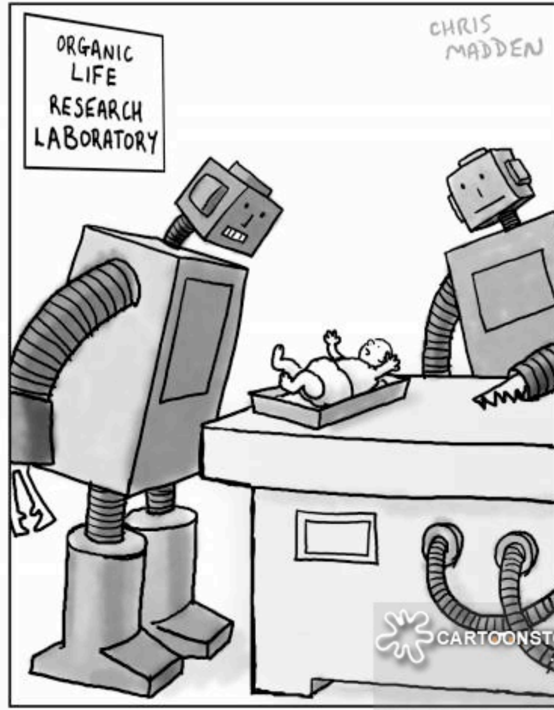
EE 360C - ALGORITHMS

Lecture 9
Graphs 2

Vallath Nandakumar
University of Texas at Austin



"We'll know whether to treat it with any special moral consideration when we see if it passes the Turing test."

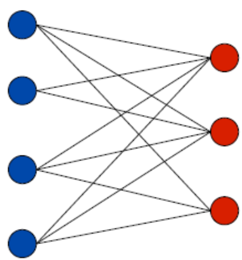
BIPARTITE GRAPH

☞ Bipartite Graph

- Undirected graph $G = (V,E)$ where nodes can be colored red or blue such that all edges have one red end and one blue end

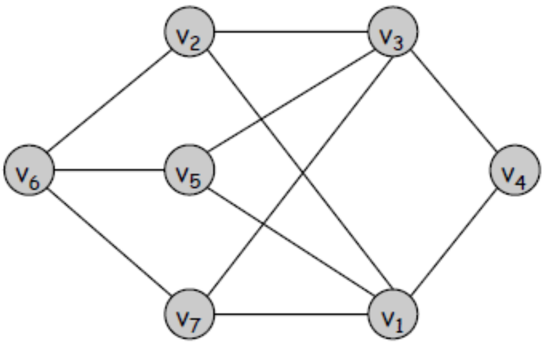
☞ Applications

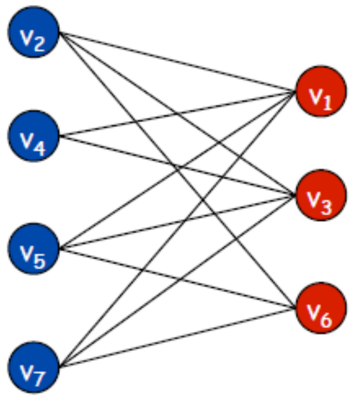
- Stable marriage: men = red, women = blue
- Scheduling: machines = red, jobs = blue



TESTING BIPARTITENESS

- Checking whether graph bipartite
  - Many graph problems become
    - Easier if underlying graph bipartite (matching)
    - Tractable if underlying graph bipartite (independent set)





BIPARTITE GRAPH

- ☞ Lemma
  - If graph bipartite, cannot contain odd length cycle
- ☞ Proof Sketch
  - Not possible to “2-color” odd cycle

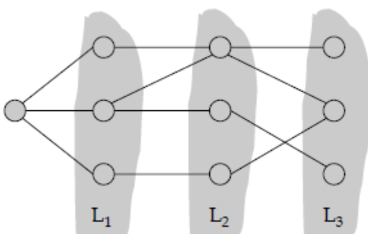
bipartite
(2-colorable)

not bipartite
(not 2-colorable)

