Parul® University

NAAC A++

Exploring Graphs:
Chapter-4

Mrs. Bhumi Shah
Assistant Professor
Department of Computer Science and Engineering

Parul® University

NAAC A++

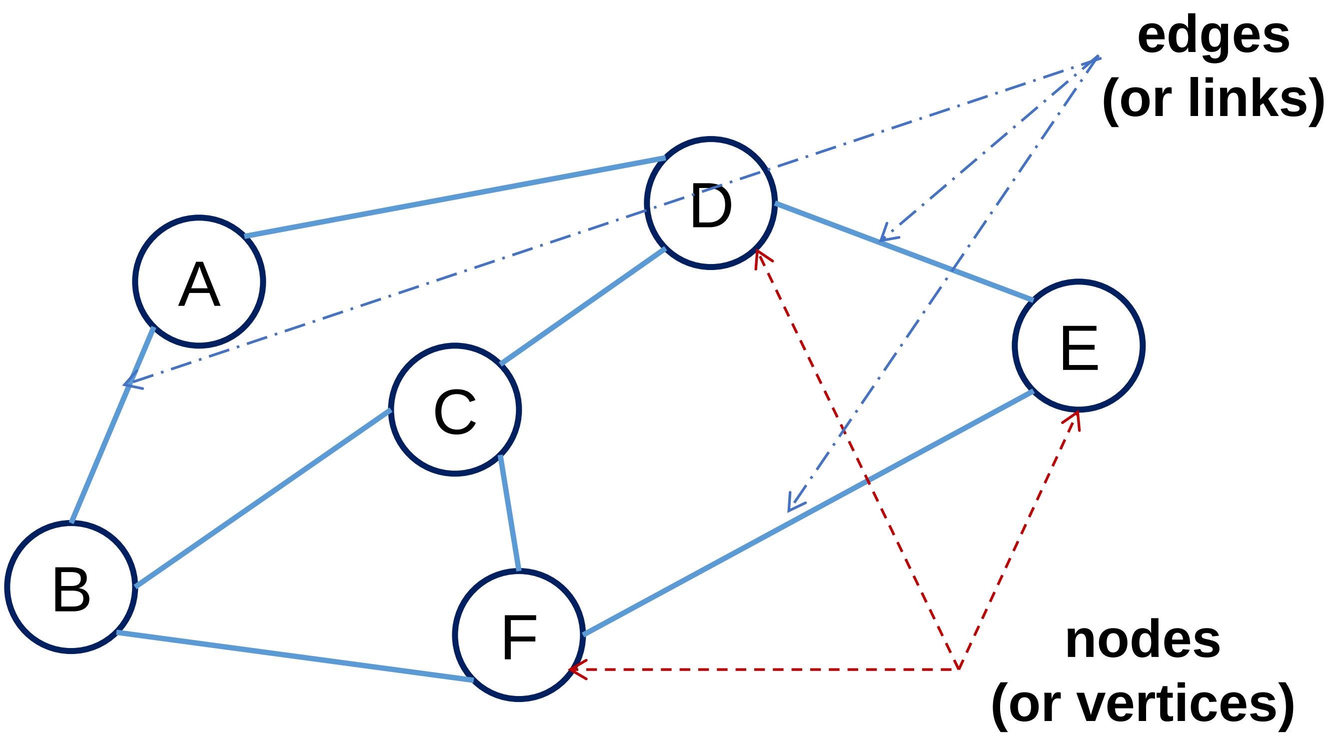
Content

1. An introduction using graphs :
   Undirected Graph
   Directed Graph
2. Traversing Graphs
   Depth First Search,
   Breath First Search,
   Topological sort

INDEX

Graph - Definition

A graph $\mathbf{G} = \langle N, A \rangle$ 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.



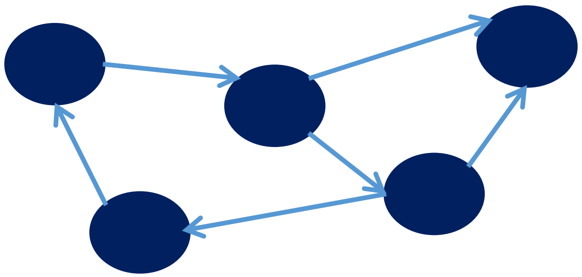
Directed Graph

Directed Graph: A graph in which **every edge is directed** from one node to another is called a directed graph or digraph.

