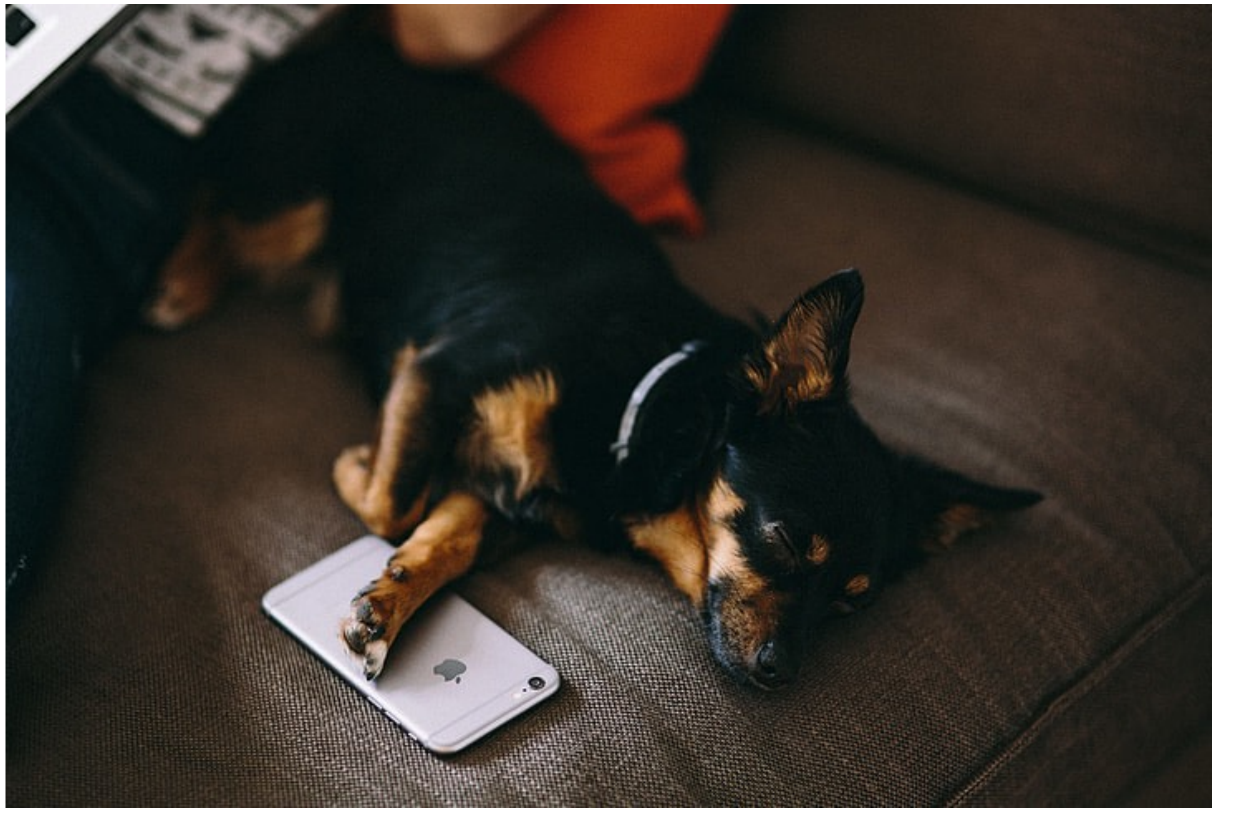
CS 256

Admin

- Videos posted on website
- Assignment 0 delayed to Saturday (Assignment 1 still released tomorrow, due Thursday)
- Zoom link on Glow and slack (no longer emailed)
- TA Office hours coming soon
- I’ll stay after class for questions

Quick Latex Note

- The final X is a chi (the Greek letter), not an X
- So it's pronounced lay-tech
- (or lah-tech)
- But not "latex"



Matching Med-Students to Hospitals

Input. A set H of n hospitals, a set S of n students and their preferences (each hospital ranks each student, each students ranks each hospital)

	1st	**2nd**	**3rd**
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	**2nd**	**3rd**
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

Perfect Matchings

Definition. A matching M is a set of ordered pairs (h, s) where $h \in H$ and $s \in S$ such that

- Each hospital h is in at most one pair in M
- Each student s is in at most one pair in M

A matching M is **perfect** if each hospital is matched to exactly one student and vice versa (i.e., $|M| = |H| = |S|$)

	1st	**2nd**	**3rd**
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	**2nd**	**3rd**
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

Unstable Pairs

Definition. A perfect matching M is **unstable** if there exists an unstable pair $(h, s) \in H \times S$, that is,

- h prefers s to its current match in M
- s prefers h to its current match in M

Can you point out an unstable pair in this matching?

	1st	**2nd**	**3rd**
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	**2nd**	**3rd**
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched. In each round,

- Each hospital makes offer to its top available candidate
- Each student accepts its top offer (irrecoverable contract) and rejects others

What goes wrong?

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Take a Step Back

- Imagine you are one of these students
- Why is it a bad idea to accept the best offer you get in the first round?
  - You might get a better offer later!
- Can we come up with an example where this happens, causing an unstable matching?

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris

	1st	2nd	3rd
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	2nd	3rd
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH

	1st	2nd	3rd
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	2nd	3rd
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

Is this a stable matching?

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

Is this a stable matching?

- Unstable pair: (MA, Chris). What could have avoided it?

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

What Should Students Do?

- Don’t accept immediately (of course)
- What if they can have a *preliminary* accept?
  - “I’m interested, but I also want to wait to see if I get a better offer”
- That seems to solve the problem we mentioned, but does it always give a stable matching?

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched.* In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched.* In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched.* In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched.* In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Algorithm

GALE–SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

 $s \leftarrow$ first student on h's list to whom h has not yet proposed.

 IF (s is unmatched)

  Add h–s to matching M.

 ELSE IF (s prefers h to current partner h')

  Replace h'–s with h–s in matching M.

 ELSE

  s rejects h.

RETURN stable matching M.

Analyzing Gale-Shapely

Questions to ask

Efficiency:

- How long does it take to produce a matching?
- How can we efficiently implement each step?

Correctness:

- Does it match everyone? (produce a perfect matching)
- Does it produce a stable matching?

Analyzing the Algorithm: Performance

- Each hospital makes an offer to each student at most once, so the algorithm makes at most $O(n^2)$ iterations
- What do we do in each iteration?
    - Select a free hospital h
    - Find top ranked s not yet offered a post by h
    - Find s*'s* ranking of a given hospital
    - Add to & delete from set of matched pairs
    - (possibly) Add a hospital back into the free list
- How long does it take?
    - Depends on how we implement each of these!

Analyzing the Algorithm: Performance II

- **Input representation.** Index students and hospitals $1, \ldots, n$
  - Each student provides a sorted list of hospitals (most to least preferred) and each hospital provides a sorted list of students
- Of students not yet offered a post by h, find most preferred: $O(1)$
- Does s prefer h to the current hospital h'?
  - For each s, create inverse of preference list of hospitals (Identify efficient data structures for operations)

student prefers hospital 4 to 6 since rank[4] < rank[6]

Student preference list indexed by rank

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

Inverse pref-list indexed by hospital #

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

```
for i = 1 to n
    rank[pref[i]] = i
```

Analyzing the Algorithm: Performance III

Analyzing running time:

- Creating the inverse-list for each student (preprocessing): $O(n^2)$
- Once created, $O(1)$ time to accept/reject proposal by student
- Maintain free hospitals: Queue: $O(1)$ for get() and put()
- Add to & delete from set of matched pairs:
  - Array, Matched(*s*) = *h* currently matched to s (or ‘free’) : Creation time (preprocessing) $O(n)$; update time $O(1)$

Each iteration thus takes $O(1)$ time

Overall, $O(n^2)$ time preprocessing + $O(n^2)$ time in iterations: $O(n^2)$

- Linear time? Yes! Here input size is $O(n^2)$ size, linear in input size

Analyzing the Algorithm: Correctness

Does it match everyone? (Perfect matching)

- Once a student receives an offer, she has at least a tentative match for the rest of time.
- Equivalently, if any student is unmatched, then no hospital has offered them which implies that the hospitals have not exhausted their preference lists.
- When the algorithm terminates, everyone is matched (i.e., it produces a *perfect matching)*.

Does it produce a stable matching?

- Key idea: students always 'trade up'
    - s breaks match with h in favor of h' only if s prefers h' to h

Analyzing the Algorithm: Correctness II

Lemma. The Gale Shapely Algorithm produces a stable matching.

Proof. (By contradiction) Let M be the resulting matching. Suppose $\exists (h, s)$ such that $(h, s'), (h', s) \in M$ and

- h prefers s over s' and s prefers h over h'

Thus h must have offered to s before s'

- 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 s must prefer final match h' over h'', which they prefer over h. $(\Rightarrow\!\Leftarrow)$ ∎

Historical Perspective

- In 1952, the National Resident Matching Program (NRMP) adopted the “Boston Pool” algorithm named after regional clearinghouses in Boston
- In 1962, David Gale and Lloyd Shapley formally analyzed a generalization of the Boston Pool algorithm
- Shapley & Roth (who extended his work) were awarded **the 2012 Nobel Prize in Economics** (Gale did not share the prize, because he died in 2008.)
- Used to be called the **stable marriage problem/algorithm**
- Read https://www.nobelprize.org/uploads/2018/06/popular-economicsciences2012-1.pdf

Acknowledgements

- Slides adapted from Shikha Singh's slides, in turn adapted from Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
- Some material taken from Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)

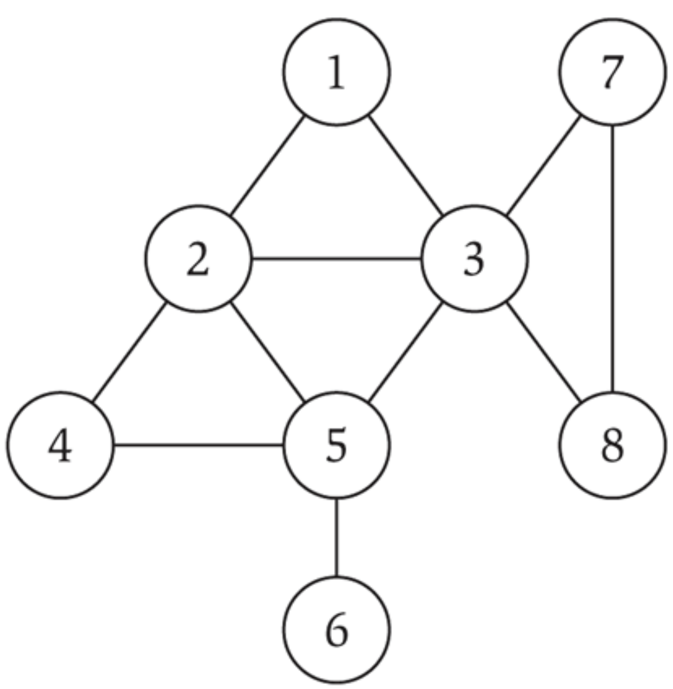
Graphs and Traversals

Review: Undirected Graphs

An undirected graph $G = (V, E)$

- V is the set of nodes, E is the set of edges
- Captures pairwise relations between objects
- Graph size parameters: $n = |V|$, $m = |E|$

Sometimes we consider weighted graphs, where each edge e has a weight $w(e)$



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

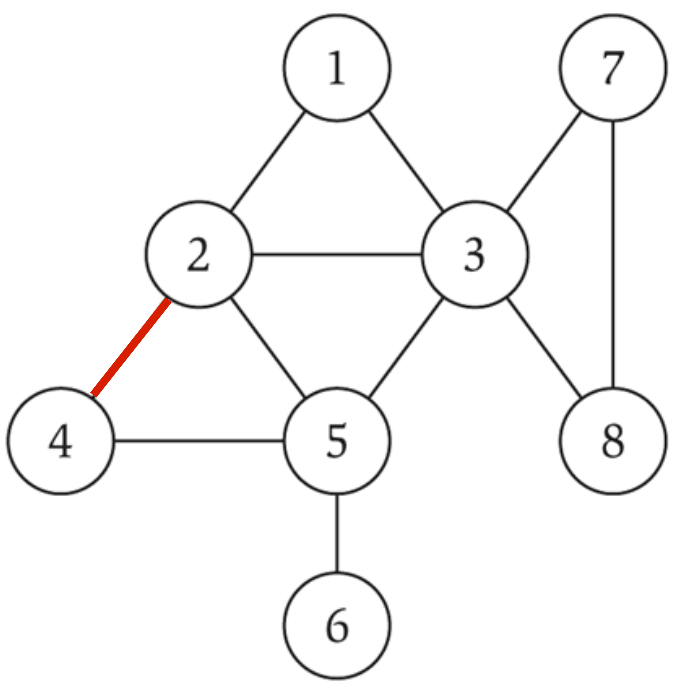
$E = \{ 1\text{–}2, 1\text{–}3, 2\text{–}3, 2\text{–}4, 2\text{–}5, 3\text{–}5, 3\text{–}7, 3\text{–}8, 4\text{–}5, 5\text{–}6, 7\text{–}8 \}$

$m = 11, n = 8$

Representing Graphs (Review)

Adjacency matrix.

