Solution Review: Detect Cycle in a Directed Graph

This review provides a detailed analysis of the different ways to solve a cycle in a graph challenge.

We'll cover the following

Solution: Recursion stack

main.py

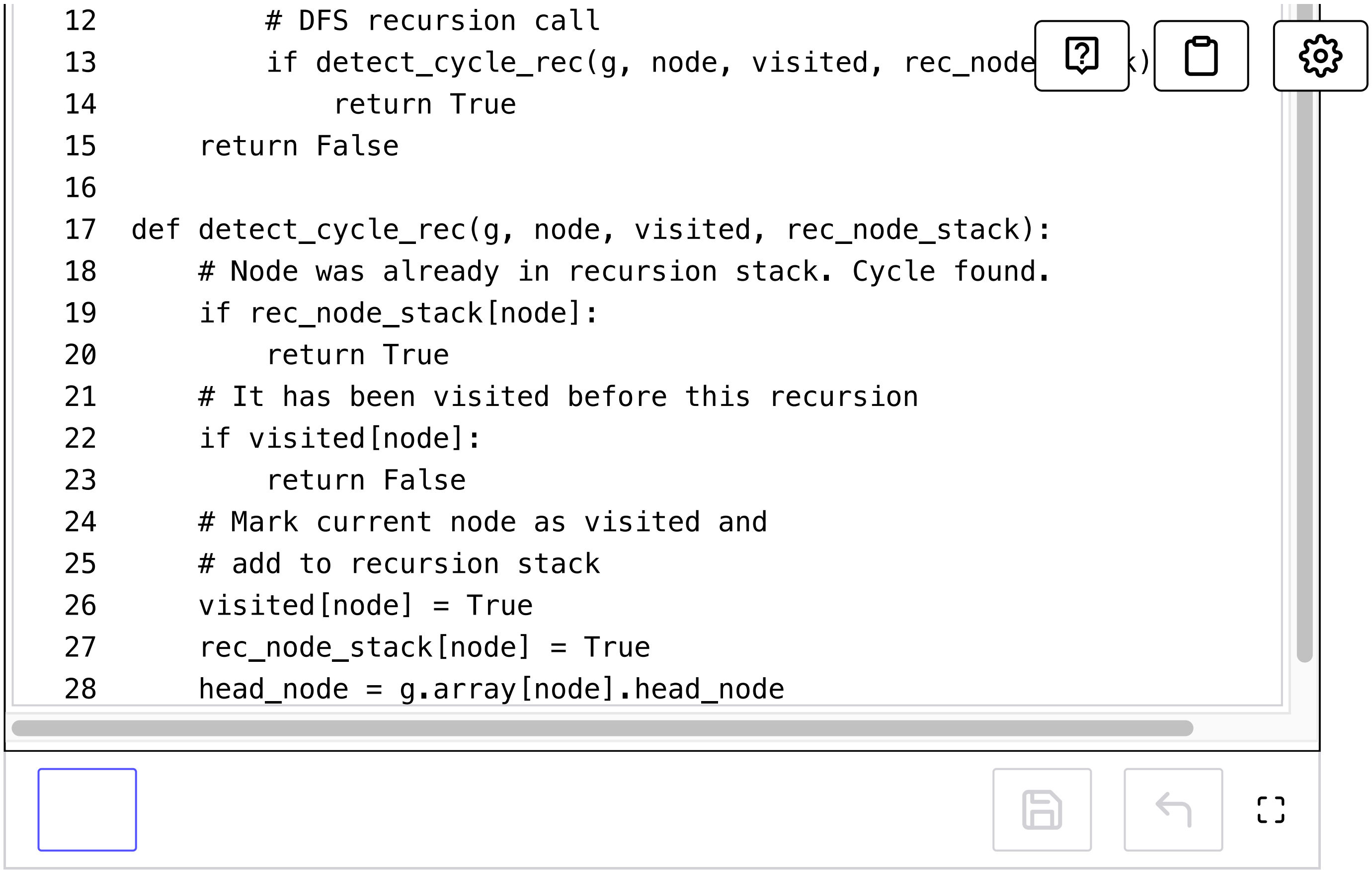
Graph.py

LinkedList.py

Node.py

```
from Graph import Graph
# We only need Graph and Stack for this Challenge!

def detect_cycle(g):
    # visited list to keep track of the nodes that have been visited
    # since the beginning of the algorithm
    visited = [False] * g.vertices
    # rec_node_stack keeps track of the nodes which are part of
    # the current recursive call
    rec_node_stack = [False] * g.vertices
    for node in range(g.vertices):
```



```
        # DFS recursion call
        if detect_cycle_rec(g, node, visited, rec_node k):
            return True
    return False

def detect_cycle_rec(g, node, visited, rec_node_stack):
    # Node was already in recursion stack. Cycle found.
    if rec_node_stack[node]:
        return True
    # It has been visited before this recursion
    if visited[node]:
        return False
    # Mark current node as visited and
    # add to recursion stack
    visited[node] = True
    rec_node_stack[node] = True
    head_node = g.array[node].head_node
```

The solution might look confusing at first, but the logic behind it is pretty straight forward.

For each node, we start a recursive call with `detect_cycle_rec`. The function maintains a stack (not to be confused with the stack data structure) of nodes called `rec_node_stack` along with a `visited` list.

The vertices that have been traversed in the current recursion are added to `rec_node_stack` and `visited` keeps a record of all the nodes that have been traversed regardless of the recursive call.

For a cycle to occur, we must reach a node that was already present in the recursion stack. If the recursion ends and no such node is found, the stack is reset again and the next iteration of `detect_cycle` starts.

Time complexity

O(V+E), which we already know is the complexity of traversi e a er list that represents our graph.

← Back

Challenge 3: Detect Cycle in a Directe...

Next →

Challenge 4: Find a "Mother Vertex" in...

Report an Issue