CSL 356: Analysis and Design of Algorithms

Ragesh Jaiswal

CSE, IIT Delhi

Graph algorithms: BFS

- Problem: Given a graph $G = (V, E)$ check if the graph is *bipartite*.

isBipartite (G)

- Run BFS(G) and check if two vertices in the same layer has an edge between them.

If yes then output("no") else output("yes")

- Running time: $O(n + m)$

- Question: Given a strongly connected bipartite graph, does it have a unique bipartition?

Graph Algorithms: DFS

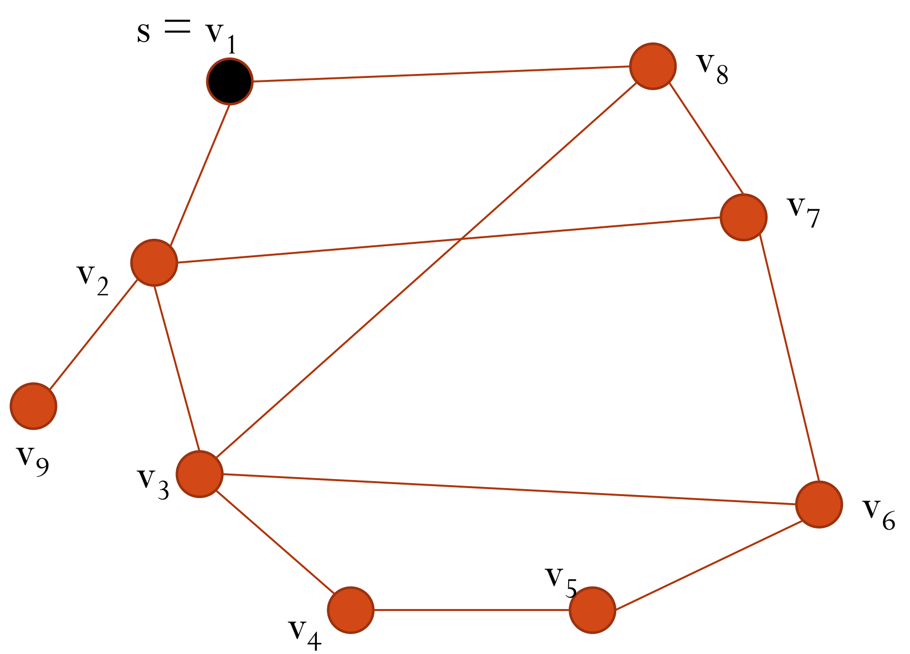
- Depth First Search (DFS):

DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)

Graph Algorithms: DFS

- Depth First Search (DFS):

DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
    - Recursively call DFS(v)

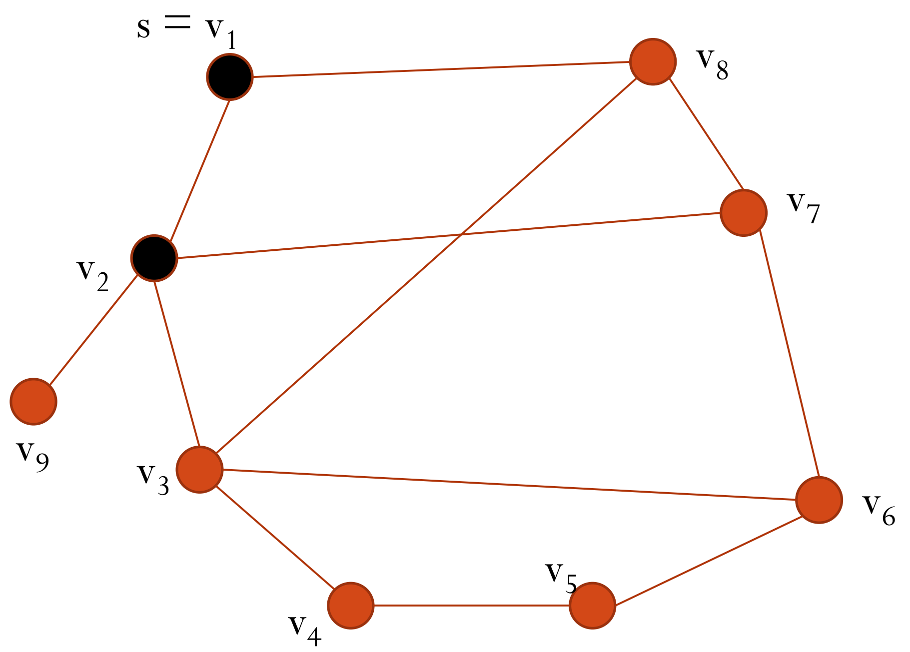


Graph Algorithms: DFS

- Depth First Search (DFS):

DFS(s)