- n-by-n matrix where $A[u][v] = 1$ if $(u, v) \in E$
- Space $O(n^2)$
- Checking if $(u, v) \in E$ takes _____ time?

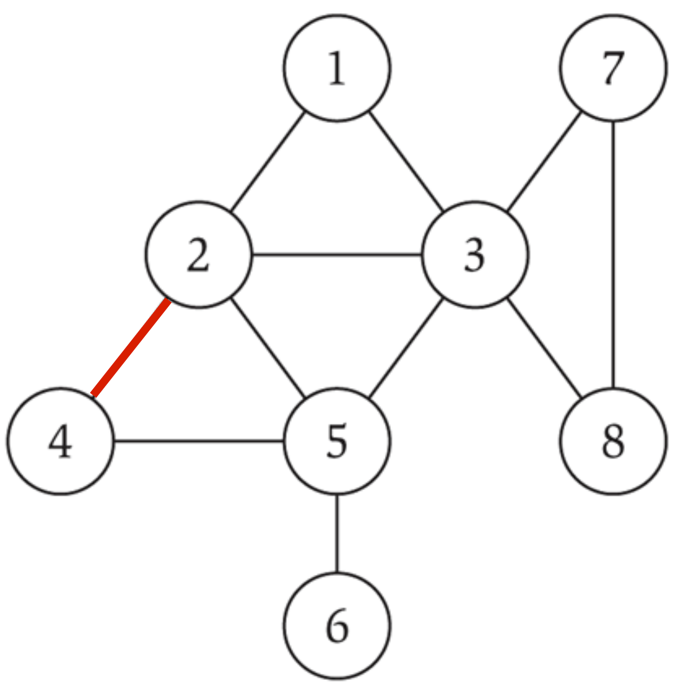


	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Representing Graphs (Review)

Adjacency matrix.

- n-by-n matrix where $A[u][v] = 1$ if $(u, v) \in E$
- Space $O(n^2)$
- Checking if $(u, v) \in E$ takes $O(1)$ time

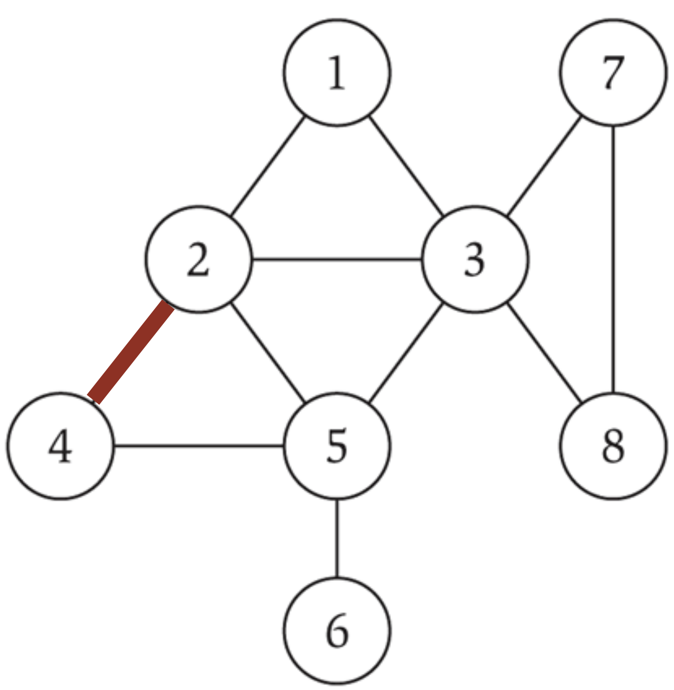


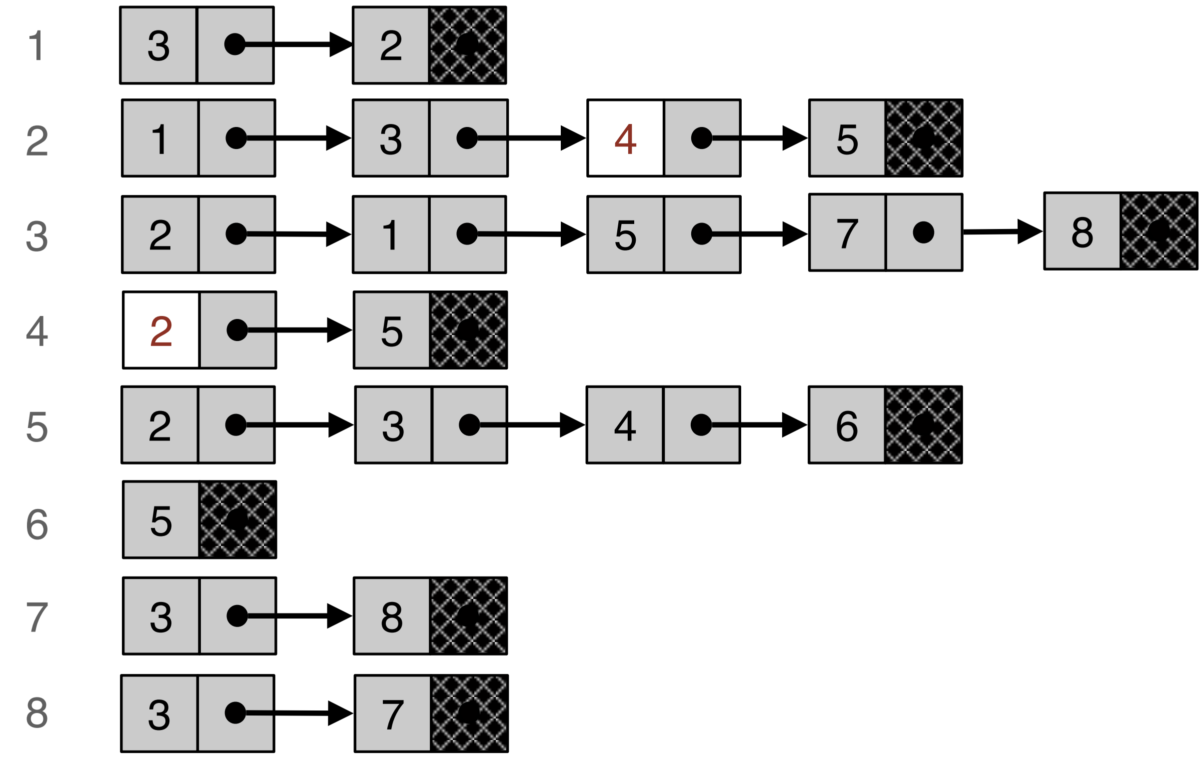
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Representing Graphs (Review)

Adjacency list.

