

Data Structures and Algorithms

Prepared by Prof. Shereen Taie

IT251
Information Technology Department

Lecture 9



Data Structures and Algorithms

Lecture 9

IT251

Information Technology Department

Prepared by
Prof. Shereen Taie

www.midocean.ae

Learning Objectives:

- Applications of Breadth First Search
- Advantages of Breadth First Search
- Disadvantages of Breadth First Search
- DEPTH-FIRST SEARCH
- Advantages of Depth First Search
- Disadvantages of Depth First Search
- Difference between BFS and DFS

Applications of Breadth First Search

Chapter

04

1. Shortest Path for unweighted graph:

In an unweighted graph, the shortest path is the path with the least number of edges. With Breadth First, we always reach a vertex from a given source using the minimum number of edges.

2. Peer-to-Peer Networks:

In Peer-to-Peer Networks like [BitTorrent](#), Breadth First Search is used to find all neighbor nodes.

[BitTorrent](#) is a hyper distribution communications protocol for peer-to-peer file sharing (“P2P”) which is used to distribute data and electronic files over the Internet.

3. Crawlers in Search Engines:

Crawlers build an index using Breadth First. The idea is to start from the source page and follow all links from the source and keep doing the same. Depth First Traversal can also be used for crawlers, but the advantage of Breadth First Traversal is, the depth or levels of the built tree can be limited.

4. Social Networking Websites:

In social networks, we can find people within a given distance 'k' from a person using Breadth First Search till 'k' levels.

5. GPS Navigation systems:

Breadth First Search is used to find all neighboring locations.

6. Broadcasting in Network:

In networks, a broadcasted packet follows Breadth First Search to reach all nodes.

7. To test if a graph is Bipartite:

We can either use Breadth First or Depth First Traversal.

8. Path Finding:

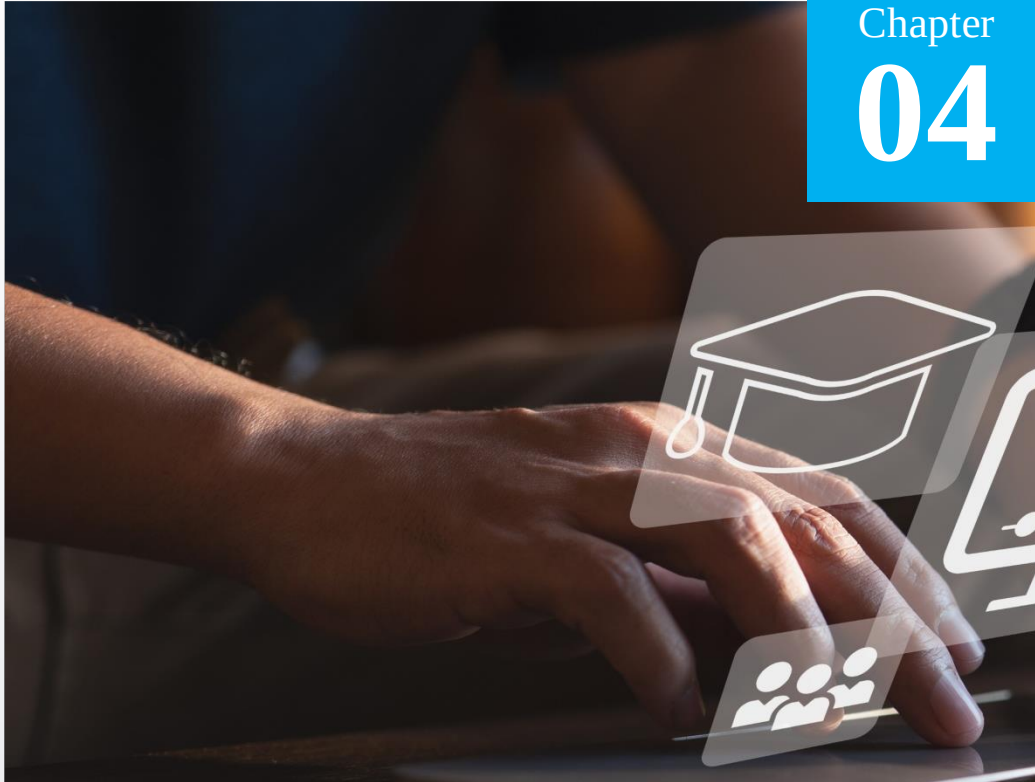
We can either use Breadth First or Depth First Traversal to find if there is a path between two vertices.

9. Cycle detection in undirected graph:

In undirected graphs, either Breadth First Search or Depth First Search can be used to detect a cycle. We can use BFS to detect cycle in a directed graph also.

Chapter
04

**Advantages and Disadvantages of
Breadth First Search**



Advantages of Breadth First Search:

- BFS will never get trapped exploring the useful path forever.
- If there is a solution, BFS will definitely find it.
- If there is more than one solution then BFS can find the minimal one that requires less number of steps.

Advantages of Breadth First Search:

- Low storage requirement – linear with depth.
- Easily programmable.

Disadvantages of Breadth First Search:

- The main drawback of BFS is its memory requirement.
- Since each level of the graph must be saved in order to generate the next level and the amount of memory is proportional to the number of nodes stored the space complexity of BFS is $O(bd)$, where b is the branching factor (the number of children at each node, the outdegree) and d is the depth.
- As a result, BFS is severely space-bound in practice so will exhaust the memory available on typical computers in a matter of minutes.

DEPTH-FIRST SEARCH

One way to explore a graph!

DFS (Iterative Version)

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.

