>
<th><strong>Edges</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>transportation</td>
<td>street intersections</td>
<td>highways</td>
</tr>
<tr>
<td>communication</td>
<td>computers</td>
<td>fiber optic cables</td>
</tr>
<tr>
<td>World Wide Web</td>
<td>web pages</td>
<td>hyperlinks</td>
</tr>
<tr>
<td>social</td>
<td>people</td>
<td>relationships</td>
</tr>
<tr>
<td>food web</td>
<td>species</td>
<td>predator-prey</td>
</tr>
<tr>
<td>software systems</td>
<td>functions</td>
<td>function calls</td>
</tr>
<tr>
<td>scheduling</td>
<td>tasks</td>
<td>precedence constraints</td>
</tr>
<tr>
<td>circuits</td>
<td>gates</td>
<td>wires</td>
</tr>
</tbody>
</table>
World Wide Web

Web graph.
- Node: web page.
- Edge: hyperlink from one page to another.
Ecological Food Web

Food web graph.
- Node = species.
- Edge = from prey to predator.
Graph Representation: Adjacency Matrix

Adjacency matrix. n-by-n matrix with $A_{uv} = 1$ if $(u, v)$ is an edge.
- Two representations of each edge.
- Space proportional to $n^2$.
- Checking if $(u, v)$ is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.
Graph Representation: Adjacency List

Adjacency list. Node indexed array of lists.
- Two representations of each edge.
- Space proportional to $m + n$.
- Checking if $(u, v)$ is an edge takes $O(\text{deg}(u))$ time.
- Identifying all edges takes $\Theta(m + n)$ time.
**Paths and Connectivity**

Def. A path in an undirected graph \( G = (V, E) \) is a sequence \( P \) of nodes \( v_1, v_2, ..., v_{k-1}, v_k \) with the property that each consecutive pair \( v_i, v_{i+1} \) is joined by an edge in \( E \).

Def. A path is **simple** if all nodes are distinct.

Def. An undirected graph is **connected** if for every pair of nodes \( u \) and \( v \), there is a path between \( u \) and \( v \).
Def. A cycle is a path \( v_1, v_2, \ldots, v_{k-1}, v_k \) in which \( v_1 = v_k, \ k > 2 \), and the first \( k-1 \) nodes are all distinct.

Cycle \( C = 1-2-4-5-3-1 \)
Def. An undirected graph is a tree if it is connected and does not contain a cycle.

Theorem. Let $G$ be an undirected graph on $n$ nodes. Any two of the following statements imply the third.

- $G$ is connected.
- $G$ does not contain a cycle.
- $G$ has $n-1$ edges.
Rooted Trees

Rooted tree. Given a tree $T$, choose a root node $r$ and orient each edge away from $r$.

Importance. Models hierarchical structure.
Connectivity

s-t connectivity problem. Given two node s and t, is there a path between s and t?

s-t shortest path problem. Given two node s and t, what is the length of the shortest path between s and t?

Applications.

- Friendster.
- Maze traversal.
- Kevin Bacon number.
- Fewest number of hops in a communication network.
Breadth First Search

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.

BFS algorithm.
- \( L_0 = \{ s \} \).
- \( L_1 \) = all neighbors of \( L_0 \).
- \( L_2 \) = all nodes that do not belong to \( L_0 \) or \( L_1 \), and that have an edge to a node in \( L_1 \).
- \( L_{i+1} \) = all nodes that do not belong to an earlier layer, and that have an edge to a node in \( L_i \).

Theorem. For each \( i \), \( L_i \) consists of all nodes at distance exactly \( i \) from \( s \). There is a path from \( s \) to \( t \) iff \( t \) appears in some layer.
**Connected Component**

*Connected component.* Find all nodes reachable from $s$.

*Connected component containing node 1* = \{1, 2, 3, 4, 5, 6, 7, 8\}.

**Algorithms:**  
Breadth First Search  BFS  
Depth First Search  DFS
CSE 202
Matching algorithms

Fan Chung Graham
UC San Diego

An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.
1. **Representative Problems**

- stable matching
- five representative problems
Matching med-school students to hospitals

**Goal.** Given a set of preferences among hospitals and med-school students, design a *self-reinforcing* admissions process.

**Unstable pair:** student $x$ and hospital $y$ are *unstable* if:
- $x$ prefers $y$ to its assigned hospital.
- $y$ prefers $x$ to one of its admitted students.

**Stable assignment.** Assignment with no unstable pairs.
- Natural and desirable condition.
- Individual self-interest prevents any hospital–student side deal.
Stable matching problem

**Goal.** Given a set of $n$ men and a set of $n$ women, find a "suitable" matching.

- Participants rank members of opposite sex.
- Each man lists women in order of preference from best to worst.
- Each woman lists men in order of preference from best to worst.

