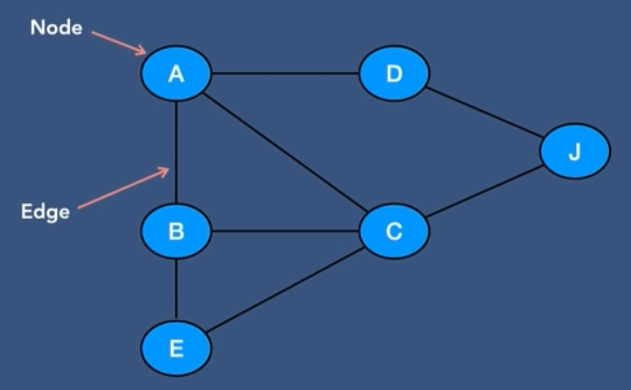
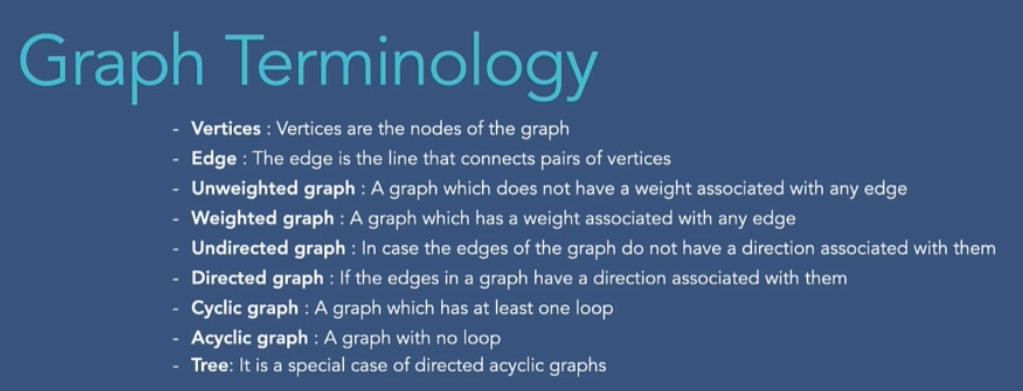
**Graph:** Graph is a finite set of vertices (nodes) and set of edges which connect a pair of vertices. Graphs are used for many real-life problems. It represents a network. Network may represent path of a city, telephone network, electric circuit. It is also used in social networks for connecting different persons (vertices).





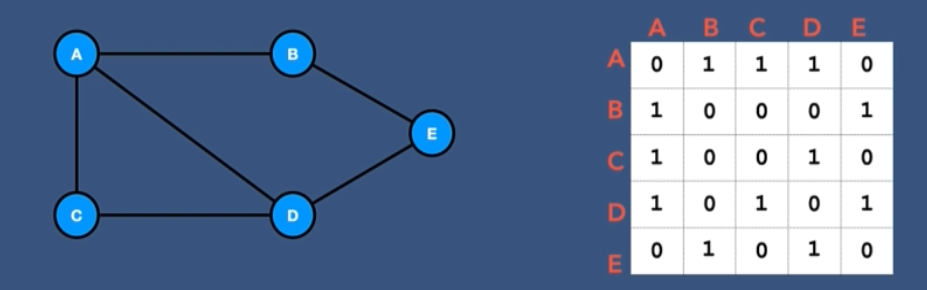
**Tree vs Graph:**

Tree is a special case of directed acyclic graph. Main difference is that tree don’t have cycles while graph can have cycle. Another difference is in tree we have a root node but in graph there is no root node

**Graph Representation:**

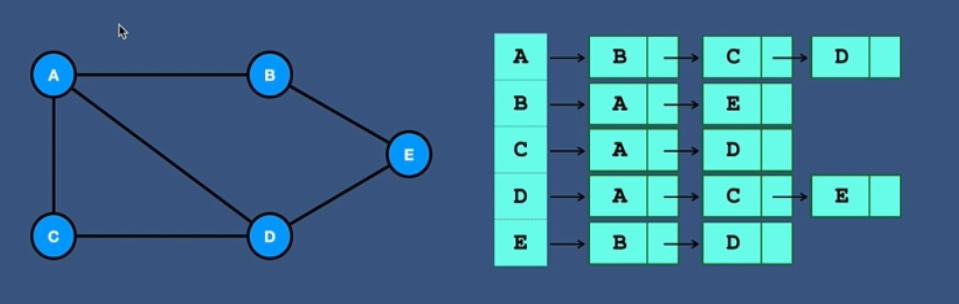
1. **Adjacency matrix**: This is a 2D array which indicates connection between pairs.

* Uses O(n^2) memory
* It is fast to look up and check for presence or absence of a specific edge  
  between any two nodes O (1)
* It is slow to iterate over all edges
* It is slow to add/delete a node; a complex operation O(n^2)
* It is fast to add a new edge O (1)



1. **Adjacency list**: It is a collection of unordered lists that represent a graph. Each list describes the set of neighbors of a vertex in the graph.

* Memory usage depends more on the number of edges (and less on the number of nodes), which might save a lot of memory if the adjacency matrix is sparse
* Finding the presence or absence of specific edge between any two nodes is slightly slower than with the matrix O(k); where k is the number of neighbors nodes
* It is fast to iterate over all edges because you can access any node neighbors directly
* It is fast to add/delete a node; easier than the matrix representation
* It fast to add a new edge O (1)

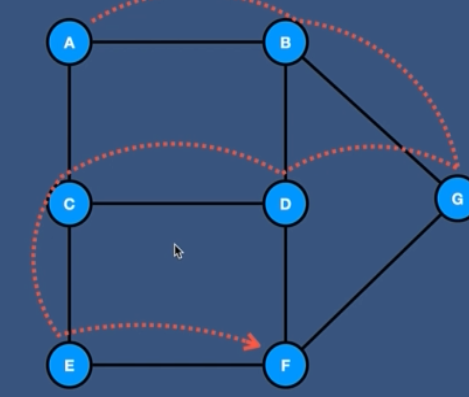
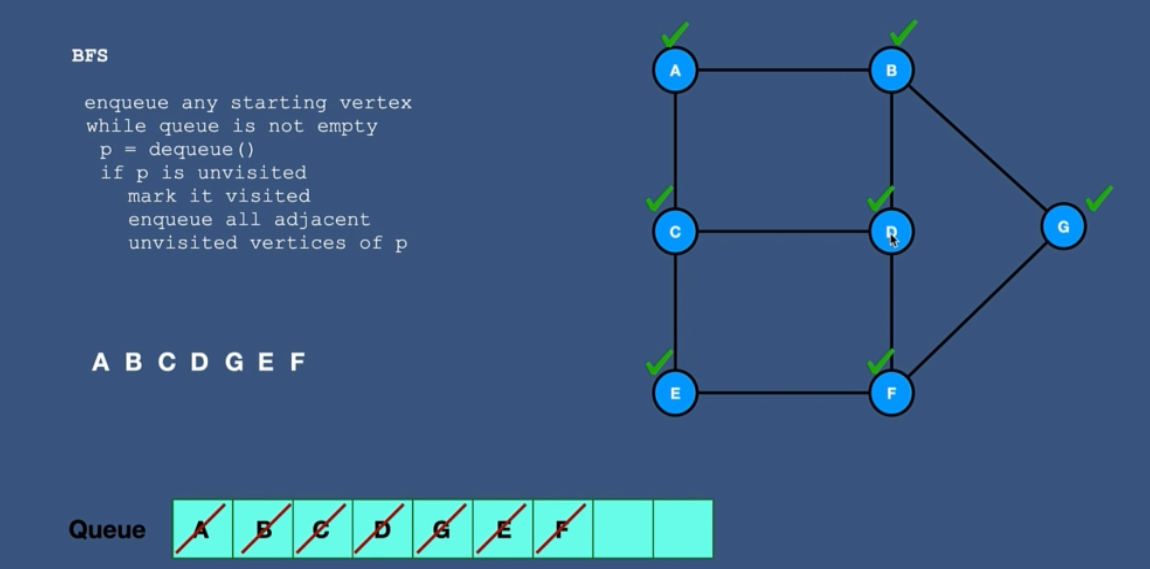


