## 133. Clone Graph

Medium 1357 1218 Add t

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 val (`int`) and a list (`List[Node]`) of its neighbors.

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

**Test case format:**

For simplicity sake, 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.

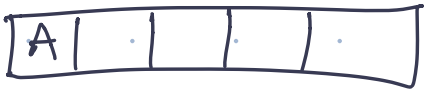
**Adjacency list** is a collection of unordered **lists** used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

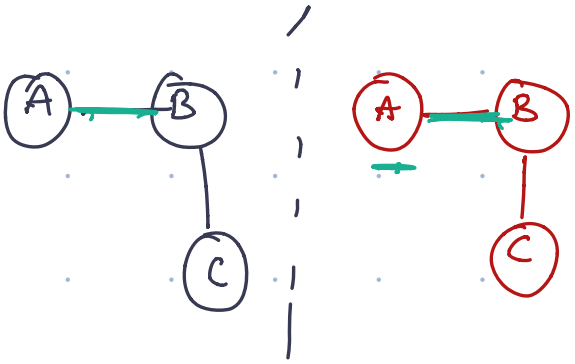
The given node will always be the first node with `val = 1`. You must return the **copy of the given node** as a reference to the cloned graph.

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> neighbors;

    Node() {
        val = 0;
        neighbors = vector<Node*>();
    }

    Node(int _val) {
        val = _val;
        neighbors = vector<Node*>();
    }

    Node(int _val, vector<Node*> _neighbo
        val = _val;
        neighbors = _neighbors;
    }
};
*/
class Solution {
public:
    Node* cloneGraph(Node* node) {

    }
};
```

# BFS

1) Queue




Hashmap

{

A : (A)

B : (B)

C : (C)

}

Input : [[2,4], [1,3], [2,4], [1,3]]

1 2 3 4

graph : {

1 : [2,4] → Represented as

2 : [1,3]

3 : [2,4]

4 : [1,3]

}

Node {

val : 1

neighbors : [2,4]

}

Node.val

Node.neighbors.

Node1 : node 1

Node2 : node 2

Node3 : node 3

Node4 : node 4

Node1 :

val : 1

neighbors = [Node2, Node4]

```
for k, y in map.items():
    for nei in key.neighbors:
        c_node = hashmap[nei]

        y.neighbors.append(c_node)
```

Node1 : [Node2, Node4]