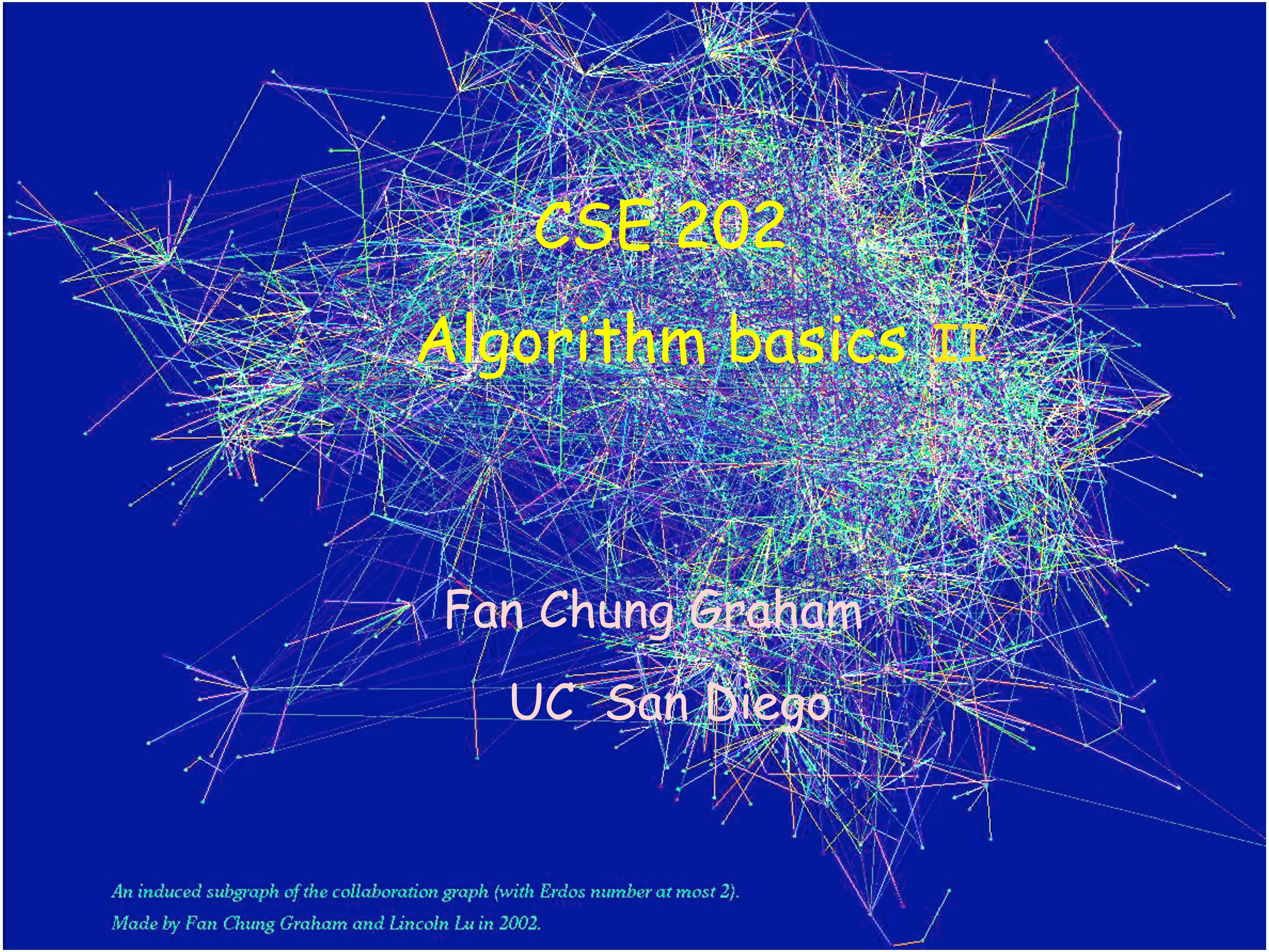
CSE 202

Algorithm basics II

Fan Chung Graham

UC San Diego



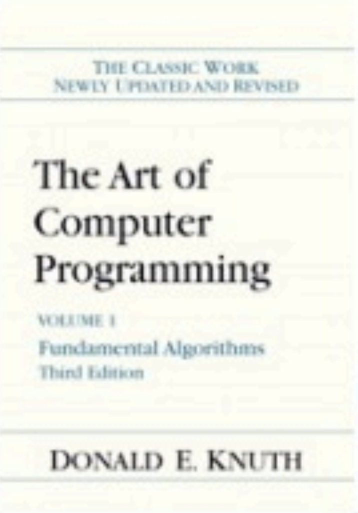
An induced subgraph of the collaboration graph (with Erdos number at most 2). Made by Fan Chung Graham and Lincoln Lu in 2002.

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*

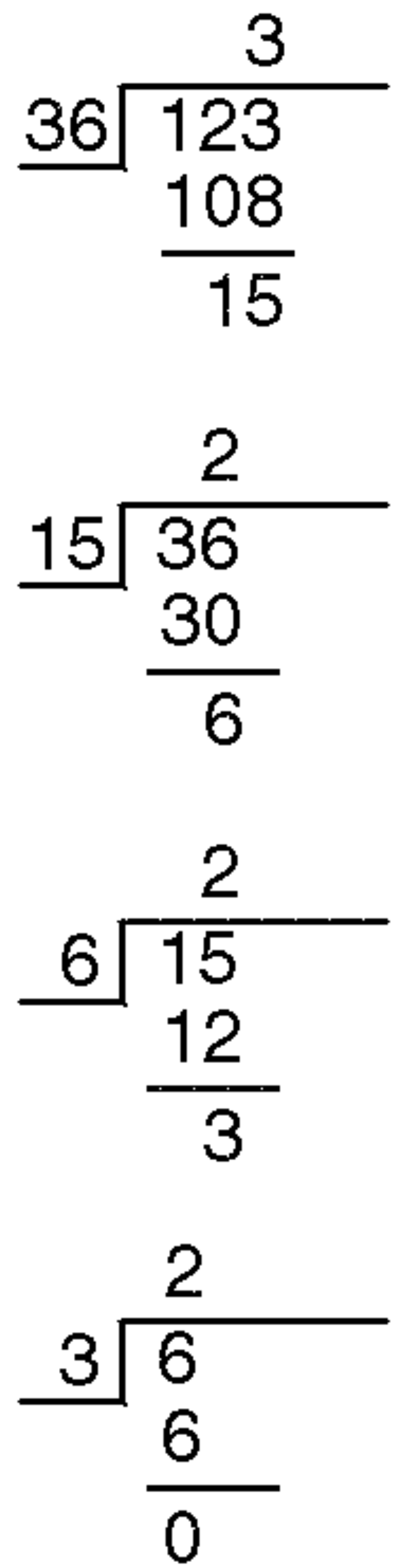
" An algorithm is a finite, definite, effective procedure, with some input and some output. "

— *Donald Knuth*



Euclidean algorithm:

Find the largest common factor between 36 and 123.



$$123 = 3(36)+15$$

$$36 = 2(15)+6$$

$$15 = 2(6)+3$$

$$6 = 2(3)$$

Euclidean algorithm:

Find the largest common factor between a and b.

Algorithm #1:

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

Algorithm #2:

```
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 algorithr 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?

size / complexity	n: 10	20	30	40	50	100	1000
$100n$	1 μsec	2 μsec	3 μsec	4 μsec	5 μsec	10 μsec	100 μsec
n^2	.1 μsec	.4 μsec	.9 μsec	1.6 μsec	2.5 μsec	10 μsec	1 msec
n^3	1 μsec	8 μsec	27 μsec	64 μsec	.13 msec	1 msec	1 sec
2^n	1 μsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
$n!$							

Polynomial vs. **exponential** growth

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

complexity \ size	n = 10	20	30	40	50	100	1000
100 n	1 μsec	2 μsec	3 μsec	4 μsec	5 μsec	10 μsec	100 μsec
n^2	.1 μsec	.4 μsec	.9 μsec	1.6 μsec	2.5 μsec	10 μsec	1 msec
n^3	1 μsec	8 μsec	27 μsec	64 μsec	.13 msec	1 msec	1 sec
2^n	1 μsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
n!	3.6 msec	1.8 yr					

Polynomial vs. **exponential** growth

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

complexity \ size	10	20	30	40	50	100	1000
100 n	1 μsec	2 μsec	3 μsec	4 μsec	5 μsec	10 μsec	100 μsec
n^2	.1 μsec	.4 μsec	.9 μsec	1.6 μsec	2.5 μsec	10 μsec	1 msec
n^3	1 μsec	8 μsec	27 μsec	64 μsec	.13 msec	1 msec	1 sec
2^n	1 μsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
$n!$	3.6 msec	1.8 yr	2.0×10^{15} cent				

Polynomial vs. **exponential** growth

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

complexity \ size	10	20	30	40	50	100	1000
100 n	1 µsec	2 µsec	3 µsec	4 µsec	5 µsec	10 µsec	100 µsec
n^2	.1 µsec	.4 µsec	.9 µsec	1.6 µsec	2.5 µsec	10 µsec	1 msec
n^3	1 µsec	8 µsec	27 µsec	64 µsec	.13 msec	1 msec	1 sec
2^n	1 µsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
n!	3.6 msec	1.8 yr	2.0×10^{15} cent	!			

Polynomial vs. **exponential** growth

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

size / complexity	n = 10	20	30	40	50	100	1000
100n	1 μsec	2 μsec	3 μsec	4 μsec	5 μsec	10 μsec	100 μsec
n^2	.1 μsec	.4 μsec	.9 μsec	1.6 μsec	2.5 μsec	10 μsec	1 msec
n^3	1 μsec	8 μsec	27 μsec	64 μsec	.13 msec	1 msec	1 sec
2^n	1 μsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
n!	3.6 msec	1.8 yr	2.0×10^{15} cent	!	forget it		

Polynomial vs. **exponential** growth

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