- Array of lists, where each list represents the neighbors of a given node
- Space $O(n + m)$
- Checking if $(u, v) \in E$ takes _____ time?

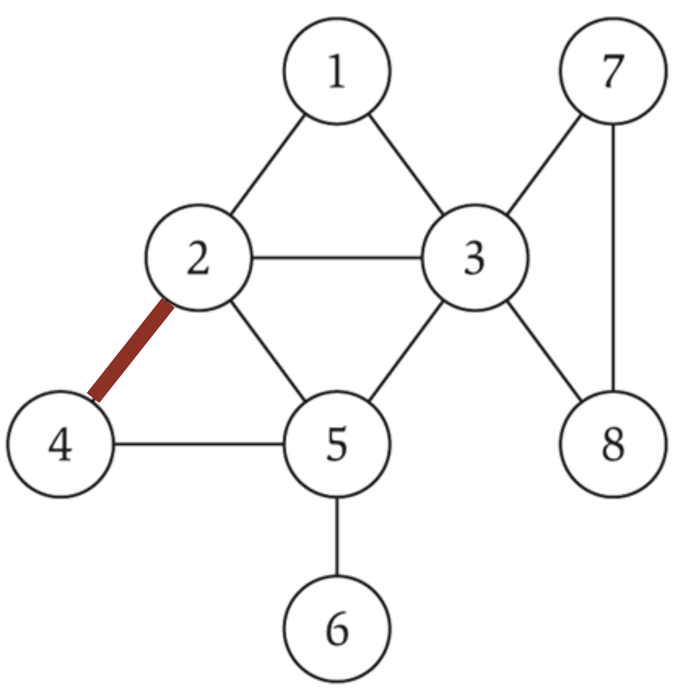


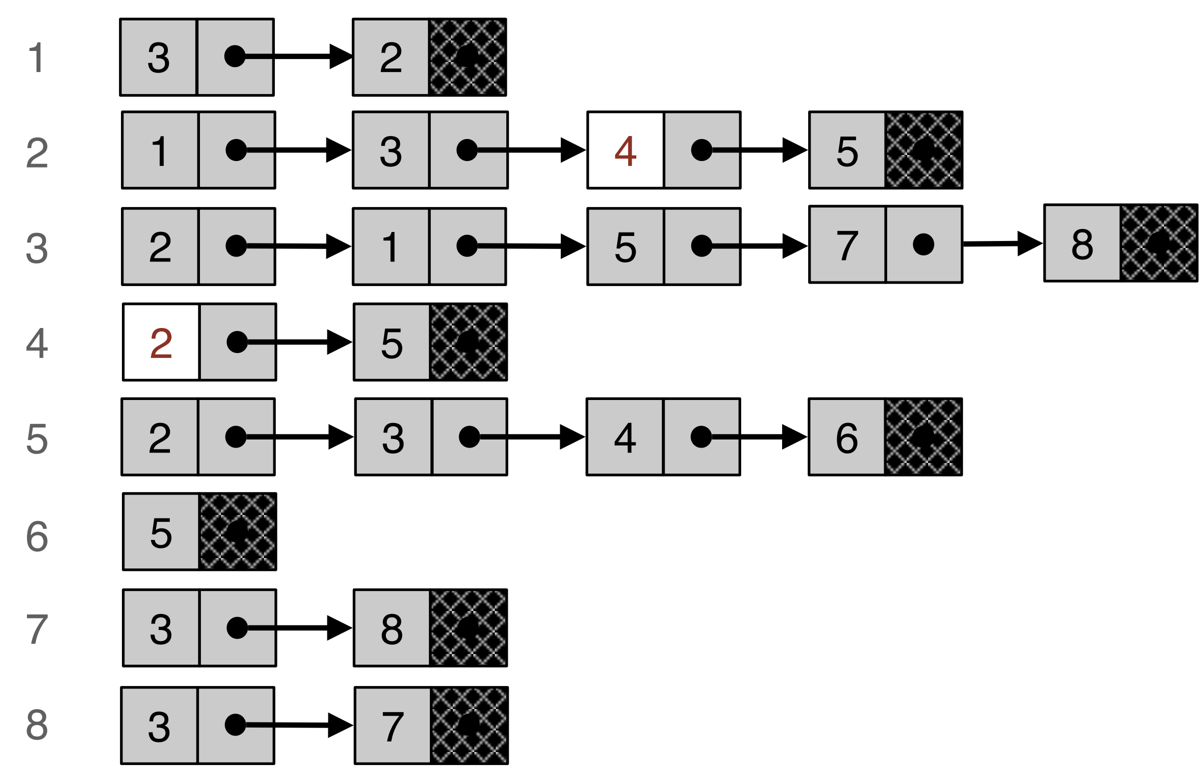


Representing Graphs (Review)

Adjacency list.