Postcondition: a vertex is reachable from s if and only if it is marked as “explored.”

mark all vertices as unexplored

$S :=$ a stack data structure, initialized with s

while S is not empty **do**

 remove (“pop”) the vertex v from the front of S

if v is unexplored **then**

 mark v as explored

for each edge (v, w) in v ’s adjacency list **do**

 add (“push”) w to the front of S

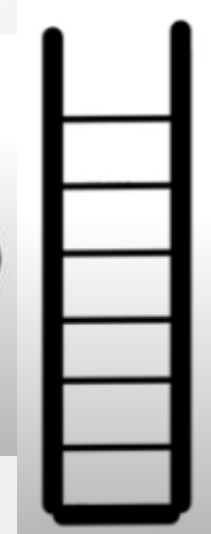
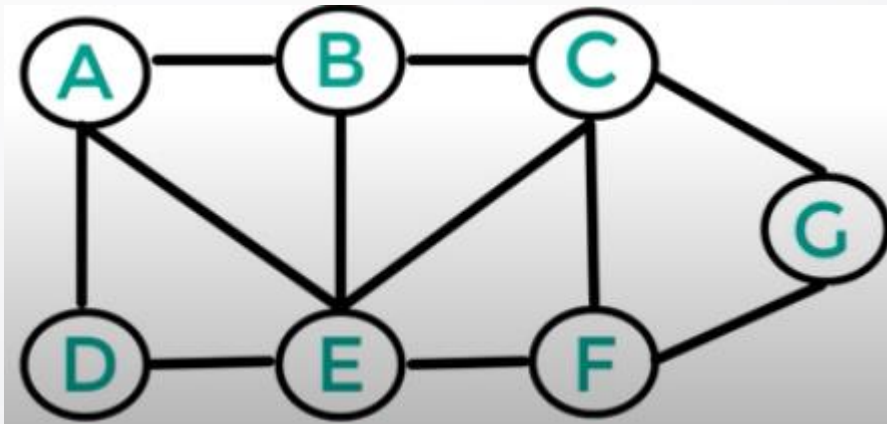
Literally just BREADTH vs DEPTH:

While

BFS first **explores the nodes closest to the “source”** and
then **moves outwards in layers,**

DFS **goes as far down a path** as it can **before it comes
back to explore other options.**