complexity \ size	n = 10	20	30	40	50	100	1000
100n	1 μsec	2 μsec	3 μsec	4 μsec	5 μsec	10 μsec	100 μsec
n^2	.1 μsec	.4 μsec	.9 μsec	1.6 μsec	2.5 μsec	10 μsec	1 msec
n^3	1 μsec	8 μsec	27 μsec	64 μsec	.13 msec	1 msec	1 sec
2^n	1 μsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
n!	3.6 msec	1.8 yr	2.0×10^{15} cent	!	forget it	?	

Polynomial vs. **exponential** growth

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

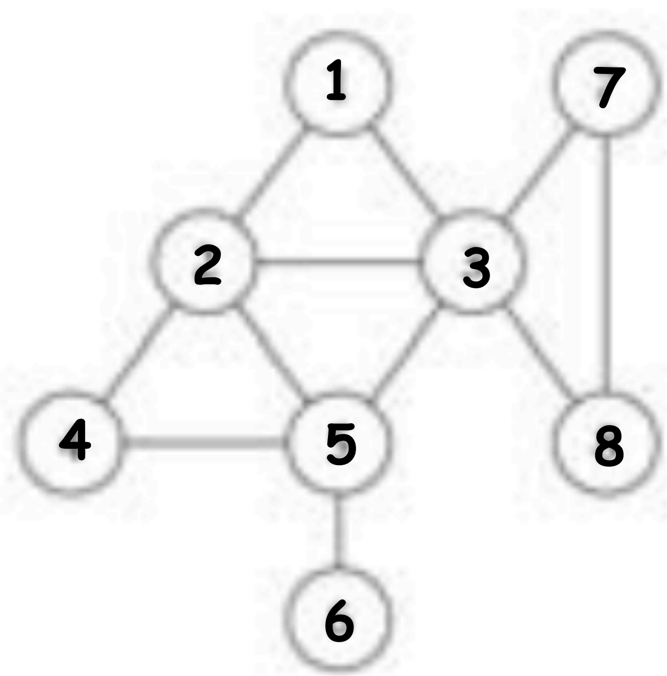
complexity \ size (n)	10	20	30	40	50	100	1000
$100n$	1 μsec	2 μsec	3 μsec	4 μsec	5 μsec	10 μsec	100 μsec
n^2	.1 μsec	.4 μsec	.9 μsec	1.6 μsec	2.5 μsec	10 μsec	1 msec
n^3	1 μsec	8 μsec	27 μsec	64 μsec	.13 msec	1 msec	1 sec
2^n	1 μsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
$n!$	3.6 msec	1.8 yr	2.0×10^{15} cent	!	forget it	?	∞

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: $n = |V|$, $m = |E|$.



$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

$E = \{ 1\text{-}2, 1\text{-}3, 2\text{-}3, 2\text{-}4, 2\text{-}5, 3\text{-}5, 3\text{-}7, 3\text{-}8, 4\text{-}5, 5\text{-}6 \}$

$n = 8$

$m = 11$

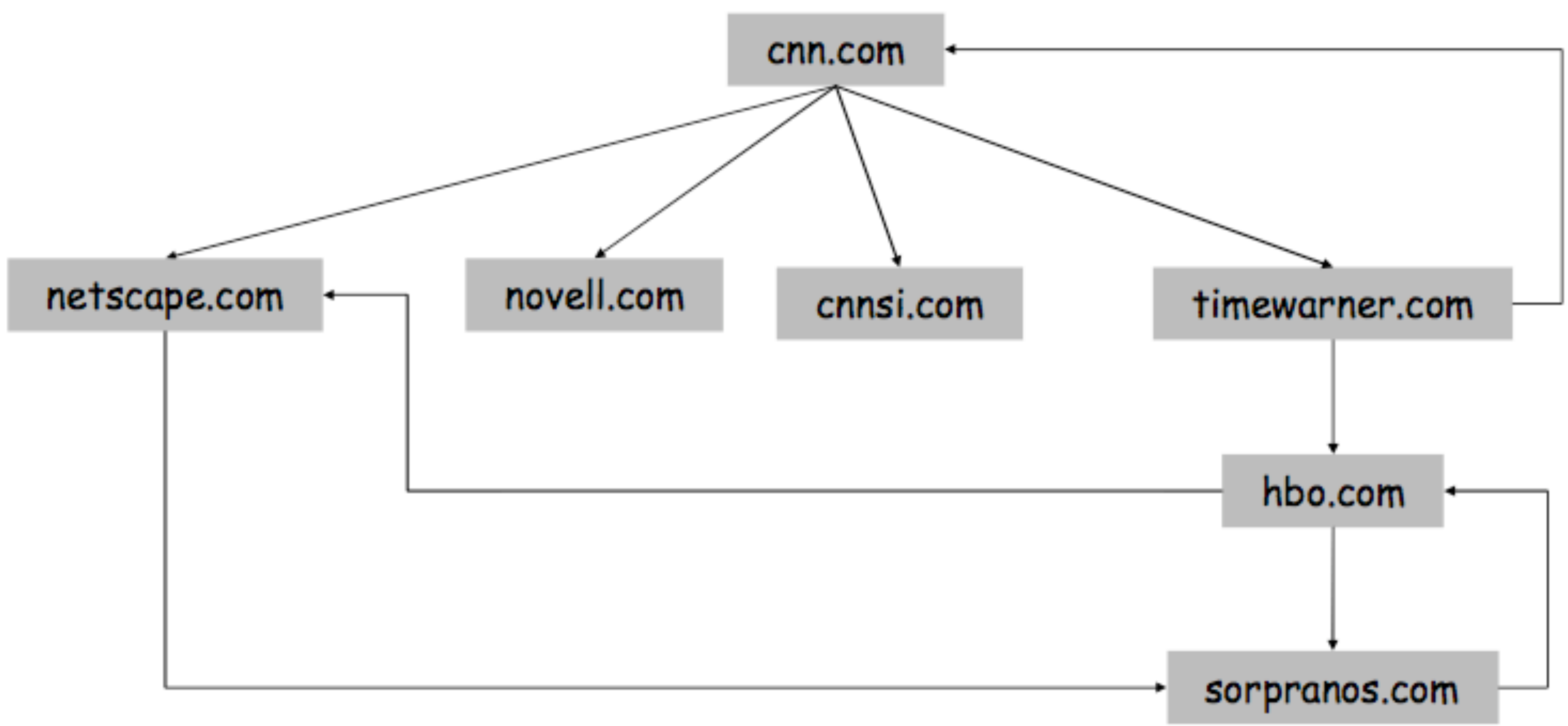
Some Graph Applications

Graph	*Nodes*	*Edges*
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

World Wide Web

Web graph.

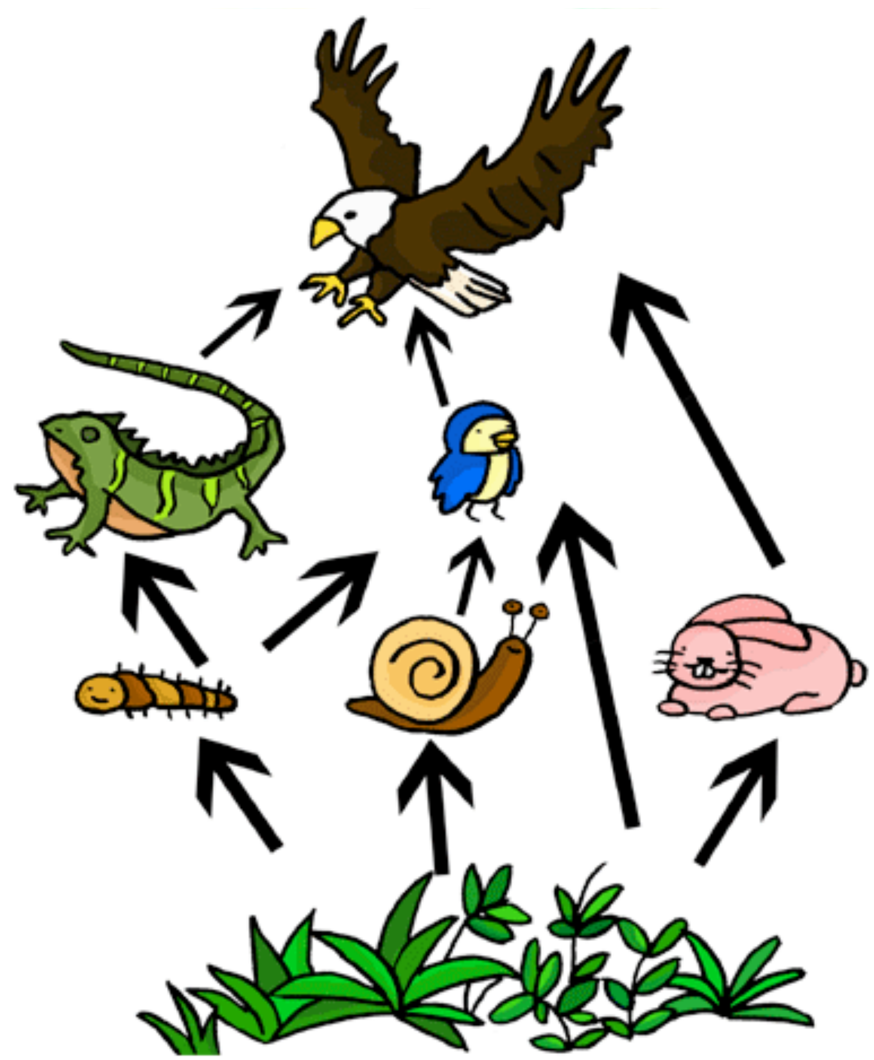
- Node: web page.
- Edge: hyperlink from one page to another.



Ecological Food Web

Food web graph.

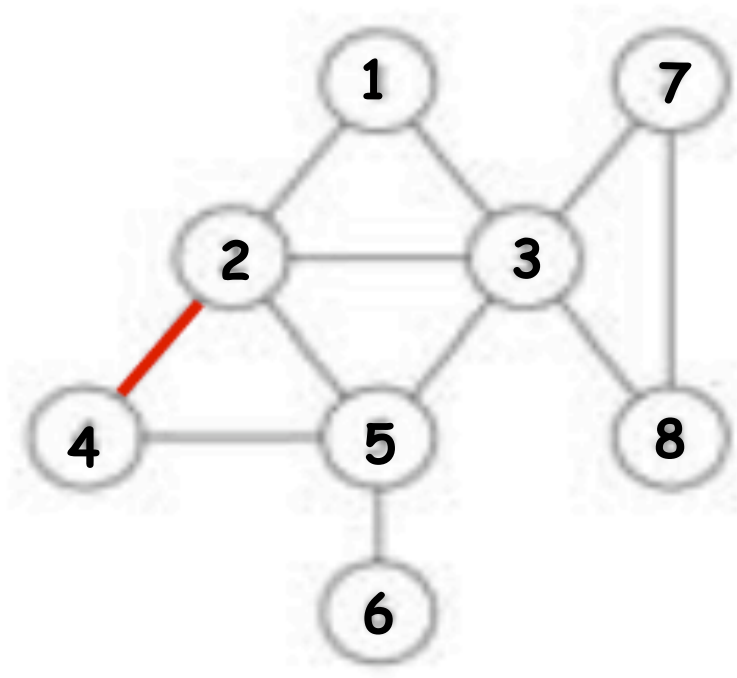
- Node = species.
- Edge = from prey to predator.



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.



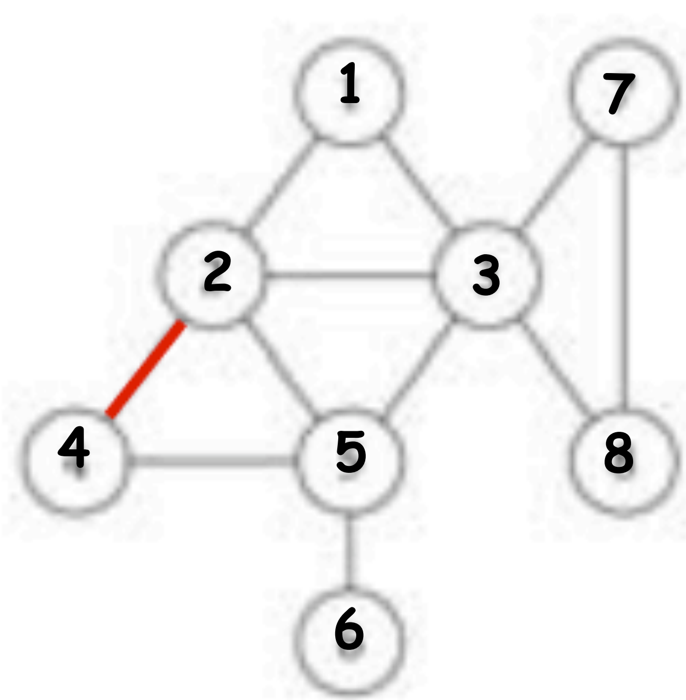
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

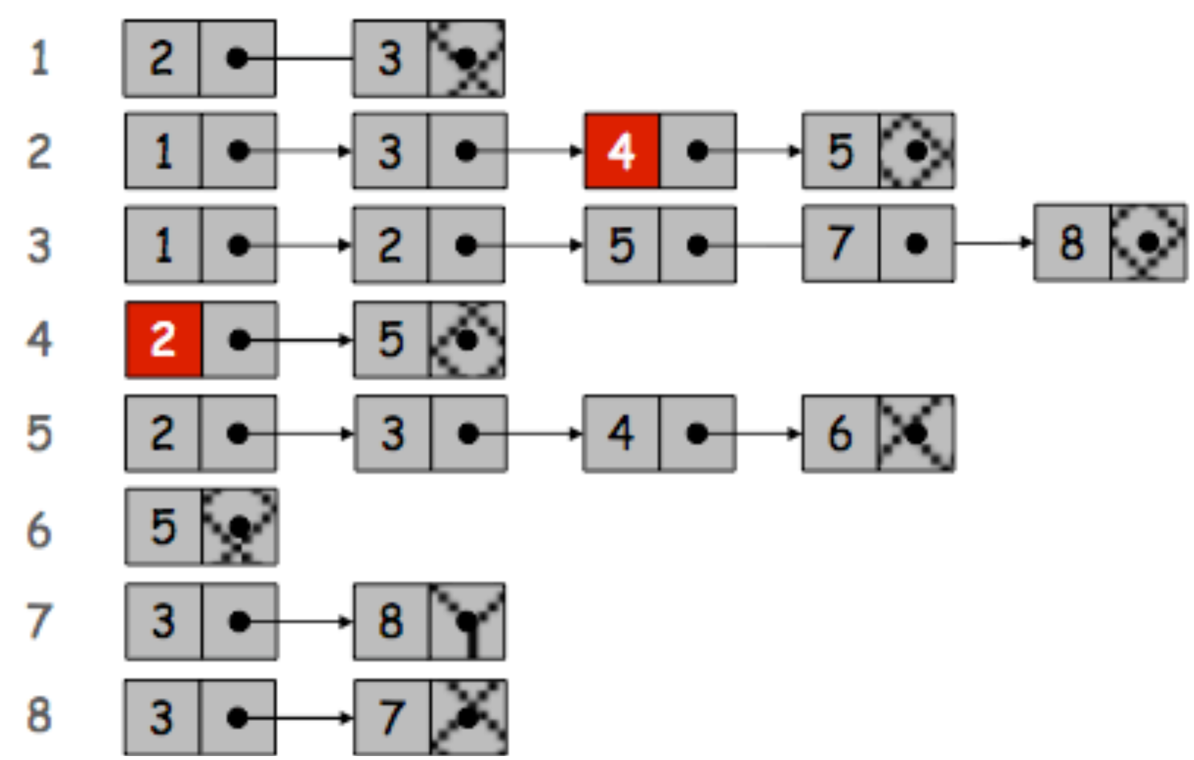
Graph Representation: Adjacency List

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(deg(u)) time.
- Identifying all edges takes Θ(m + n) time.

degree = number of neighbors of u



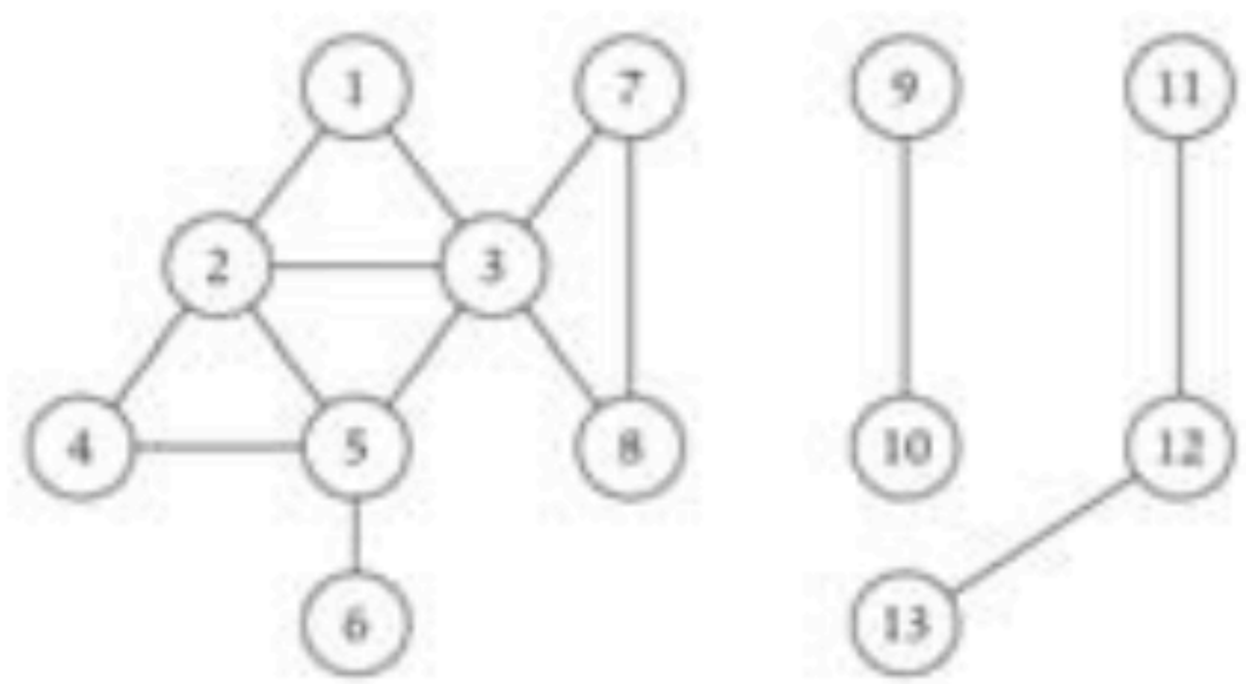


Paths and Connectivity

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

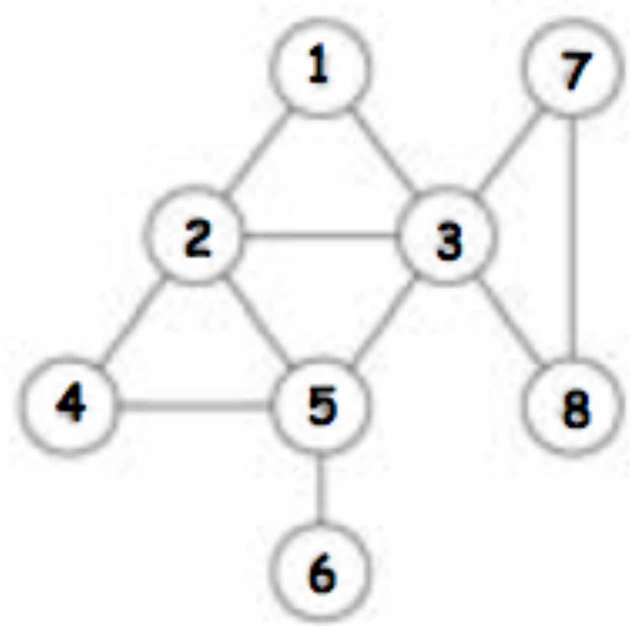
Def. A path is simple if all nodes are distinct.

Def. An undirected graph is connected if for every pair of nodes u and v, there is a path between u and v.



