Input:

GraphNode node

Output:

GraphNode node

Sample 1:

1 -> 2,3

2 -> 3,4

3 -> 4

Sample 2:

Null

Sample 3:

1

Approach:

BFS AdjList:

1 -> 2,3

2 -> 1,3,4

3 -> 1,2,4

4 -> 2,3

1 -> Node – 2, 3

2 -> Node – 1, 3, 4

3 -> Node – 1, 2, 4

4 -> Node – 2, 3

Time complexity : O(V+E)

Space complexity : O(V)

Code:

public class GraphNode {

private int value;

private List<GraphNode> neighbors = new ArrayList<>();

public GraphNode(int value){

this.value = value;

}

public void addNeighbor(GraphNode node){

neighbors.add(Node);

}

public List<GraphNode> getNeighbors(){

return neighbors;

}

public int getValue(){

return value;

}

public GraphNode clone(){

return new GraphNode(this.value);

}

}

public class GraphCloner {

private HashSet<Integer> visitedNodes = new HashSet<>();

private GraphNode resultNode = null;

private Queue<GraphNode> queue = null;

private Hashmap<Integer, GraphNode> nodeMap = new HashMap<>();

private void cloneBFS(){

while(!queue.isEmpty){

node = queue.remove();

visitedNodes.add(node);

GraphNode currentNode = fetchNewNode(node.getValue());

for(GraphNode neighbor : node.getNeighbors()){

currentNode.addNeighbors(fetchNewNode(neighbor.getValue());

if(!visitedNode.conatins(neighbor)){

queue.add(neighbor);

}

}

}

}

private GraphNode fetchNewNode(int value){

if(nodesMap.containsKey(value)){

return nodesMap.get(value);

}

GraphNode newNode = node.clone();

nodeMap.put(value, newNode);

return newNode;

}

public GraphNode clone(GraphNode node) {

if(node == null){

return null;

}

queue = new LinkedList<>();

queue.add(node);

resultNode = fetchNewNode(node.getValue());

cloneByBFS();

return resultNode;

}

}

Testing:

1 -> 2,3

2 -> 1, 3,4

3 -> 4

q:

2 3

map:

1 -> 1

2 -> 2

3 -> 3

visited:

1, 2

output:

1 -> 2, 3

2 -> 1, 3, 4

3 -> 1, 2, 4

4 -> 2, 3

return 1

Test2:

null

return null

Test 3:

1

return 1