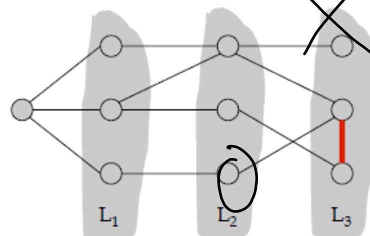
BIPARTITE GRAPH

- Lemma
  - Let G be connected graph and L_0, …, L_k be layers produced by BFS starting at node s
  - Exactly one of following holds
    - 1) No edge of G joins two nodes of same layer
      - – G is bipartite
    - 2) Edge of G joins two nodes in same layer
      - – G contains odd length cycle and not bipartite

L_3



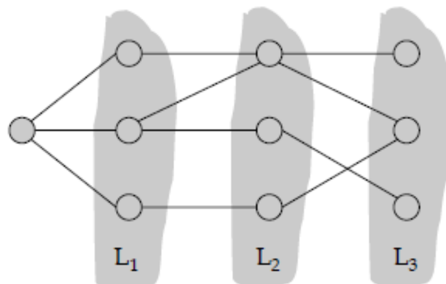
Case 1



Case 2

BIPARTITE GRAPH

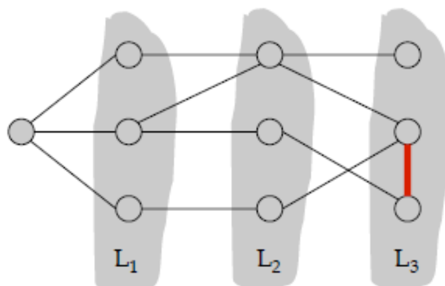
- ☞ Prove that if no edge of *G* joins two nodes of same layer of a BFS search tree, *G* is bipartite



Case 1

BIPARTITE GRAPH

☞ **Prove that if edge of *G* joins two nodes in same layer of a BFS search tree, *G* contains odd length cycle and not bipartite**

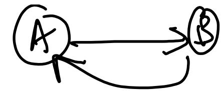


Case 2

DIRECTED GRAPH

- ☞ **Directed Reachability**
  - **Given node *s*, find all nodes reachable from *s***
- ☞ **Graph Search**
  - **BFS and DFS extend naturally to Directed Graph**

STRONG CONNECTIVITY



☞ **Mutually Reachable**

- **If path from node *u* to *v* and from *v* to *u*, then *u* to *v* are mutually reachable**

☞ **Strongly Connected**

- **Every pair of nodes mutually reachable**

☞ **Lemma**

- ***G* strongly connected iff every node reachable from *s* and *s* reachable from every node**

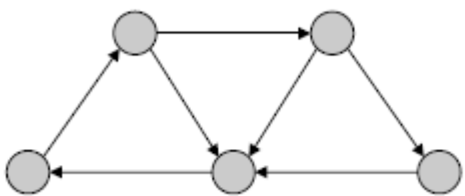
DETERMINING STRONG CONNECTIVITY

☞ **Theorem**

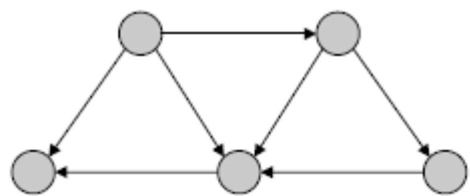
- **Can determine if *G* strongly connected in $O(m+n)$**

☞ **Algorithm**

- **Pick any node *s***
- **Run BFS from *s* in *G***
- **Run BFS from *s* in G_{rev} (reverse direction edges)**
- **Return true iff all nodes reached in both BFS runs**