Cycles

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 th
first k-1 nodes are all distinct.



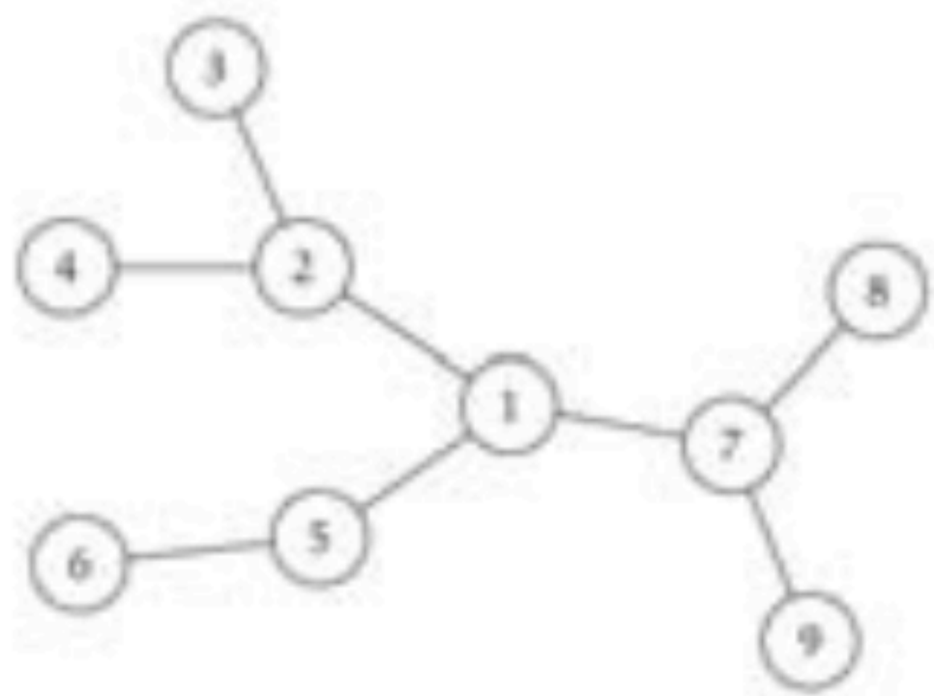
cycle C = 1-2-4-5-3-1

Trees

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.

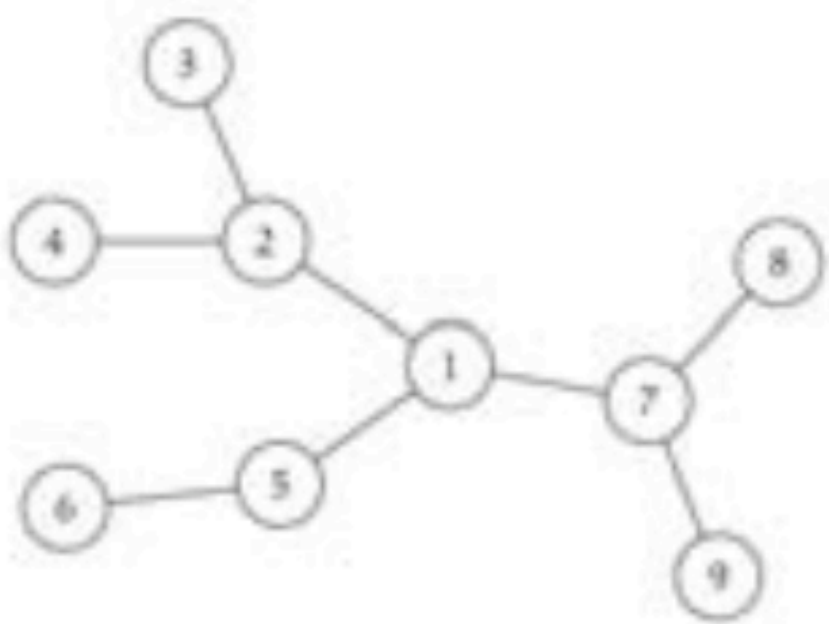
- G is connected.
- G does not contain a cycle.
- G has $n-1$ edges.



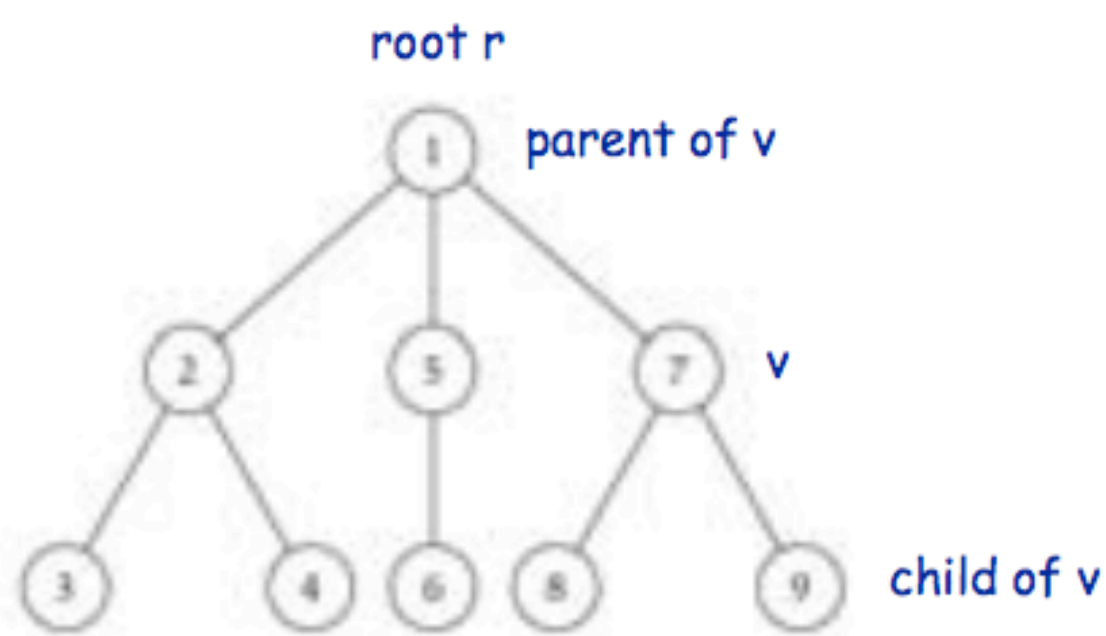
Rooted Trees

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

Importance. Models hierarchical structure.



a tree



the same tree, rooted at 1

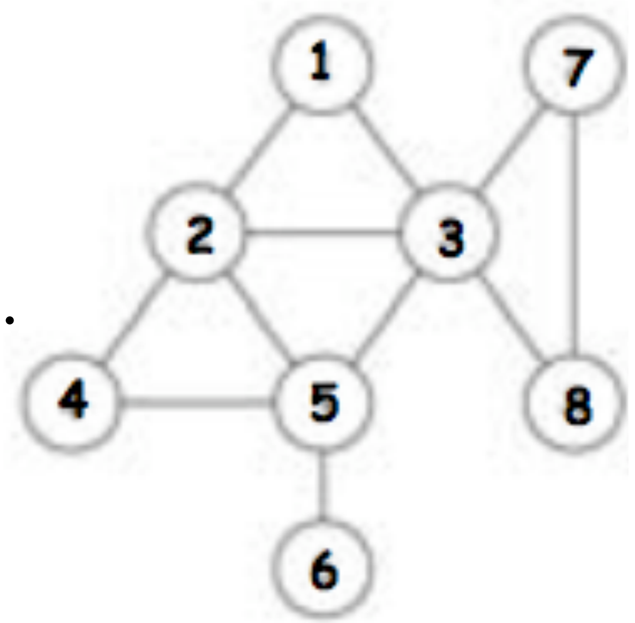
Connectivity

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?

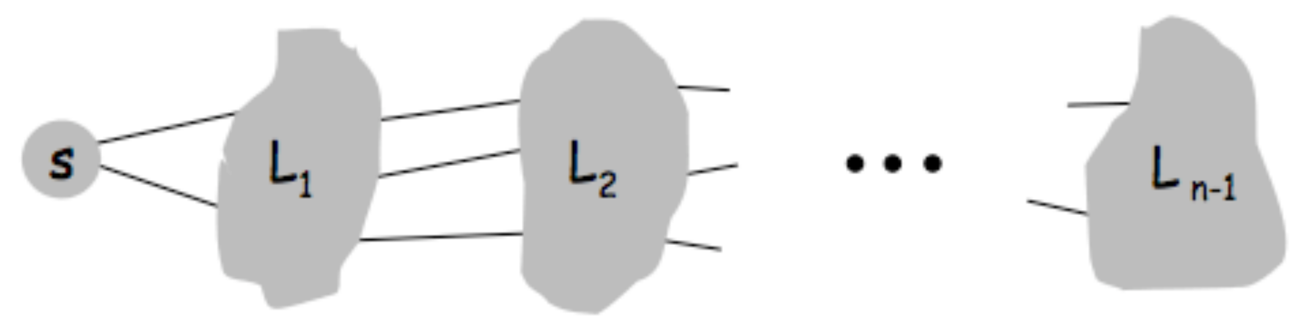
Applications.

- 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.



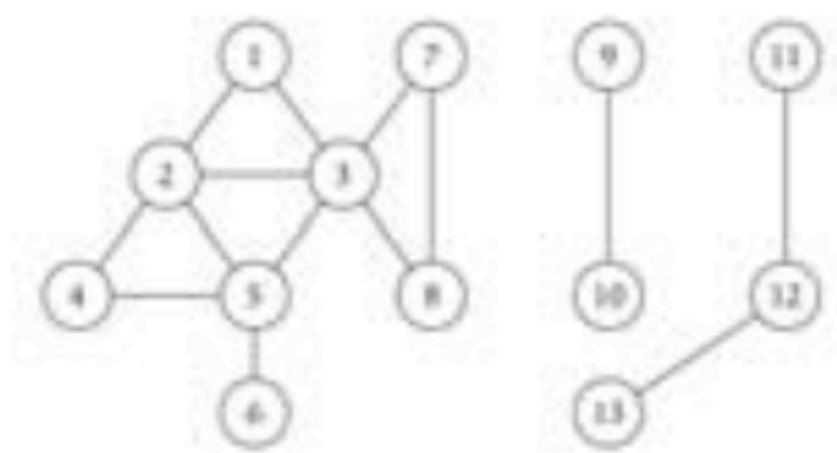
BFS algorithm.

- $L_0 = \{ s \}$.
- 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

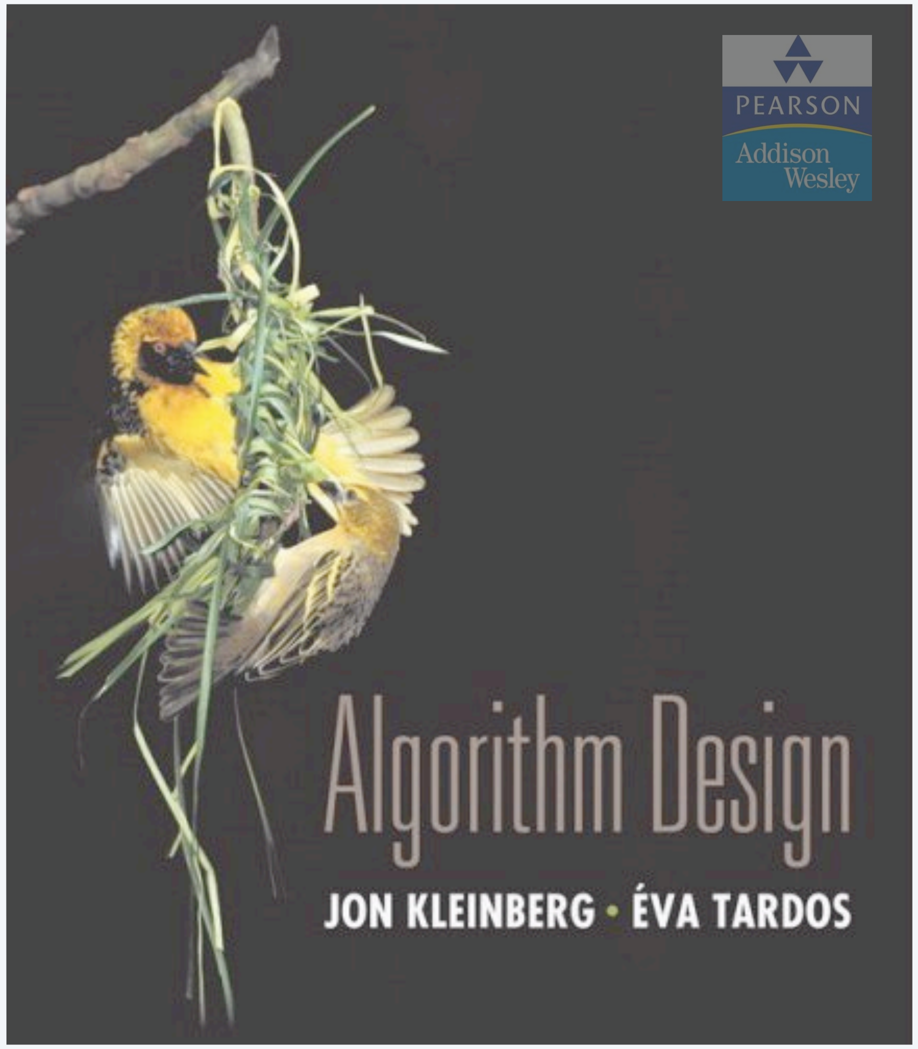
Depth First Search DFS

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.



SECTION 1.1

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.

favorite ↓ least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

men's preference list

favorite ↓ least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

women's preference list

Perfect matching

Def. A matching S is a set of ordered pairs $m\text{–}w$ 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 $|S| = |M| = |W| = n$.

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

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.

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

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.

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

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.

	1st	2nd	3rd
Adam	B	C	D
Bob	C	A	D
Chris	A	B	D
Doofus	A	B	C

no perfect matching is stable

A–B, C–D $\Rightarrow$ B–C unstable

A–C, B–D $\Rightarrow$ A–B unstable

A–D, B–C $\Rightarrow$ A–C unstable

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)

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

    IF (w is unmatched)

        Add pair m–w to matching S.

    ELSE IF (w prefers m to her current partner m')

        Remove pair m'–w from matching S.