**Which one to use?**

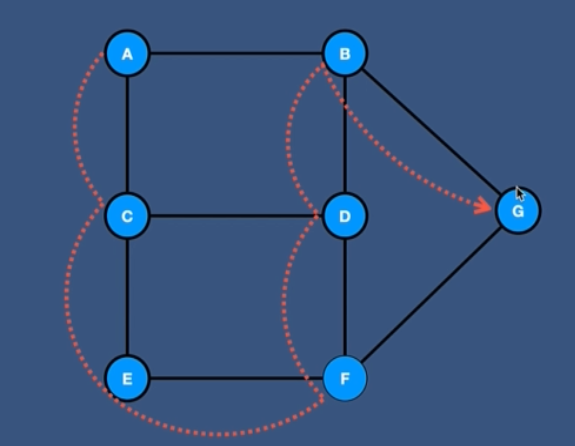
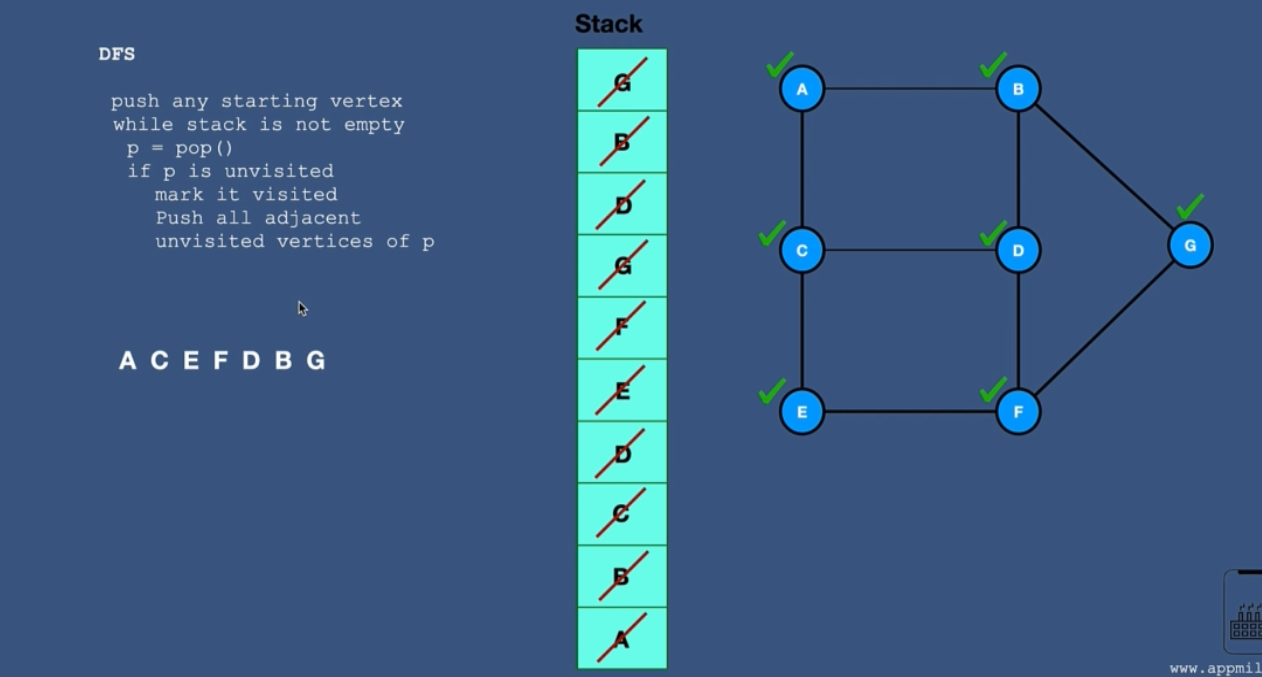
* For complete or almost complete graph we should use adjacency matrix as it will use the matrix efficiently.
* If the number of edges is low, we should use Adjacency list.

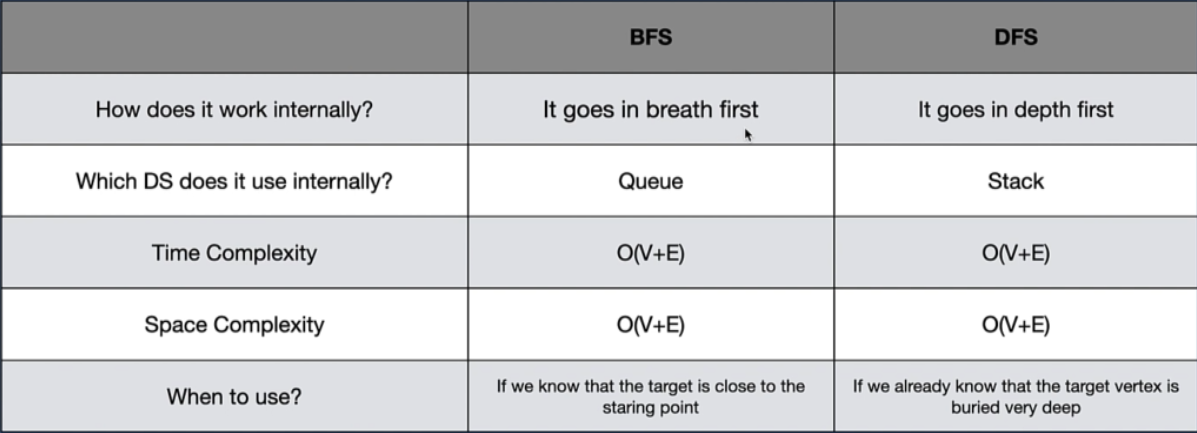
**Graph Traversal:** It is the process of visiting each graph.

