Solution Review: Check if Given Undirected Graph is a Tree or Not

This review provides a detailed analysis of the different ways to check if an undirected graph is a tree or not.

We'll cover the following

Solution: Recursion stack#

main.py

Graph.py

Stack.py

Queue.py

LinkedList.py

Node.py

```
from Graph import Graph
# We only need Graph for this Question!


def is_tree(g):
    # All vertices unvisited
    visited = [False] * g.vertices
```

```
 8
 9      # Check cycle using recursion stack
10      # Also mark nodes visited to check connectivity
11      if check_cycle(g, 0, visited, -1):
12          return False
13
14      # Check if all nodes we visited from the source (graph is connect
15      for i in range(len(visited)):
16          # Graph is not connected
17          if not visited[i]:
18              return False
19      # Not cycle and connected graph
20      return True
21
22
23  def check_cycle(g, node, visited, parent):
24      # Mark node as visited
25      visited[node] = True
26
27      # Pick adjacent node and run recursive DFS
28      adjacent = g.array[node].head_node
```

The logic for this problem is the same as Challenge 3 where you have to detect a cycle in the graph. We make a stack (not to be confused with the stack data structure) of vertices in `check_cycle()`. This stack grows recursively (line **31**). The only difference is that we keep track of the `parent` vertex since a backward link to the parent does not count as a cycle (undirected graph). If a cycle is found in the graph, `check_cycle` will return `True`.

At the end of our recursion, two things must be true if the graph is a tree:

- All elements of `visited` must be true
- `check_cycle` should return `False`

Whenever these two conditions are true, our graph is a tree!

Time complexity#

The graph is traversed in both functions. Hence, the time complexity is $O(V + E)$.