Characteristics:

Asymmetrical: An edge from vertex A to vertex B does not imply a connection back from B to A.

Use Cases: Webpage links (where one page links to another), task scheduling.



Directed Graph

Here, the arrows indicate the direction of the relationships between vertices.

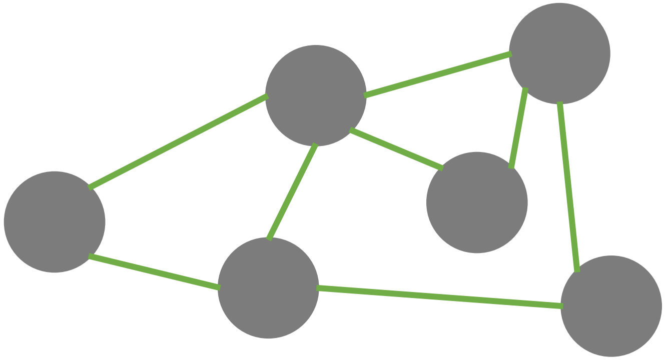
Undirected Graph

Undirected Graph: A graph in which **every edge is undirected and no direction is associated with them** is called an undirected graph.

Characteristics:

Symmetrical: If there is an edge between vertex A and vertex B, one can move from A to B and from B to A.

Use Cases: Social networks, where relationships (like friendships) are mutual.



Undirected Graph

In this undirected graph, connections between nodes are bidirectional.

Graphs in Games

- Graphs can represent various aspects of games, particularly in strategy and route planning. Here’s how they are utilized:
- Strategy Games: Players can be represented as vertices, with potential moves as edges. The strategies can be analyzed based on connectivity.
- Pathfinding Algorithms: In games, directing characters or units from point A to point B can be modeled using directed graphs, utilizing algorithms like Dijkstra's or A*.
- Social Dynamics: Collaboration and competition among players can be modeled as undirected graphs, showing alliances or rivalries.

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/

Parul® University NAAC A++

Exploring Graphs: Chapter-4

Mrs. Bhumi Shah
Assistant Professor
Department of Computer Science and Engineering

Parul® University

NAAC A++

Content

1. An introduction using graphs :
   Undirected Graph
   Directed Graph
2. Traversing Graphs
   Depth First Search,
   Breath First Search,
   Topological sort

INDEX

Traversing Graph

Preorder

i. Visit the **root**.
ii. Traverse the **left sub tree** in preorder.
iii. Traverse the **right sub tree** in preorder.

