CSE 331: Algorithms & Complexity

"More BFS and DFS"

Prof. Charlie Anne Carlson (She/Her)

Lecture 14

Monday September 29th, 2025

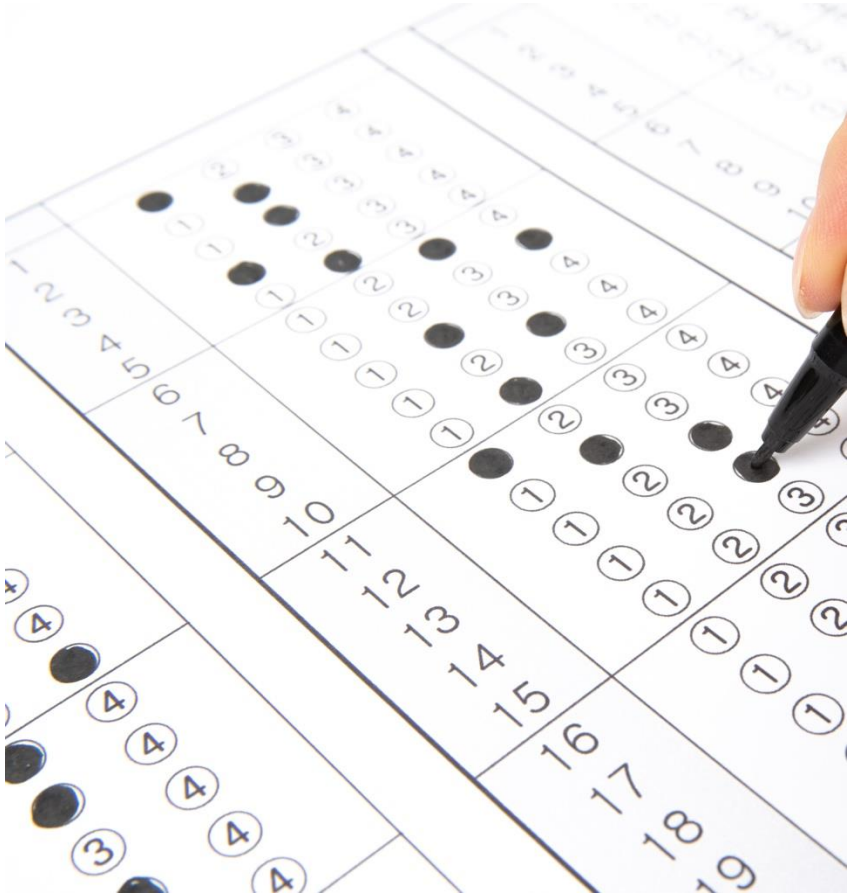
University at Buffalo®

Schedule

1. Quiz #1
2. Course Updates
3. Bipartite Graph
4. Directed Graphs

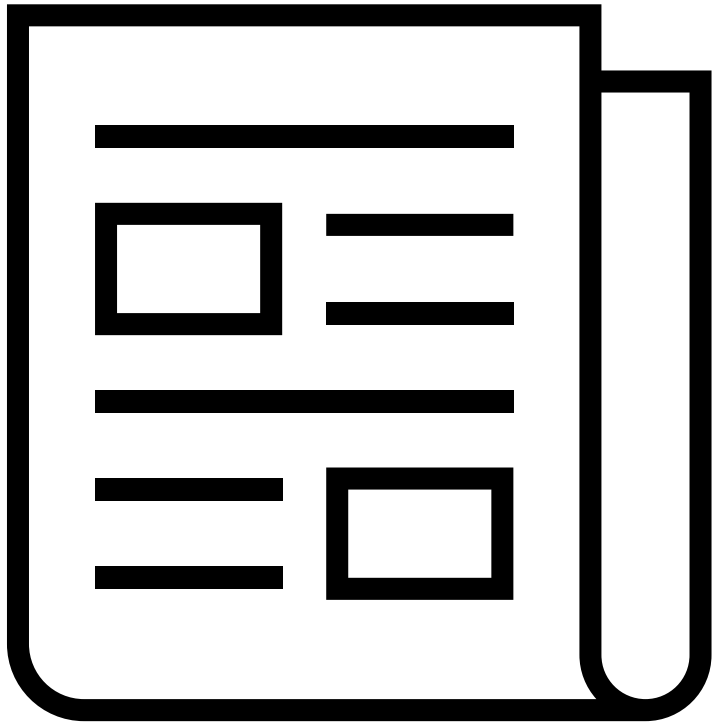
Quiz #1

- You will have 10 minutes once I say start.
- I will remind you at 5,2, and 1 minute.
- Do not cheat!



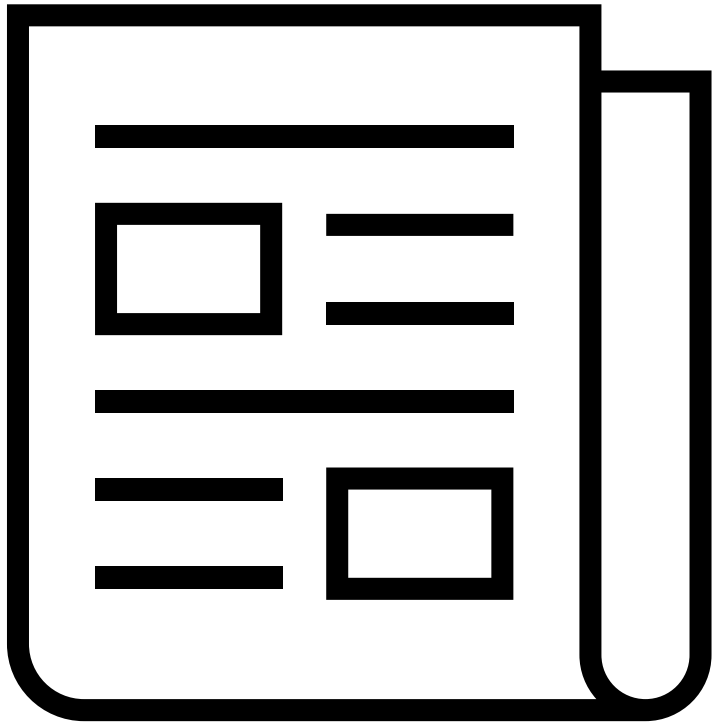
Course Updates

- HW 2 Grading Out Soon
- HW 3 Solutions Out Wednesday
- HW 4 Out Tomorrow
- Group Project
  - Team Emails Out
  - First Autolab Up Soon
- Sample Midterms Out
- Midterms Oct 6 and Oct 8



