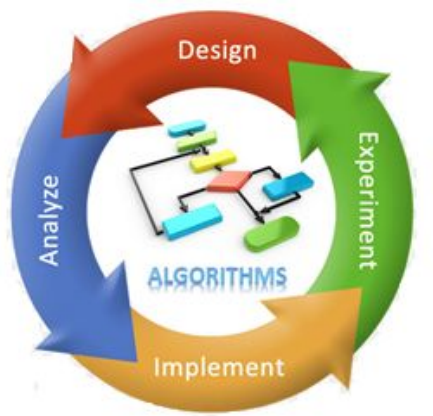
Analysis and Design of Algorithms
(ADA)
GTU # 3150703

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Unit-6:
Exploring **Graphs**



Dr. Gopi Sanghani
Computer Engineering Department
Darshan Institute of Engineering & Technology,
Rajkot
gopi.sanghani@darshan.ac.in
9825621471

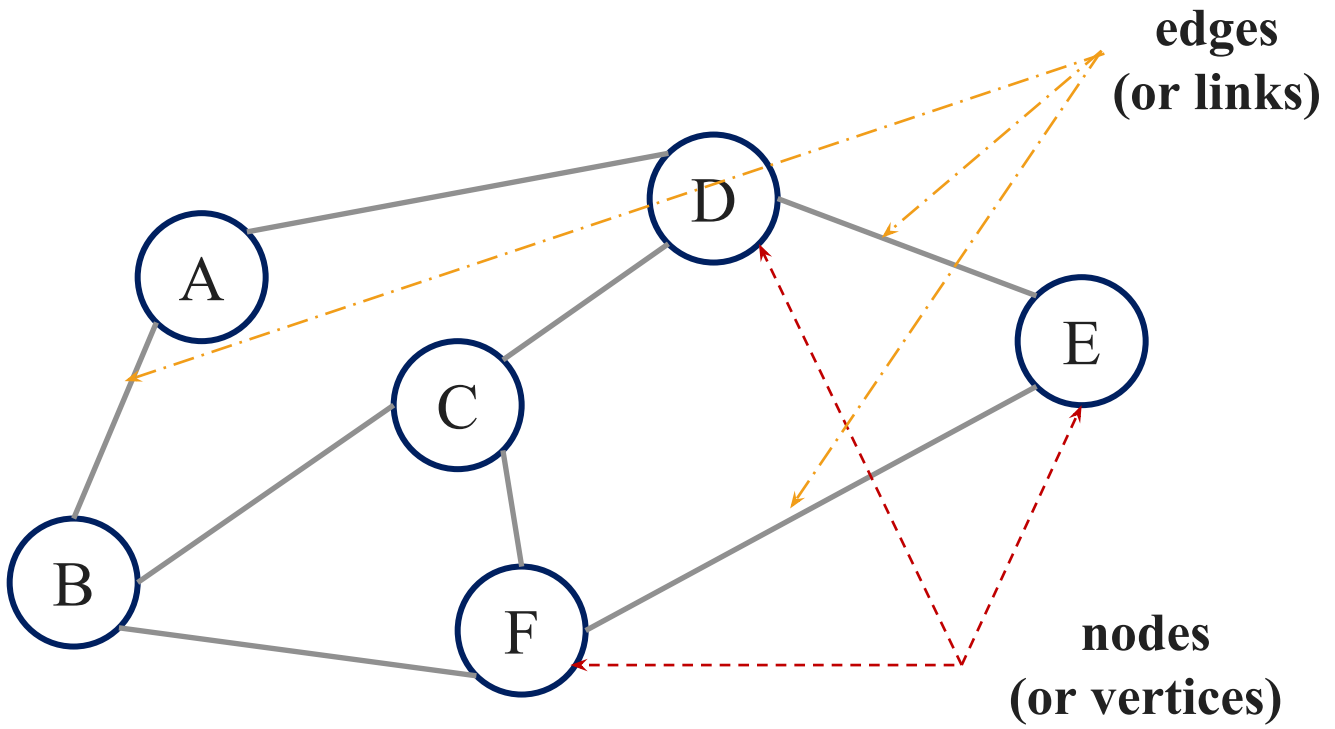
Outline

- An introduction using Graphs
- Undirected Graph
- Directed Graph
- Traversing Graphs
- Depth First Search (DFS)
- Breath First Search (BFS)
- Topological sort
- Connected components
- Articulation point