        Add pair m–w to matching S.

    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. ▪

	1st	2nd	3rd	4th	5th
Victor	A	B	C	D	E
Wyatt	B	C	D	A	E
Xavier	C	D	A	B	E
Yancey	D	A	B	C	E
Zeus	A	B	C	D	E

	1st	2nd	3rd	4th	5th
Amy	W	X	Y	Z	V
Bertha	X	Y	Z	V	W
Clare	Y	Z	V	W	X
Diane	Z	V	W	X	Y
Erika	V	W	X	Y	Z

n(n-1) + 1 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. ← men propose in decreasing order of preference
  - $\Rightarrow$ 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. ← women only trade up
  - $\Rightarrow$ A–Z is stable.
- In either case, the pair A–Z is stable. ▪

$A - Y$

$B - Z$

$\vdots$

Gale–Shapley matching S*

Summary

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

Theorem. [Gale-Shapley 1962] The Gale-Shapley algorithm guarantees to find a stable matching for any problem instance.

Q. How to implement GS algorithm efficiently?

Q. If there are multiple stable matchings, which one does GS find?

COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept. Accordingly, in order for a college to receive q acceptances, it will generally have to offer to admit more than q applicants. The problem of determining how many and which ones to admit requires some rather involved guesswork. It may not be known (a) whether a given applicant has also applied elsewhere; if this is known it may not be known (b) how he ranks the colleges to which he has applied; even if this is known it will not be known (c) which of the other colleges will offer to admit him. A result of all this uncertainty is that colleges can expect only that the entering class will come reasonably close in numbers to the desired quota, and be reasonably close to the attainable optimum in quality.

Efficient implementation

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

Representing men and women.

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

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.

Efficient implementation (continued)

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.

pref[]

1st	2nd	3rd	4th	5th	6th	7th	8th
8	3	7	1	4	5	6	2

woman prefers man 3 to 6
since `inverse[3] < inverse[6]`

inverse[]

1	2	3	4	5	6	7	8
4th	8th	2nd	5th	6th	7th	3rd	1st