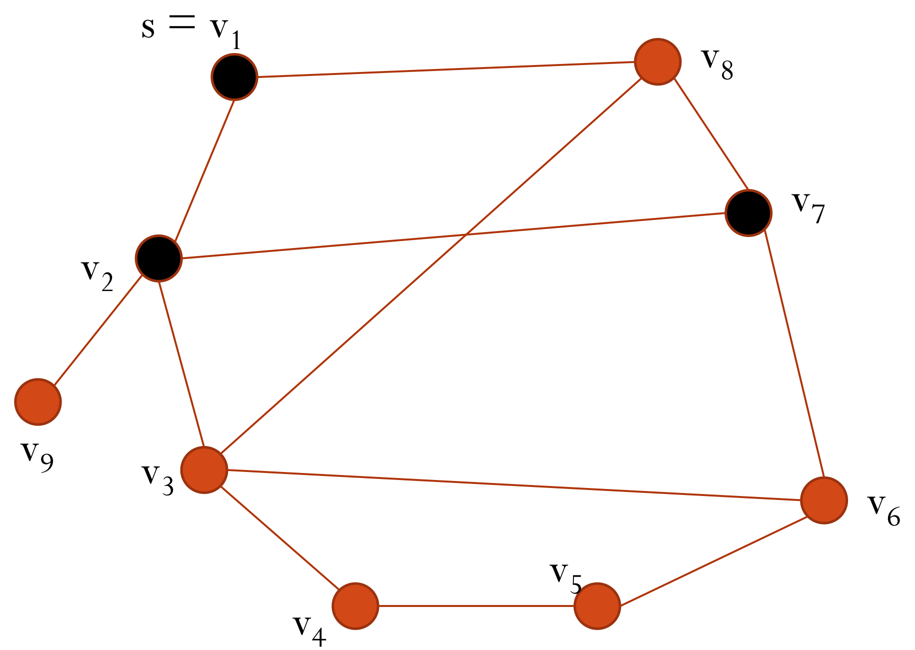
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

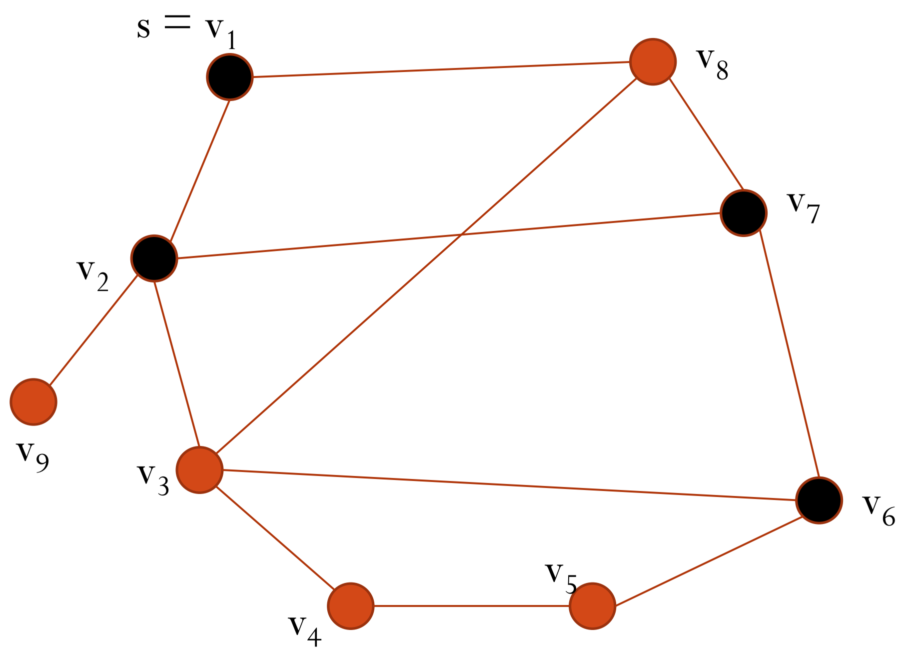
DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

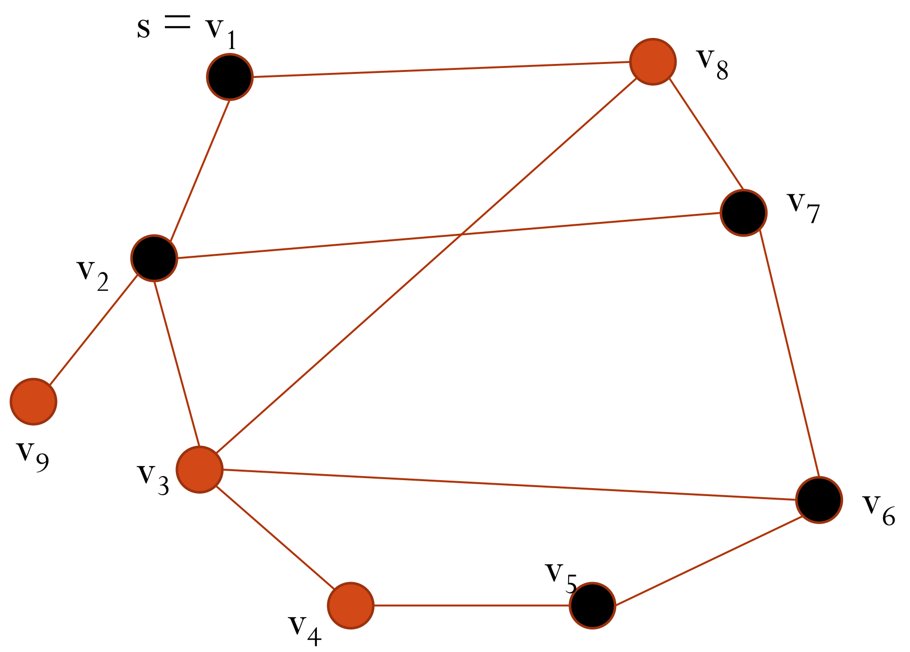
DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

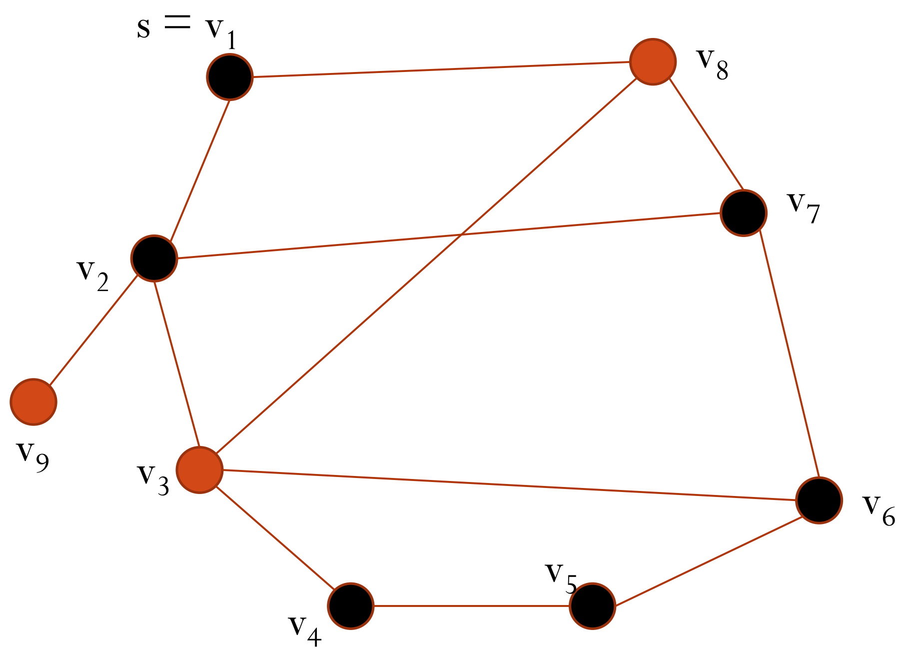
DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

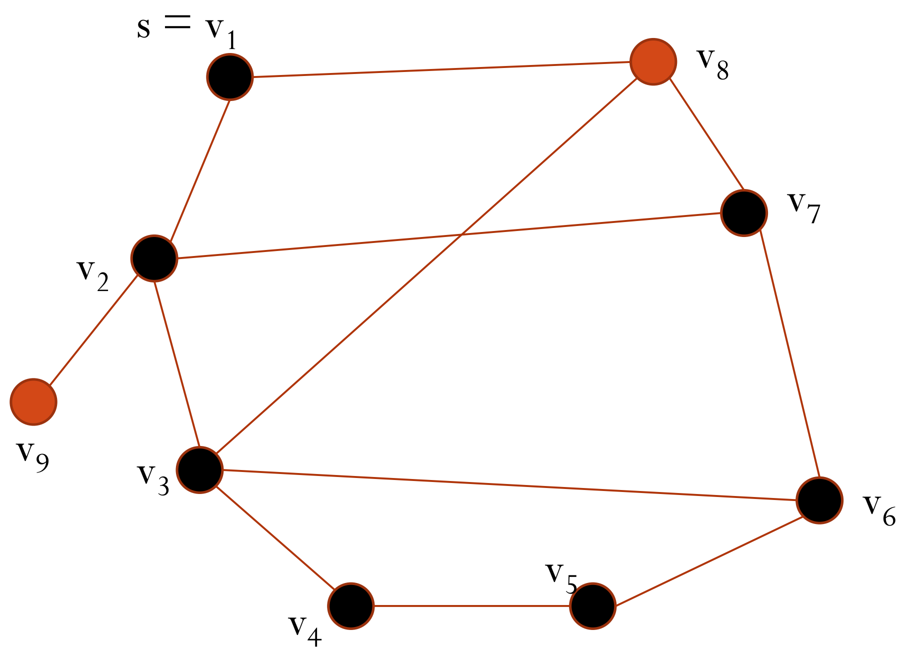
DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)