- Array of lists, where each list represents the neighbors of a given node
- Space $O(n + m)$
- Checking if $(u, v) \in E$ takes $O(\text{degree}(u))$ time





Graph Terminology

- A **path** in an undirected graph $G = (V, E)$ is a sequence of nodes $u_1, u_2, \ldots, u_k$ such that every pair $(u_{i-1}, u_i) \in E$.
- A path is **simple** if all nodes are distinct.
- The **length** of a path is **the number of edges on the** path

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

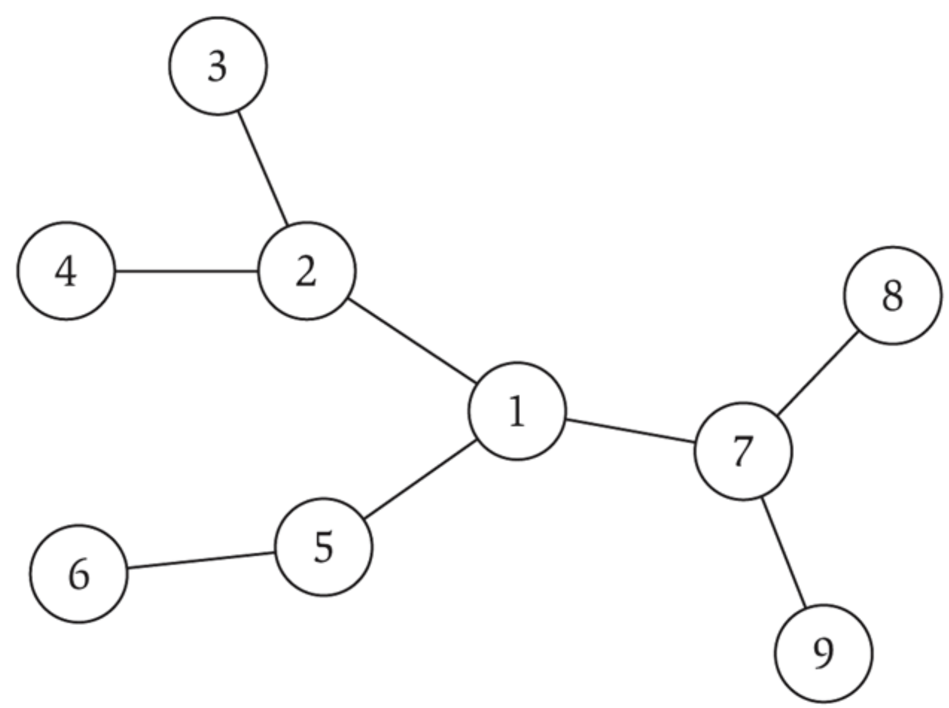
- A **cycle** is path $u_1, u_2, \ldots, u_k$ where $u_1 = u_k$ $(k \geq 2)$
- A cycle is **simple** if all internal nodes are distinct

Trees

- An undirected graph is a tree if it is connected and does not contain a cycle

Lemma. Let G be an undirected graph with n nodes. Then any two of these conditions imply the third

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

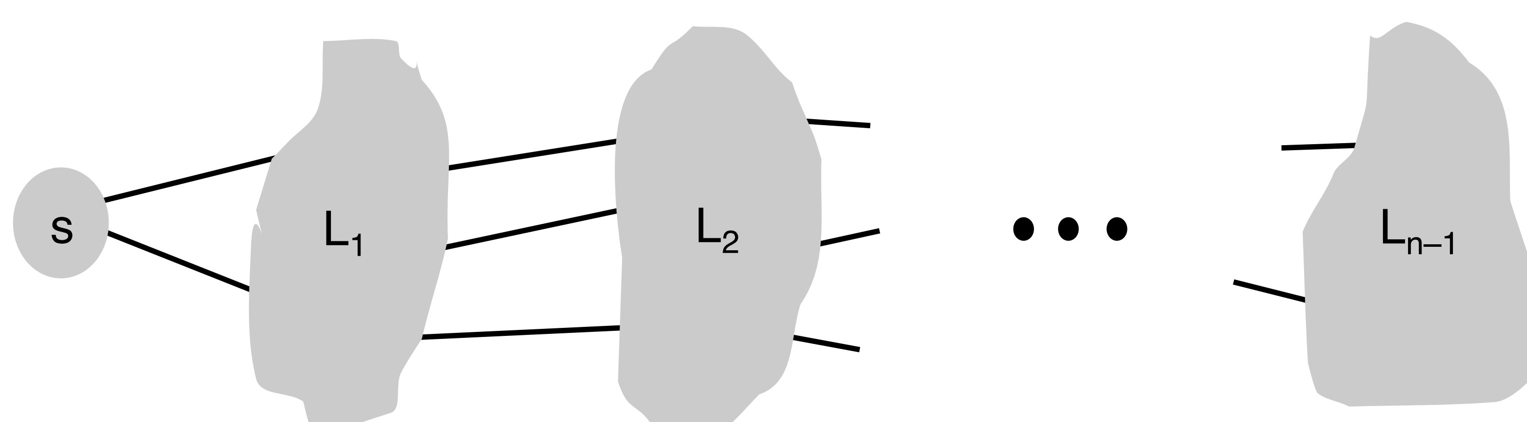


Graph Traversals

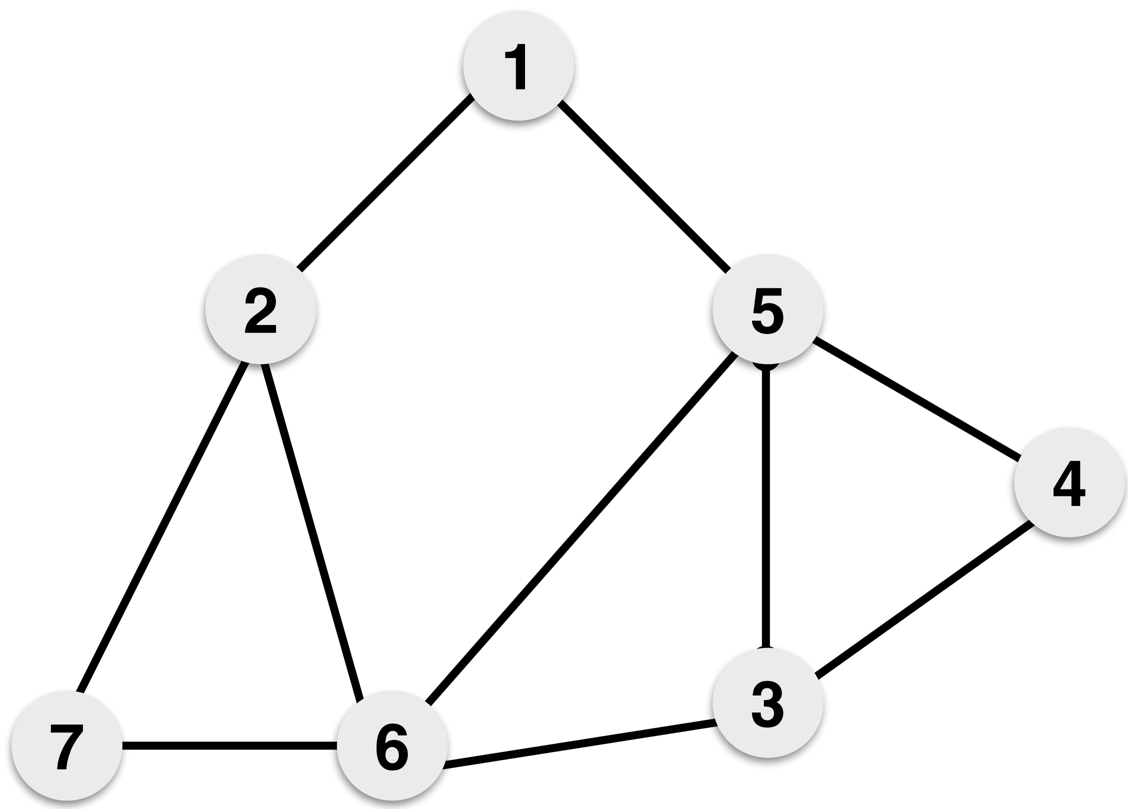
- **Connectivity.** How do we verify if a graph is connected?
- **Path.** Given $s, t \in V$, is there a path between them?
- Determined by "traversing the graph"
- Two classic graph traversal algorithms:
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Both have different applications
    - Bipartite testing (BFS)
    - Topological ordering (DFS), etc