1. **Breadth First Search (BFS):** This is an algorithm for traversing graph data structure. It starts at some arbitrary node of graph and explore the neighbor nodes first. Then it moves to next level neighbors.

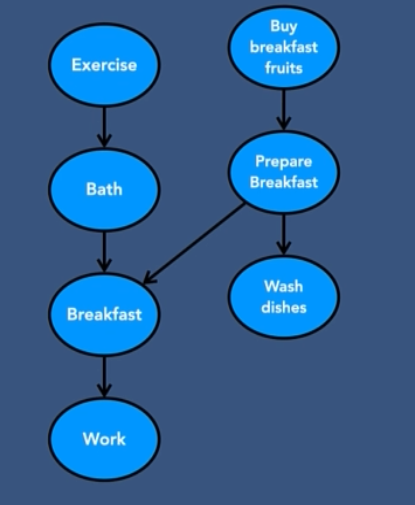
 

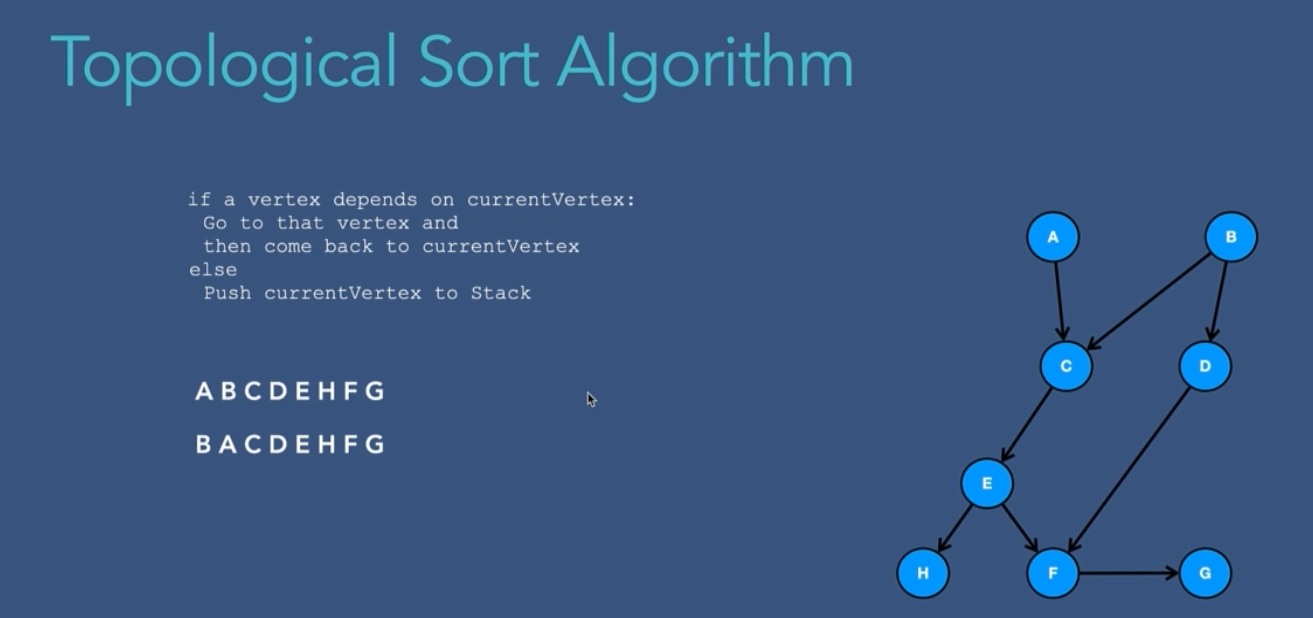
1. **Depth First Search (DFS):** This is an algorithm for traversing graph data structure. It starts at some arbitrary node of graph and explores as far as possible along each edge before backtracking.



**Topological sort:** If there is dependency of one action (for example, Bath will always come after exercise), then the dependent action (Bath) will always come after the parent (Exercise) action

****

****

**Time and space complexity:** O (E + V)

**Single Source Shortest Path problem (SSSP):** This problem is about finding a path between a given vertex (source) and to all other vertices in the graph such that the total distance between them (source to destination) is minimum.

There are basically three main algorithms for solving this problem

1. **BFS:**
2. **Dijkstra algorithm:**
3. **Bellman-Ford algorithm:**