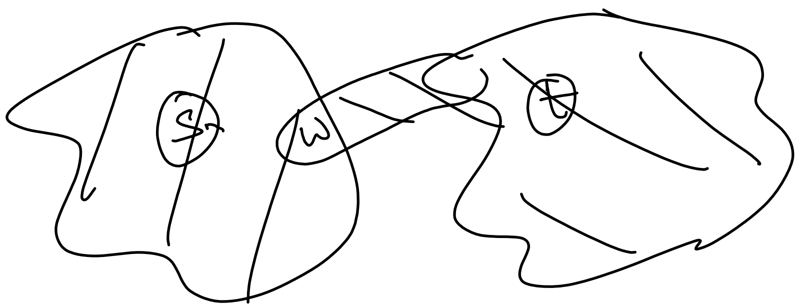
strongly connected



not strongly connected

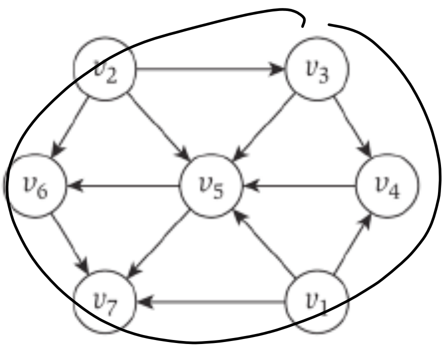
STRONG COMPONENTS

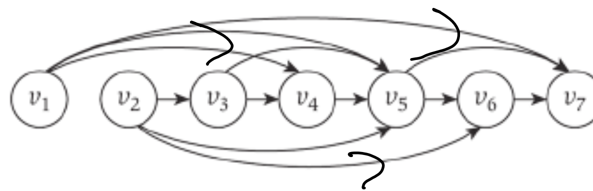
- Strong Component containing node *s* in directed graph
  - Set of all *v* such that *s* and *v* mutually reachable
- Theorem
  - For any two nodes *s* and *t* in directed graph, their strong components are either identical or disjoint



DIRECTED ACYCLIC GRAPHS

- Directed Acyclic Graph (DAG)
  - Contains no directed cycles
- Topological Order
  - Ordering of nodes $v_1, v_2, \ldots, v_n$ in directed graph so that $i > j$ for every edge (v_i, v_j)





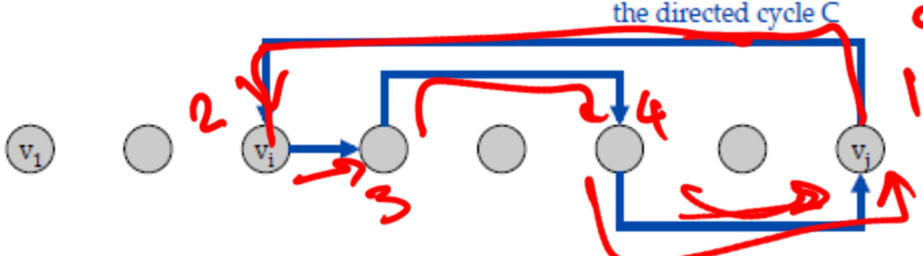
DIRECTED ACYCLIC GRAPHS

- ☞ **Precedence constraints**
  - **edge (v_i, v_j) means v_i must precede v_j**
- ☞ **Example Applications**
  - **Course prerequisite graph**
    - **Course v_i must be taken before v_j**
  - **Compilation**
    - **Module v_i must be compiled before v_j**
  - **Pipeline of computing jobs**
    - **Output of job v_i needed to determine input of job v_j**

DIRECTED ACYCLIC GRAPHS

- Lemma
  - If *G* has topological order, then *G* is DAG
- Proof (by contradiction)
  - Suppose *G* has topological order $v_1,\ldots,v_n$ and *G* has cycle
  - Let v_i be lowest-indexed node in cycle and let v_j be node just before v_i
  - By choice of *i*, we have $i < j$
  - Contradiction because topological order requires $j < i$ for edge (v_j, v_i)