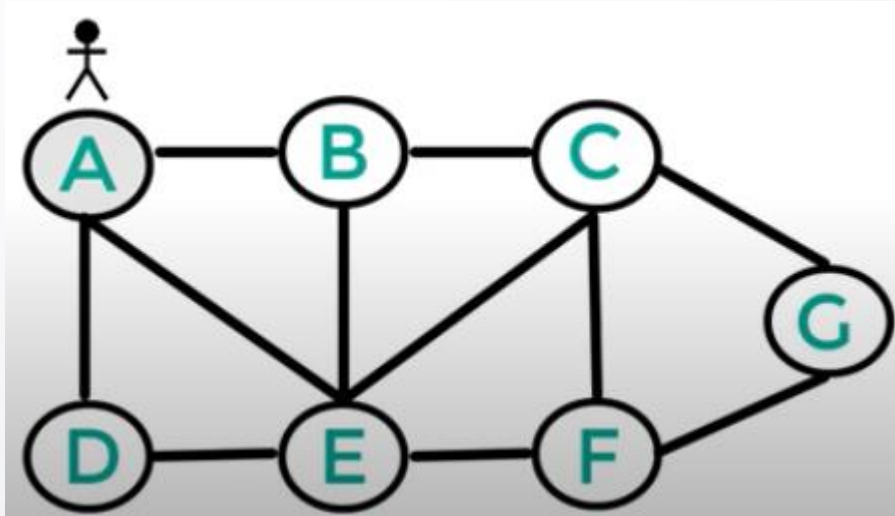
DFS stack example



STACK

1. Define a **stack**, size = # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

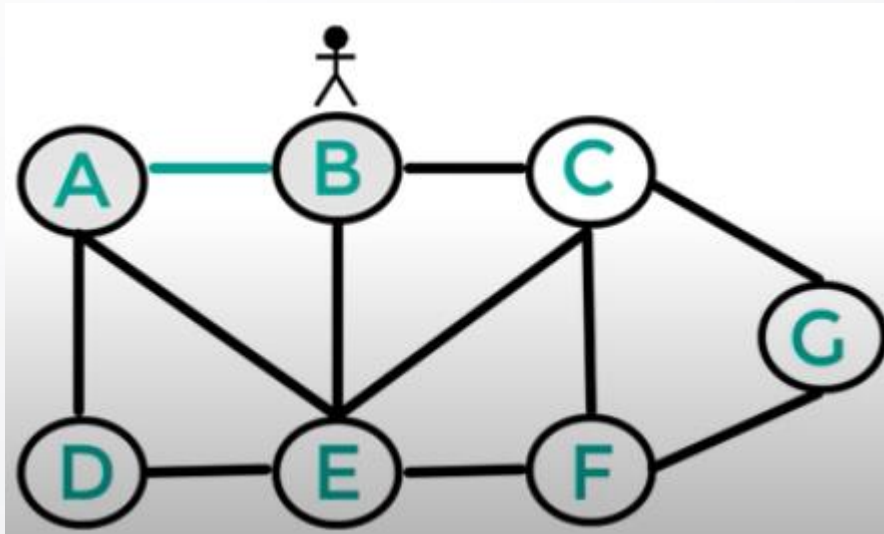


STACK

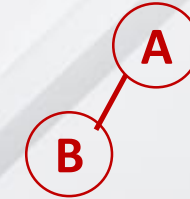
A

1. Define a **stack**, size= # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

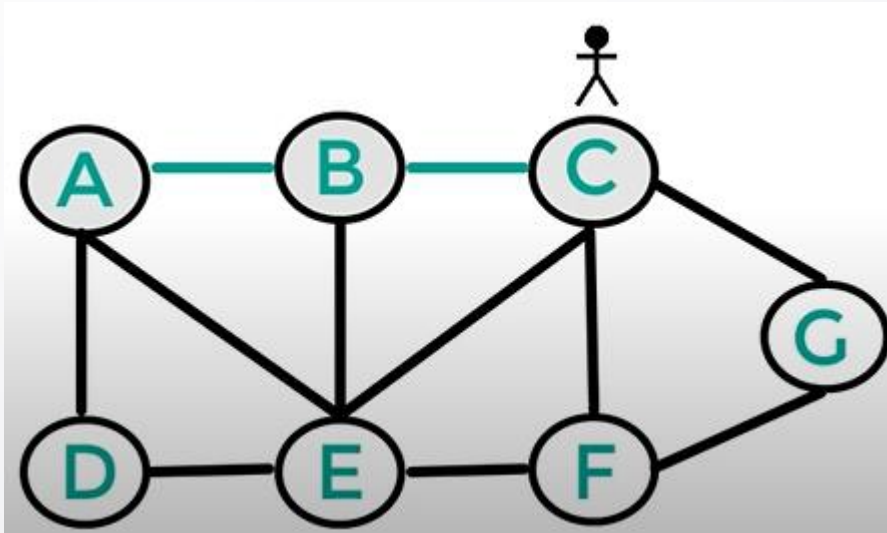


STACK

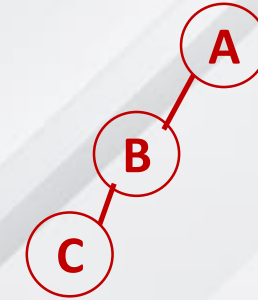


1. Define a **stack**, size= # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

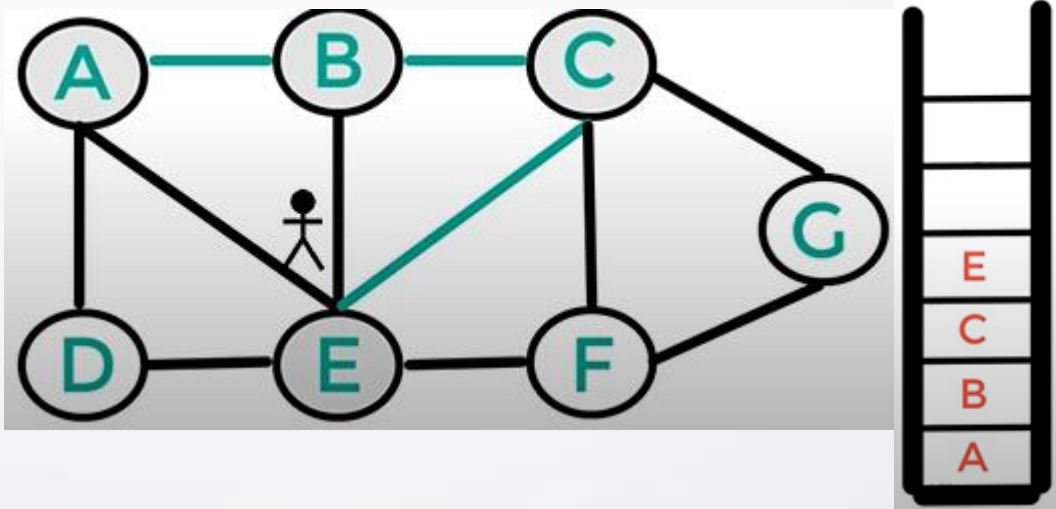


STACK

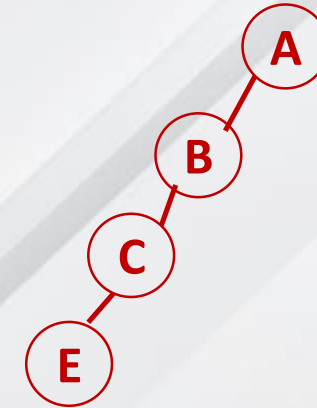


1. Define a **stack** ,size= # of Vertices in graph.
2. Select **any vertex** as starting point , visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3 ,**until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

