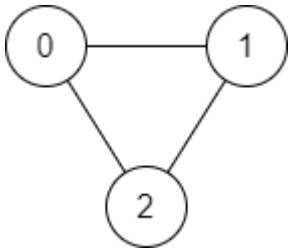
Find If Path Exists in Graph LeetCode

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to n - 1 (**inclusive**). The edges in the graph are represented as a 2D integer array edges, where each edges[i] = [u_i, v_i] denotes a bi-directional edge between vertex u_i and vertex v_i. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex source to vertex destination.

Given edges and the integers n, source, and destination, return true *if there is a **valid path** from* source *to* destination*, or* false *otherwise.*

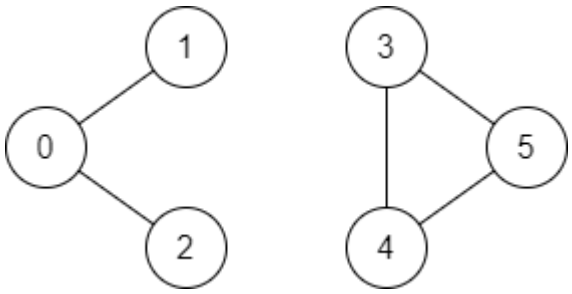
Example:



Source = 0, Destination = 2

Output: True

Example 1:



Source = 0, Destination = 5

Output: False

Approach 1: BFS traversal to check if a valid path exists from source to destination

- **Explanation:**
    - The **validPathBFS** function uses BFS traversal to explore nodes and check if a valid path exists from the source to the destination.
    - It maintains a queue for BFS and marks visited nodes to avoid infinite loops.

- **Time Complexity:**
    - **The time complexity is O(V + E), where V is the number of vertices, and E is the number of edges.**
        - **V: Visiting each vertex once**
        - **E: Exploring each edge once**
- **Space Complexity:**
    - **The space complexity is O(V), attributed to the queue and the visited vector.**
        - **V: Storing information about each vertex during traversal**

Approach 2: DFS-based function to check if a valid path exists from source to destination

- **Explanation:**
    - The **validPathDFS** function employs DFS traversal to explore nodes and check if a valid path exists from the source to the destination.
    - It uses recursive DFS calls and marks visited nodes to prevent revisiting.
- **Time Complexity:**
    - **The time complexity is O(V + E), where V is the number of vertices, and E is the number of edges.**
        - **V: Visiting each vertex once**
        - **E: Exploring each edge once**
- **Space Complexity:**
    - **The space complexity is O(V), attributed to the recursion call stack and the visited vector.**
        - **V: Storing information about each vertex during traversal**

Conclusion:

Both BFS and DFS approaches effectively determine the existence of a valid path in a graph. The choice between them depends on specific project requirements or graph

characteristics. In this scenario, both approaches demonstrate similar time and space complexities, making either approach suitable.