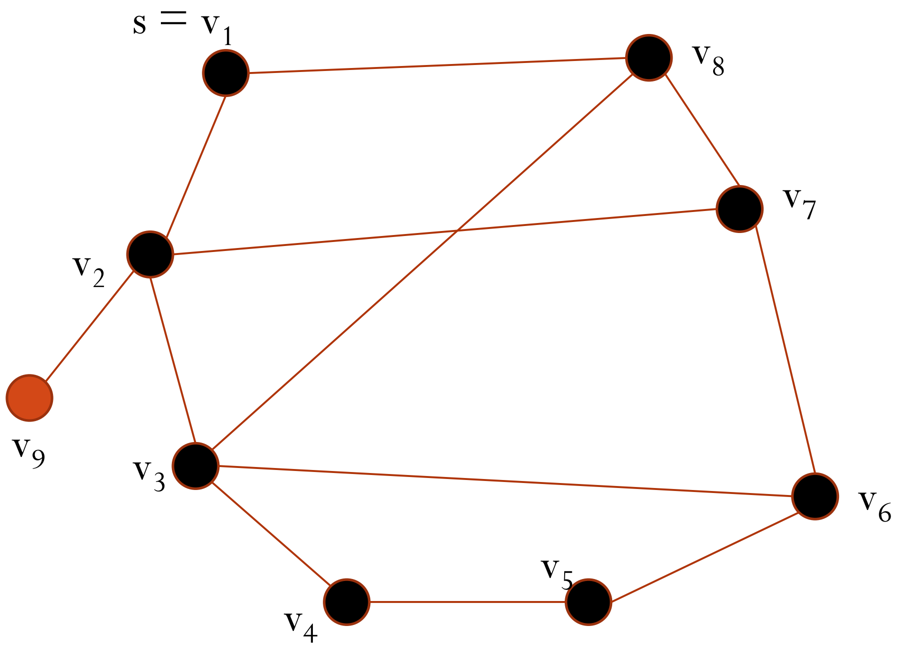


Graph Algorithms: DFS

- Depth First Search (DFS):

DFS(s)

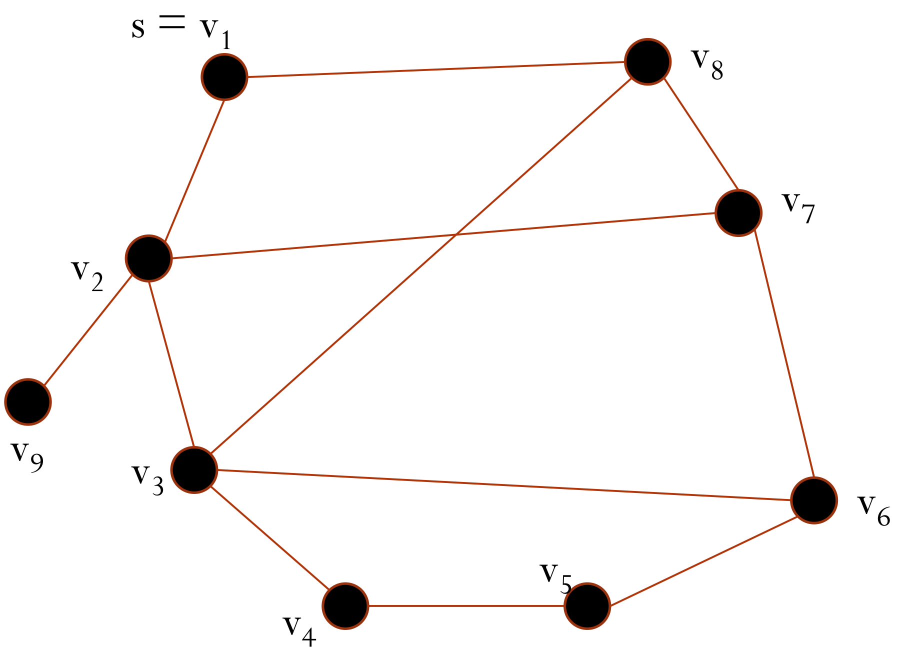
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