<table>
<thead>
<tr>
<th>favorite</th>
<th>least favorite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
</tr>
<tr>
<td>Xavier</td>
<td>Amy</td>
</tr>
<tr>
<td>Yancey</td>
<td>Bertha</td>
</tr>
<tr>
<td>Zeus</td>
<td>Amy</td>
</tr>
</tbody>
</table>

*men's preference list*

<table>
<thead>
<tr>
<th>favorite</th>
<th>least favorite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
</tr>
<tr>
<td>Amy</td>
<td>Yancey</td>
</tr>
<tr>
<td>Bertha</td>
<td>Xavier</td>
</tr>
<tr>
<td>Clare</td>
<td>Xavier</td>
</tr>
</tbody>
</table>

*women's preference list*
**Perfect matching**

**Def.** A matching $S$ is a set of ordered pairs $m–w$ with $m \in M$ and $w \in W$ s.t.
- Each man $m \in M$ appears in at most one pair of $S$.
- Each woman $w \in W$ appears in at most one pair of $S$.

**Def.** A matching $S$ is **perfect** if $|S| = |M| = |W| = n$.

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xavier</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
<tr>
<td>Yancey</td>
<td>Bertha</td>
<td>Amy</td>
</tr>
<tr>
<td>Zeus</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Yancey</td>
<td>Xavier</td>
</tr>
<tr>
<td>Bertha</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
<tr>
<td>Clare</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
</tbody>
</table>

*a perfect matching $S = \{ X–C, Y–B, Z–A \}$*
Unstable pair

**Def.** Given a perfect matching $S$, man $m$ and woman $w$ are unstable if:
- $m$ prefers $w$ to his current partner.
- $w$ prefers $m$ to her current partner.

**Key point.** An unstable pair $m$–$w$ could each improve partner by joint action.

Bertha and Xavier are an unstable pair
Stable matching problem

**Def.** A **stable matching** is a perfect matching with no unstable pairs.

**Stable matching problem.** Given the preference lists of $n$ men and $n$ women, find a stable matching (if one exists).

- Natural, desirable, and self-reinforcing condition.
- Individual self-interest prevents any man–woman pair from eloping.

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xavier</td>
<td>Amy</td>
<td>Bertha</td>
<td>Clare</td>
</tr>
<tr>
<td>Yancey</td>
<td>Bertha</td>
<td>Amy</td>
<td>Clare</td>
</tr>
<tr>
<td>Zeus</td>
<td>Amy</td>
<td>Bertha</td>
<td>Clare</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Yancey</td>
<td>Xavier</td>
<td>Zeus</td>
</tr>
<tr>
<td>Bertha</td>
<td>Xavier</td>
<td>Yancey</td>
<td>Zeus</td>
</tr>
<tr>
<td>Clare</td>
<td>Xavier</td>
<td>Yancey</td>
<td>Zeus</td>
</tr>
</tbody>
</table>

*a perfect matching $S = \{ X-A, Y-B, Z-C \}$*
Stable roommate problem

Q. Do stable matchings always exist?
A. Not obvious a priori.

Stable roommate problem.
- 2n people; each person ranks others from 1 to 2n – 1.
- Assign roommate pairs so that no unstable pairs.

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>Bob</td>
<td>C</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>Chris</td>
<td>A</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>Doofus</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

**Observation.** Stable matchings need not exist for stable roommate problem.
Gale-Shapley deferred acceptance algorithm

An intuitive method that guarantees to find a stable matching.

**Gale–Shapley (preference lists for men and women)**

**Initialize** $S$ to empty matching.

**While** (some man $m$ is unmatched and hasn't proposed to every woman)

1. $w \leftarrow$ first woman on $m$'s list to whom $m$ has not yet proposed.

2. **If** ($w$ is unmatched)
   
   Add pair $m$–$w$ to matching $S$.

3. **Else if** ($w$ prefers $m$ to her current partner $m'$)

   Remove pair $m'$–$w$ from matching $S$.
   
   Add pair $m$–$w$ to matching $S$.

4. **Else**

   $w$ rejects $m$.

**Return** stable matching $S$. 
Proof of correctness: termination

**Observation 1.** Men propose to women in decreasing order of preference.

**Observation 2.** Once a woman is matched, she never becomes unmatched; she only "trades up."

**Claim.** Algorithm terminates after at most $n^2$ iterations of while loop.

**Pf.** Each time through the while loop a man proposes to a new woman. There are only $n^2$ possible proposals. □

\[
\text{n}(n-1) + 1 \text{ proposals required}
\]
Proof of correctness: perfection

**Claim.** In Gale-Shapley matching, all men and women get matched.

**Pf.** [by contradiction]

- Suppose, for sake of contradiction, that Zeus is not matched upon termination of GS algorithm.
- Then some woman, say Amy, is not matched upon termination.
- By Observation 2, Amy was never proposed to.
- But, Zeus proposes to everyone, since he ends up unmatched. □
Proof of correctness: stability

Claim. In Gale-Shapley matching, there are no unstable pairs.