```
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?

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

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.

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

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.

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.
  $\Rightarrow$ $\boxed{A \text{ prefers } Z \text{ to } Y.}$
- 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.
  because this is the first rejection by a valid partner
- Thus, Z has not yet proposed to B when he proposes to A.
  $\Rightarrow$ $\boxed{Z \text{ prefers } A \text{ to } B.}$
- Thus A–Z is unstable in S, a contradiction. ▪

$A - Y$

$B - Z$

$\vdots$

stable matching S

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 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.
  $\Rightarrow$ $\boxed{A \text{ prefers } Z \text{ to } Y.}$
- Let B be the partner of Z in S. By man-optimality, A is the best valid partner for Z.
  $\Rightarrow$ $\boxed{Z \text{ prefers } A \text{ to } B.}$
- Thus, A–Z is an unstable pair in S, a contradiction. ▪

$A - Y$

$B - Z$

$\vdots$

stable matching S

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.

men's preference list

	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C

women's preference list

	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

Amy lies

	1st	2nd	3rd
A	Y	Z	X
B	X	Y	Z
C	X	Y	Z

Extensions: matching residents to hospitals

Ex: Men ≈ hospitals, Women ≈ med school residents.

Variant 1. Some participants declare others as unacceptable.

resident A unwilling to work in Cleveland

Variant 2. Unequal number of men and women.

Variant 3. Limited polygamy. ⟵ hospital X wants to hire 3 residents

Def. Matching is S unstable if there is a hospital h and resident r such that:

- h and r are acceptable to each other; and
- Either r is unmatched, or r prefers h to her assigned hospital; and
- Either h does not have all its places filled, or h prefers r to at least one of its assigned residents.

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.

hospitals began making offers earlier and earlier, up to 2 years in advance

stable matching is no longer guaranteed to exist

The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design

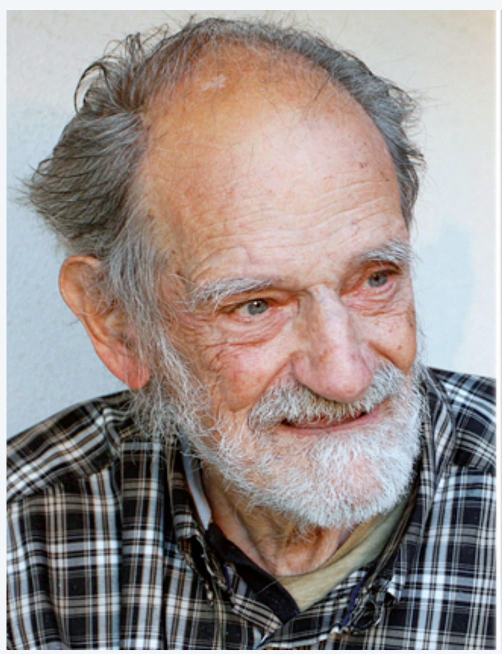
By Alvin E. Roth and Elliott Peranson*

We report on the design of the new clearinghouse adopted by the National Resident Matching Program, which annually fills approximately 20,000 jobs for new physicians. Because the market has complementarities between applicants and between positions, the theory of simple matching markets does not apply directly. However, computational experiments show the theory provides good approximations. Furthermore, the set of stable matchings, and the opportunities for strategic manipulation, are surprisingly small. A new kind of "core convergence" result explains this; that each applicant interviews only a small fraction of available positions is important. We also describe engineering aspects of the design process. (JEL C78, B41, J44)

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.



