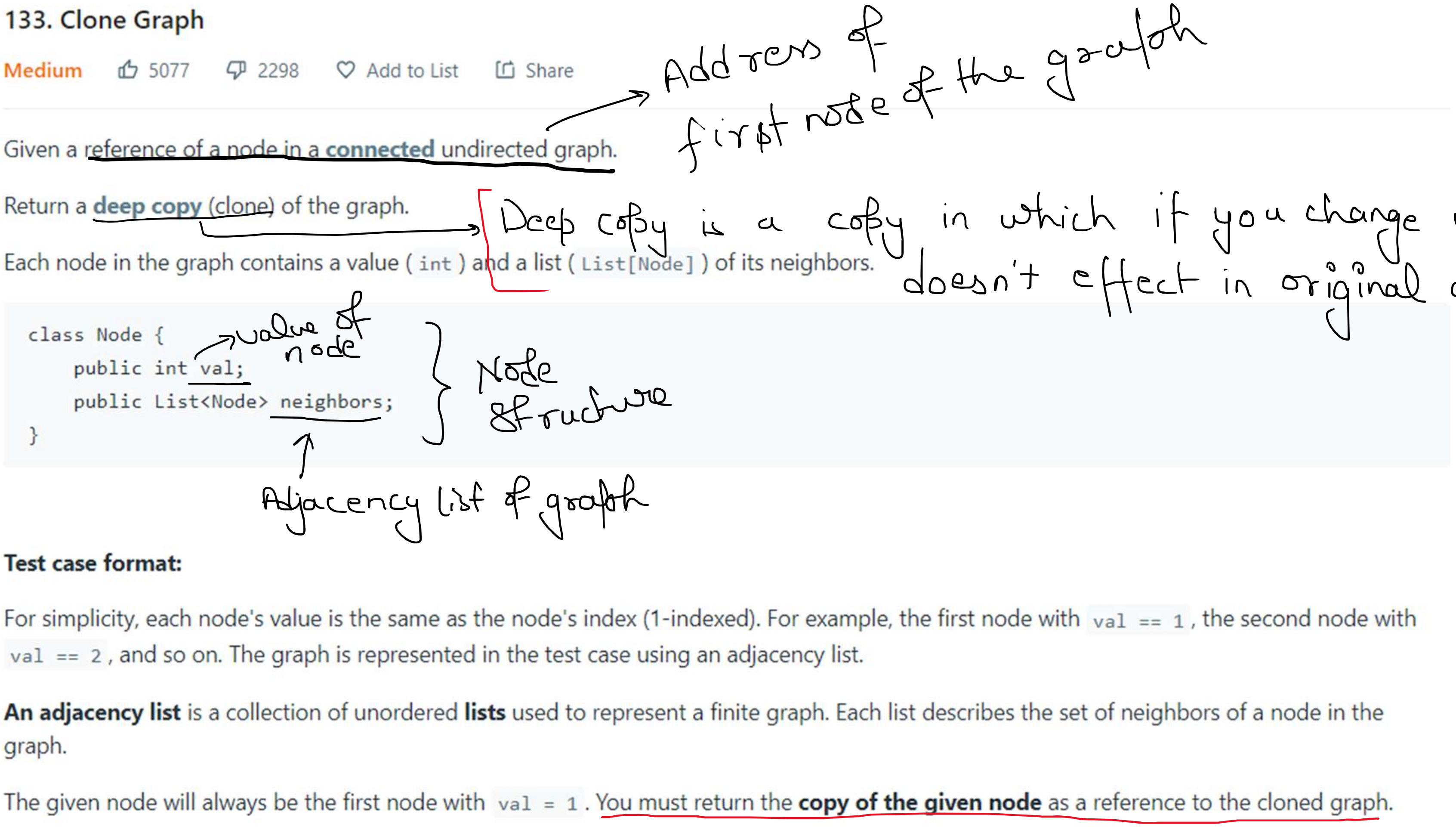
133. Clone Graph

https://leetcode.com/problems/clone-graph/

February 23, 2022

-Priyanshu Arya



133. Clone Graph

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

Address of first node of the graph

Return a **deep copy** (clone) of the graph.

Deep copy is a copy in which if you change value it doesn't effect in original one

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

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

value of node

Node structure

Adjacency list of graph

Test case format:

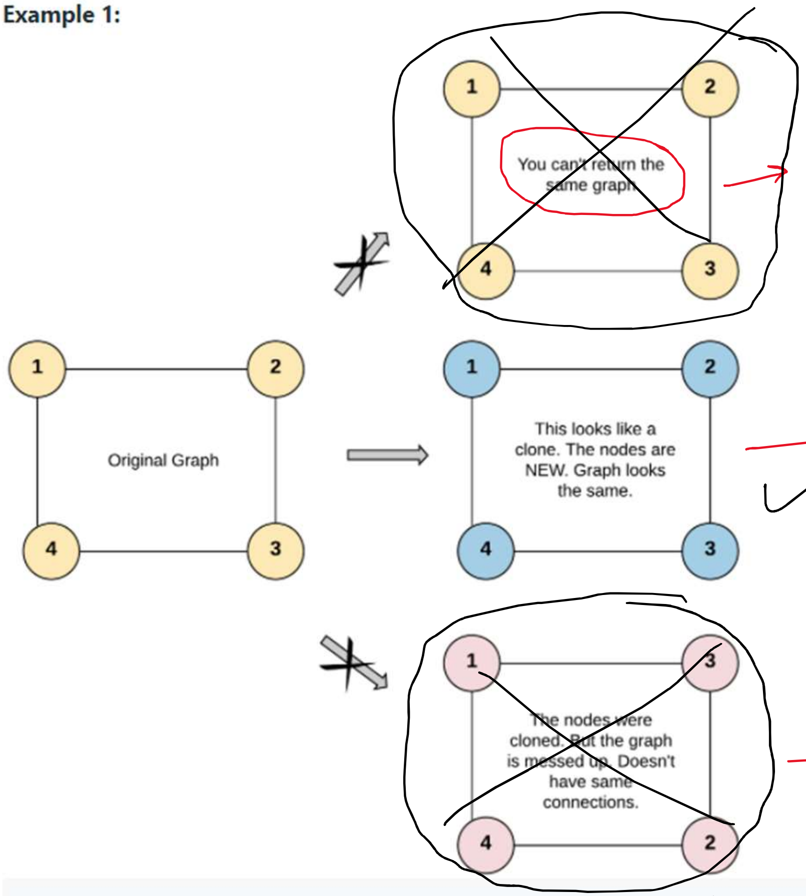
For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with `val == 1`, the second node with `val == 2`, and so on. The graph is represented in the test case using an adjacency list.

An adjacency list is a collection of unordered **lists** used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with `val = 1`. You must return the **copy of the given node** as a reference to the cloned graph.

return the address of cloned graph

Example 1:


1
2
3
4
Original Graph
You can't return the same graph.
This looks like a clone. The nodes are NEW. Graph looks the same.
The nodes were cloned. But the graph is messed up. Doesn't have same connections.