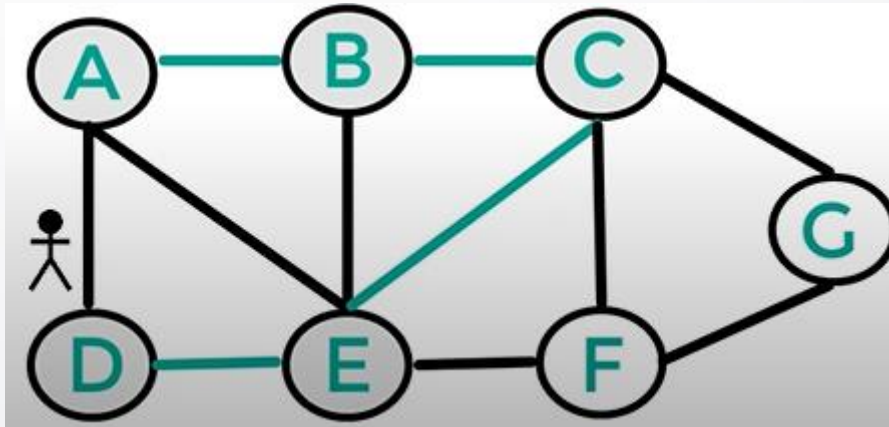


STACK

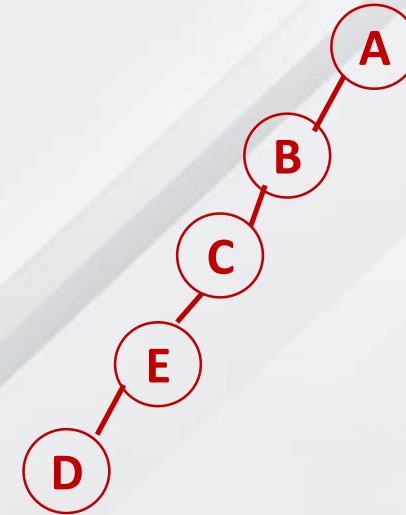


1. Define a **stack**, size = # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

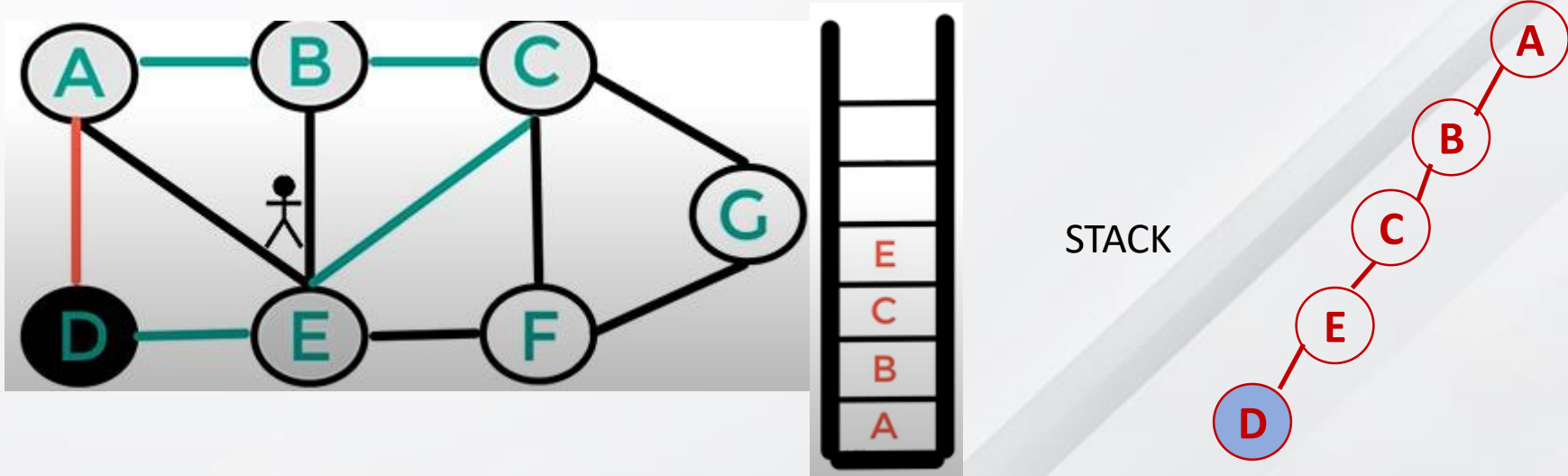


STACK



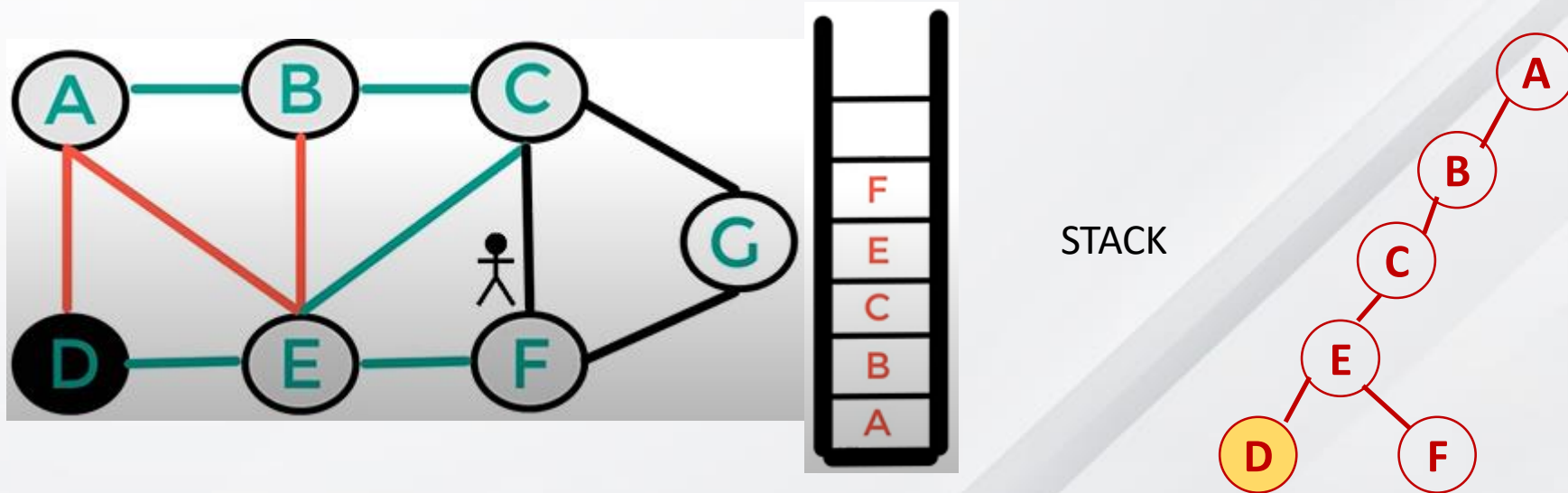
1. Define a **stack**, size= # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example