Pf. Suppose the GS matching $S^*$ does not contain the pair $A-Z$.

- Case 1: $Z$ never proposed to $A$.
  - $\Rightarrow$ $Z$ prefers his GS partner $B$ to $A$.
  - $\Rightarrow$ $A-Z$ is stable.

- Case 2: $Z$ proposed to $A$.
  - $\Rightarrow$ $A$ rejected $Z$ (right away or later)
  - $\Rightarrow$ $A$ prefers her GS partner $Y$ to $Z$.
  - $\Rightarrow$ $A-Z$ is stable.

- In either case, the pair $A-Z$ is stable. ■
Summary

Stable matching problem. Given $n$ men and $n$ women, and their preferences, find a stable matching if one exists.


Q. How to implement GS algorithm efficiently?
Q. If there are multiple stable matchings, which one does GS find?
Efficient implementation

Efficient implementation. We describe an $O(n^2)$ time implementation.

Representing men and women.

- Assume men are named 1, ..., $n$.
- Assume women are named 1', ..., $n'$.

Representing the matching.

- Maintain a list of free men (in a stack or queue).
- Maintain two arrays $wife[m]$ and $husband[w]$.
  - if $m$ matched to $w$, then $wife[m] = w$ and $husband[w] = m$
  - set entry to 0 if unmatched

Men proposing.

- For each man, maintain a list of women, ordered by preference.
- For each man, maintain a pointer to woman in list for next proposal.
Women rejecting/accepting.

- Does woman $w$ prefer man $m$ to man $m'$?
- For each woman, create inverse of preference list of men.
- Constant time access for each query after $O(n)$ preprocessing.

```
for i = 1 to n
    inverse[pref[i]] = i
```
Understanding the solution

For a given problem instance, there may be several stable matchings.

- Do all executions of GS algorithm yield the same stable matching?
- If so, which one?

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xavier</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
<tr>
<td>Yancey</td>
<td>Bertha</td>
<td>Amy</td>
</tr>
<tr>
<td>Zeus</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Yancey</td>
<td>Xavier</td>
</tr>
<tr>
<td>Bertha</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
<tr>
<td>Clare</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
</tbody>
</table>

an instance with two stable matching: $M = \{A-X, B-Y, C-Z\}$ and $M' = \{A-Y, B-X, C-Z\}$
Understanding the solution

**Def.** Woman $w$ is a **valid partner** of man $m$ if there exists some stable matching in which $m$ and $w$ are matched.

**Ex.**
- Both Amy and Bertha are valid partners for Xavier.
- Both Amy and Bertha are valid partners for Yancey.
- Clare is the only valid partner for Zeus.

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xavier</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
<tr>
<td>Yancey</td>
<td>Bertha</td>
<td>Amy</td>
</tr>
<tr>
<td>Zeus</td>
<td>Amy</td>
<td>Bertha</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Yancey</td>
<td>Xavier</td>
</tr>
<tr>
<td>Bertha</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
<tr>
<td>Clare</td>
<td>Xavier</td>
<td>Yancey</td>
</tr>
</tbody>
</table>

an instance with two stable matching: $M = \{ \text{A–X, B–Y, C–Z} \}$ and $M' = \{ \text{A–Y, B–X, C–Z} \}$
Understanding the solution

**Def.** Woman \( w \) is a valid partner of man \( m \) if there exists some stable matching in which \( m \) and \( w \) are matched.

**Man-optimal assignment.** Each man receives best valid partner.
- Is it perfect?
- Is it stable?

**Claim.** All executions of GS yield man-optimal assignment.

**Corollary.** Man-optimal assignment is a stable matching!
Man optimality

Claim. GS matching $S^*$ is man-optimal.

