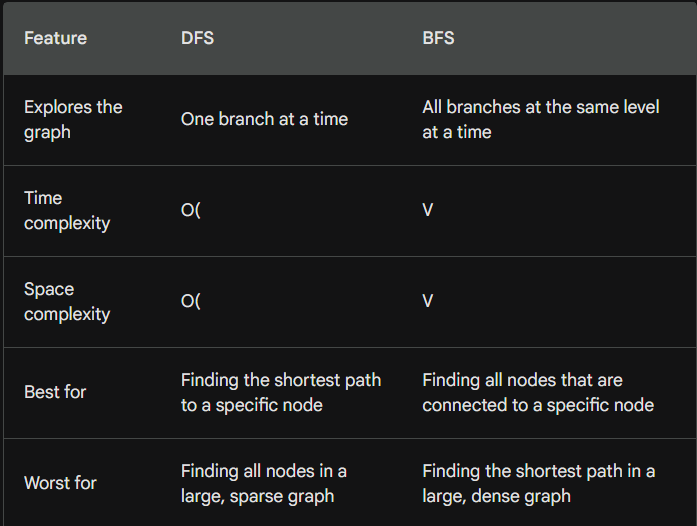
Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure

**DFS & BFS**



In [15]:

*# DFS*

def dfs(graph, start, visited):

visited.add(start)

for neighbor in graph[start]:

if neighbor not in visited:

dfs(graph, neighbor, visited)

*# BFS*

def bfs(graph, start):

visited = set([start])

queue = [start]

while queue:

node = queue.pop(0)

for neighbor in graph[node]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

*# Example*

graph = {

'A': ['B', 'C'],

'B': ['D'],

'C': ['E', 'F'],

'D': [],

'E': [],

'F': []

}

*# DFS*

visited = set()

dfs(graph, 'A', visited)

print(visited)

*# BFS*

visited = set()

queue = ['A']

while queue:

node = queue.pop(0)

print(node)

for neighbor in graph[node]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

{'A', 'E', 'F', 'B', 'D', 'C'}

A

B

C

D

E

F

**RECURSIVE DFS**

Recursive DFS is a graph traversal algorithm that explores the graph one branch at a time.

The algorithm starts at a given node and then recursively explores all of the node's children.

If a child node has not been visited, the algorithm will recursively explore that child node.

The algorithm continues to recursively explore nodes until all of the nodes in the graph have been visited.

Recursive DFS is a powerful tool for finding paths in graphs, but it can be inefficient for large graphs.

For large graphs, it is often better to use an iterative graph traversal algorithm, such as breadth-first search.

In [16]:

def dfs\_recursive(graph, node, visited):

"""

Performs a recursive depth-first search on the given graph, starting at the given node.

Args:

graph: The graph to search.

node: The node to start the search at.

visited: A set of nodes that have already been visited.

Returns:

A list of all the nodes that were visited during the search.

"""

if node in visited:

return

visited.add(node)

for neighbor in graph[node]:

dfs\_recursive(graph, neighbor, visited)

return visited

*# Perform a recursive depth-first search*

visited = dfs\_recursive(graph, "A", set())

*# Print the visited nodes*

print(visited)

{'A', 'E', 'F', 'B', 'D', 'C'}