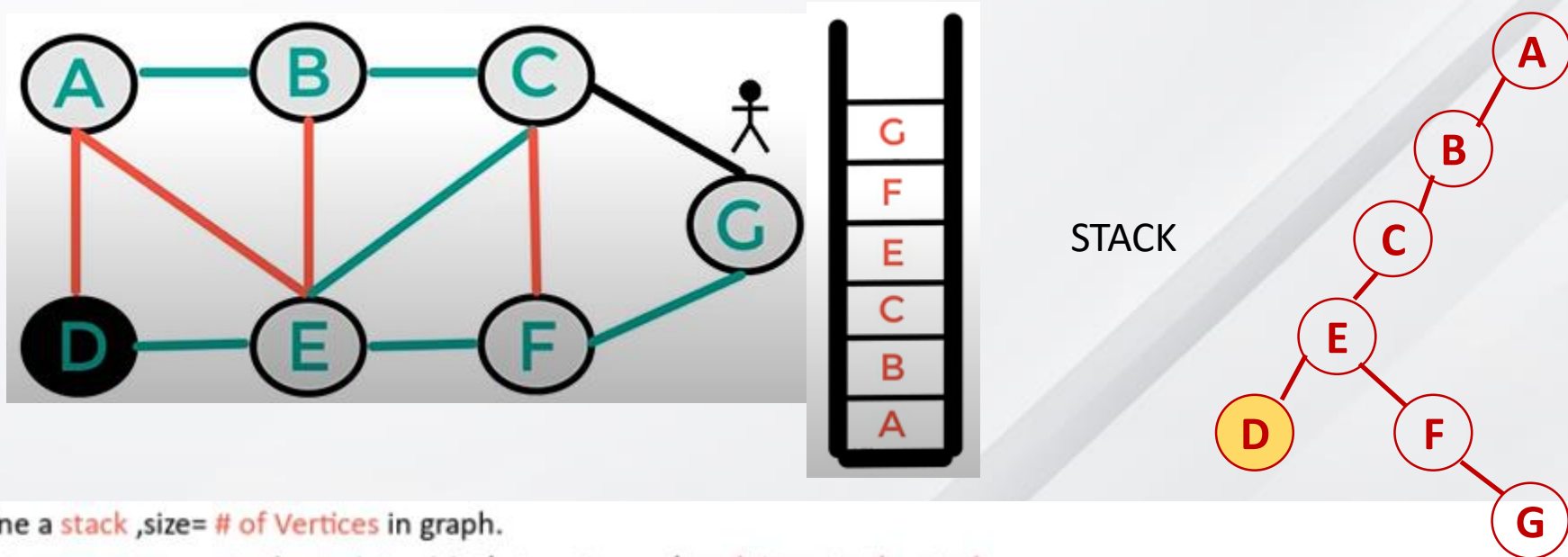
1. Define a **stack**, size = # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example



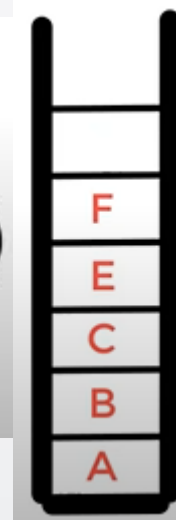
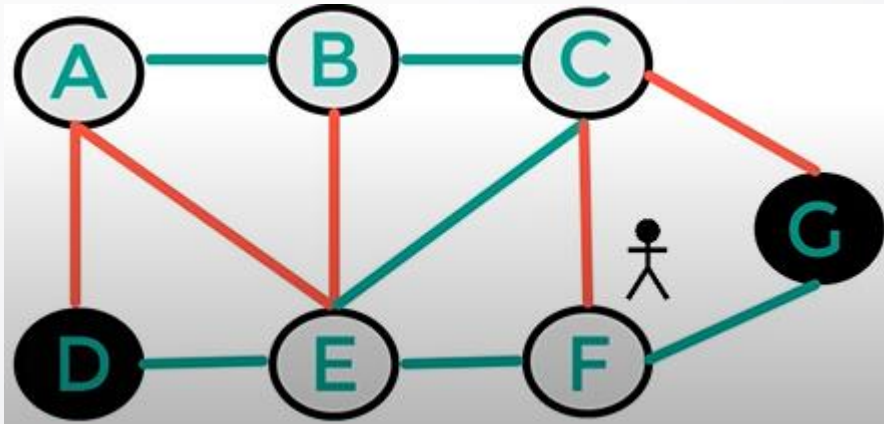
1. Define a **stack**, size = # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