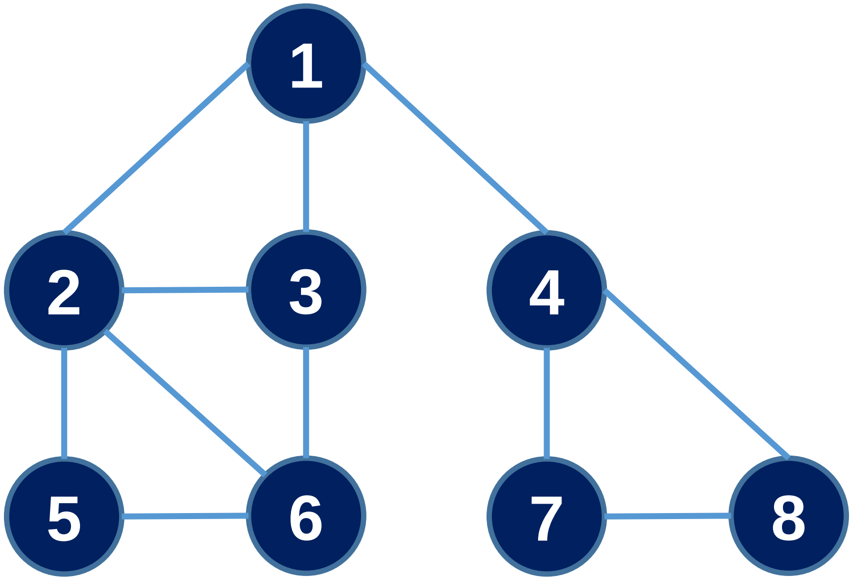
In order

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

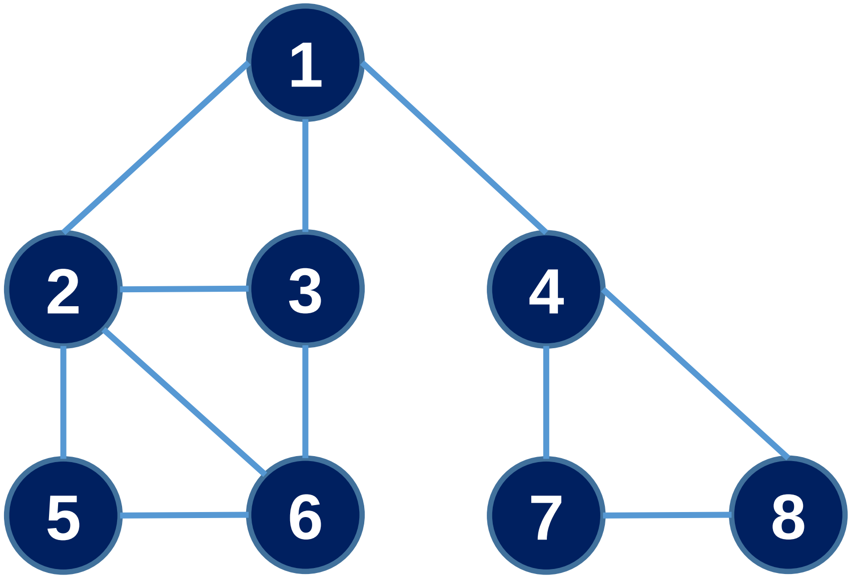
- Let $\boldsymbol{G = (N, A)}$ be an undirected graph all of whose nodes we wish to visit.
- It is somehow possible **to mark a node** to show it has already been visited.
- To carry out a **depth-first traversal** of the graph, choose any node $v \in N$ as the starting point.
- Mark this node to show it has been **visited**.
- If there is a node adjacent to $\boldsymbol{v}$ 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 $\boldsymbol{v}$ **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)