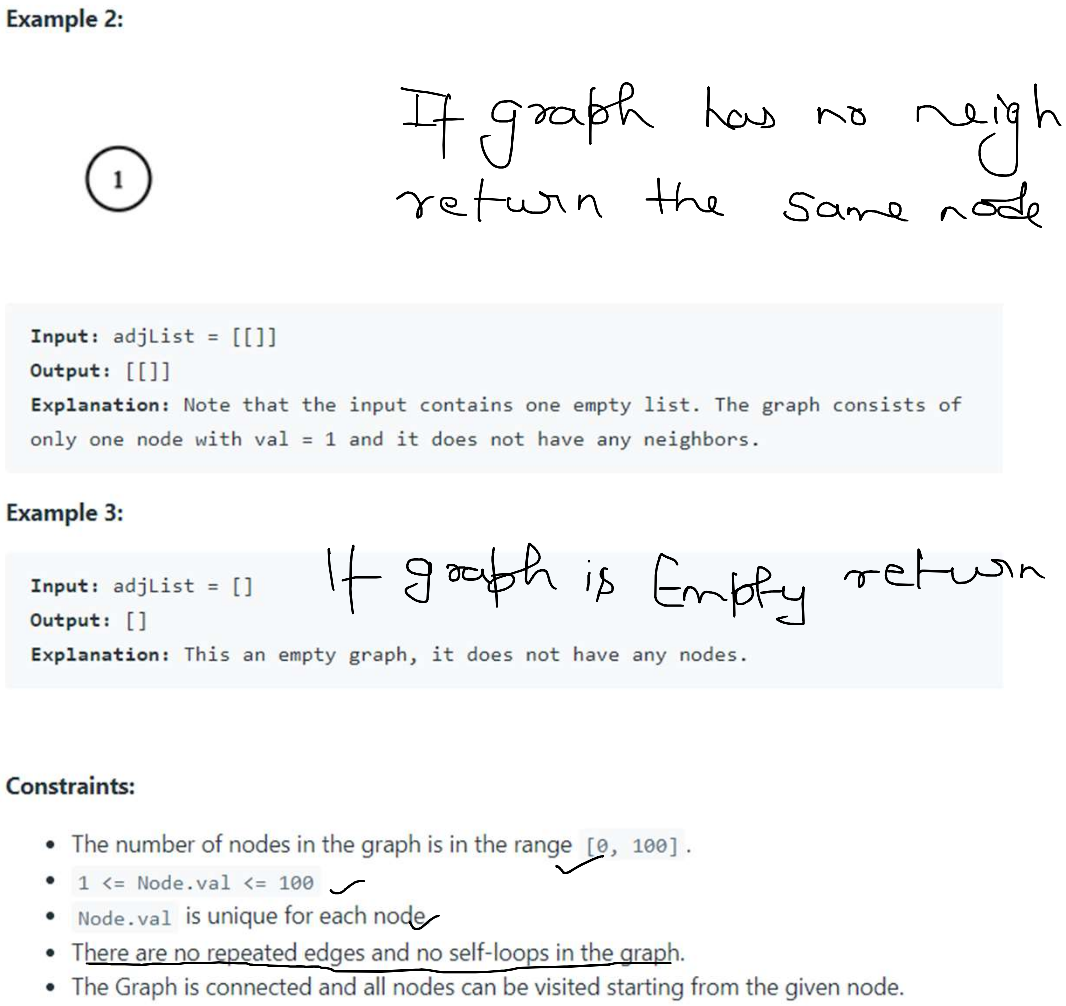
If you change in value of this graph it will reflect back in original graph ✗

If you change in value of this graph then it will not reflect back in original one & the structure is also same. ✓

You cannot change the order of the graph elements ✗

```
Input: adjList = [[2,4],[1,3],[2,4],[1,3]]
Output: [[2,4],[1,3],[2,4],[1,3]]
Explanation: There are 4 nodes in the graph.
1st node (val = 1)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
2nd node (val = 2)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
```

Input (Adjacency List)



Example 2:

1

If graph has no neighbour return the same node

```
Input: adjList = [[]]
Output: [[]]
Explanation: Note that the input contains one empty list. The graph consists of
only one node with val = 1 and it does not have any neighbors.
```

Example 3:

If graph is Empty return none

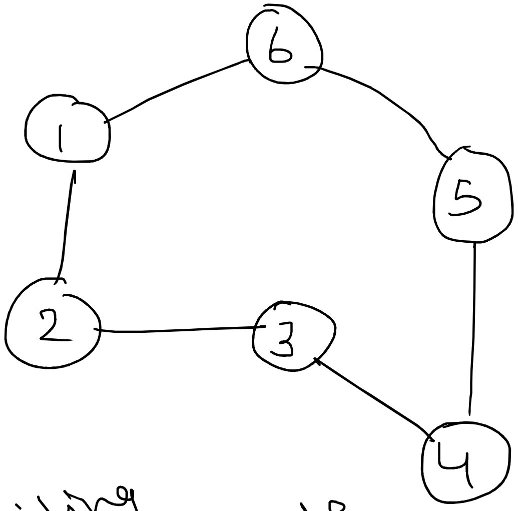
```
Input: adjList = []
Output: []
Explanation: This an empty graph, it does not have any nodes.
```

Constraints:

- The number of nodes in the graph is in the range `[0, 100]`.
- `1 <= Node.val <= 100`
- `Node.val` is unique for each node.
- There are no repeated edges and no self-loops in the graph.
- The Graph is connected and all nodes can be visited starting from the given node.

For Ex

we should create same graph like this of same structure also



For Visiting every node in graph you can use any one of them

BFS | DFS

Intution

1) Visit to every node of the graph

2) Create deep copy of the node

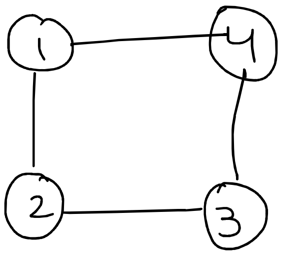
3) Use any data structure to make sure that we don't have that node previously in our new graph

4) Link all nodes with same structure as original one

5) Return new node

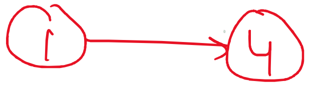
For Ex

We are performing DFS (Depth First Search)

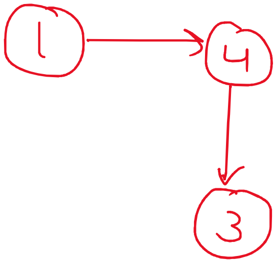


Create a copy of node

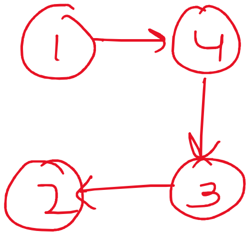
Now we have two nodes to explore 4 and 3



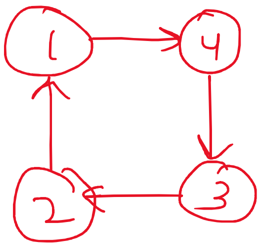
Now we have to explore 3 $ 1



Now we have to explore 2$4



Now we have explore 1 $ 3



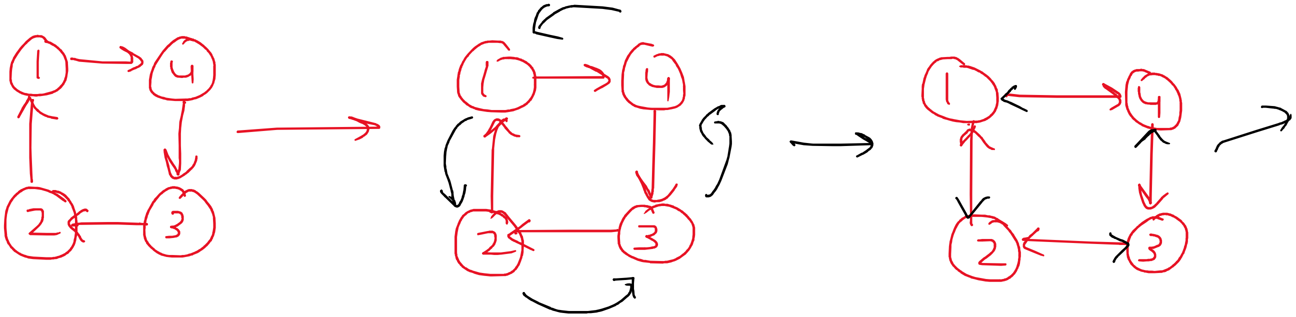
1 has already in our Hashmap it means we get to our maximum depth now we are returning back.

HashMap

Old	New
1 →	1
4 →	4
3 →	3
2 →	2

{Please don't miss the arrows sign in the graph it was representing that the graph is only connected by one side not both side. }

Now see



Now it is connected from both side hence our graph is get successfully cloned.