Breadth-first Search

- Explore outwards in all possible direction from starting point, peeling "one layer after another"
- BFS algorithm: Initialize $L_0 = \{s\}$
    - $L_1 =$ all neighbors of L_0
    - $L_2 =$ all nodes that do not belong to L_0 or L_1 that are adjacent to a node in L_1
    - …
    - $L_{i+1} =$ all nodes that do not belong an earlier layer that are adjacent to a node in L_i

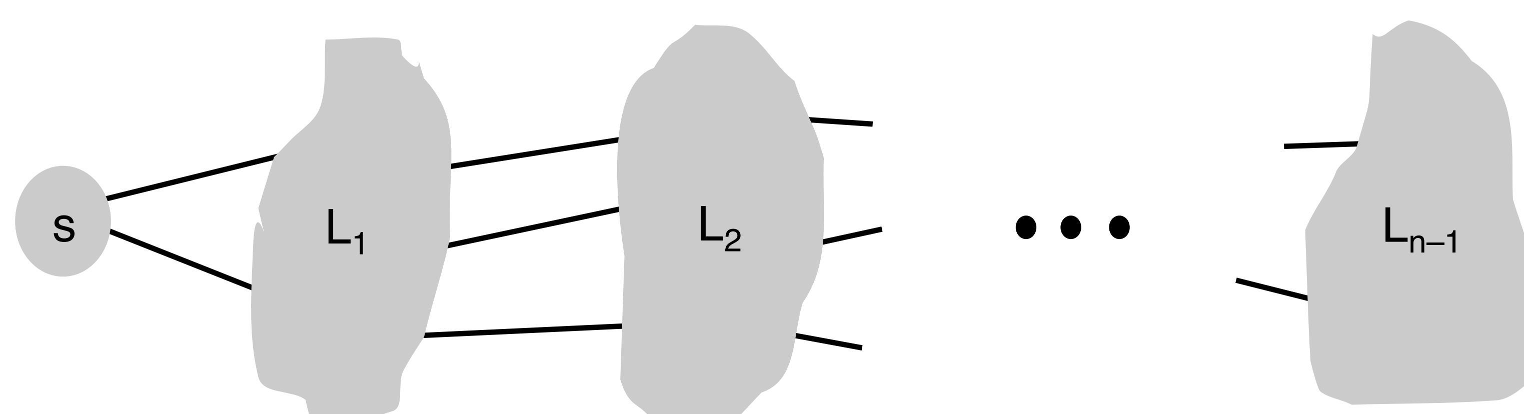


BFS Example



BFS Implementation

- Nodes that we have not seen yet
- Nodes that we have visited
- Nodes that have been "explored" (visited all its neighbors as well)
    - Suppose we are currently exploring u
    - Its neighbors will be marked but when should they be explored compared to other marked unexplored nodes?
    - Want to explore all nodes at level i before moving on to level $i+1$ (first visited is first to be explored)
    - Which data structure?
        - Queue



BFS Implementation: Queue

- Nodes that we have not seen yet (never been added to queue)
- Nodes that we have visited (added to queue but not marked)
- When a node is marked (after extraction from queue), all its neighbors are visited: next time we see it we can ignore it —-its been explored!

```
BFS (G, s):
 Put s in the queue Q
 While Q is not empty
   Extract v from Q
     If v is unmarked
       Mark v
       For each edge (v, w):
          Put w into the queue Q
```

The BFS Tree

- Can remember parent nodes (the node at level i that lead us to a given node at level $i + 1$)

```
BFS-Tree(G, s):
 Put (∅, s) in the queue Q
 While Q is not empty
   Extract (p, v) from Q
     If v is unmarked
       Mark v
       parent(v) = p
       For each edge (v, w):
          Put (v, w) into the queue Q
```