Introduction to Graph

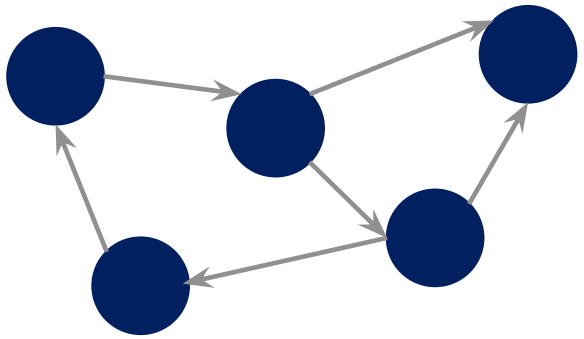
Graph - Definition

- A graph $G = \langle N, A \rangle$ consists of a non-empty set N called the set of nodes (vertices) of the graph, a set A called the set of edges that also represents a mapping from the set of edges A to a set of pairs of elements N.

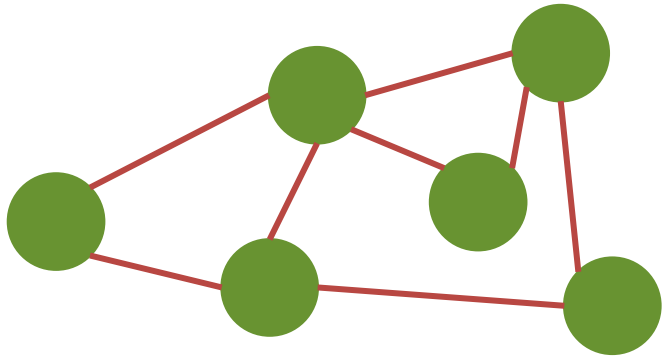


Directed & Undirected Graph

- **Directed Graph**: A graph in which **every edge is directed** from one node to another is called a directed graph or digraph.
- **Undirected Graph**: A graph in which **every edge is undirected and no direction is associated with them** is called an undirected graph.



Directed Graph



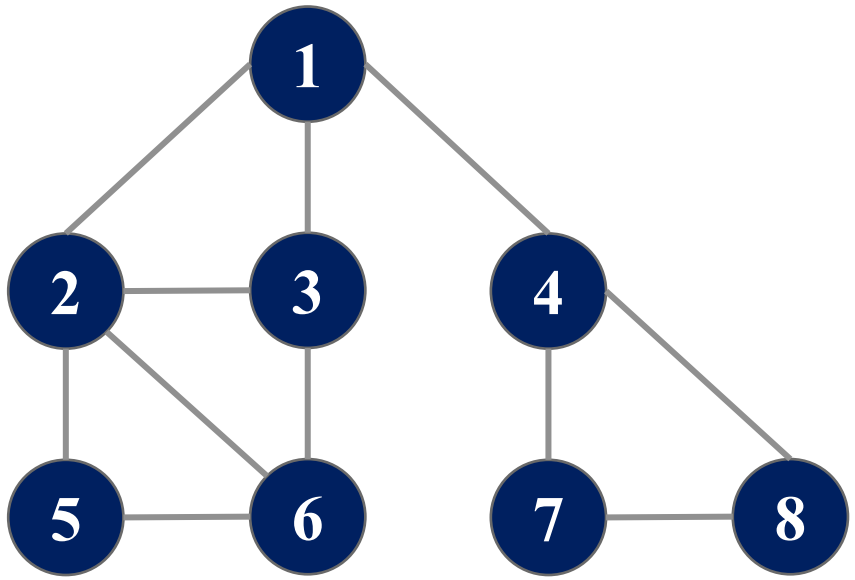
Undirected Graph

Traversing Graphs

Traversing Graph/Tree

- **Preorder**
  i. Visit the **root**.
  ii. Traverse the **left sub tree** in preorder.
  iii. Traverse the **right sub tree** in preorder.
- **In order**
  i. Traverse the **left sub tree** in in order.
  ii. Visit the **root**.
  iii. Traverse the **right sub tree** in in order.
- **Post order**
  i. Traverse the **left sub tree** in post order.
  ii. Traverse the **right sub tree** in post order.
  iii. Visit the **root.**