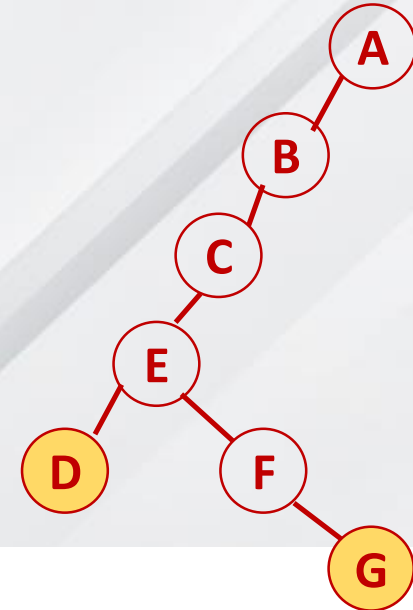


1. Define a **stack**, size= # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

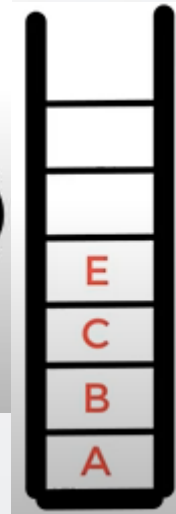
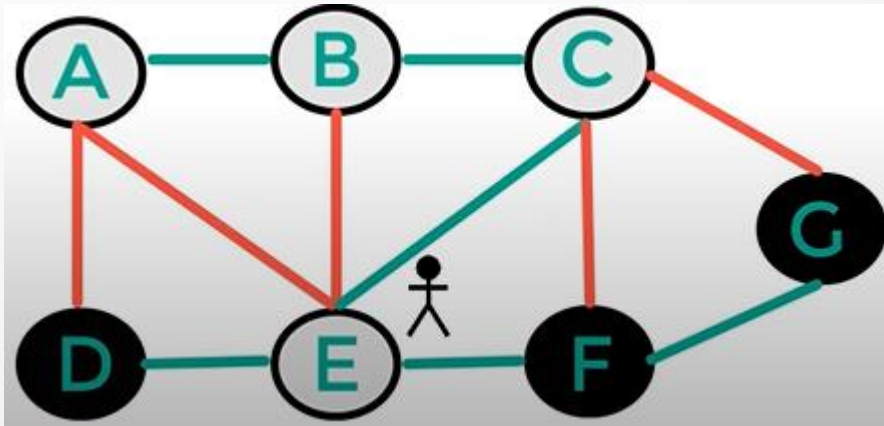


STACK

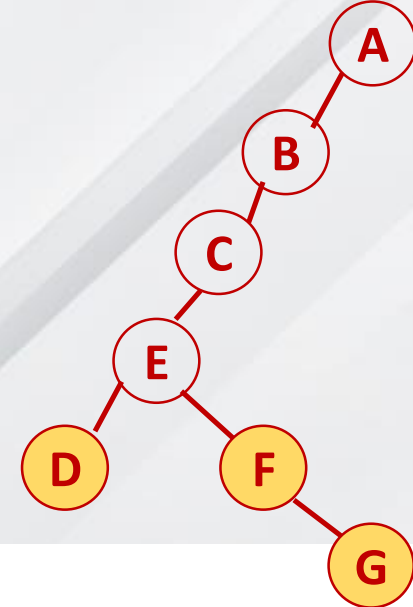


1. Define a **stack**, size = # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

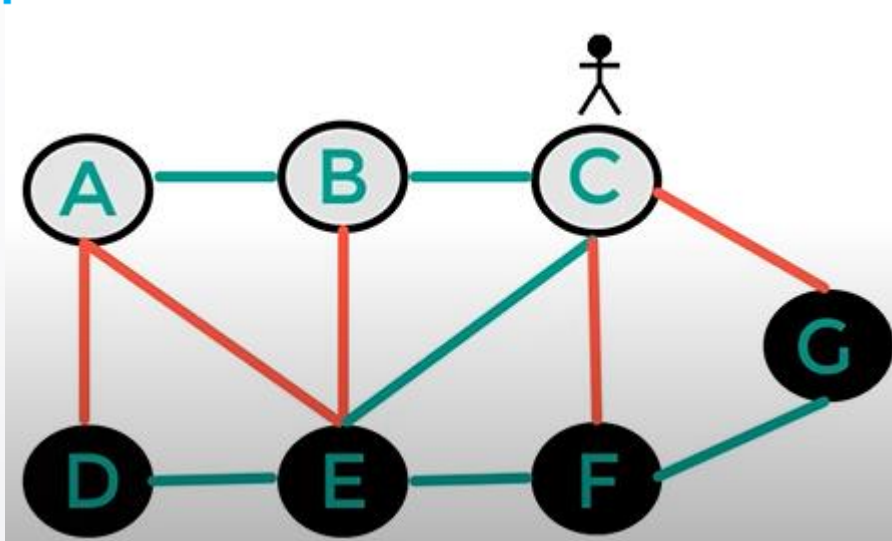


STACK

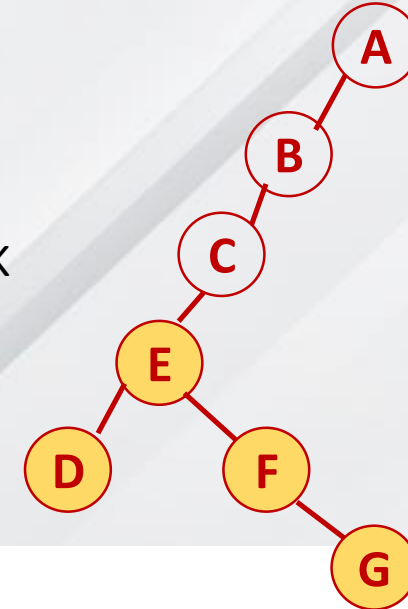


1. Define a **stack**, size= # of Vertices in graph.
2. Select **any vertex** as starting point, visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3, **until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

