# Clone Graph

Clone an undirected graph. Each node in the graph contains a `label` and a list of its `neighbors`.

**OJ's undirected graph serialization:**
Nodes are labeled uniquely.

We use `#` as a separator for each node, and `,` as a separator for node label and each neighbor of the node.
As an example, consider the serialized graph `{0,1,2#1,2#2,2}`.

The graph has a total of three nodes, and therefore contains three parts as separated by `#`.

1. First node is labeled as `0`. Connect node `0` to both nodes `1` and `2`.
2. Second node is labeled as `1`. Connect node `1` to node `2`.
3. Third node is labeled as `2`. Connect node `2` to node `2` (itself), thus forming a self-cycle.

Visually, the graph looks like the following:

```
       1
      / \
     /   \
    0 --- 2
         / \
         \_/
```

## Solution 1

```
public class Solution {
    private HashMap<Integer, UndirectedGraphNode> map = new HashMap<>();
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        return clone(node);
    }

    private UndirectedGraphNode clone(UndirectedGraphNode node) {
        if (node == null) return null;

        if (map.containsKey(node.label)) {
            return map.get(node.label);
        }
        UndirectedGraphNode clone = new UndirectedGraphNode(node.label);
        map.put(clone.label, clone);
        for (UndirectedGraphNode neighbor : node.neighbors) {
            clone.neighbors.add(clone(neighbor));
        }
        return clone;
    }
}
```

written by mohamed+ebrahim original link here

## Solution 2

The solution is same as https://oj.leetcode.com/discuss/22244/simple-c-solution-using-dfs-and-recursion I just make it shorter;

```
/**
*  author : s2003zy
*  weibo  : http://weibo.com/songzy982
*  blog   : s2003zy.com
*  date   : 2015.02.27
*/
class Solution {
public:
    unordered_map<UndirectedGraphNode*, UndirectedGraphNode*> hash;
    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
       if (!node) return node;
       if(hash.find(node) == hash.end()) {
           hash[node] = new UndirectedGraphNode(node -> label);
           for (auto x : node -> neighbors) {
                (hash[node] -> neighbors).push_back( cloneGraph(x) );
           }
       }
       return hash[node];
    }
};
```

written by s2003zy original link here

## Solution 3

Use HashMap to look up nodes and add connection to them while performing BFS.

```
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null) return null;

        UndirectedGraphNode newNode = new UndirectedGraphNode(node.label); //new node for return
        HashMap<Integer, UndirectedGraphNode> map = new HashMap(); //store visited nodes

        map.put(newNode.label, newNode); //add first node to HashMap

        LinkedList<UndirectedGraphNode> queue = new LinkedList(); //to store **original** nodes need to be visited
        queue.add(node); //add first **original** node to queue

        while (!queue.isEmpty()) { //if more nodes need to be visited
            UndirectedGraphNode n = queue.pop(); //search first node in the queue
            for (UndirectedGraphNode neighbor : n.neighbors) {
                if (!map.containsKey(neighbor.label)) { //add to map and queue if this node hasn't been searched before
                    map.put(neighbor.label, new UndirectedGraphNode(neighbor.label));
                    queue.add(neighbor);
                }
                map.get(n.label).neighbors.add(map.get(neighbor.label)); //add neighbor to new created nodes
            }
        }

        return newNode;
    }
}
```

written by shu3 original link here

From Leetcoder.