Depth-First Search / Traversal



Select any node $v \in N$ as starting point mark that node as visited

Select one of the unvisited adjacent of current node.
Make it new starting point and mark it as visited

If new node has no unvisited adjacent then move to parent and make it starting point

Visited 1 2 3 6 5 4 7 8

DFS – Procedure

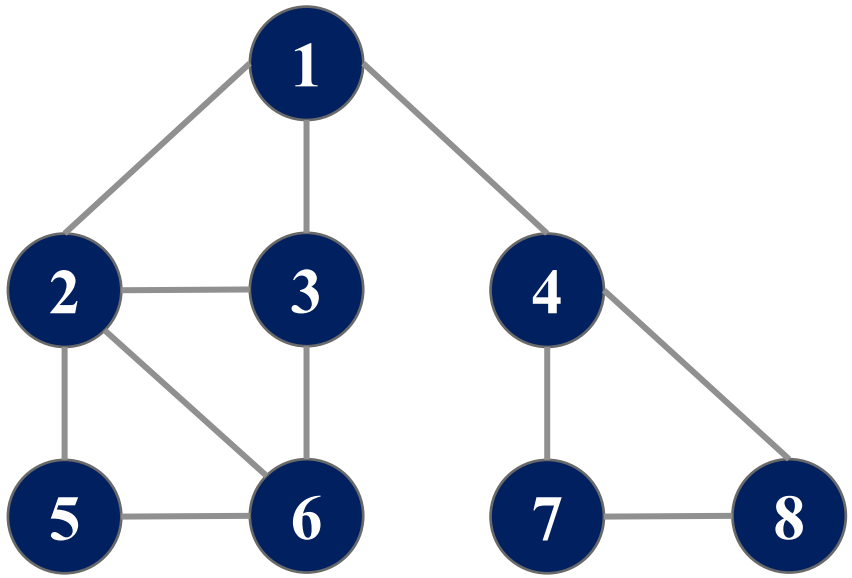
- Let $G = (N, A)$ be an undirected graph all of whose nodes we wish to visit.
- It is somehow possible **to mark a node** to show it has already been visited.
- To carry out a **depth-first traversal** of the graph, choose any node $v \in N$ as the starting point.
- Mark this node to show it has been **visited**.
- If there is a node adjacent to v that has not yet been visited, choose this node as a new starting point and call the depth-first search procedure **recursively**.
- When all the nodes adjacent to v **are marked**, the search starting at v is finished.
- If there remain any nodes of G that **have not been visited**, choose any one of them as a **new starting point**, and call the procedure again.

Depth-First Search Algorithm

```
procedure dfsearch(G)
    for each v ∈ N do
        mark[v] ← not-visited
    for each v ∈ N do
        if mark[v] ≠ visited
        then dfs(v)
procedure dfs(v)
    {Node v has not previously been visited}
    mark[v] ← visited
    for each node w adjacent to v do
        if mark[w] ≠ visited
        then dfs(w)
```

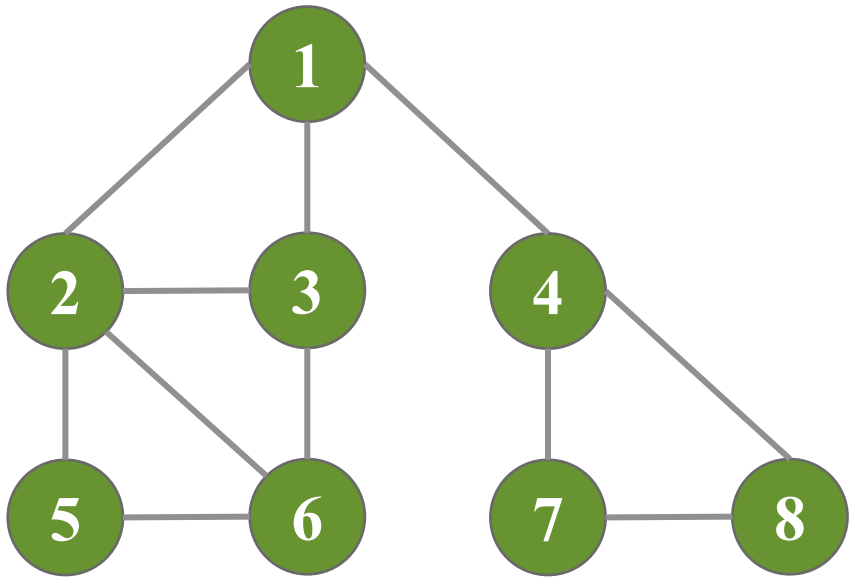
Depth-First Search - Algorithm

```
1. dfs(1)              Initial call
2.   dfs(2)            recursive call
3.     dfs(3)          recursive call
4.       dfs(6)        recursive call
5.         dfs(5)      recursive      call;
   progress is blocked
6.   dfs(4)            a neighbour of node
   1 that has not been visited
7.     dfs(7)          recursive call
8.       dfs(8)        recursive call
9. There are no more nodes to visit
```




```
procedure dfs(v)
   mark[v] ← visited
   for each node w adjacent to v do
    if mark[w] ≠ visited
        then dfs(w)
```