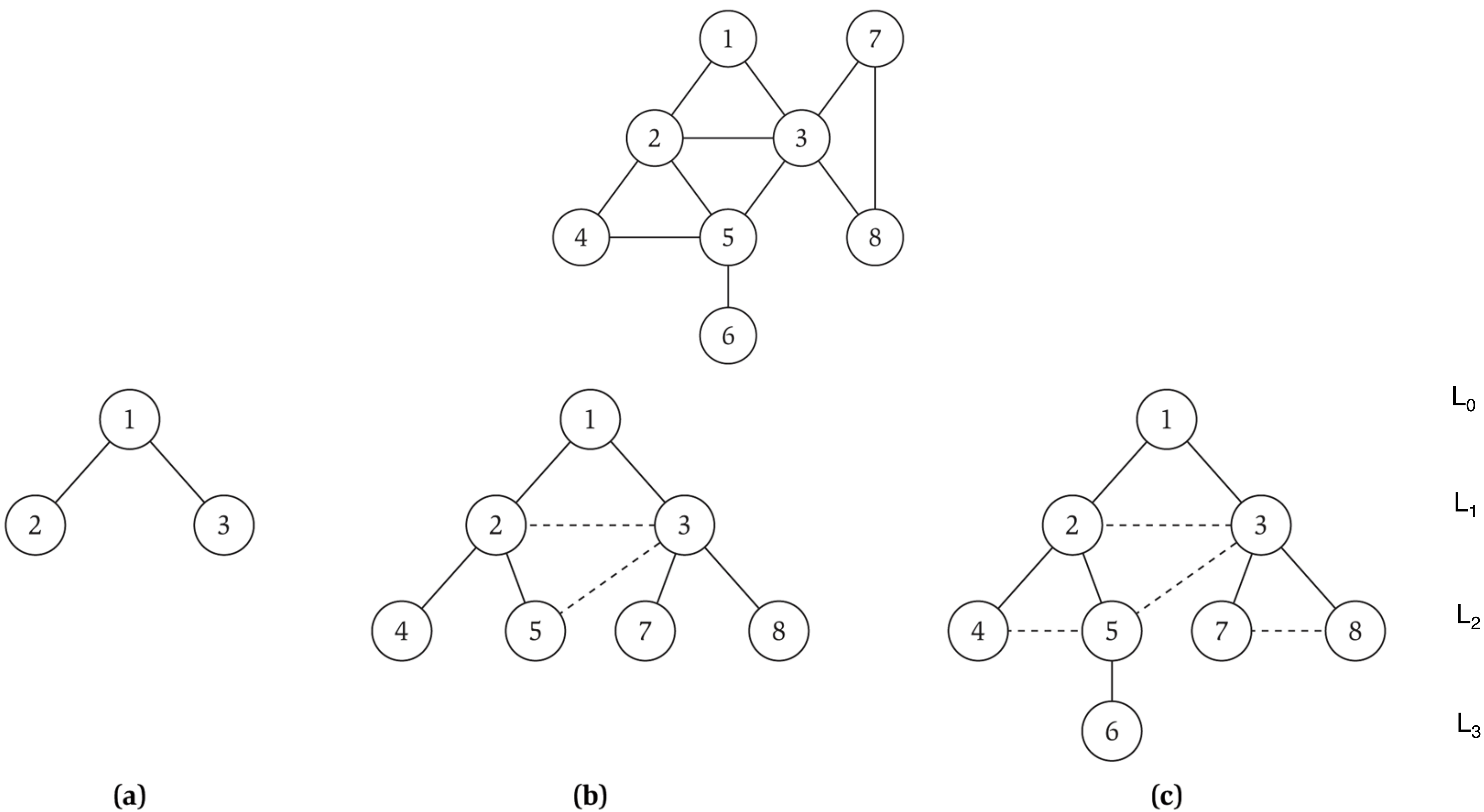
BFS Analysis

- Inserting and extracting from a queue
    - $O(1)$ time
- Extracting edges of node v (assuming **adjacency list**)
    - $O(1)$
- Overall running time?
    - Easy to prove $O(n^2)$ time
    - Can improve the analysis to $O(n+m)$
        - Node u has degree(u) incident edges (u, v)
        - Total time processing edges: $\sum_{u \in V} \text{degree(u)} = 2m$

each edge (u, v) is counted exactly twice
in sum: once in degree(u) and once in degree(v)

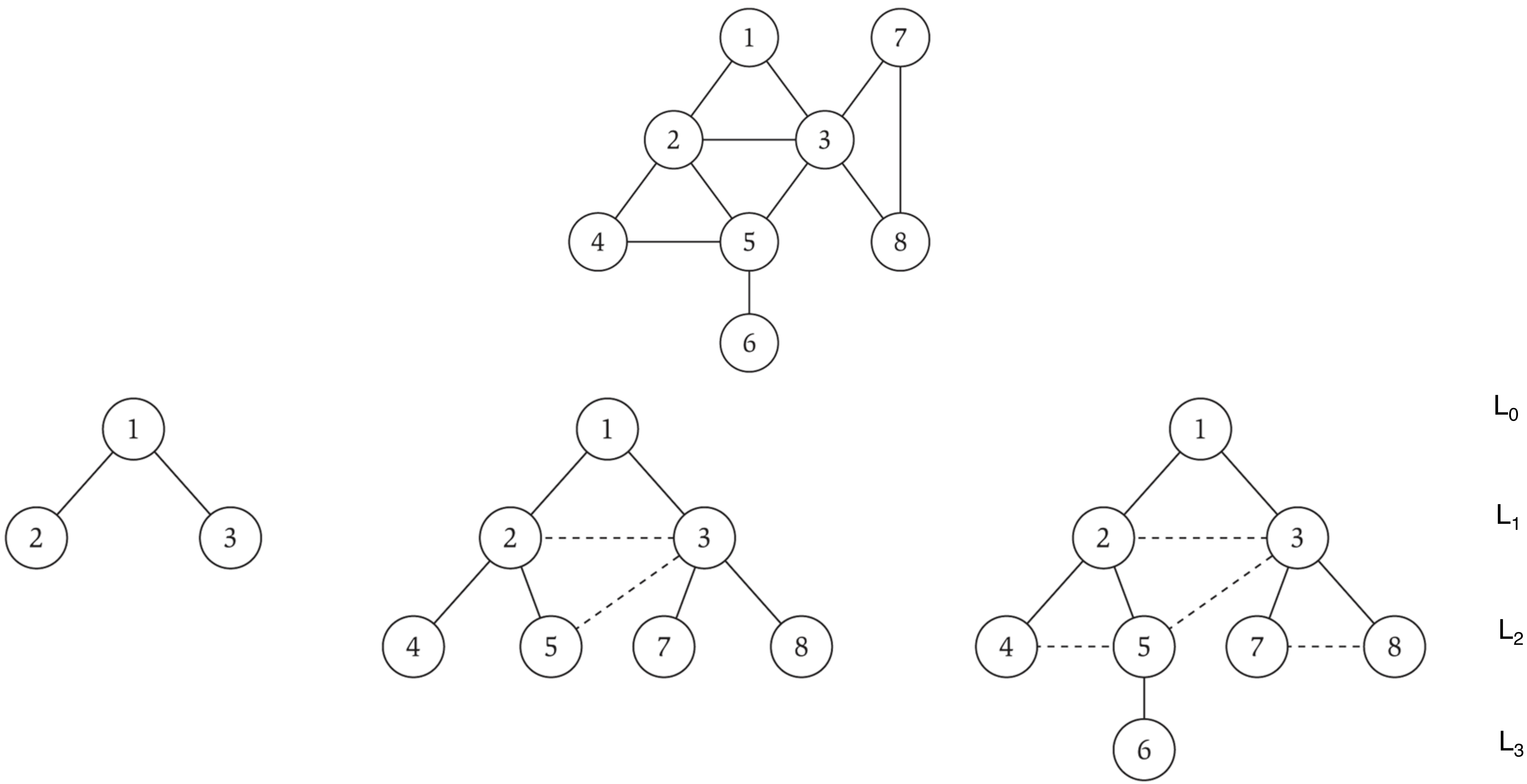
BFS Tree Structure

- Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then, the levels of x and y differ by at most 1.



