Solution Review: Count the Number of Edges in an Undirected Graph

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

We'll cover the following

Solution #1: Iteration#

main.py

Graph.py

Stack.py

Queue.py

LinkedList.py

Node.py

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


```

```
def num_edges(g):
    # For undirected graph, just sum up the size of
    # all the adjacency lists for each vertex
    sum = 0
    for i in range(g.vertices):
        temp = g.array[i].head_node
        while temp is not None:
            sum += 1
            temp = temp.next_element

    # Half the total sum as it is an undirected graph
    return sum//2

if __name__ == "__main__" :

    g = Graph(9)
    g.add_edge(0, 2)
    g.add_edge(0, 5)
    g.add_edge(2, 3)
    g.add_edge(2, 4)
    g.add_edge(5, 3)
    g.add_edge(5, 6)
    g.add_edge(3, 6)
    g.add_edge(6, 7)
```

Nothing too tricky going on here. We simply traverse through the complete adjacency list and count the size of each linked list. In an undirected graph, the number of edges is always even as the edges are bidirectional. Hence, to get the number of bidirectional edges, we half the total `sum`.

Time complexity#

O(V + E)

Solution #2:#

main.py

Graph.py

Stack.py

Queue.py

LinkedList.py

Node.py

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


def num_edges(g):
    # For undirected graph, just sum up the size of
    # all the adjacency lists for each vertex
    return sum([g.array[i].length() for i in range(g.vertices)]) // 2

if __name__ == "__main__" :

    g = Graph(9)
    g.add_edge(0, 2)
    g.add_edge(0, 5)
    g.add_edge(2, 3)
    g.add_edge(2, 4)
    g.add_edge(5, 3)
    g.add_edge(5, 6)
    g.add_edge(3, 6)
    g.add_edge(6, 7)
    g.add_edge(6, 8)
    g.add_edge(6, 4)
    g.add_edge(7, 8)

    g2 = Graph(7)
    g2.add_edge(1, 2)
    g2.add_edge(1, 3)
    g2.add_edge(3, 4)
    g2.add_edge(3, 5)
    g2.add_edge(2, 5)
    g2.add_edge(2, 4)
    g2.add_edge(4, 6)
    g2.add_edge(4, 5)
    g2.add_edge(6, 5)

    print(num_edges(g))

    print(num_edges(g2))
```

Nothing too tricky going on here. It is just a compact version of writing the code. We are using the length function to get the size and we half the total `sum`.

Time complexity#

O(V + E)

Back

Challenge 5: Count Number of Edges i...

Next

Challenge 6: Check if a Path Exists Bet...

Mark as Completed

Report an Issue