Breadth First Search / Traversal



Select any node v ∈ N as starting point mark that node as visited.

Enqueue visited v node into queue Q

Dequeue a node from the front of queue.
Find it's all unvisited adjacent nodes, mark as visited, enqueue into queue

1	2	3	4	5	6	7	8

Queue Q

Visited 1 2 3 4 5 6 7 8

Breadth First Search - Algorithm

```
procedure search(G)
    for each v ∈ N do
        mark[v] ← not visited
    for each v ∈ N do
        if mark[v] ≠ visited
        then bfs(v)
```

```
procedure bfs(v)
    Q ← empty-queue
    mark[v] ← visited
    enqueue v into Q
    while Q is not empty do
    u ← first(Q)
    dequeue u from Q
    for each node w adjacent to u do
            if mark[w] ≠ visited
                then mark[w] ← visited
        enqueue w into Q
```

Comparison of DFS and BFS

Depth First Search (DFS)	Breath First Search (BFS)
DFS traverses according to tree depth. DFS reaches up to the bottom of a subtree, then backtracks.	BFS traverses according to tree level. BFS finds the shortest path to the destination.
It uses a stack to keep track of the next location to visit.	It uses a queue to keep track of the next location to visit.
DFS requires less memory since only nodes on the current path are stored.	BFS guarantees that the space of possible moves is systematically examined; this search requires considerably more memory resources.
Does not guarantee to find solution. Backtracking is required if wrong path is selected.	If there is a solution, BFS is guaranteed to find it.

Comparison of DFS and BFS

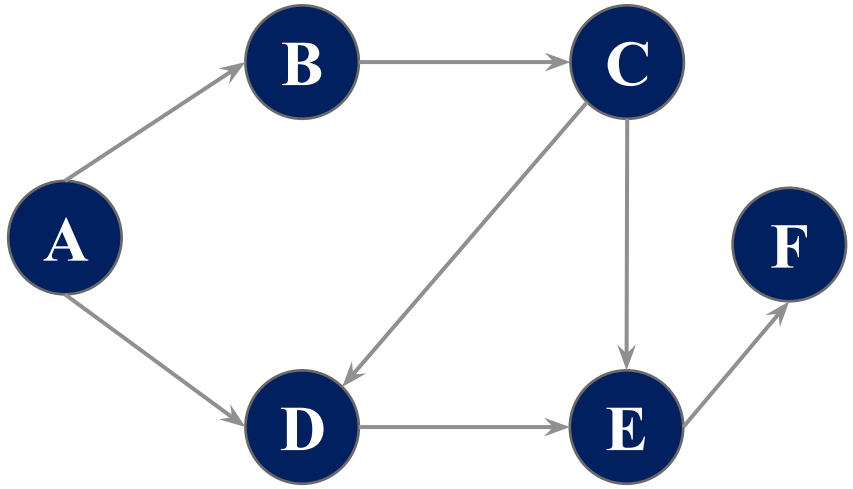
Depth First Search	Breath First Search
If the selected path does not reach to the solution node, DFS gets stuck or trapped into an infinite loops.	BFS will not get trapped exploring an infinite loops.

The Time complexity of both BFS and DFS will be O(V + E), where V is the number of vertices, and E is the number of Edges.