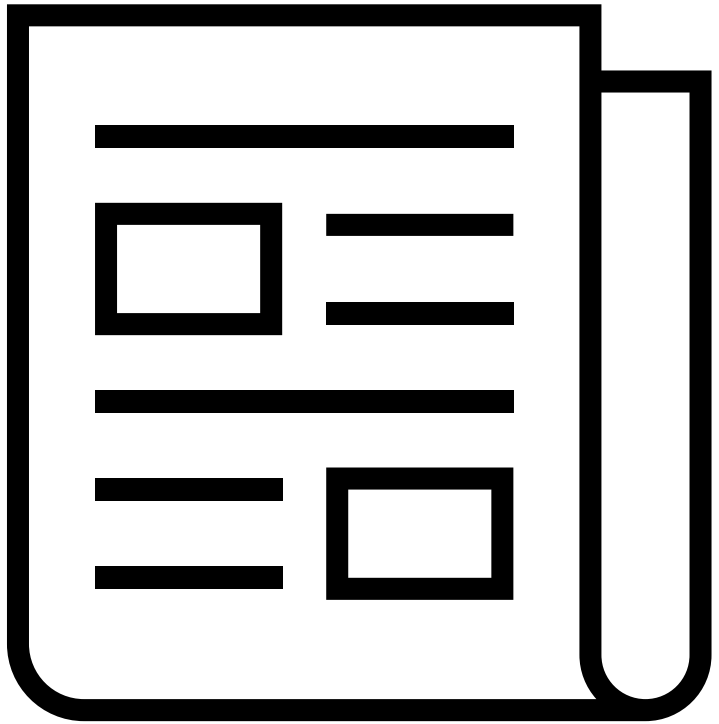
Midterms

- Two days during normal lecture times in this room.
- What do you need to know?
  - KT Chapters 1 – 3
    - Stable Matchings
    - Algorithm Analysis
    - Graph Basics
  - Lectures up until Today/Wednesday



Midterms

- There are sample midterms on Piazza along with solutions
  - These tell you the format and give you some nice sample problems but don’t tell you everything you need to know.
- You get a reference sheet!



Problem: Bipartite Graph

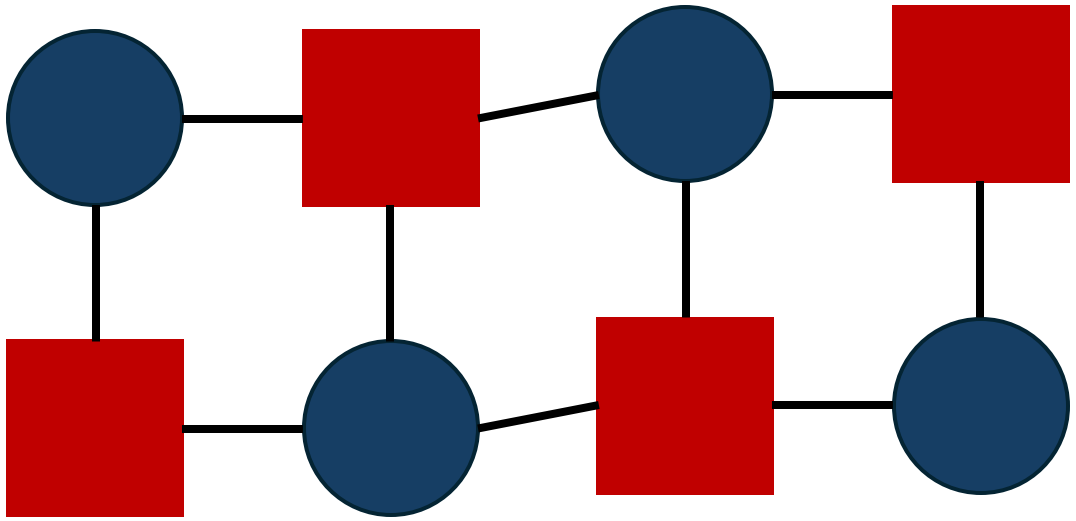
Input: A graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$

Output: True if G is bipartite and False otherwise.

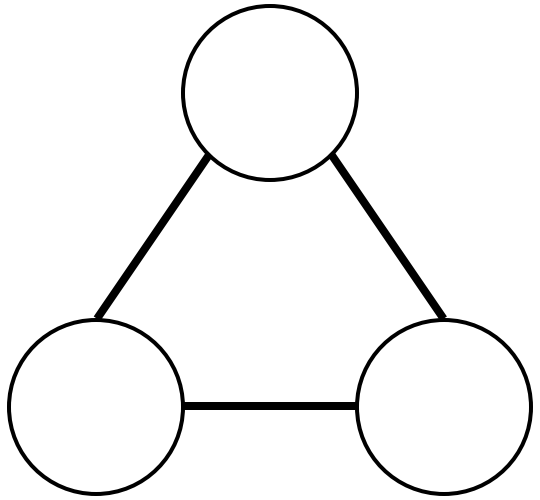
Def: We say that a graph is bipartite if we can partition the vertices V into two groups L and R such that each edge has one endpoint in L and the other endpoint in R.

Problem: Bipartite Graph

Def: We say that a graph is bipartite if we can partition the vertices V into two groups L and R such that each edge has one endpoint in L and the other endpoint in R.



Bipartite



Not Bipartite

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas:

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue.

Q: When will this fail?

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue. This will fail if there is a cross edge in one of the layers.

Q: What does this imply?

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue. This will fail if there is a cross edge in one of the layers. This implies there is an odd cycle in the graph.

Q: Is that a problem?

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue. This will fail if there is a cross edge in one of the layers. This implies there is an odd cycle in the graph. We can show that a graph is not bipartite if it contains an odd cycle.

Problem: Connected Components

Input: A graph $G = (V, E)$

Output: Return the number of connected components.

Algorithm Ideas:

Problem: Connected Components

Algorithm Ideas:

- Use an array to track of which nodes have been discovered.
- Use an array of arrays to keep track of connected components.
- While there is an undiscovered node, run BFS (or DFS) on node.
  - Mark nodes in connected component as discovered.
  - Add connected component to the array of connected components.

Question: What is the runtime?