BFS Tree Structure

- Property. Let T be a BFS tree rooted at r of a connected unweighted graph, then the path from r to any node $u \in V$ in T is **the shortest path** from r to u.



(a) (b) (c)

Spanning Trees

- **Definition.** A **spanning tree** of an undirected graph G is a connected acyclic subgraph of G that contains every node of G.
- The tree produced by the BFS algorithm (with $(u, \text{parent}(u))$ as edges) is a spanning tree of the component containing s.
- **Connected component of s:** all nodes reachable from s
- In an undirected graph, a BFS spanning tree gives the shortest path from s to every other vertex in its component
- (We will revisit shortest path in a couple of lectures)
- BFS trees in general are short and thick

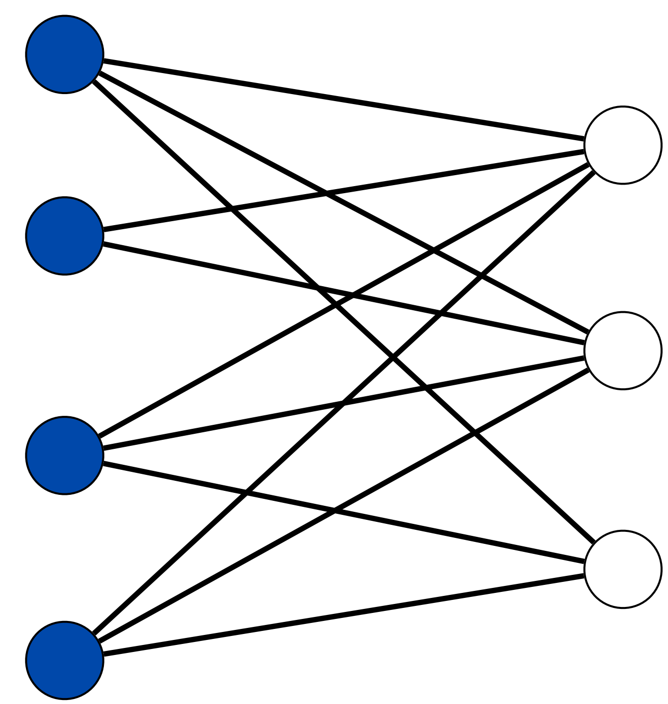
BFS Application: Connectivity

- How to whether a graph is connected using traversals?
  - If the BFS spanning tree contains all nodes of the graph, then the graph is connected
- Suppose the graph is not connected
- How can we find all connected components?
  - Start BFS with any node s, when its done, all nodes in the BFS tree of s are one component
  - Pick another node that is not visited and repeat
  - Number of trees in resulting **forest** is the number of components of the graph

BFS Application: Bipartite Testing

- **Bipartite graph.**
  - An undirected graph is **bipartite** if its nodes can be portioned into two sets S_1, S_2 such that all edges have endpoint in both sets

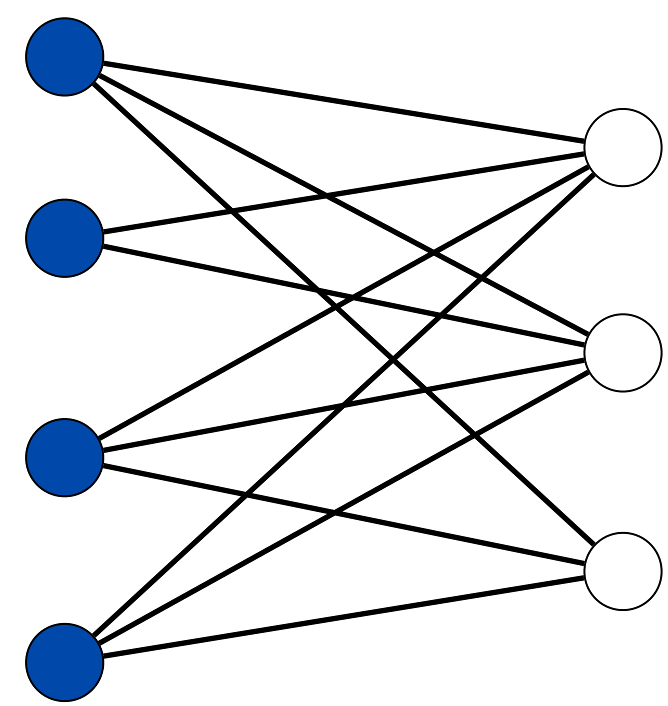
- Models many settings
  - We already encountered an application, which is…?
  - Common in scheduling, one set is machine, other set is jobs



a bipartite graph

BFS Application: Bipartite Testing

- Given a graph $G = (V, E)$ verify if it is bipartite
- Hint: need to use traversals
- But first need to understand structure of bipartite graphs
- **Question:** Can a bipartite graph contain an odd-length cycle?
- How do we prove this?
- In fact, a graph is bipartite if and only if it does have an odd length cycle
- One direction bipartite implies no odd length cycle is simple
- Will prove the other direction constructively



a bipartite graph

Bipartite Testing: Using BFS

Theorem. The following statements are **equivalent** for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Note: Conditions (a) and (b) seem hard to check directly; but conditions (c) and (d) allow an easy check!

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (a) ⇒ (b)

Vertices must alternate between V_1 and V_2.

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (b) $\Rightarrow$ (c)

Contradiction: Such an edge implies an odd cycle

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (c) ⇒ (d)

If all BFS trees have a property then some do as well

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (d) ⇒ (a)

Edges must span consecutive levels: levels provide bipartition of G

Implications of the Theorem

How to check if a graph is bipartite?

- When we visit an edge during BFS, we know the level of both of its endpoints
- So if both ends have the same level, then we can stop ! (G is not bipartite)
- If no such edge is found during traversal, G is bipartite
- Alternate levels give the bipartition

Running time?

- Still $O(n + m)$
- **Certificate.** If G is not bipartite this algorithm gives us a proof of it (the odd cycle that is found)!