Time Complexity : $O(V+E)$

$V \rightarrow$ No. of Vertices

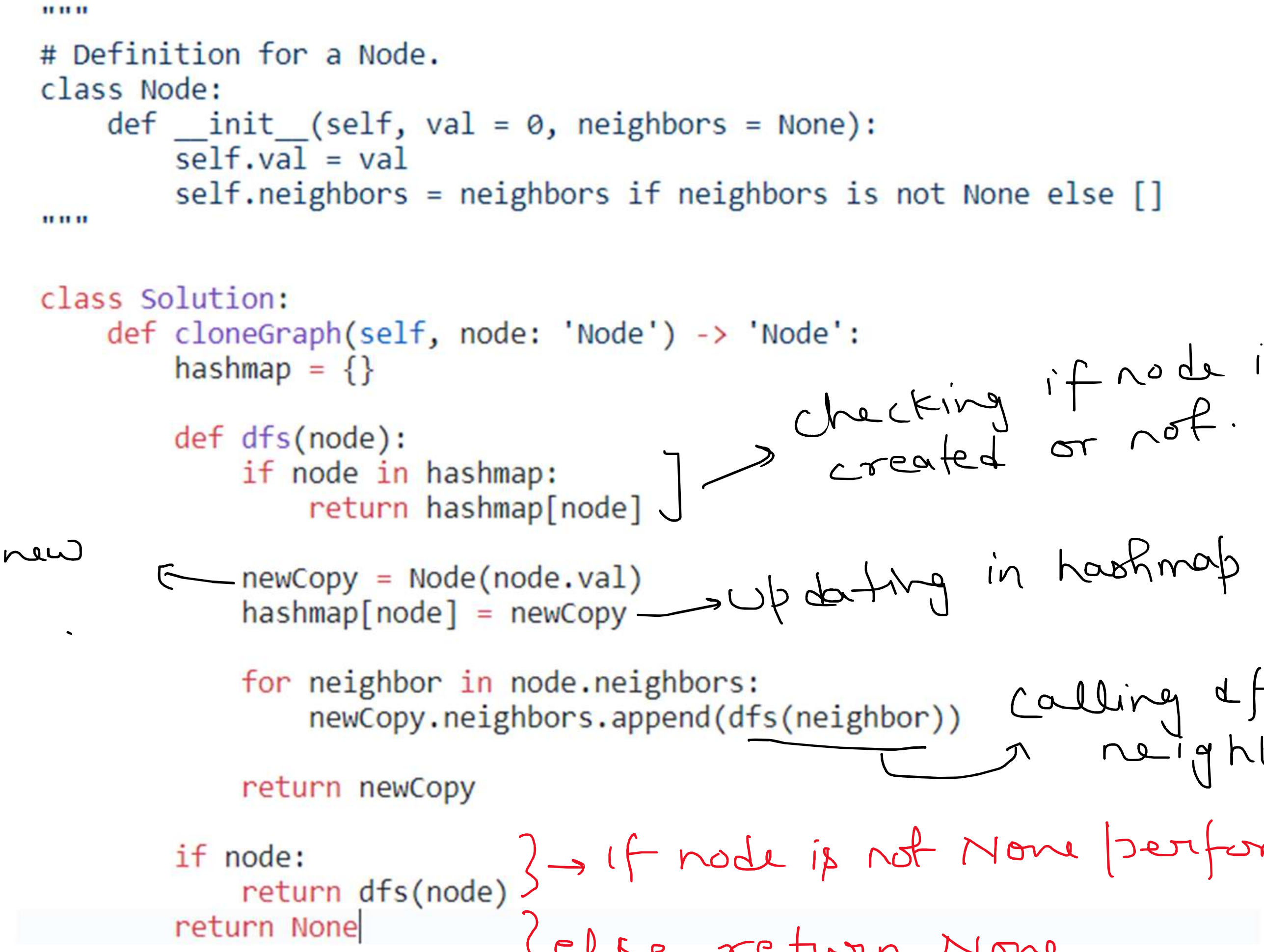
$E \rightarrow$ No. of Edges

Space Complexity: $O(V+E)$

```
Algorithm Clone graph (node):
        map = { }

        dfs (node):
            if node in map:                              } If copy is already
                return map[node]                         }   created

            newcopy = Node(node.val)         } → creating new node
     dfs    map[node] = newcopy              } → updating new created node in
                                                   map
           for neighbour in node.neighbours:                     } calling
               newcopy.neighbours.append(dfs(neighbour))         } for its
                                                                   all neighbours
           return newcopy
                  └→ Returning new copy
        return dfs(node)
```



```
"""
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []
"""

class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        hashmap = {}

        def dfs(node):
            if node in hashmap:
                return hashmap[node]

            newCopy = Node(node.val)
            hashmap[node] = newCopy

            for neighbor in node.neighbors:
                newCopy.neighbors.append(dfs(neighbor))

            return newCopy

        if node:
            return dfs(node)
        return None
```

checking if node is already created or not.

creating new node

updating in hashmap

calling dfs for all neighbors

→ if node is not None perform dfs

else return None

Thank you

If you like Please share this and feel free to connect for any queries.
GitHub: https://github.com/priyanshu-arya/DSA/tree/master/Leetcode%201
Discord: https://discord.gg/qPer56TP
Mail: priyanshuarya2482000@gmail.com