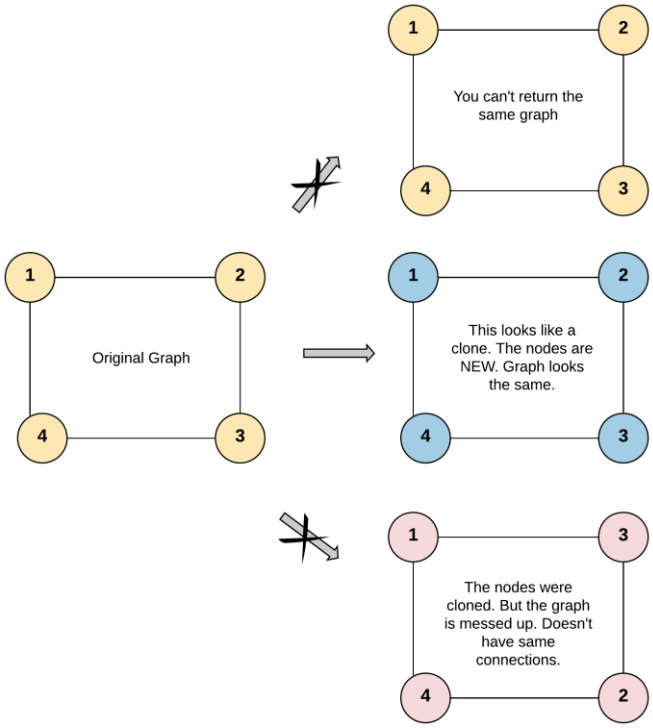
Clone Graph LeetCode

Given a reference of a node in a **connected** undirected graph.

Return a deep copy (clone) of the graph.

Each node in the graph contains a value (int) and a list (List[Node]) of its neighbors.

Example:



Approach 1: Function to clone the graph using DFS

- **Explanation:**
  - The program provides a DFS-based cloning function **cloneGraphDFS**.
  - It uses a recursive DFS traversal to clone the graph, creating new nodes for unvisited neighbors.
- **Time Complexity:**
  - **The time complexity is O(V + E), where V is the number of vertices and E is the number of edges.**
- **Space Complexity:**
  - **The space complexity is O(V), where V is the number of vertices, due to the recursive call stack.**

Approach 2: Function to clone the graph using BFS

- **Explanation:**
    - The program provides a BFS-based cloning function **cloneGraphBFS**.
    - It uses a queue for BFS traversal, creating new nodes for unvisited neighbors.
- **Time Complexity:**
    - **The time complexity is O(V + E), where V is the number of vertices and E is the number of edges.**
- **Space Complexity:**
    - **The space complexity is O(V), where V is the number of vertices, due to the queue.**

Conclusion:

Both DFS and BFS approaches offer effective strategies for cloning a graph. The choice between the two depends on specific requirements and preferences. In terms of simplicity and ease of implementation, DFS provides a straightforward recursive approach. Both approaches have the same time and space complexity, making them efficient for graph cloning. While DFS might be preferable for its simplicity.