Algorithm Ideas:

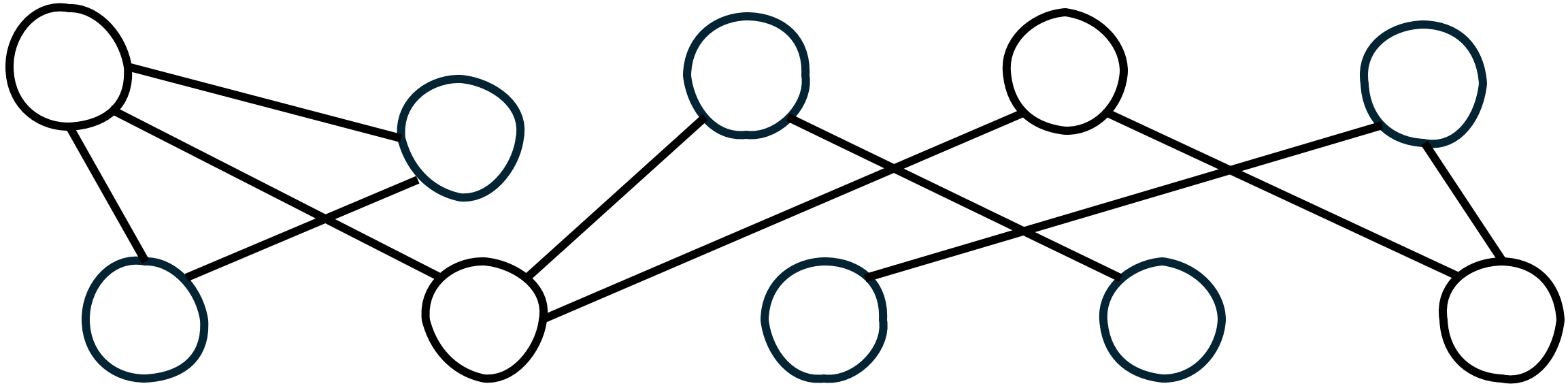
- Use an array to track of which nodes have been discovered.
- Use an array of arrays to keep track of connected components.
- While there is an undiscovered node, run BFS (or DFS) on node.
  - Mark nodes in connected component as discovered.
  - Add connected component to the array of connected components.

Answer: O(m+n)

Algorithm Ideas:

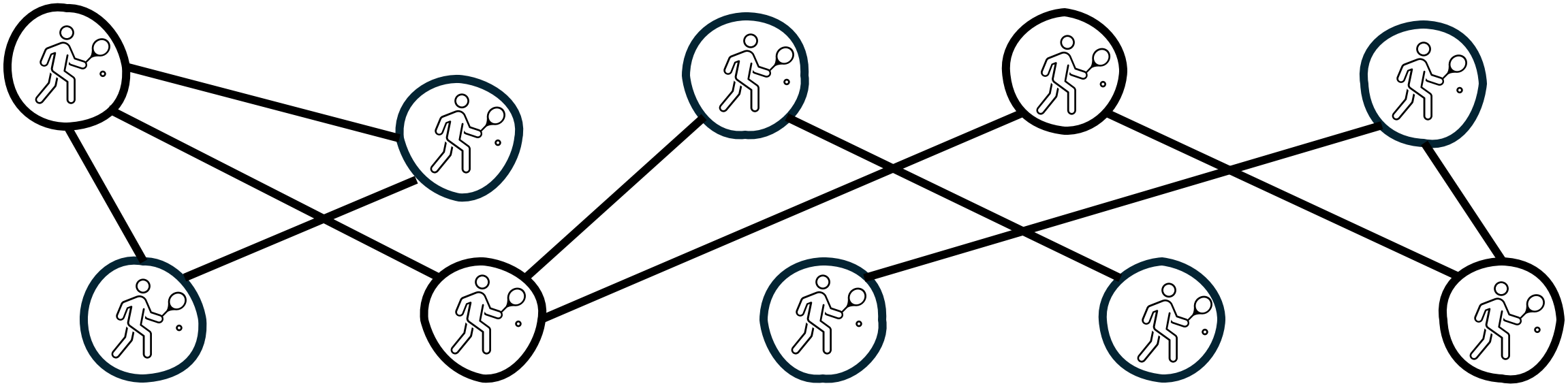
- Use an array to track of which nodes have been discovered. O(n)
- Use an array of arrays to keep track of connected components. O(1)
- While there is an undiscovered node s O(n)
  - Run BFS (or DFS) on node s to find CC(S). $\sum O(CC(s)) = O(n)$
  - Mark nodes in connected component as discovered. “”
  - Add connected component to the array of connected components. “”

Question: What do graphs represent?

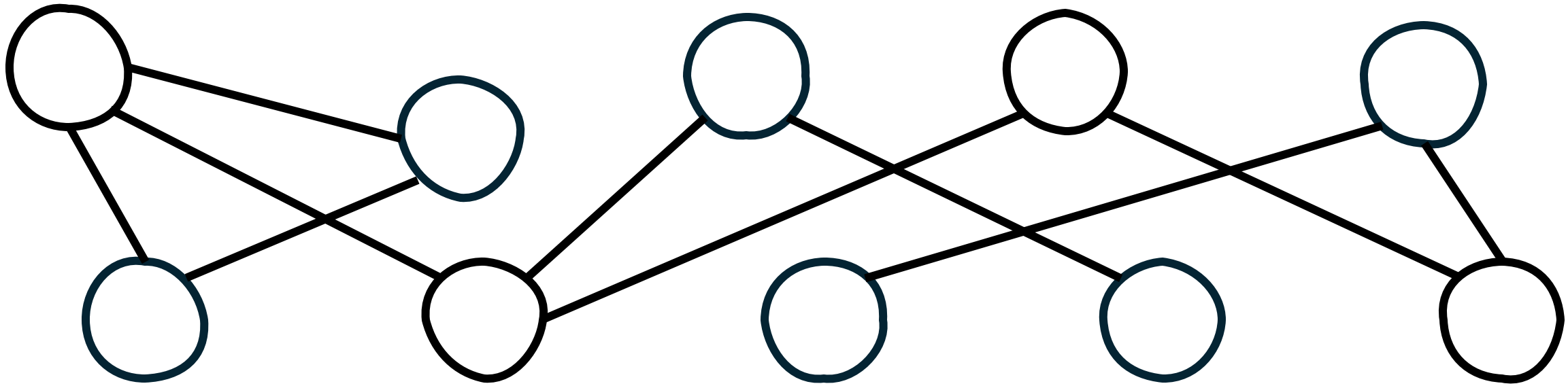


Answer: Symmetric relationships

E.g.: Each node represents a player, and an edge represents if they have played a game together today.