Pf. [by contradiction]

• Suppose a man is matched with someone other than best valid partner.
• Men propose in decreasing order of preference
  $\Rightarrow$ some man is rejected by valid partner during GS.
• Let $Y$ be first such man, and let $A$ be the first valid woman that rejects him.
• Let $S$ be a stable matching where $A$ and $Y$ are matched.
• When $Y$ is rejected by $A$ in GS, $A$ forms (or reaffirms) engagement with a man, say $Z$.
  $\Rightarrow$ **A prefers $Z$ to $Y$.**
• Let $B$ be partner of $Z$ in $S$.
• $Z$ has not been rejected by any valid partner (including $B$) at the point when $Y$ is rejected by $A$.
• Thus, $Z$ has not yet proposed to $B$ when he proposes to $A$.
  $\Rightarrow$ **Z prefers $A$ to $B$.**
• Thus $A$–$Z$ is unstable in $S$, a contradiction. $\blacksquare$
**Woman pessimality**

**Q.** Does man-optimality come at the expense of the women?
**A.** Yes.

**Woman-pessimal assignment.** Each woman receives worst valid partner.

**Claim.** GS finds \textit{woman-pessimal} stable matching $S^*$.  

**Pf.** [by contradiction]

- Suppose $A-Z$ matched in $S^*$ but $Z$ is not worst valid partner for $A$.
- There exists stable matching $S$ in which $A$ is paired with a man, say $Y$, whom she likes less than $Z$.
  \[ \Rightarrow A \text{ prefers } Z \text{ to } Y. \]
- Let $B$ be the partner of $Z$ in $S$. By man-optimality, $A$ is the best valid partner for $Z$.
  \[ \Rightarrow Z \text{ prefers } A \text{ to } B. \]
- Thus, $A-Z$ is an unstable pair in $S$, a contradiction. \[ \square \]
Deceit:  Machiavelli meets Gale-Shapley

**Q.** Can there be an incentive to misrepresent your preference list?
- Assume you know men’s propose-and-reject algorithm will be run.
- Assume preference lists of all other participants are known.

**Fact.** No, for any man; yes, for some women.

<table>
<thead>
<tr>
<th>men's preference list</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Y</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>women's preference list</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

**Amy lies**

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>
Extensions: matching residents to hospitals

Ex: Men \approx hospitals, Women \approx med school residents.

Variant 1. Some participants declare others as unacceptable.

Variant 2. Unequal number of men and women.

Variant 3. Limited polygamy.

Def. Matching is \textit{S unstable} if there is a hospital $h$ and resident $r$ such that:

\begin{itemize}
  \item $h$ and $r$ are acceptable to each other; and
  \item Either $r$ is unmatched, or $r$ prefers $h$ to her assigned hospital; and
  \item Either $h$ does not have all its places filled, or $h$ prefers $r$ to at least one of its assigned residents.
\end{itemize}
Historical context

National resident matching program (NRMP).

• Centralized clearinghouse to match med-school students to hospitals.
• Began in 1952 to fix unraveling of offer dates.
• Originally used the "Boston Pool" algorithm.
• Algorithm overhauled in 1998.
  - med-school student optimal
  - deals with various side constraints (e.g., allow couples to match together)
• 38,000+ residents for 26,000+ positions.
2012 Nobel Prize in Economics

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

**Alvin Roth.** Applied Gale-Shapley to matching new doctors with hospitals, students with schools, and organ donors with patients.
Lessons learned

Powerful ideas learned in course.
• Isolate underlying structure of problem.
• Create useful and efficient algorithms.

Potentially deep social ramifications. [legal disclaimer]
• Historically, men propose to women. Why not vice versa?
• Men: propose early and often; be honest.
• Women: ask out the men.
• Theory can be socially enriching and fun!
• COS majors get the best partners (and jobs)!
1. **Representative Problems**

- stable matching
- five representative problems
Interval scheduling

**Input.** Set of jobs with start times and finish times.

**Goal.** Find maximum cardinality subset of mutually compatible jobs.

jobs don't overlap
Weighted interval scheduling

**Input.** Set of jobs with start times, finish times, and weights.

**Goal.** Find maximum weight subset of mutually compatible jobs.
Bipartite matching

**Problem.** Given a bipartite graph $G = (L \cup R, E)$, find a max cardinality matching.

**Def.** A subset of edges $M \subseteq E$ is a matching if each node appears in exactly one edge in $M$. 
Independent set

**Problem.** Given a graph $G = (V, E)$, find a max cardinality independent set.

**Def.** A subset $S \subseteq V$ is **independent** if for every $(u, v) \in E$, either $u \notin S$ or $v \notin S$ (or both).
Competitive facility location

Input. Graph with weight on each node.

Game. Two competing players alternate in selecting nodes. Not allowed to select a node if any of its neighbors have been selected.

Goal. Select a **maximum weight** subset of nodes.

Second player can guarantee 20, but not 25.
Five representative problems

Variations on a theme: independent set.

Interval scheduling: $O(n \log n)$ greedy algorithm.
Weighted interval scheduling: $O(n \log n)$ dynamic programming algorithm.
Bipartite matching: $O(n^k)$ max-flow based algorithm.
Independent set: NP-complete.
Competitive facility location: PSPACE-complete.