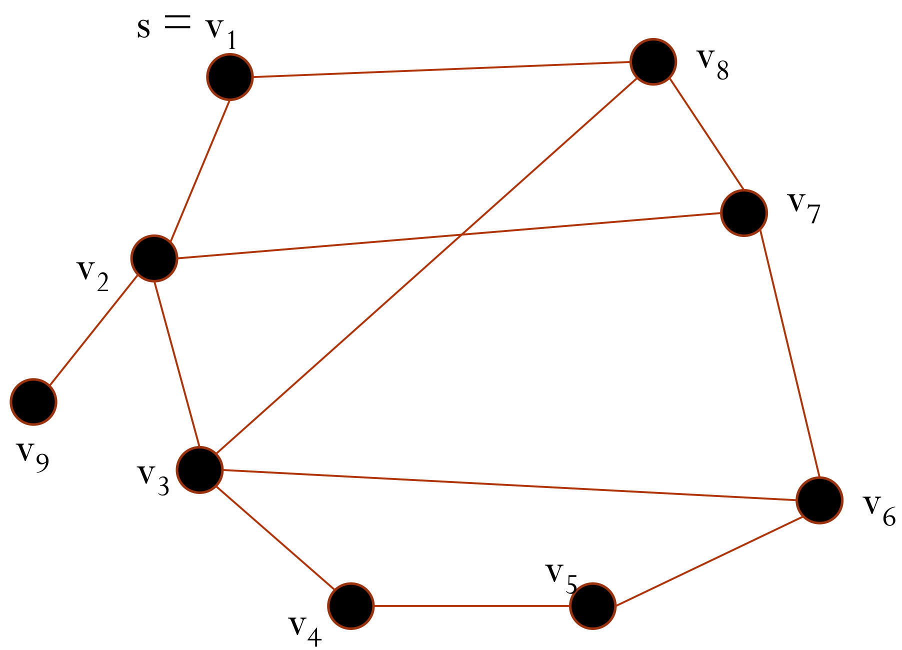
DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



Graph Algorithms: DFS

- Depth First Search (DFS):

DFS(s)
- Mark s as "explored"
- For each unexplored neighbor v of s
  - Recursively call DFS(v)



- Running time: $O(n + m)$

Graph Algorithms: DFS

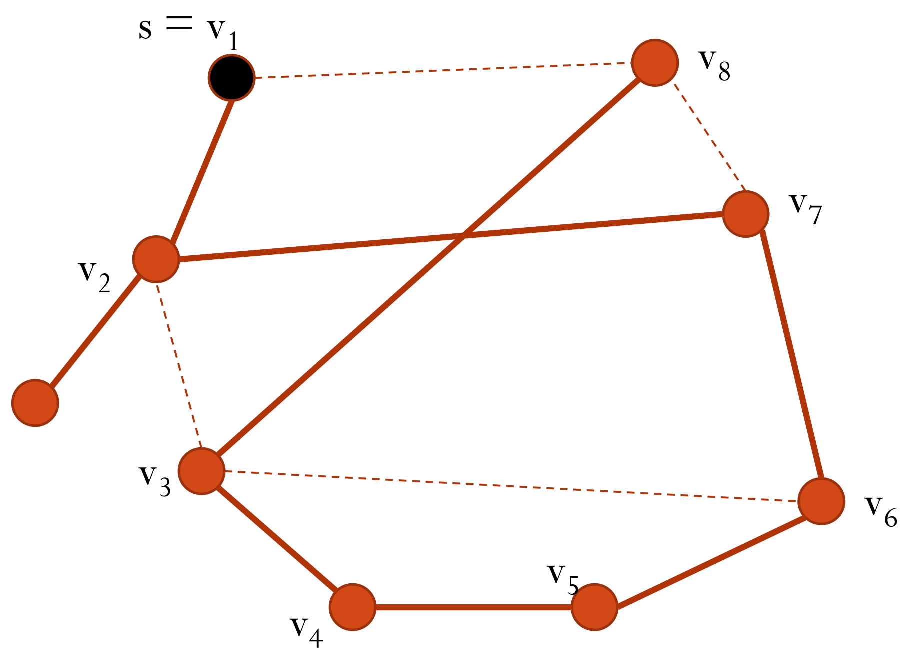
- Depth First Search (DFS):

> DFS(s)
> - Mark s as "explored"
> - For each unexplored neighbor v of s
>   - Recursively call DFS(v)

- The DFS algorithm defines the following "DFS tree" rooted at s:
  - Vertex u is the parent of vertex v if u caused the immediate discovery of v.