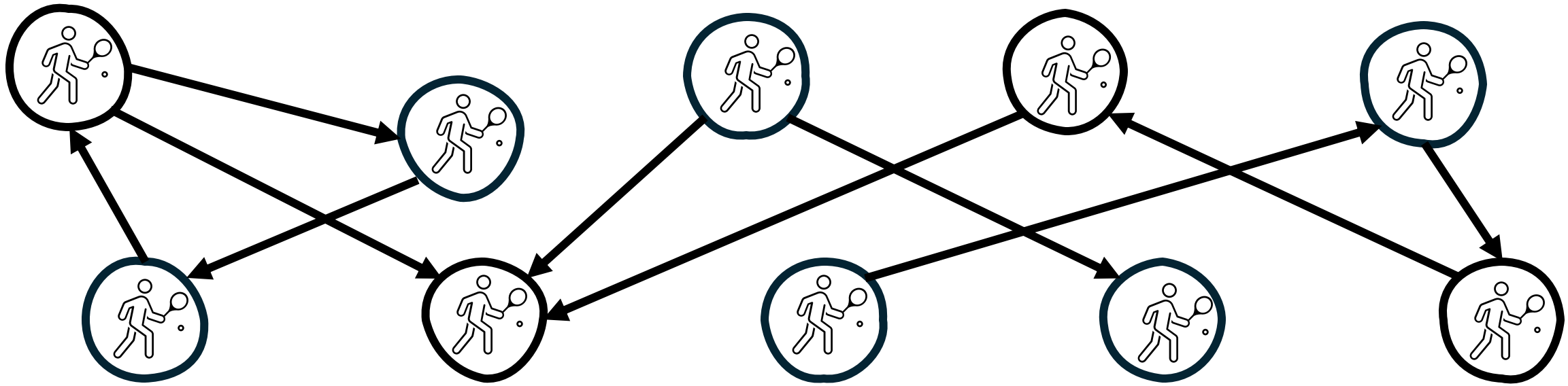


Question: How can we make graphs more general?



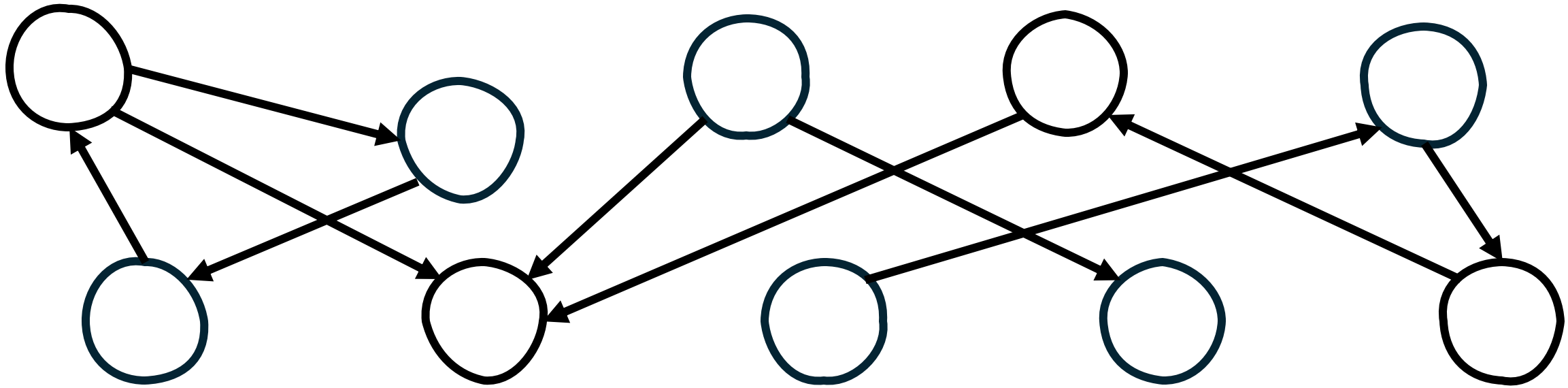
Answer: Asymmetric relationships

E.g.: Each node represents a player, and an edge represents if they have played a game together today and who won!



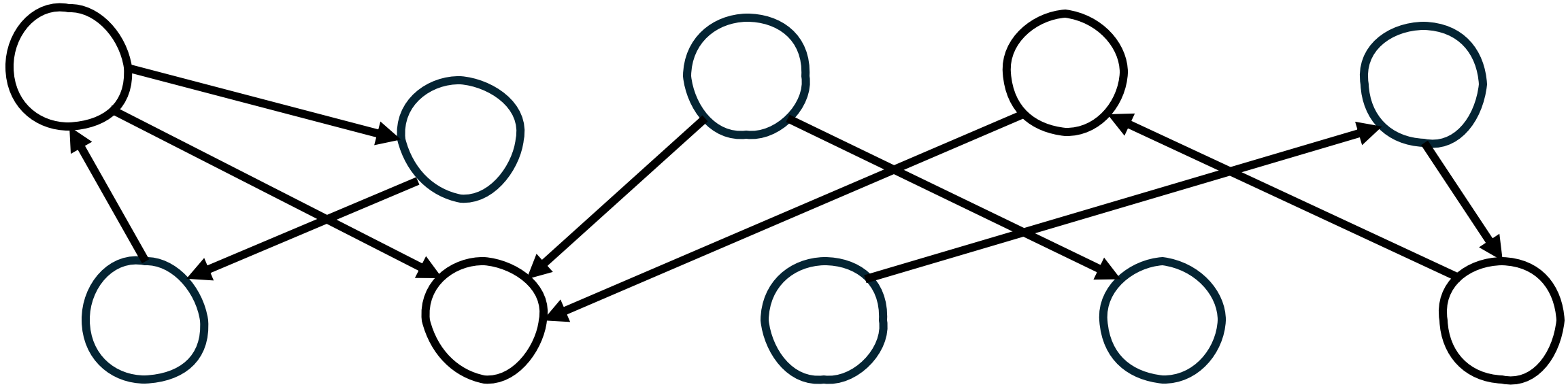
Answer: Asymmetric relationships

Definition: A directed graph is a graph $G = (V, E)$ such that each edge has a direction (e.g. (u,v) is an edge from u to v).