Topological Sorting

- A **topological sort** or **topological ordering** of a directed acyclic graph is a linear ordering of its vertices such that for every directed edge (u, v) from vertex u to vertex v, the vertex u comes before the vertex v in the ordering.
- Topological Sorting for a graph is not possible if the graph is not a DAG.
- In DFS, we print a vertex and then recursively call DFS for its adjacent vertices. In topological sorting, we need to print a vertex before its adjacent vertices.
- Few important applications of topological sort are-
  - Scheduling jobs from the given dependencies among jobs
  - Instruction Scheduling
  - Determining the order of compilation tasks to perform in makefiles

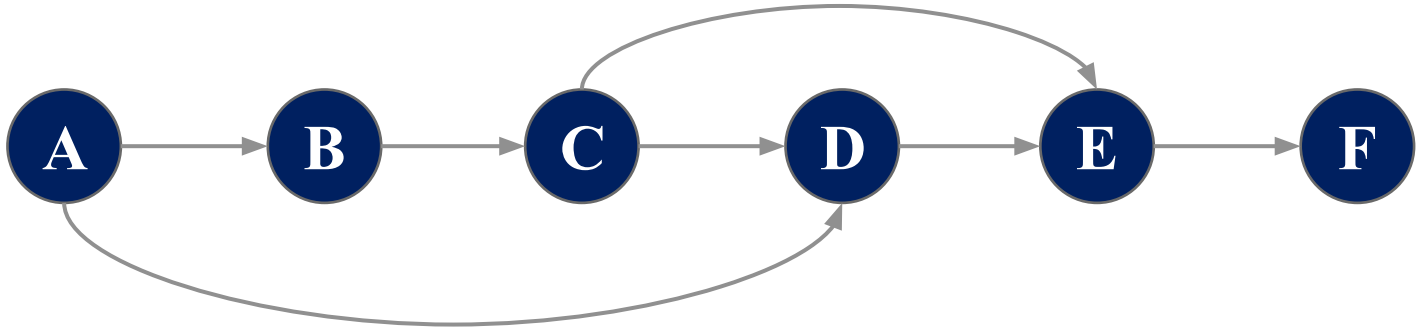
Topological Sorting – Example 1



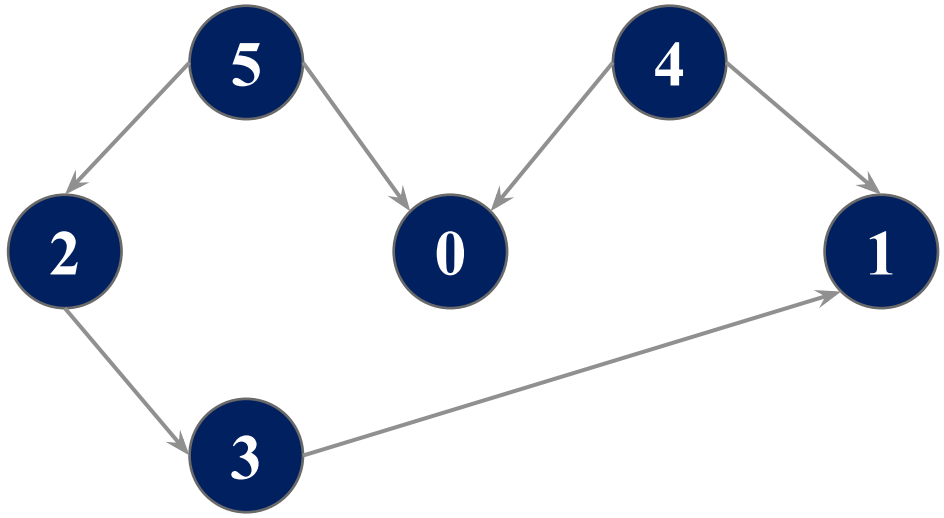
Identify nodes having in degree ‘0’