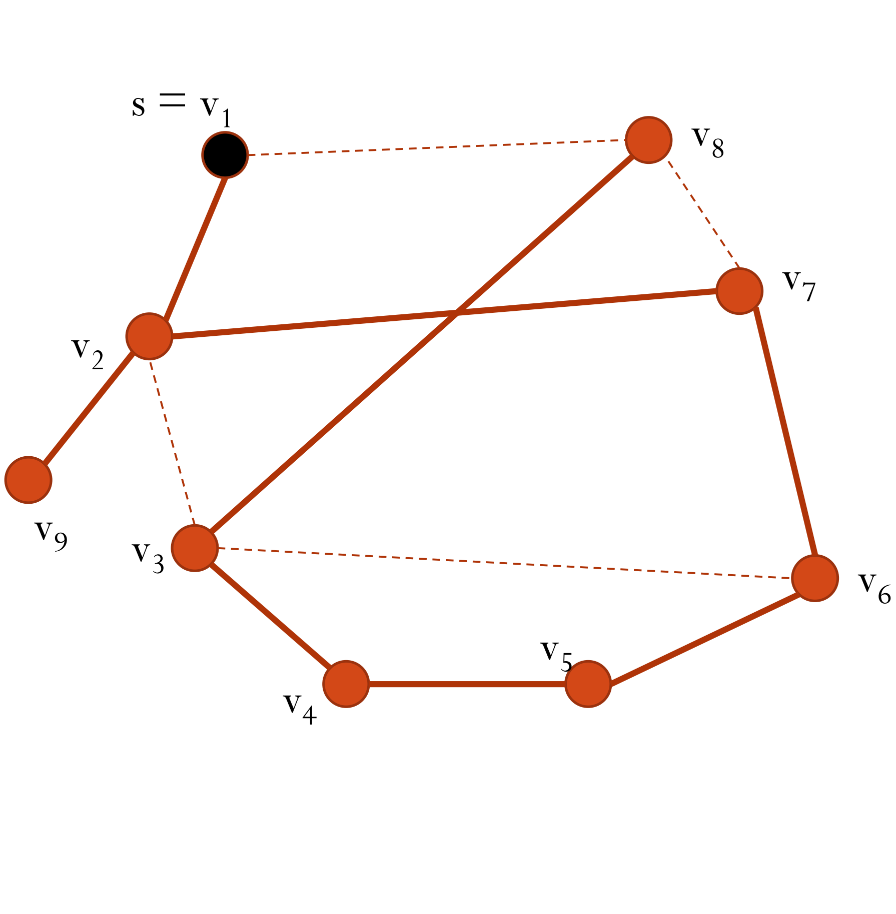
Graph Algorithms: DFS

- Depth First Search (DFS):
- The DFS algorithm defines the following "DFS tree" rooted at s:
  - Vertex u is the parent of vertex v if u caused the immediate discovery of v.



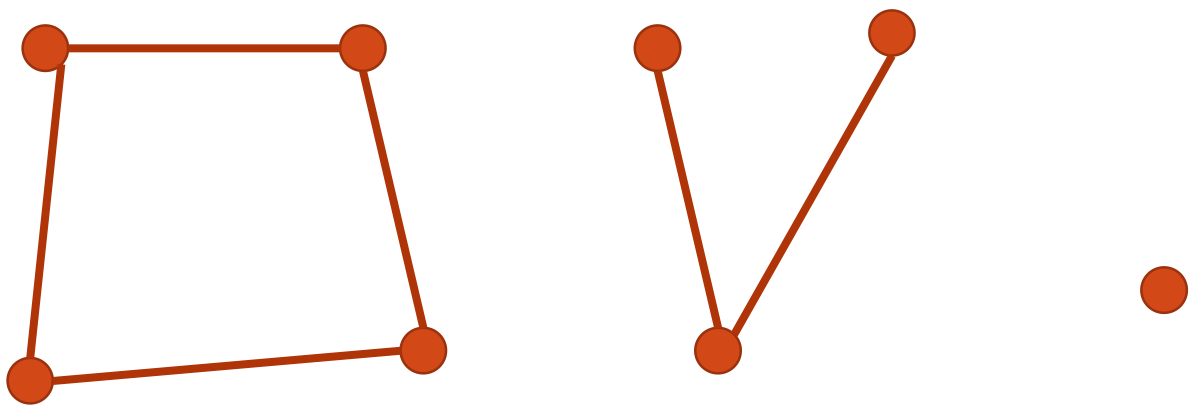
Graph Algorithms: DFS/BFS

- DFS tree versus BFS tree



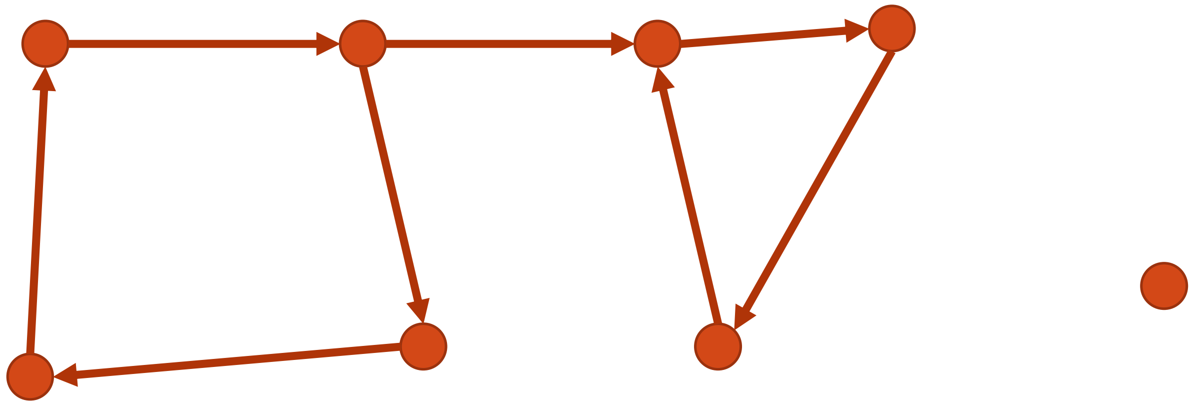
Graph algorithms: Connectivity

- A graph may not always be "connected".
- A connected component in an undirected graph is a maximal subgraph (maximal subset of vertices along with respective edges) such that there is a path between any pair of vertices in the subset.