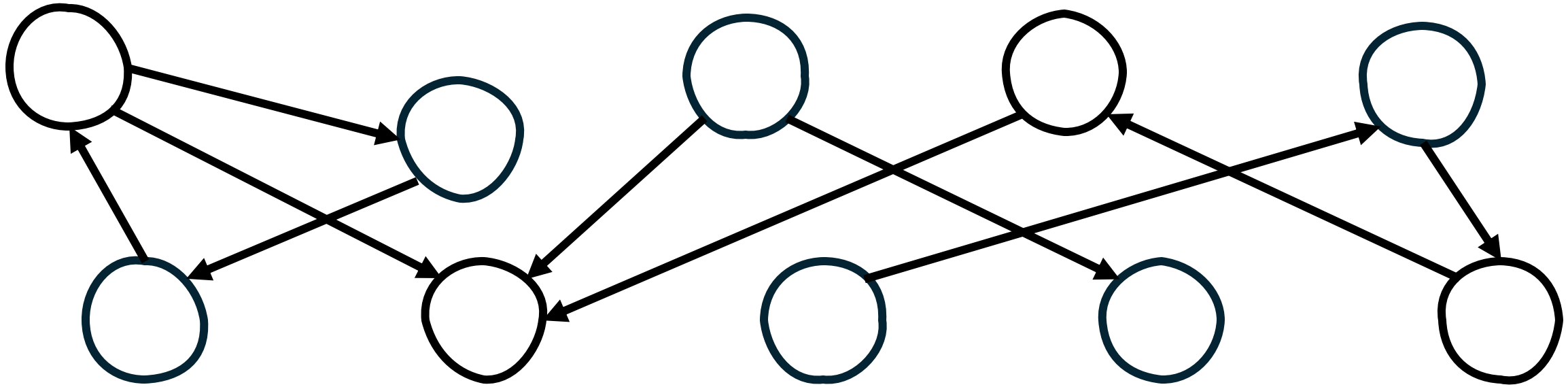


Answer: Asymmetric relationships

Notes: Adj. matrix is not symmetric, and adj. list has two lists per vertex.

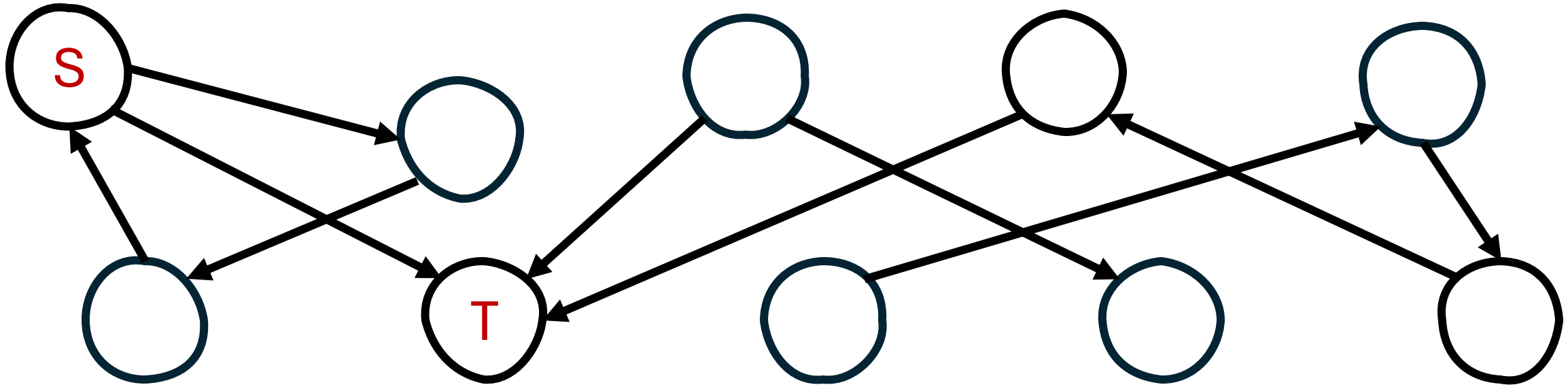


Question: Can we do BFS and DFS?



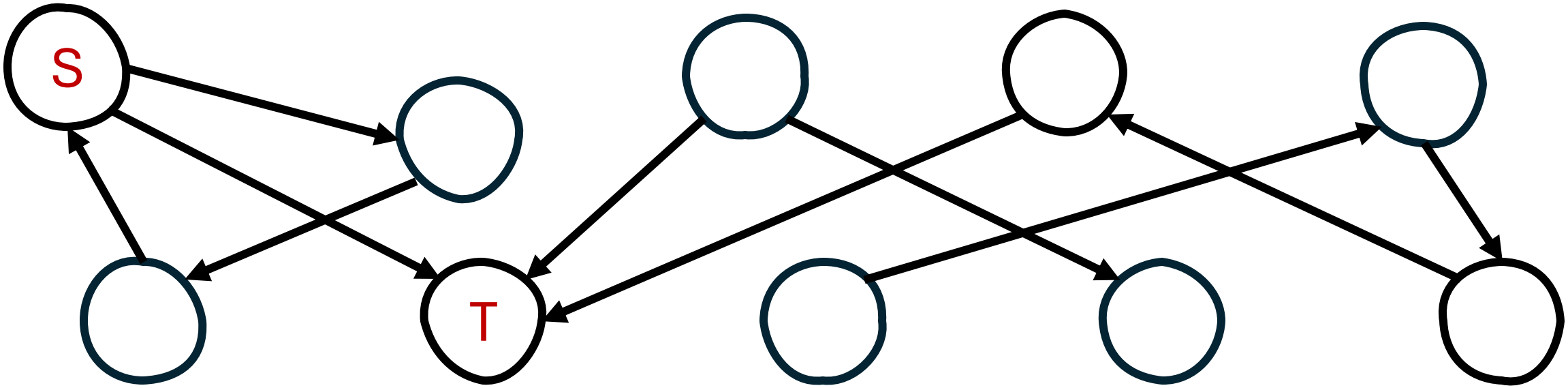
Answer: BFS and DFS mostly work.

- Only look at outgoing edges from a vertex.
- BFS and DFS return the nodes that can be reached from a starting vertex s but not every node in that set can reach s.