```

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.

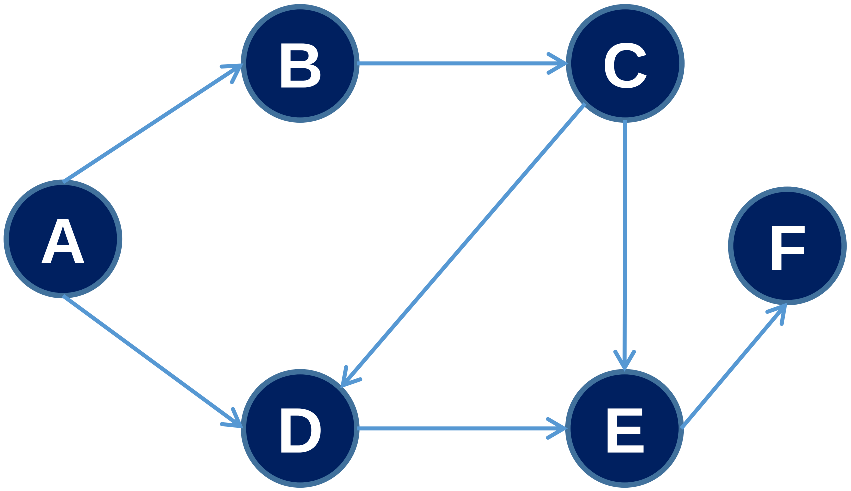
Comparison of DFS and BFS

Depth First Search (DFS)	Breath First Search (BFS)
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

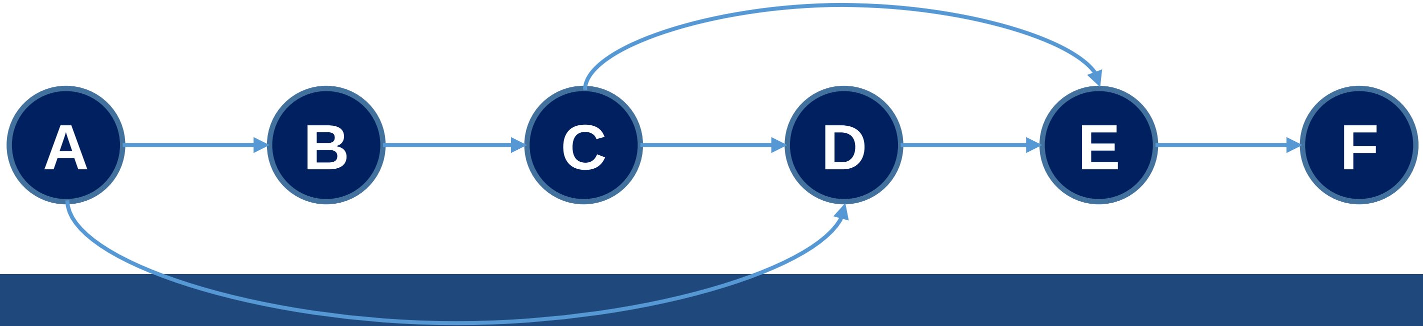
Topological Sort- Example 1



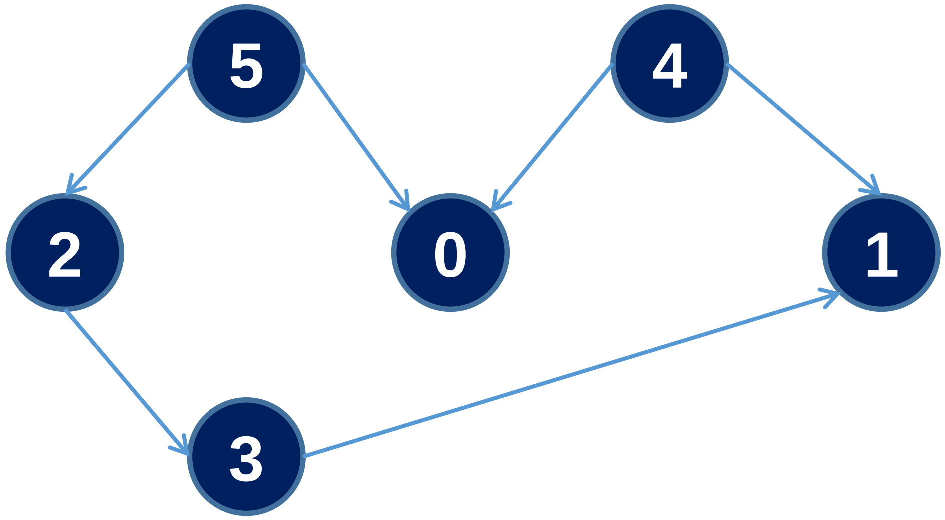
Identify nodes having in degree '0'

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

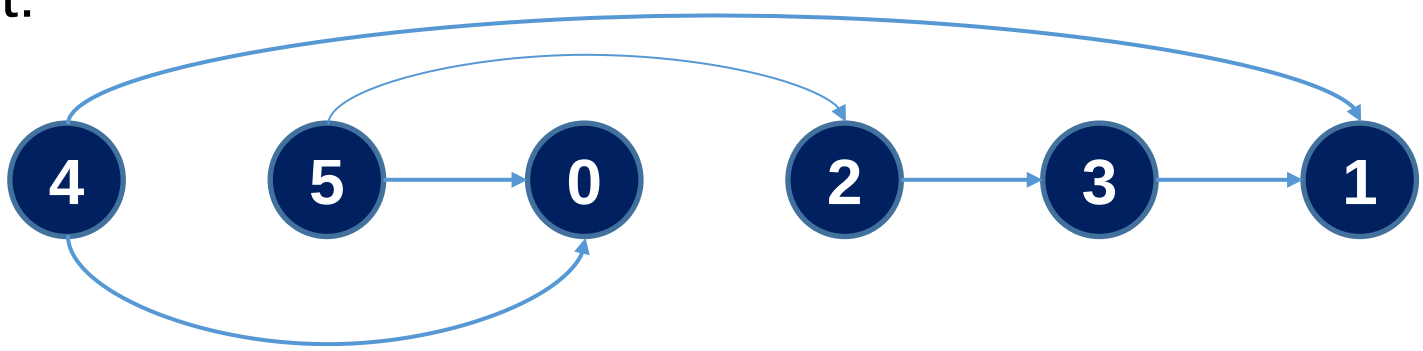
Output:



Topological Sort- Example 2



Output:



Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/