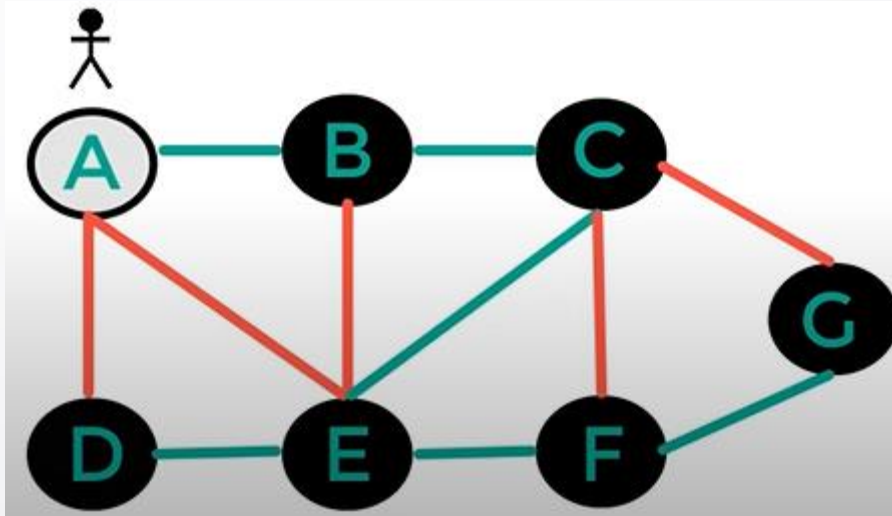


STACK

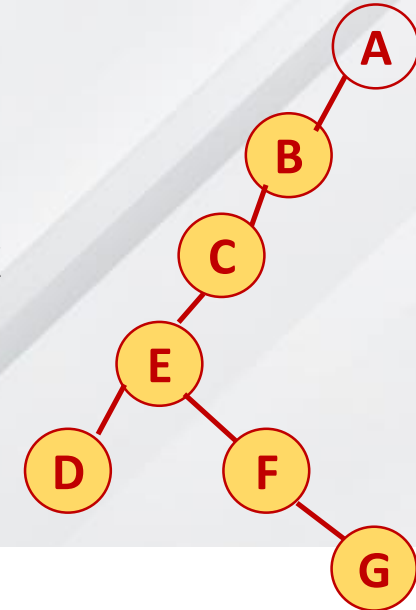


1. Define a **stack** ,size= # of Vertices in graph.
2. Select **any vertex** as starting point , visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3 ,**until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example

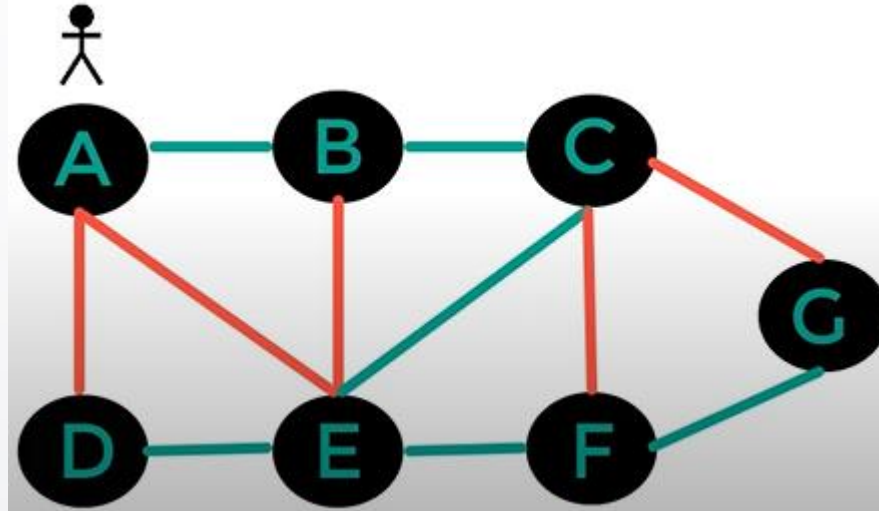


STACK

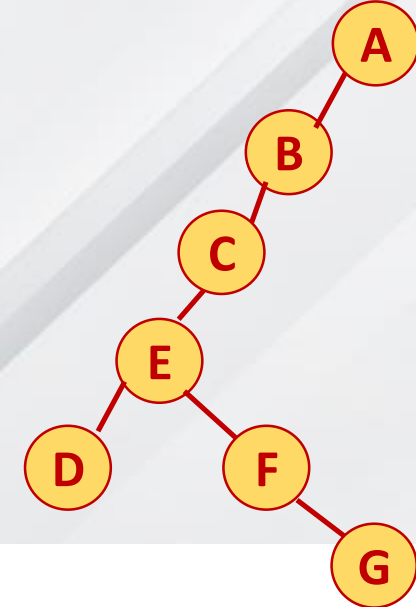


1. Define a **stack** ,size= # of Vertices in graph.
2. Select **any vertex** as starting point , visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3 ,**until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