Graph algorithms: Connectivity

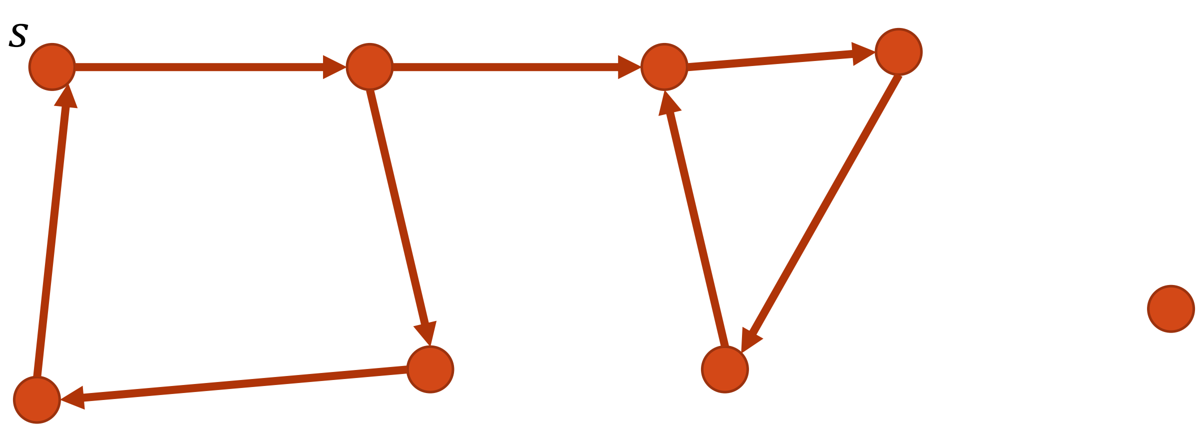
- In a directed graph, a strongly connected component is a maximal subgraph such that for each pair of vertices (u, v) in the subset, there is a path from u to v and there is a path from v to u.



Graph algorithms: Connectivity

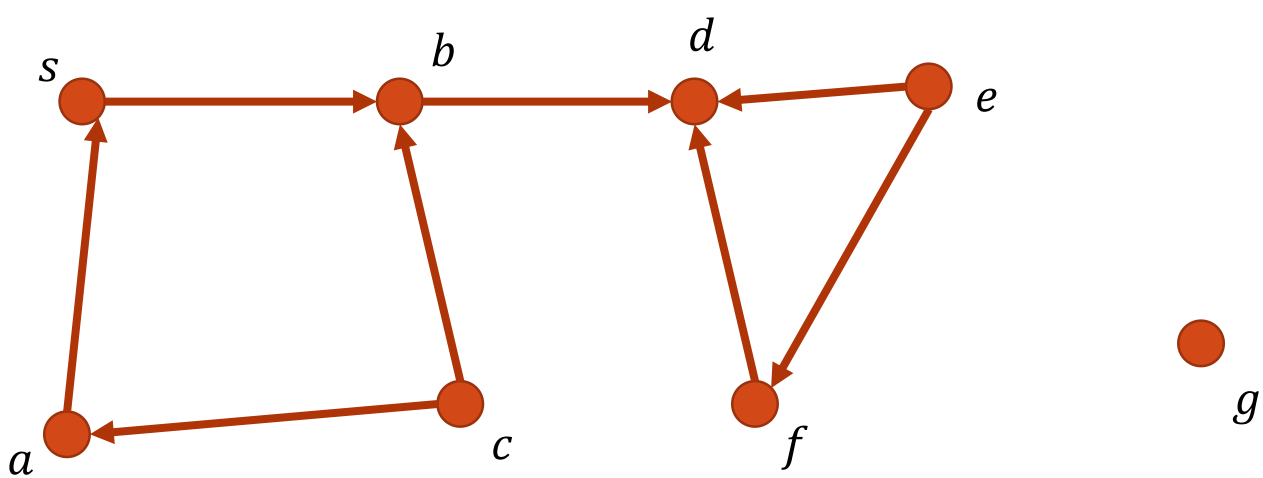
- Question: Given a directed graph, can a vertex be in two strongly connected components?

- Problem: Given a directed graph and a vertex s. Give an algorithm to find the vertices in the strongly connected component containing s. What is the running time?



