BFS of graph

Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0.

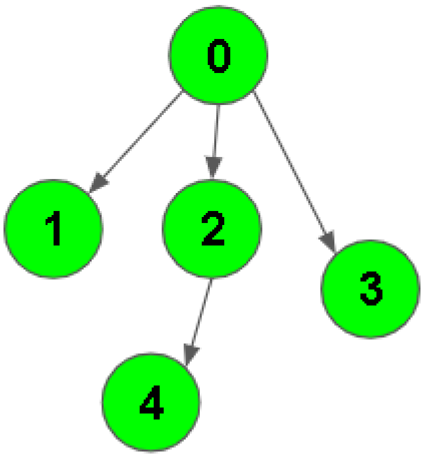
Note: One can move from node u to node v only if there's an edge from u to v. Find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the input graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration.

Example 1:

Input:

V = 5, E = 4

adj = {{1,2,3},{},{4},{},{}}



Output:

0 1 2 3 4

Explanation:

0 is connected to 1 , 2 , 3.

2 is connected to 4.

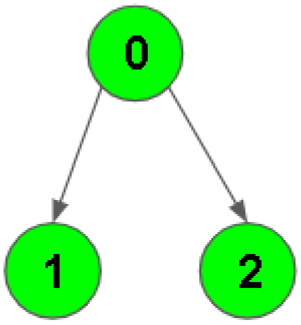
so starting from 0, it will go to 1 then 2

then 3. After this 2 to 4, thus bfs will be

0 1 2 3 4.

Example 2:

Input:
V = 3, E = 2
adj = {{1,2},{},{}}



Output:
0 1 2

Explanation:

0 is connected to 1 , 2.

so starting from 0, it will go to 1 then 2,

thus bfs will be 0 1 2.

Your task:
You dont need to read input or print anything. Your task is to complete the function **bfsOfGraph()** which takes the integer V denoting the number of vertices and adjacency list as input parameters and returns a list containing the BFS traversal of the graph starting from the 0th vertex from left to right.

Expected Time Complexity: O(V + E)
Expected Auxiliary Space: O(V)

Constraints:
$1 \leq V, E \leq 10^4$

```
from typing import List
from queue import Queue

class Solution:
    def bfsOfGraph(self, V: int, adj: List[List[int]]) ->
List[int]:
        visited = [False] * V
        bfs_traversal = []
        q = Queue()
        q.put(0)
        visited[0] = True

        while not q.empty():
            node = q.get()
            bfs_traversal.append(node)

            for neighbor in adj[node]:
                if not visited[neighbor]:
                    q.put(neighbor)
                    visited[neighbor] = True

        return bfs_traversal
```