Lloyd Shapley

Alvin Roth

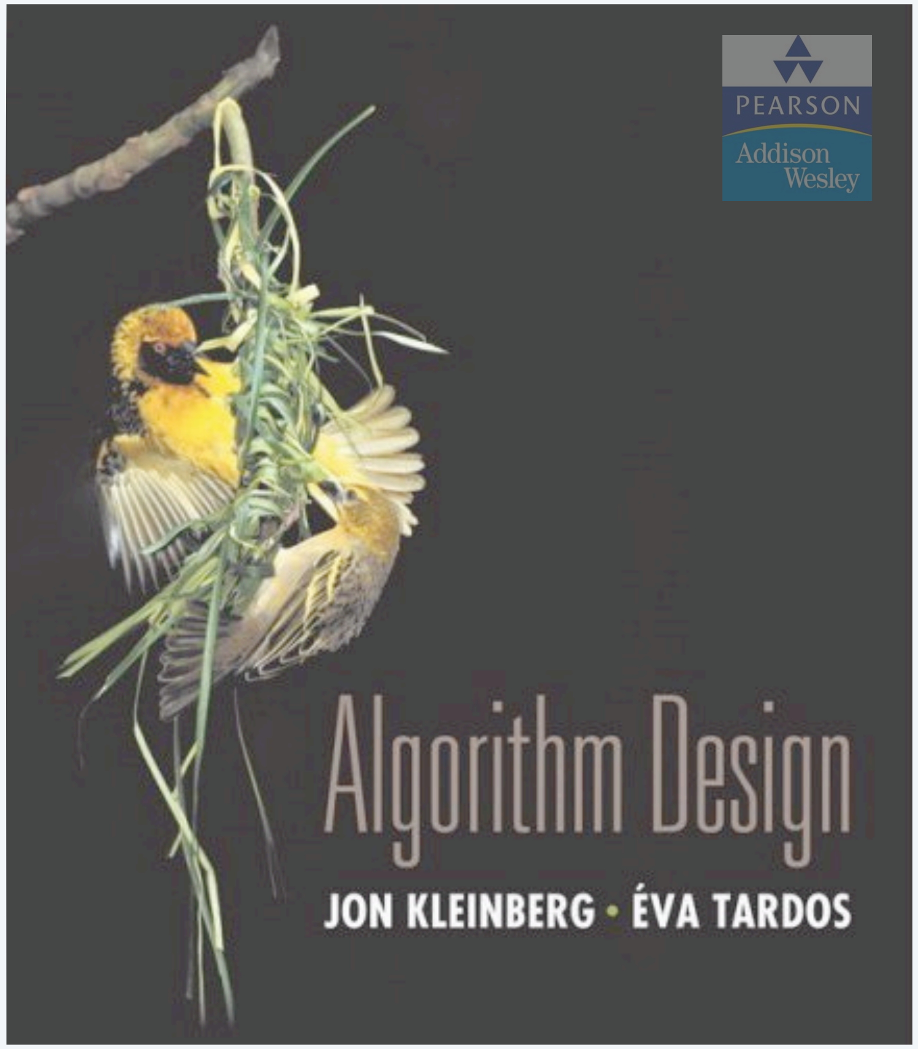
Lessons learned

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)!



SECTION 1.2

1. REPRESENTATIVE PROBLEMS

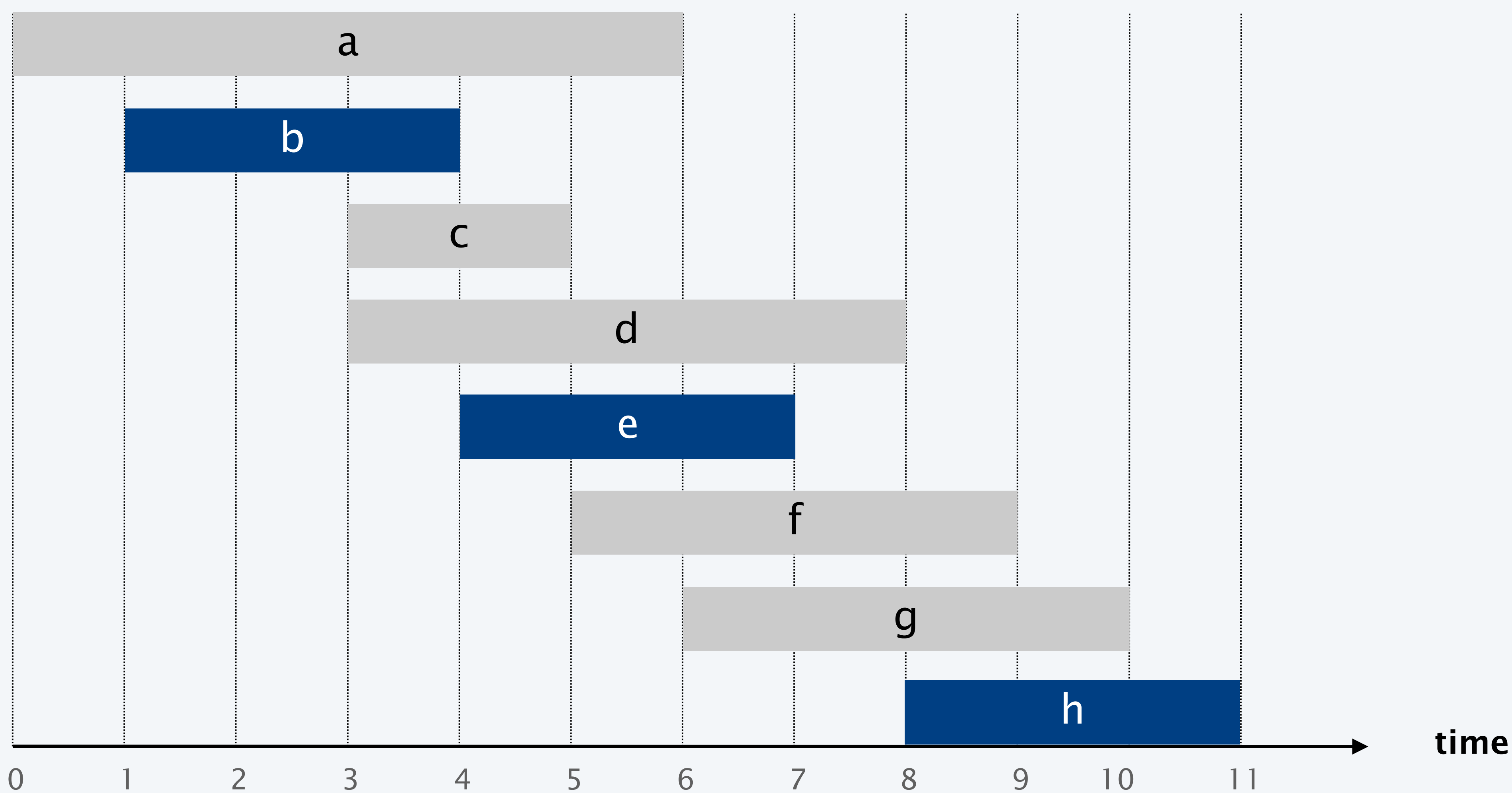
- *stable matching*
- *five representative problems*

Interval scheduling

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

Goal. Find maximum cardinality subset of mutually compatible jobs.