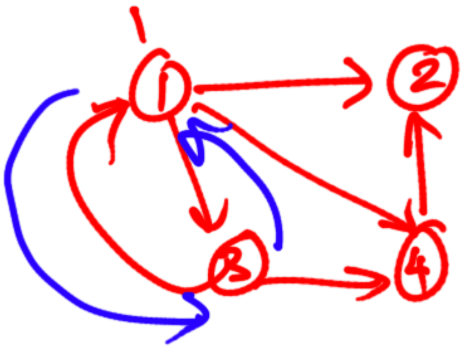
4 to 1 is decreasing

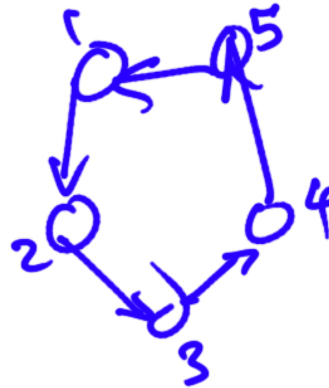


DIRECTED ACYCLIC GRAPHS

☞ **Lemma**

- **If *G* is DAG, then *G* has a node with no incoming edges**





DIRECTED ACYCLIC GRAPHS

☞ **Lemma**

- **If *G* is DAG, then *G* has topological ordering**

A E B D C

DIRECTED ACYCLIC GRAPHS

☞ **Theorem**

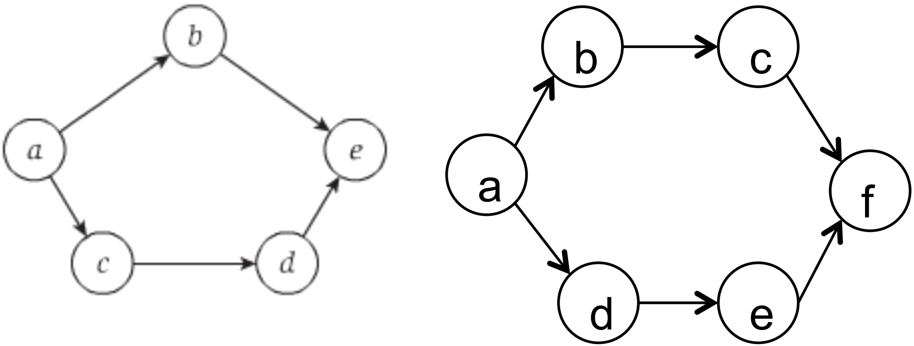
- **Algorithm finds topological order in $O(m+n)$ Time**

☞ **Proof**

- **Maintain following information**
  - ***count*[*w*] – count of incoming edges for node *w***
  - ***S* – remaining nodes with no incoming edges**
- **Initialization takes $O(m+n)$ via single scan through graph**
- **Update to delete *v* takes $O(m)$**
  - **Remove *v* from *S***
  - **Decrement *count*[*w*] for all edges *v* to *w* and add *w* to *S* if *count*[*w*] hits 0**

iClicker

☞ **How many topological orderings do the following graphs have?**

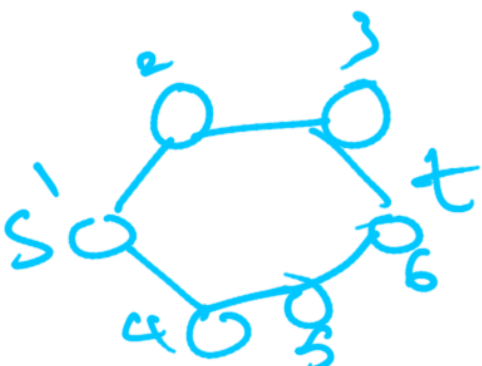


A: 3, 6 **B: 2, 4** **C: 3, 4** **D: 4, 5** **E: Other**

EXERCISE

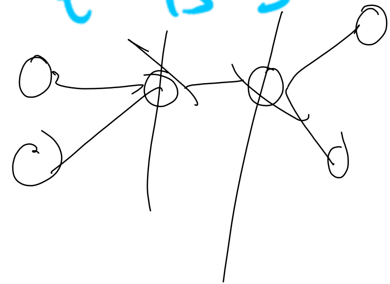
☞ *n* node undirected graph contains two nodes *s* and *t*

- If distance between *s* and *t* greater than *n*/2, show some node *v* exists such that deleting *v* destroys all paths from s to t.
- Hint: Use BFS and layer concepts



6 nodes. Distance from s to t is 3.

6



EXERCISE

- ☞ **Given undirected graph *G*, for some pair of nodes *v* and *w*, give algorithm to compute number of shortest paths from *v* to *w* in *O*(*m*+*n*).**
  - **Hint: Use BFS and layer concepts**