Solution Review: Implement Depth First Search

This review provides a detailed analysis of the different ways to solve the breadth first search challenge.

We'll cover the following

Solution: Using stacks#

main.py

Graph.py

Stack.py

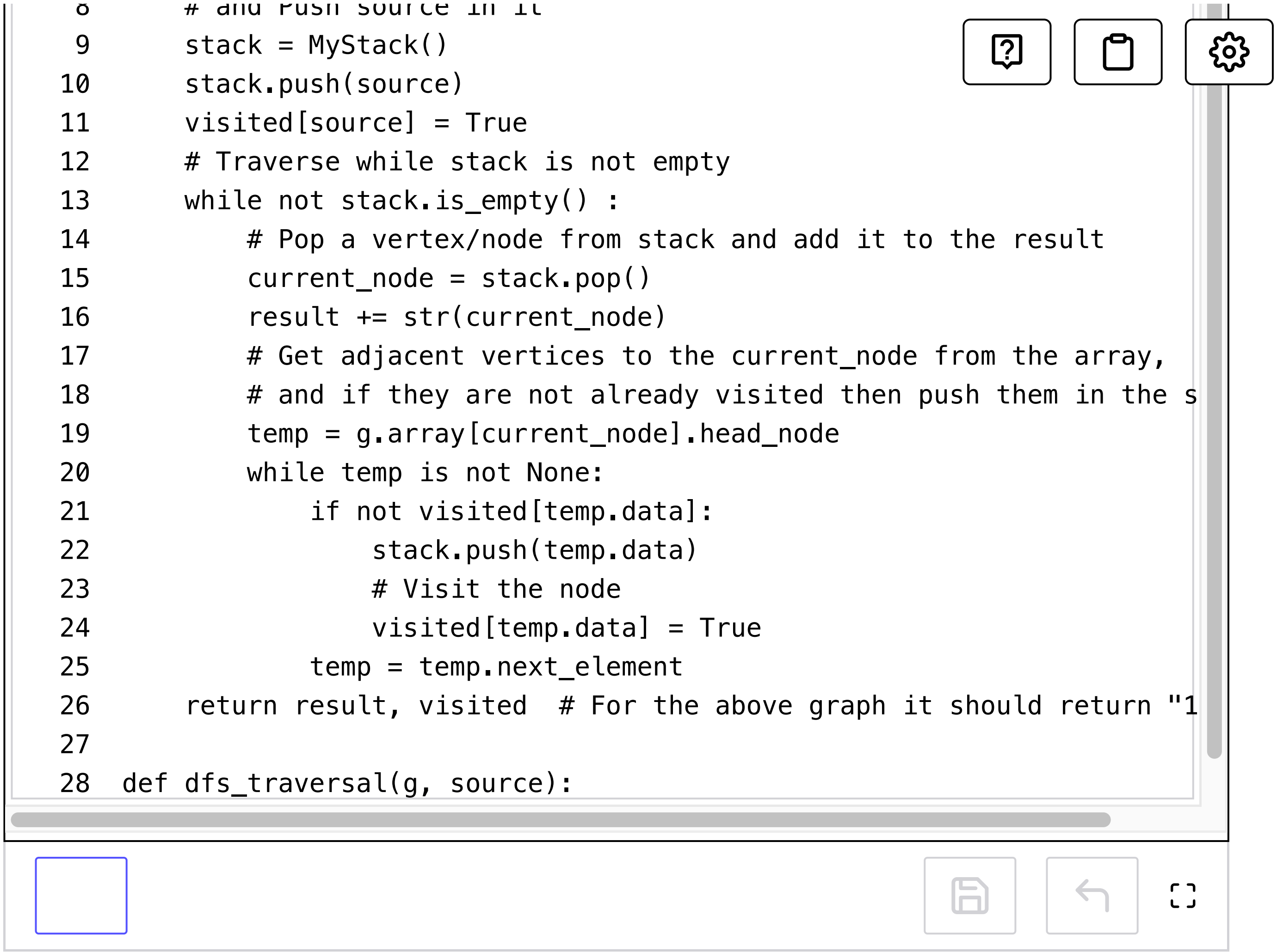
Queue.py

LinkedList.py

Node.py

```
from Graph import Graph
from Stack import MyStack
# You can check the input graph in console tab

def dfs_traversal_helper(g, source, visited):
    result = ""
    # Create Stack(Implemented in previous lesson) for Depth First Tr
    # and Push source in it
```



```
 8      # and Push source in it
 9      stack = MyStack()
10      stack.push(source)
11      visited[source] = True
12      # Traverse while stack is not empty
13      while not stack.is_empty() :
14          # Pop a vertex/node from stack and add it to the result
15          current_node = stack.pop()
16          result += str(current_node)
17          # Get adjacent vertices to the current_node from the array,
18          # and if they are not already visited then push them in the s
19          temp = g.array[current_node].head_node
20          while temp is not None:
21              if not visited[temp.data]:
22                  stack.push(temp.data)
23                  # Visit the node
24                  visited[temp.data] = True
25              temp = temp.next_element
26      return result, visited  # For the above graph it should return "1
27
28  def dfs_traversal(g, source):
```

The approach is very similar to that of the BFS solution. However, instead of a queue, we use a stack since it follows the **Last In First Out** (LIFO) approach (line **9**). We will see how that is useful here.

`dfs_traversal` calls the helper function `dfs_traversal_helper` on every vertex which is not visited. Starting from `source` which is `1`, each node is pushed into the stack (line **10**). Whenever a node is popped (line **15**), it is added to the `result`. Nodes are marked visited (line **24**) whenever they are pushed. Now we can understand why we need the stack because it keeps popping out the new adjacent nodes (gives you a node at a new **level**) instead of returning the previous nodes that we pushed in.

Time complexity#

Like the BFS, this algorithm traverses the whole list once. He t’s complexity is $O(V + E)$

← Back

Challenge 2: Implement Depth First Se...

Next →

Challenge 3: Detect Cycle in a Directe...

☑ Mark as Completed

Report an Issue