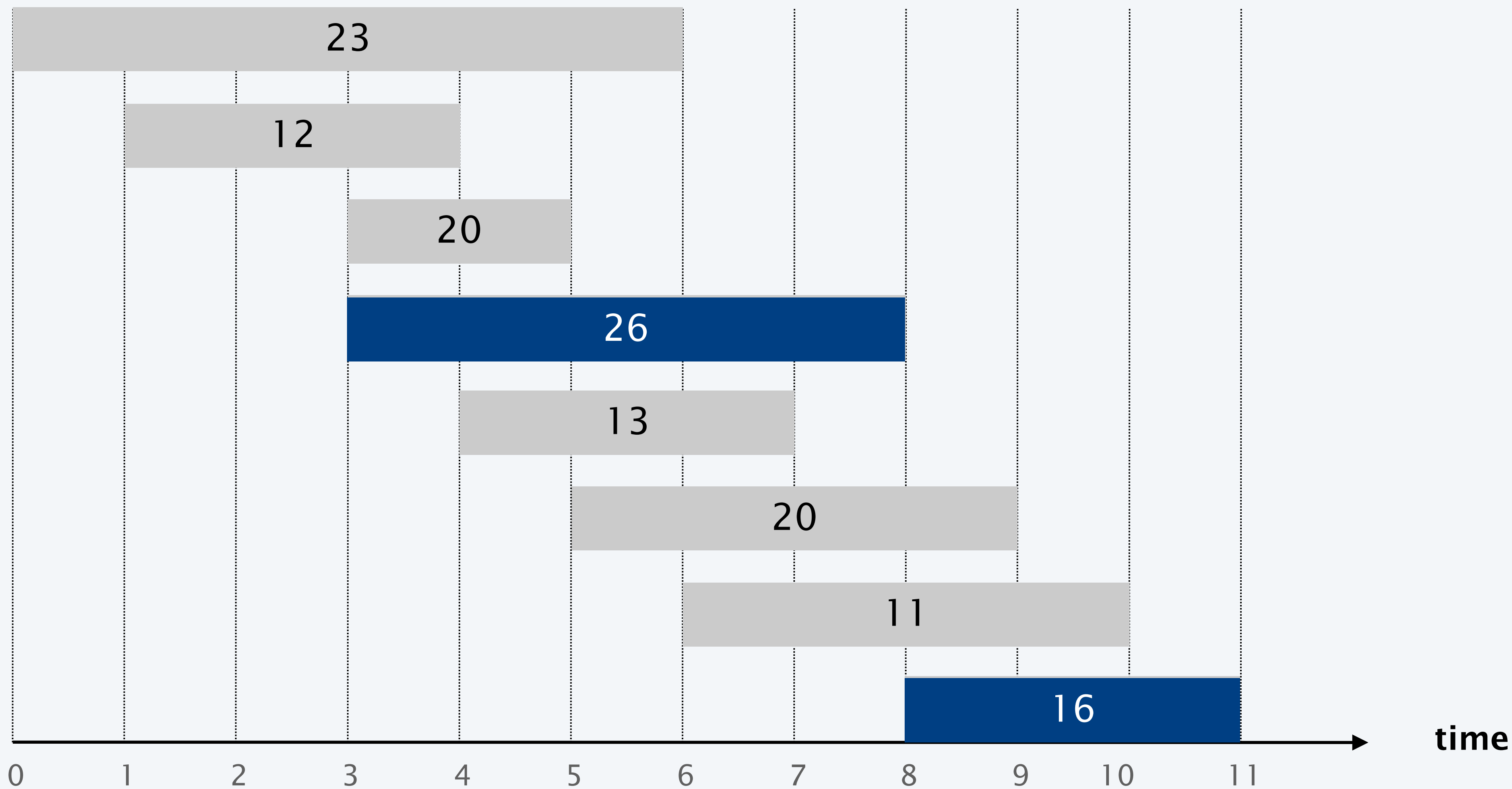
jobs don't overlap



Weighted interval scheduling

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

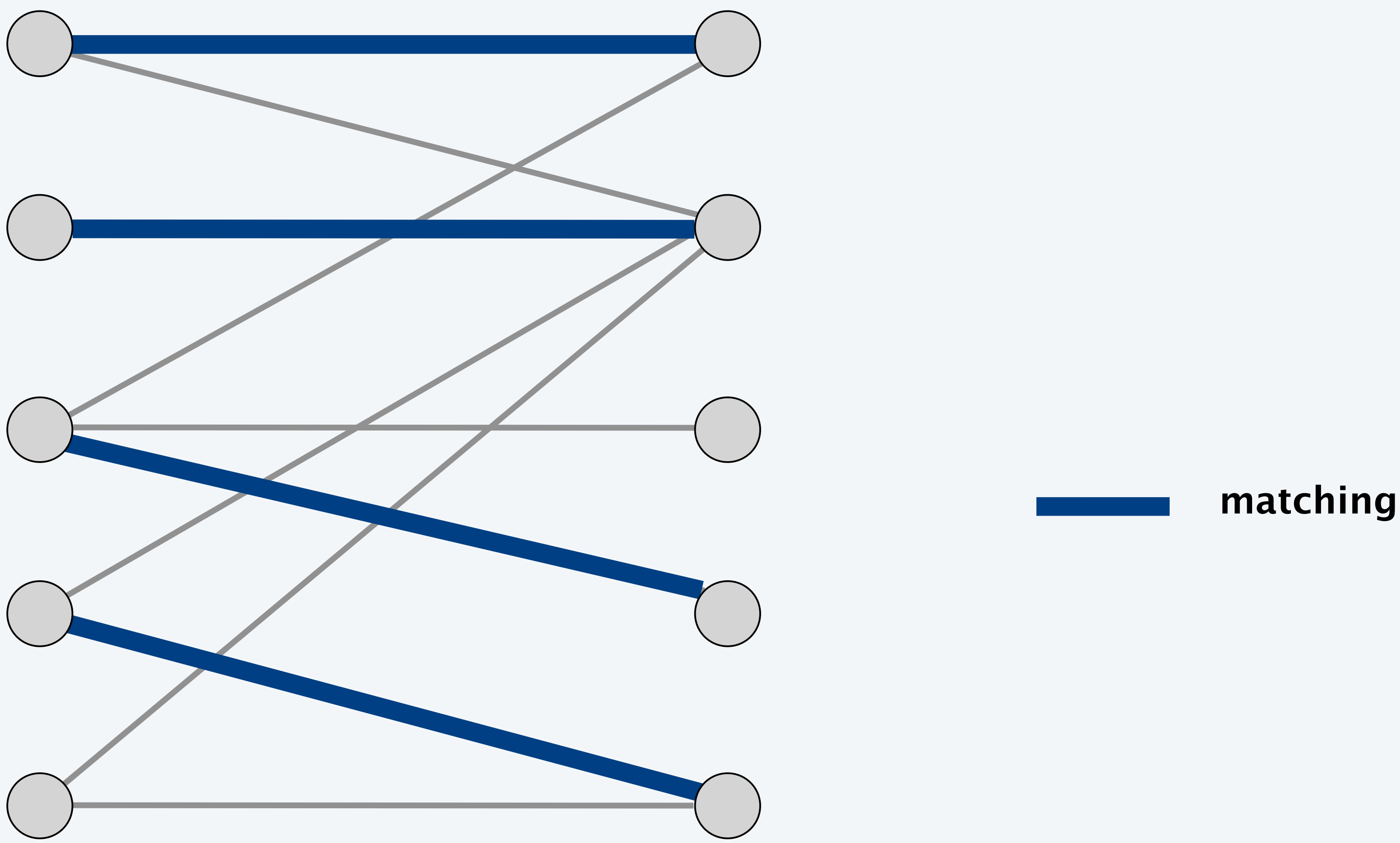
Goal. Find maximum weight subset of mutually compatible jobs.



Bipartite matching

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

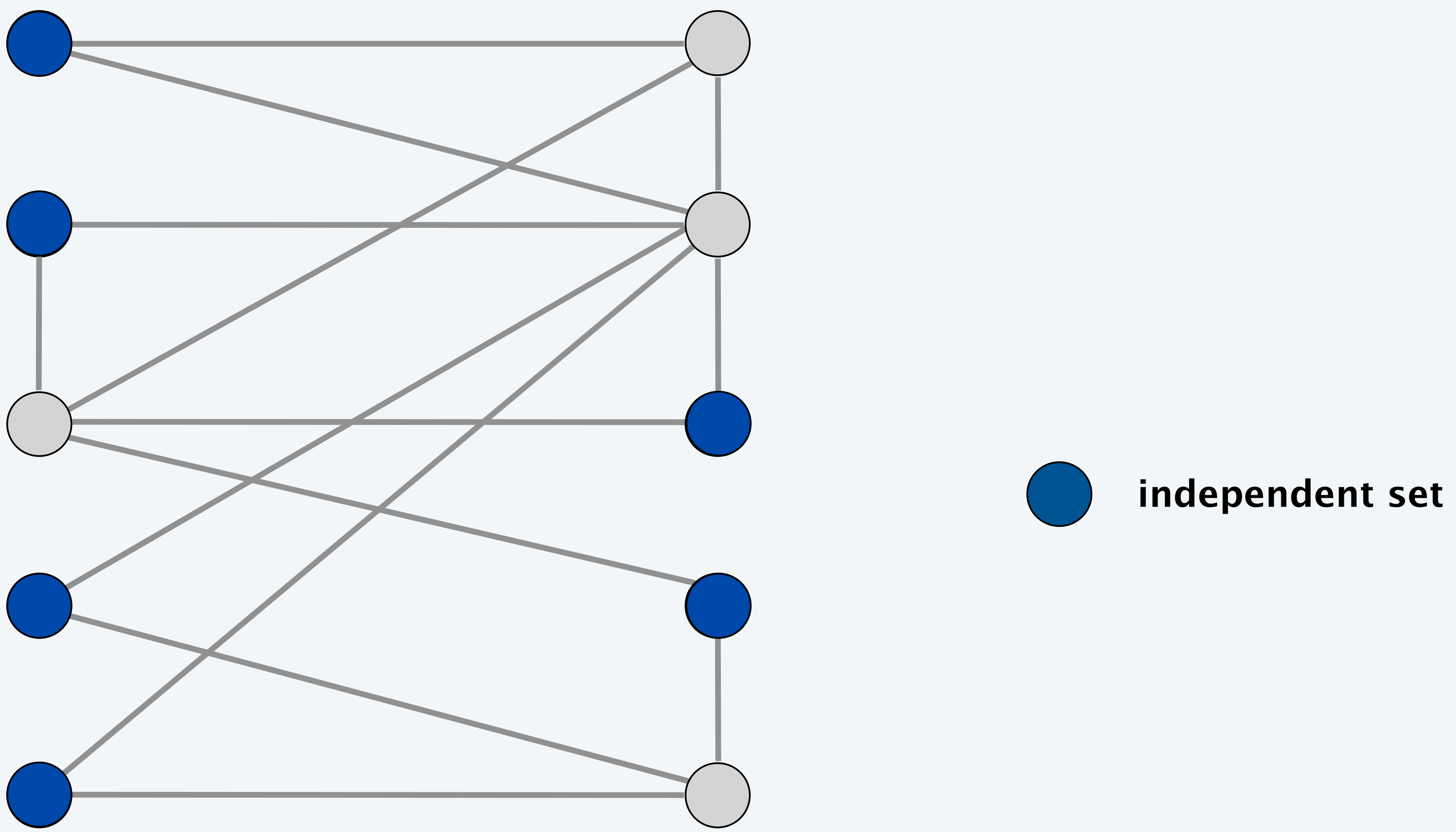
Def. A subset of edges $M \subseteq E$ is a matching if each node appears in exactly one edge in M.



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.