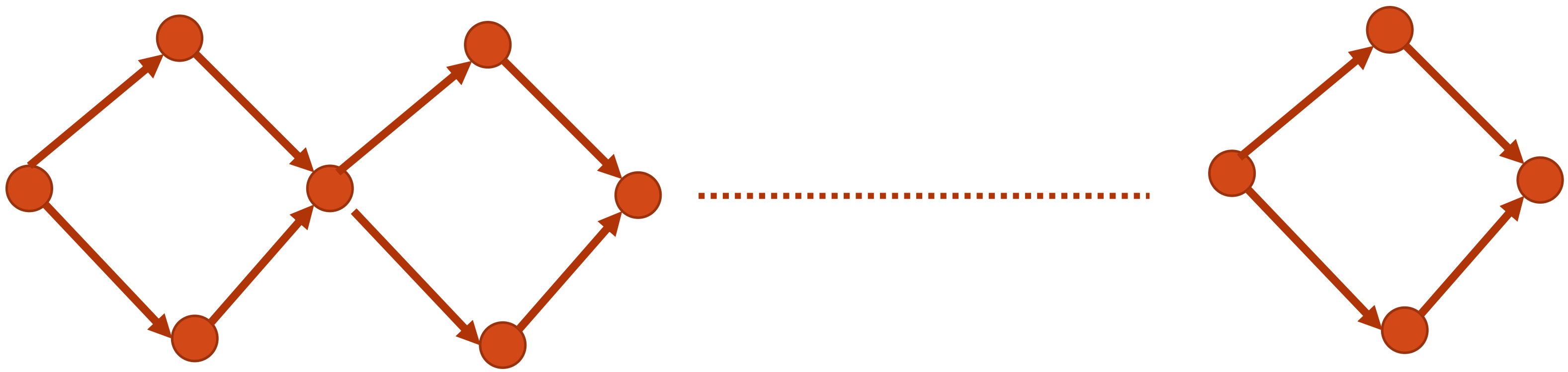
Graph algorithms: Cycles

- A "directed acyclic graph" (DAG) is a directed graph such that there are no cycles in the graph.
- Topological ordering: An ordering of the vertices of a directed graph such that there is no directed edge from a vertex that lies later in the order to another vertex that lies earlier in the order.



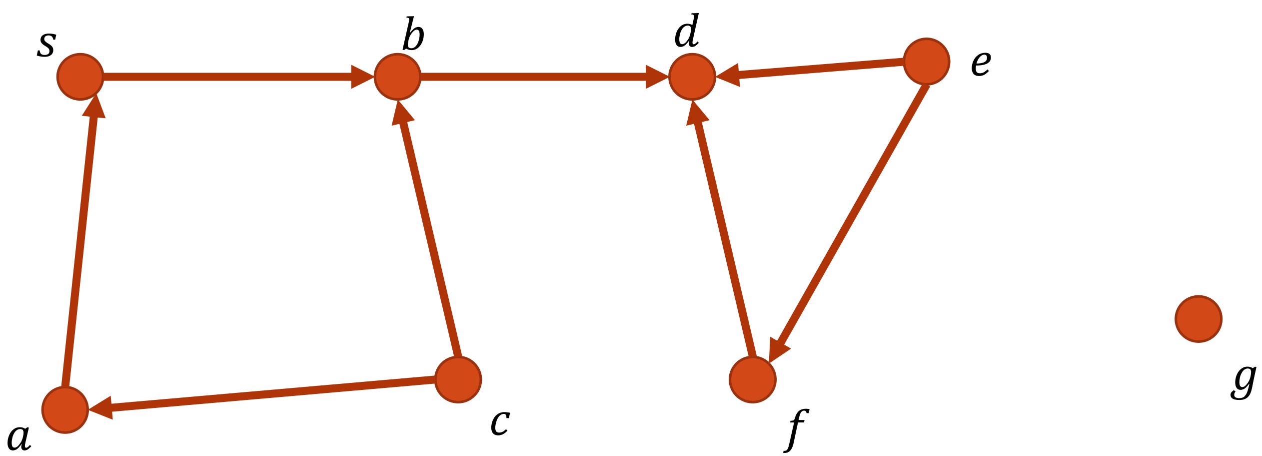
Graph algorithms: Cycles

- Question: How many topological ordering of the following graph is possible



Graph algorithms: Cycles

- <u>Question</u>: Given a directed graph that contains a cycle. Is topological ordering possible?
- <u>Question</u>: Given a DAG. Is topological ordering possible? If so give an algorithm that outputs one such order. What is the running time?



End

Problems to think about:

1. A Graph is called *semi-connected* if for any pair of vertices (u, v) in the graph either there is a path from u to v OR there is a path from v to u.
   Problem: Given a DAG check if it is *semi-connected*. What is the running time of your algorithm?