Select a node and delete it with its edges then add node to output

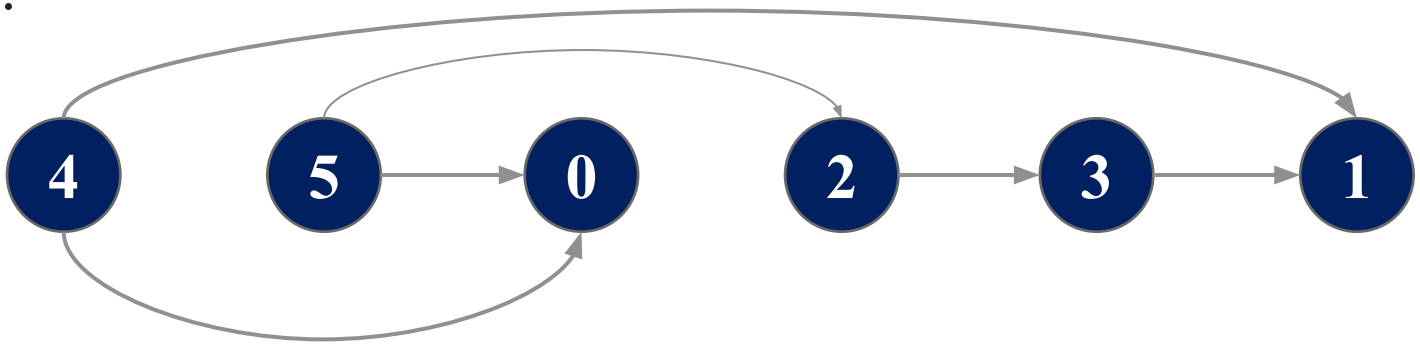
Output:



Topological Sorting – Example 2

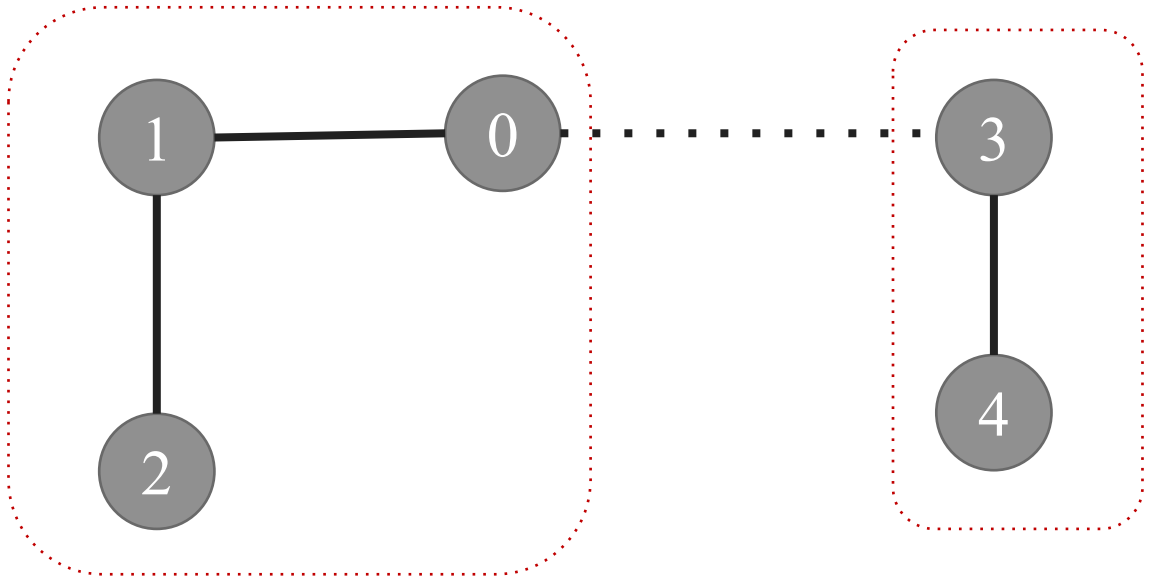


Output:



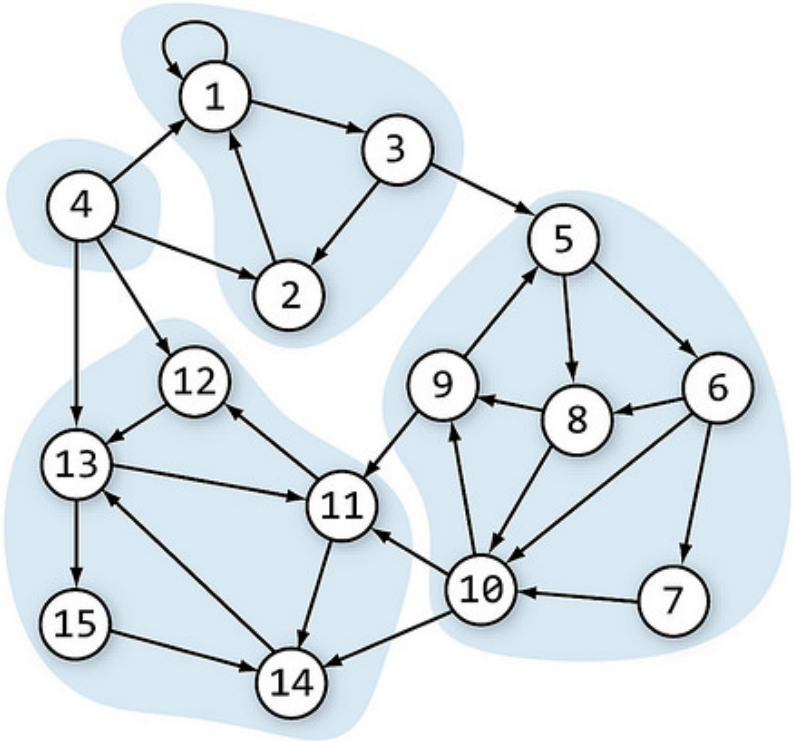
Connected Components

- A **connected component** (or just component) of an undirected graph is **a subgraph in which any two vertices are connected** to each other by paths.



- There are two connected components in the above undirected graph. **0 1 2** and **3 4**

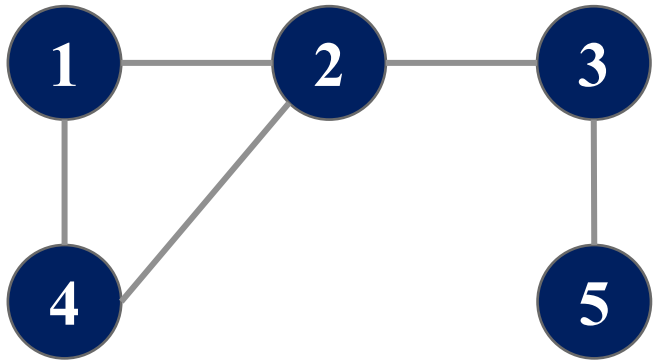
Connected Components



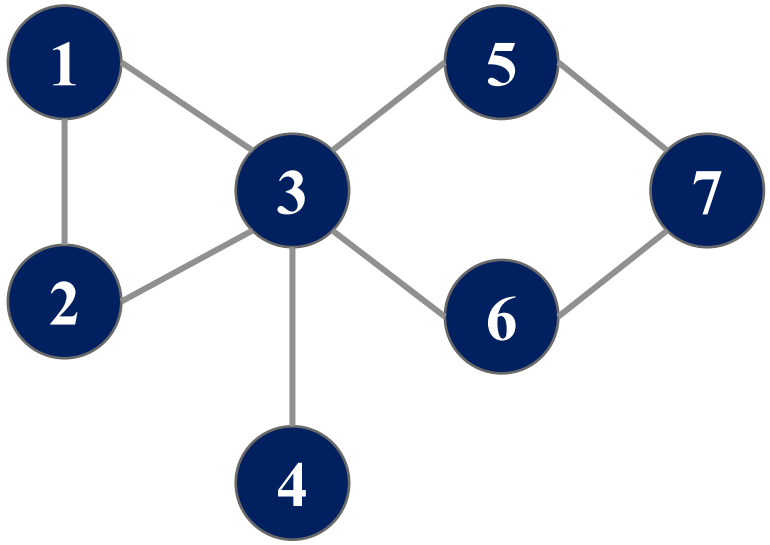
- A directed graph is strongly connected if there is a directed path from any vertex to every other vertex.
- This is same as connectivity in an undirected graph, the only difference is strong connectivity applies to directed graphs and there should be directed paths instead of just paths.
- Similar to connected components, a directed graph can be broken down into Strongly Connected Components.

Articulation Point

- An **articulation point** in a connected graph is a vertex, that if is deleted (and edges through it) then disconnects the graph.
- It represent vulnerabilities in a connected network, single points whose failure would split network into two or more disconnected components.
- For a disconnected undirected graph, an articulation point is a vertex removing which will increases number of connected components.



Articulation Points:
2,3



Articulation Points: 3

Thank You!