1. Using Adjacency List

Code:

from collections import deque

def bfs(graph, start):

visited = set() # To track visited nodes

queue = deque([start]) # Initialize a queue with the starting node

visited.add(start) # Mark the starting node as visited

while queue:

node = queue.popleft() # Dequeue a node from the front of the queue

print(node, end=" ") # Process the node (e.g., print its value)

# Enqueue all unvisited neighbors

for neighbor in graph[node]:

if neighbor not in visited:

visited.add(neighbor) # Mark the neighbor as visited

queue.append(neighbor) # Add the neighbor to the queue

# Example graph represented as an adjacency list

graph = {

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

'B': ['D', 'E'],

'C': ['F'],

'D': [],

'E': ['F'],

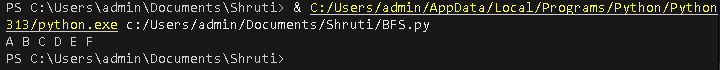
'F': []

}

# Perform BFS starting from node 'A'

bfs(graph, 'A') # Output: A B C D E F

Output:



1. Using Adjacency Matrix

Code:

from collections import deque

def bfs\_matrix(adj\_matrix, start, node\_labels):

n = len(adj\_matrix) # Number of nodes

visited = [False] \* n # Track visited nodes

queue = deque([start]) # Initialize the queue with the start node

visited[start] = True # Mark the start node as visited

while queue:

node = queue.popleft() # Dequeue a node

print(node\_labels[node], end=" ")# Process the node (print label)

# Check all adjacent nodes

for neighbor in range(n):

if adj\_matrix[node][neighbor] == 1 and not visited[neighbor]:

visited[neighbor] = True

queue.append(neighbor)

# Node labels for display

node\_labels = ['A', 'B', 'C', 'D', 'E', 'F']

# Equivalent adjacency matrix for the same graph structure

# A — B — D

# | |

# C E — F

adj\_matrix = [

# A B C D E F

[0, 1, 1, 0, 0, 0], # A

[1, 0, 0, 1, 1, 0], # B

[1, 0, 0, 0, 0, 0], # C

[0, 1, 0, 0, 0, 0], # D

[0, 1, 0, 0, 0, 1], # E

[0, 0, 0, 0, 1, 0], # F

]

# Perform BFS starting from node 'A' (index 0)

bfs\_matrix(adj\_matrix, 0, node\_labels) # Output: A B C D E F

Output:

