جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قسم الانظمة الطبية الذكية

Breadth First Search

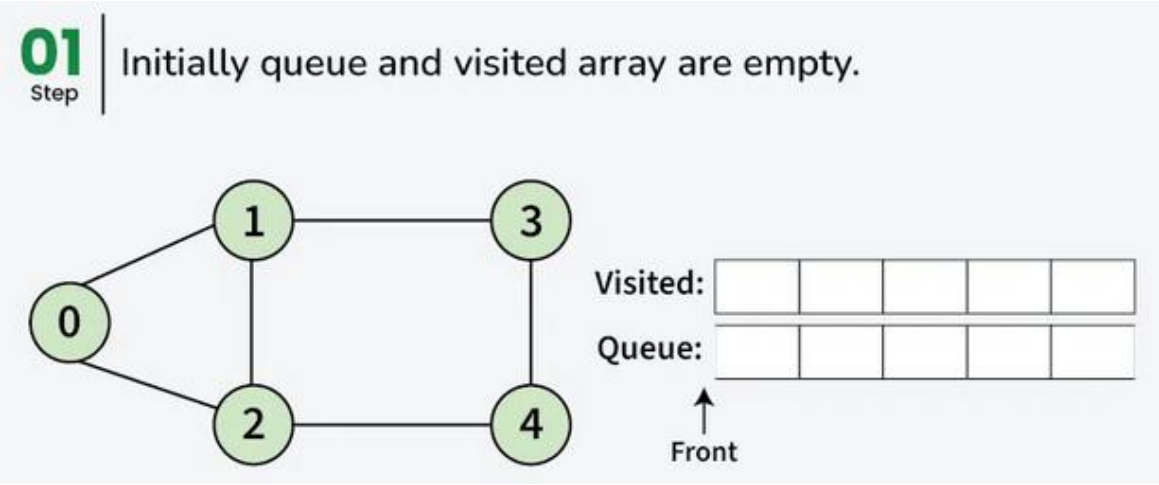
Lab: (2)

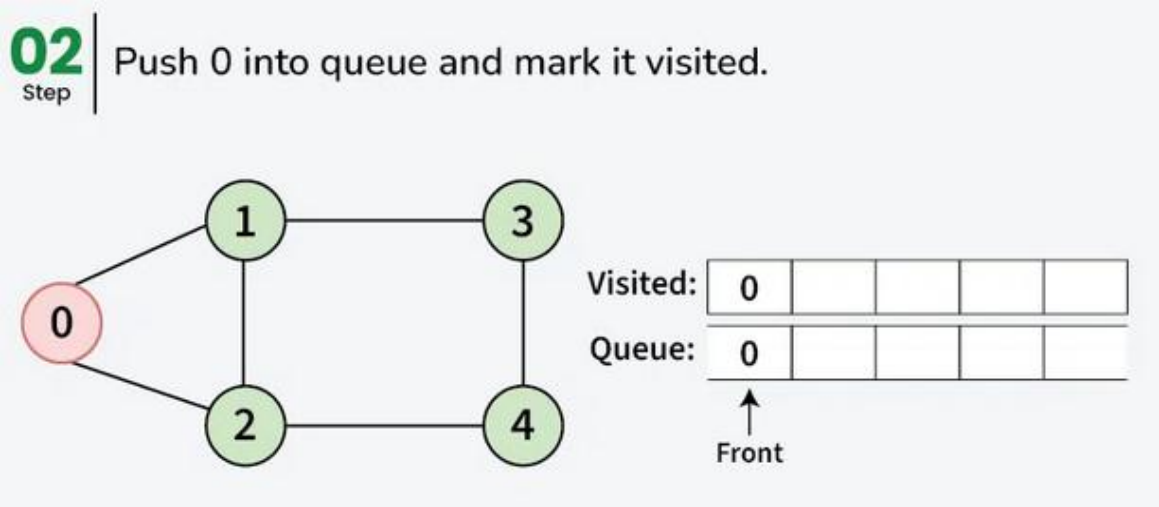
Subject: Artificial Intelligence
Class: Third
Lecturer: Dr. Maytham N. Meqdad

Breadth First Search or BFS for a Graph

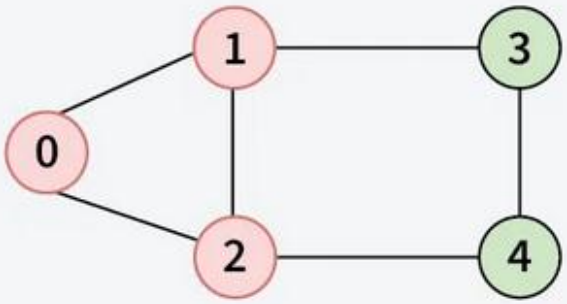
Breadth First Search (BFS) is a fundamental **graph traversal algorithm.** It begins with a node, then first traverses all its adjacent. Once all adjacent are visited, then their adjacent are traversed. This is different from DFS in a way that closest vertices are visited before others. We mainly traverse vertices level by level. A lot of popular graph algorithms like Dijkstra's shortest path, Kahn's Algorithm, and Prim's algorithm are based on BFS. BFS itself can be used to detect cycle in a directed and undirected graph, find shortest path in an unweighted graph and many more problems.

How Does the BFS Algorithm Work?





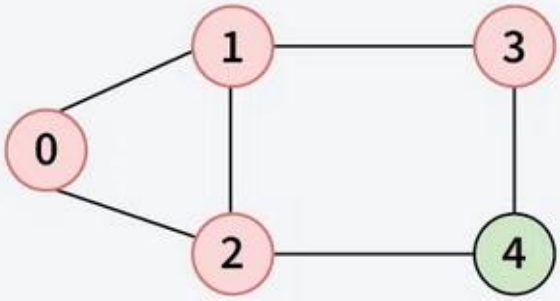
03 Step Remove 0 from the front of queue and visit the unvisited neighbours and push them into queue.



Visited:	0	1	2		
Queue:	0	1	2		

Front

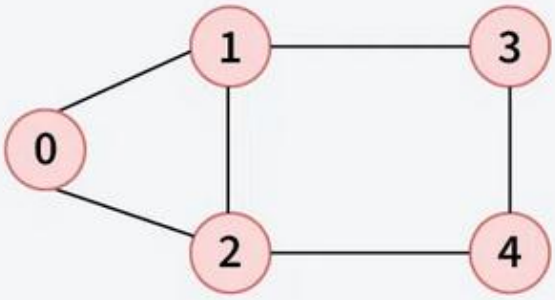
04 Step Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



Visited:	0	1	2	3	
Queue:	1	2	3		

Front

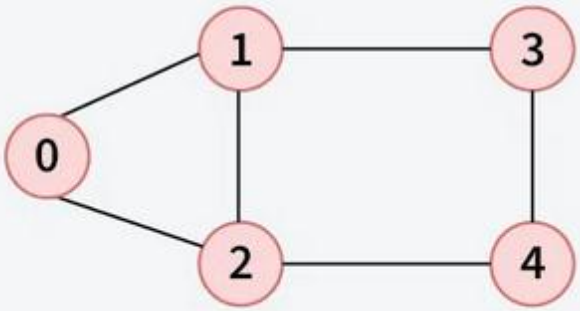
05 Step Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



Visited:	0	1	2	3	4
Queue:	2	3	4		

Front

06 Step | Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

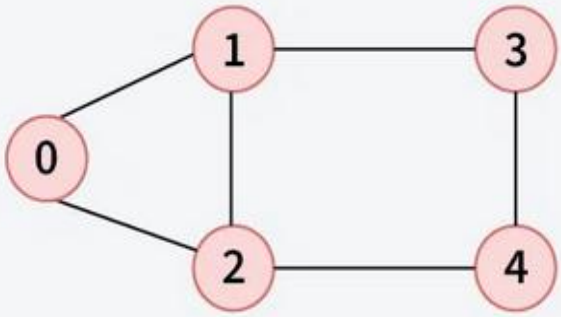


Visited:	0	1	2	3	4
Queue:	3	4			

Front

All neighbors of node 3 have been visited, proceed to the next node in the queue.

07 Step | Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.



Visited:	0	1	2	3	4
Queue:	4				

Front

All neighbors of node 4 have been visited, proceed to the next node in the queue.

```
from collections import deque

# BFS from given source s
def bfs(adj, s):

    # Create a queue for BFS
    q = deque()

    # Initially mark all the vertices as not visited
    # When we push a vertex into the q, we mark it as
    # visited
    visited = [False] * len(adj);

    # Mark the source node as visited and enqueue it
    visited[s] = True
    q.append(s)

    # Iterate over the queue
    while q:

        # Dequeue a vertex from queue and print it
        curr = q.popleft()
        print(curr, end=" ")

        # Get all adjacent vertices of the dequeued
        # vertex. If an adjacent has not been visited,
        # mark it visited and enqueue it
        for x in adj[curr]:
            if not visited[x]:
                visited[x] = True
                q.append(x)

# Function to add an edge to the graph
def add_edge(adj, u, v):
    adj[u].append(v)
    adj[v].append(u)

# Example usage
if __name__ == "__main__":

    # Number of vertices in the graph
    V = 5

    # Adjacency list representation of the graph
    adj = [[] for _ in range(V)]

    # Add edges to the graph
    add_edge(adj, 0, 1)
    add_edge(adj, 0, 2)
    add_edge(adj, 1, 3)
    add_edge(adj, 1, 4)
    add_edge(adj, 2, 4)

    # Perform BFS traversal starting from vertex 0
    print("BFS starting from 0: ")
    bfs(adj, 0)
```

Output

```
BFS starting from 0 :
0 1 2 3 4
```

https://www.programiz.com/online-compiler/45xxhTFmnVwCs