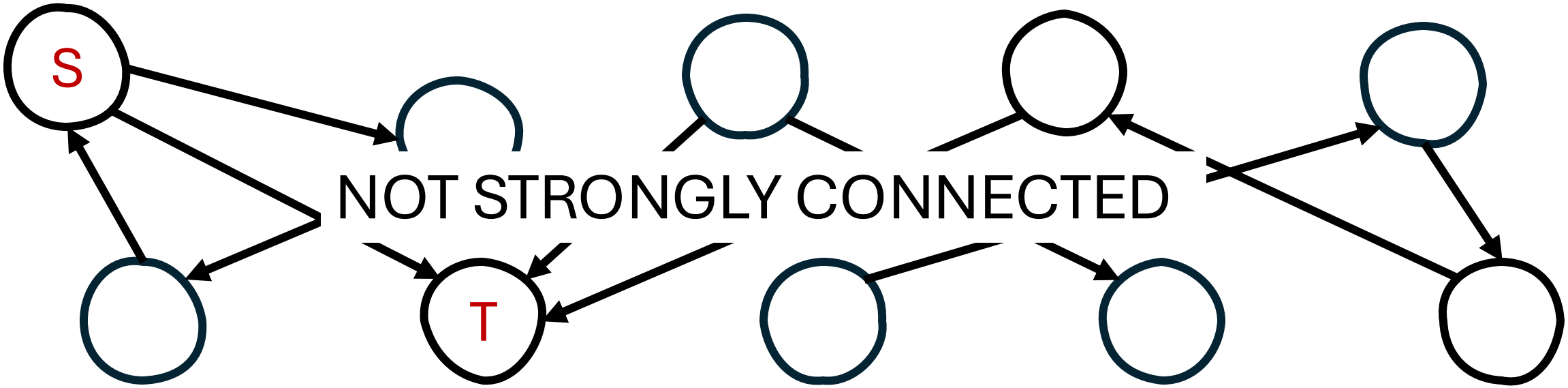
Strong Connectivity

Definition: We say that a directed graph is **strongly connected** if for every two vertices $u, v \in V$, there exists a directed path from u to v and a directed path from v to u.



Strong Connectivity

Definition: We say that a directed graph is **strongly connected** if for every two vertices $u, v \in V$, there exists a directed path from u to v and a directed path from v to u.



Strong Connectivity Property

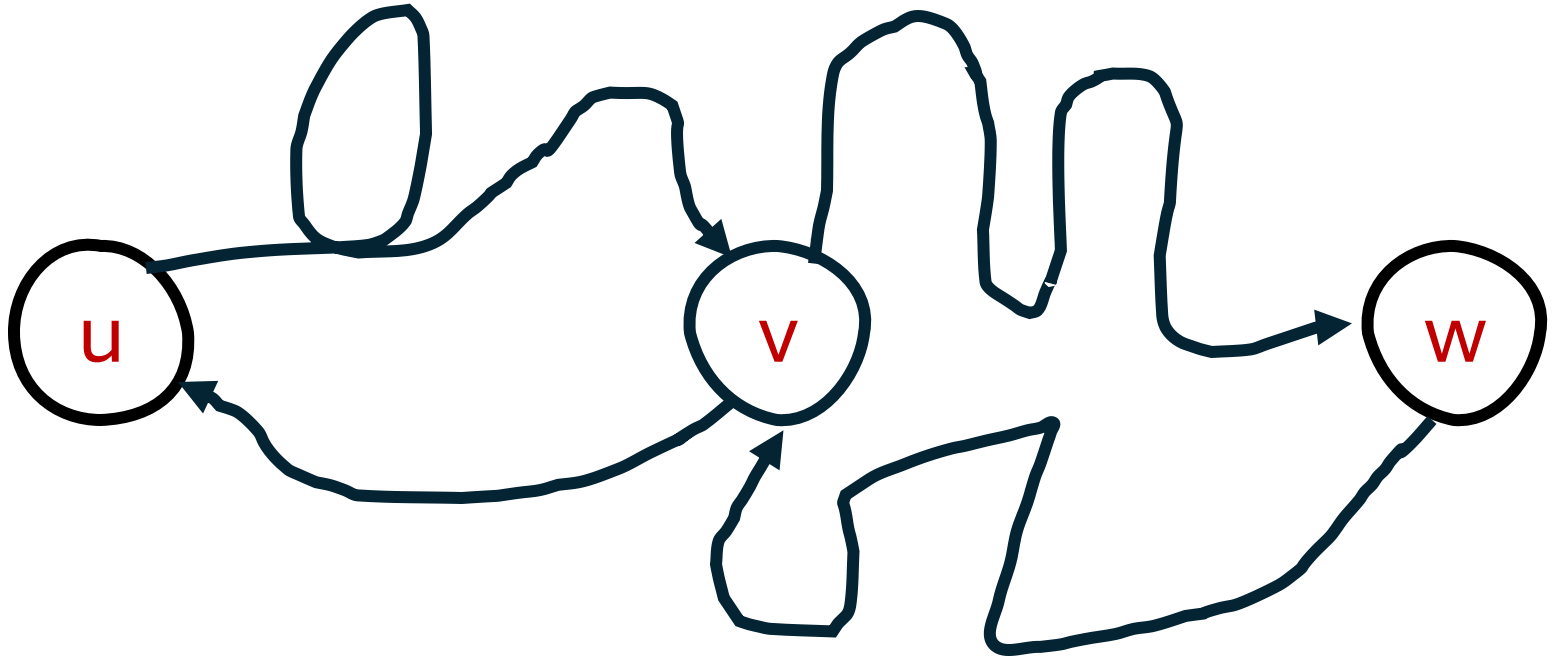
Claim: If u and v are mutually reachable, and v and w are mutually reachable, then u and w are mutually reachable.

Proof Idea:

Strong Connectivity Property

Claim: If u and v are mutually reachable, and v and w are mutually reachable, then u and w are mutually reachable.

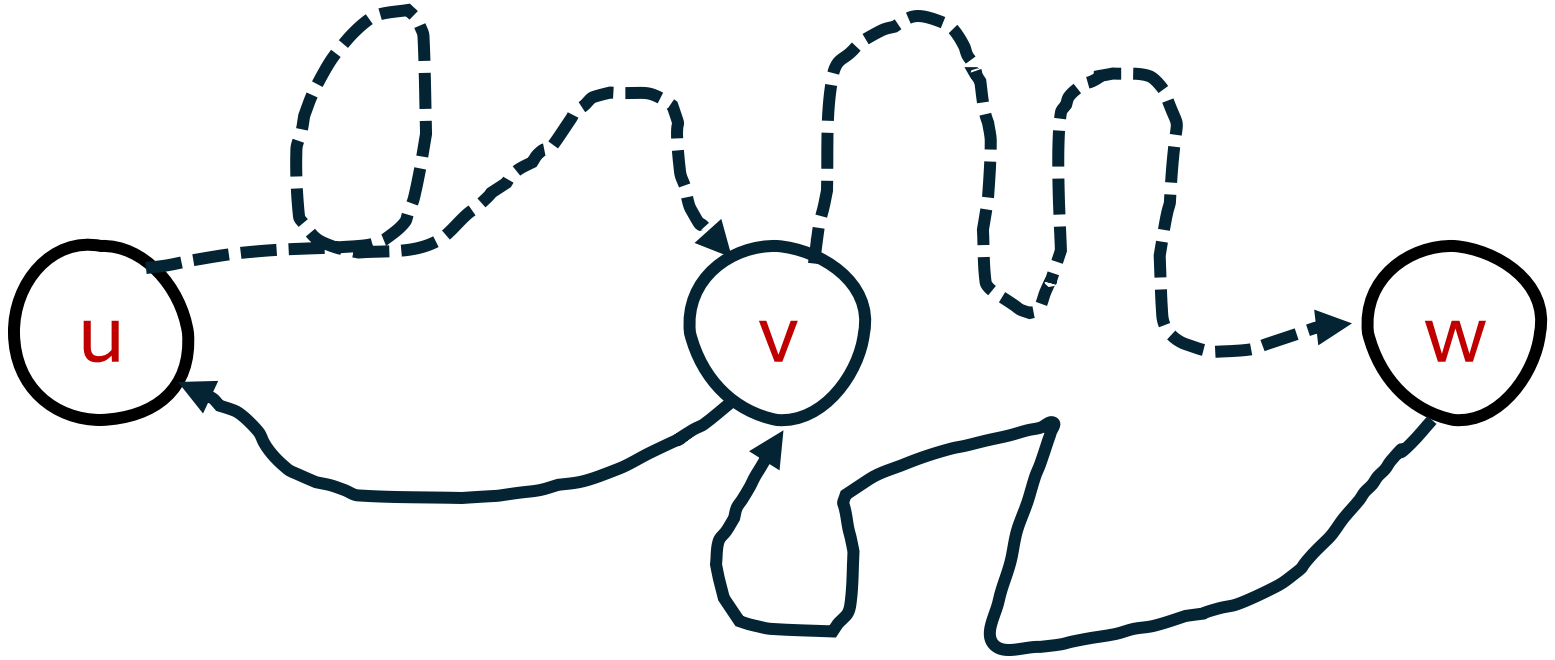
Proof Idea:



Strong Connectivity Property

Claim: If u and v are mutually reachable, and v and w are mutually reachable, then u and w are mutually reachable.

Proof Idea:



Strong Connectivity Algorithm

Input: Directed graph $G = (V, E)$
Output: True if strongly connected and False otherwise.

Proof Idea:

- Pick a vertex s in V
- Use BFS to find all vertices I can reach from s.
- Use _____ to find all vertices that can reach s.
- If both sets are equal return true and otherwise return false.

Strong Connectivity Algorithm

Input: Directed graph $G = (V, E)$
Output: True if strongly connected and False otherwise.

Proof Idea:

- Pick a vertex s in V
- Use BFS to find all vertices I can reach from s.
- Use BFS on “Reversed Graph” to find all vertices that can reach s.
- If both sets are equal return true and otherwise return false.

Strong Connectivity Algorithm

Input: Directed graph $G = (V, E)$
Output: True if strongly connected and False otherwise.

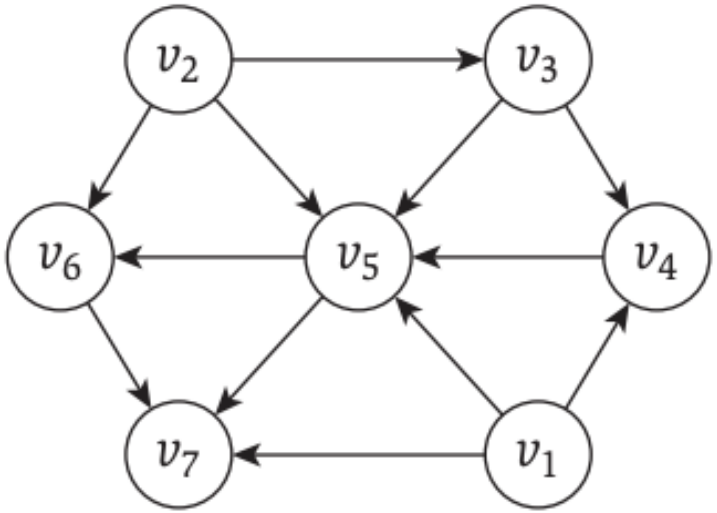
Proof Idea:

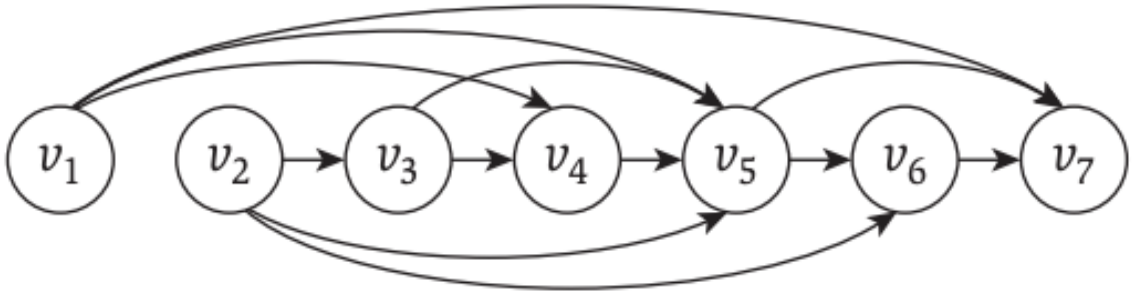
- Pick a vertex s in V
- Use BFS to find all vertices I can reach from s.
- Use BFS on "Reversed Graph" to find all vertices that can reach s. For each edge (u,v) replace with edge (v,u)
- If both sets are equal return true and otherwise return false.

Directed Acyclic Graphs (DAGs)

Definition: A directed graph is a DAG if it has no directed cycles.

Definition: A topological ordering of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j), we have i < j.

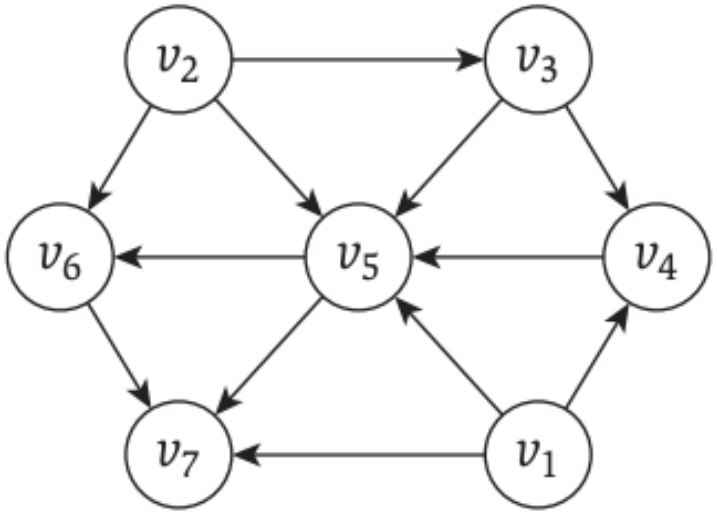


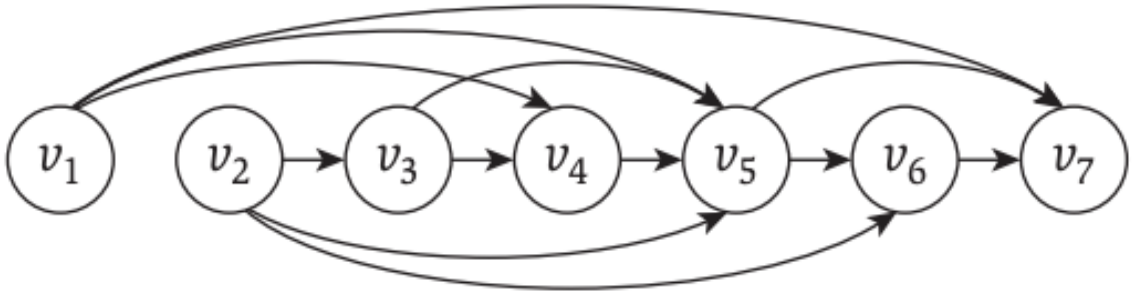


Directed Acyclic Graphs (DAGs)

Definition: A directed graph is a DAG if it has no directed cycles.

Definition: A topological ordering of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j), we have i < j.





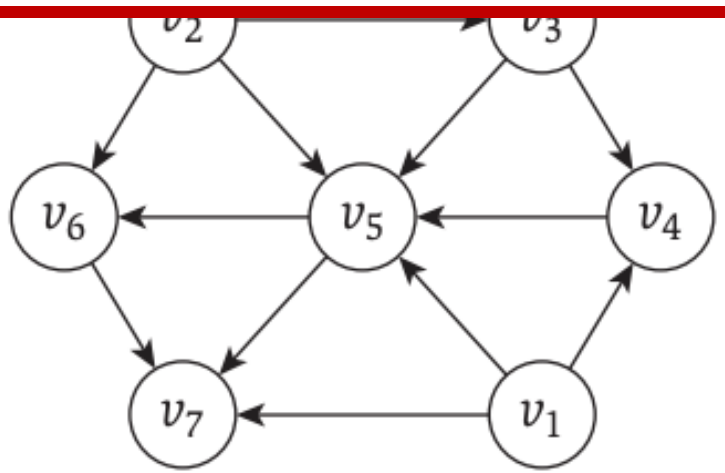
All edges are going “forward”

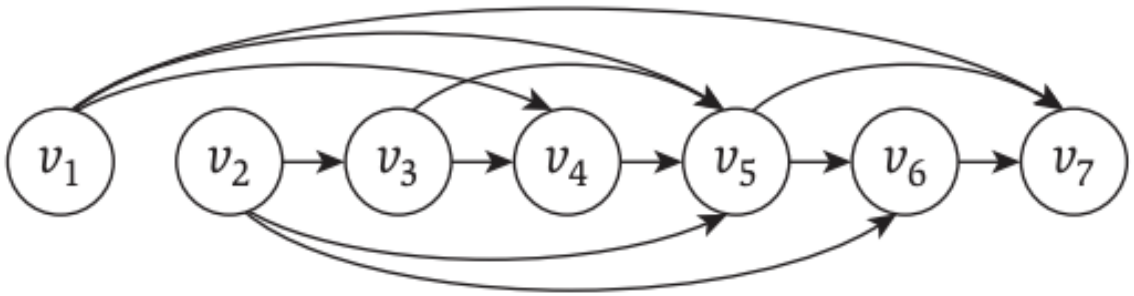
Directed Acyclic Graphs (DAGs)

Definition: A directed graph is a DAG if it has no directed cycles.

Definition: A topological ordering of a directed graph $G =$

Read KT Section 3.6 and Review Care Packaged on Topological Ordering





All edges are going “forward”

Midterm Check Point

What is next?

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flows (maybe)
- Computation Complexity

“How do we design new algorithms?”

- **Greedy Algorithms**
- **Divide and Conquer**
- **Dynamic Programming**
- **Network Flows (maybe)**
- Computation Complexity

“How do we use reduce another problem?”

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- **Network Flows (maybe)**
- **Computation Complexity**

“How do we know when to give up?”

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flows (maybe)
- **Computation Complexity**

What are Greedy Algorithm?

What are Greedy Algorithm?

- Build solution one piece at a time.
- Only look at immediate information to make choices.
- Never go back on a decision.
- NOT ALWAYS THE BEST CHOICE!

Coin Change Problem

- **Problem**: Given U.S. currency denominations {1.00, 0.25, 0.10, 0.05, 0.01} find an algorithm to pay an amount to a customer using the fewest coins possible.

Coin Change Problem

- **Problem**: Given U.S. currency denominations {1.00, 0.25, 0.10, 0.05, 0.01} find an algorithm to pay an amount to a customer using the fewest coins possible.

- **Algorithm**: At each iteration, add a coin of the largest value that is less than the amount needed to be paid.

Q: Is this algorithm always optimal?

- **Problem**: Given U.S. currency denominations {1.00, 0.25, 0.10, 0.05, 0.01} find an algorithm to pay an amount to a customer using the fewest coins possible.

- **Algorithm**: At each iteration, add a coin of the largest value that is less than the amount needed to be paid.