DFS stack example



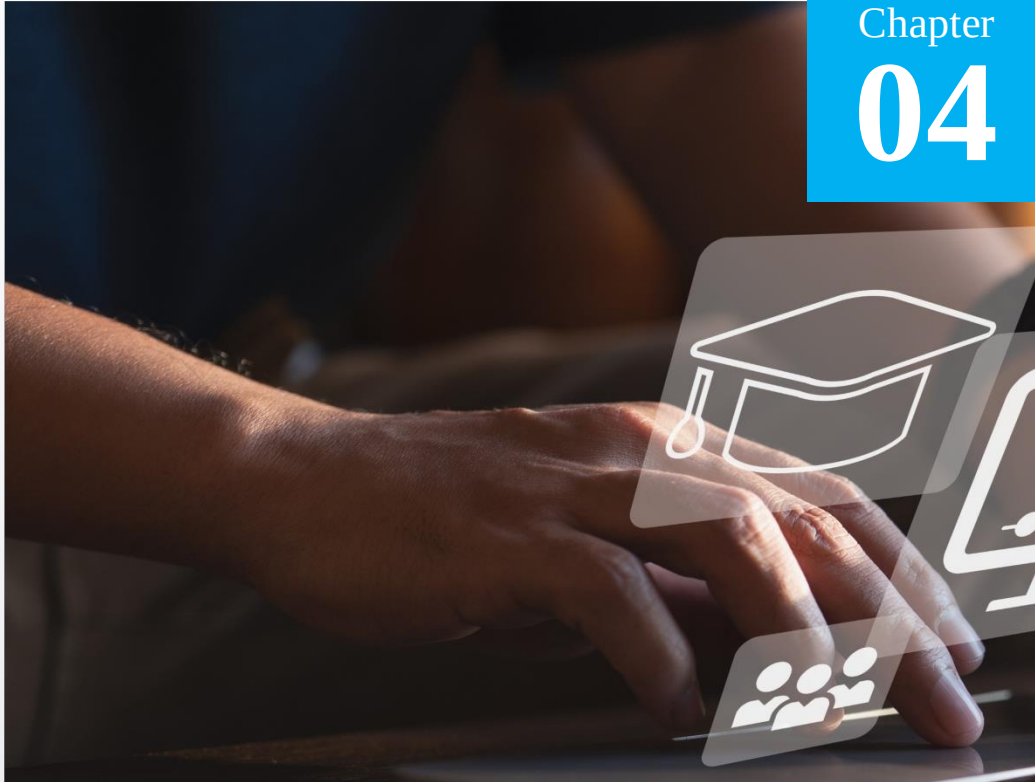
STACK



1. Define a **stack** ,size= # of Vertices in graph.
2. Select **any vertex** as starting point , visit that vertex and **push it on to the stack**.
3. Visit **any one of the adjacent vertex** of the vertex which is **at top** of the stack, Which **is not visited**, and **push it on the stack**.
4. Repeat step 3 ,**until there are no new vertex** to be Visit from the vertex on top of the stack.
5. When there **is no new vertex** to be visit then use **backtracking** and **pop one vertex** from the stack.
6. Repeat steps 3,4,5 **until stack be comes empty**.

Chapter
04

Advantages and Disadvantages of Depth First Search



Advantages of Depth First Search:

- Memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.
- The time complexity of a depth-first Search to depth d and branching factor b (the number of children at each node, the outdegree) is $O(bd)$.

Advantages of Depth First Search:

- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.
- DFS requires less memory since only the nodes on the current path are stored. By chance DFS may find a solution without examining much of the search space at all.

Disadvantages of Depth First Search:

- The disadvantage of Depth-First Search is that there is a possibility that it may down the left-most path forever.
- Depth-First Search is not guaranteed to find the solution.
- And there is no guarantee to find a minimal solution, if more than one solution.

Chapter

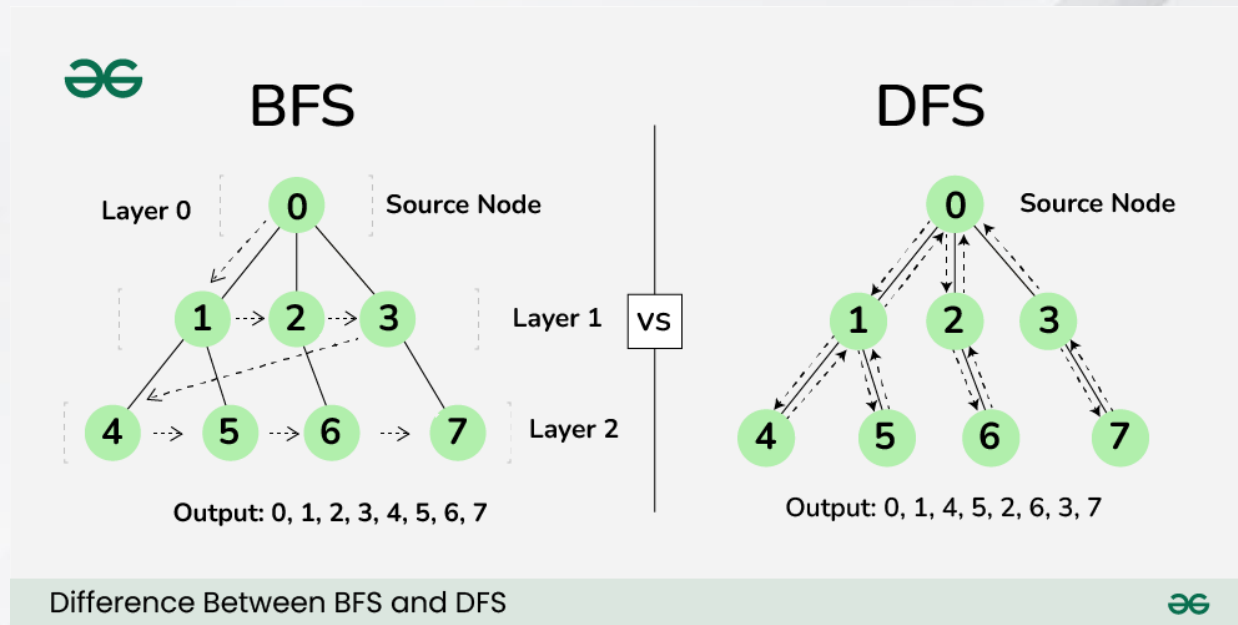
04

Difference between BFS and DFS



Difference between BFS and DFS:

- Breadth-First Search (BFS) and Depth-First Search (DFS) are two fundamental algorithms used for traversing or searching graphs and trees.



Difference Between BFS and DFS



Difference between BFS and DFS

Breadth-First Search (BFS):

BFS, Breadth-First Search, is a vertex-based technique for finding the shortest path in the graph.

It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

Depth First Search (DFS):

DFS, Depth First Search, is an edge-based technique.

It uses the Stack data structure and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.

Difference between BFS and DFS:

Parameters	BFS	DFS
Stands for	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
Data Structure	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
Definition	BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.

Difference between BFS and DFS:

Parameters	BFS	DFS
Conceptual Difference	BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.
Approach used	It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).
Suitable for	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.
Applications	BFS is used in various applications such as bipartite graphs, shortest paths, etc.	DFS is used in various applications such as acyclic graphs and finding strongly connected components etc.



Thank You