Solution Review: Find the Shortest Path Between Two Vertices

This review provides a detailed analysis to solve the shortest path between two vertices challenge.

We'll cover the following

Solution: BFS queue

main.py

Graph.py

Stack.py

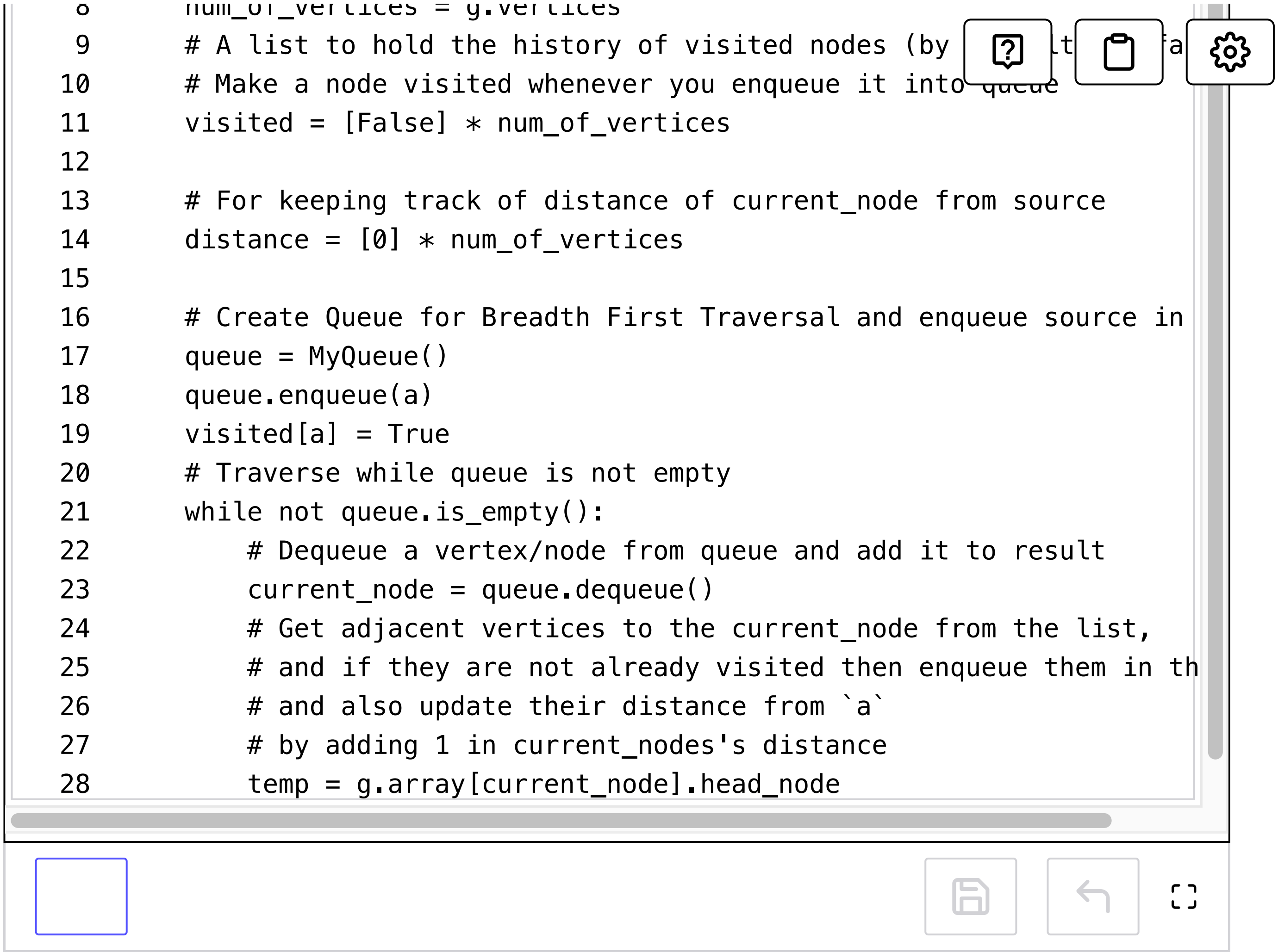
Queue.py

LinkedList.py

Node.py

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


def find_min(g, a, b):
    result = 0
    num_of_vertices = g.vertices
```



```
    num_of_vertices = g.vertices
    # A list to hold the history of visited nodes (by          lt          a
    # Make a node visited whenever you enqueue it into queue
    visited = [False] * num_of_vertices

    # For keeping track of distance of current_node from source
    distance = [0] * num_of_vertices

    # Create Queue for Breadth First Traversal and enqueue source in
    queue = MyQueue()
    queue.enqueue(a)
    visited[a] = True
    # Traverse while queue is not empty
    while not queue.is_empty():
        # Dequeue a vertex/node from queue and add it to result
        current_node = queue.dequeue()
        # Get adjacent vertices to the current_node from the list,
        # and if they are not already visited then enqueue them in th
        # and also update their distance from `a`
        # by adding 1 in current_nodes's distance
        temp = g.array[current_node].head_node
```

Once again, breadth first search comes to the rescue. The `visited` list must be familiar to you by now. The crux of this algorithm, however, lies in the `distance` list. For each node, the indexed value in `distance` shows the node's distance from `a` in terms of the number of edges where `a` is the source node.

The rest is a simple BFS traversal where the distance is incremented by 1 each time a node is visited.

We are guaranteed to find the shortest distance to `b` (destination node) because once it has been visited it won't be visited again through the longer path as it has already been marked.

Time complexity

The algorithm will have the same time complexity as BFS: *O*( ). v since we stop it as soon as we find `b`, it won't go through the whole list in the average case.

← Back

Challenge 8: Find the Shortest Path B...

Next →

Challenge 9: Remove Edge

☑ Mark as Completed

Report an Issue