TASK B: Modified BFS to Solve Six Degree of Separation

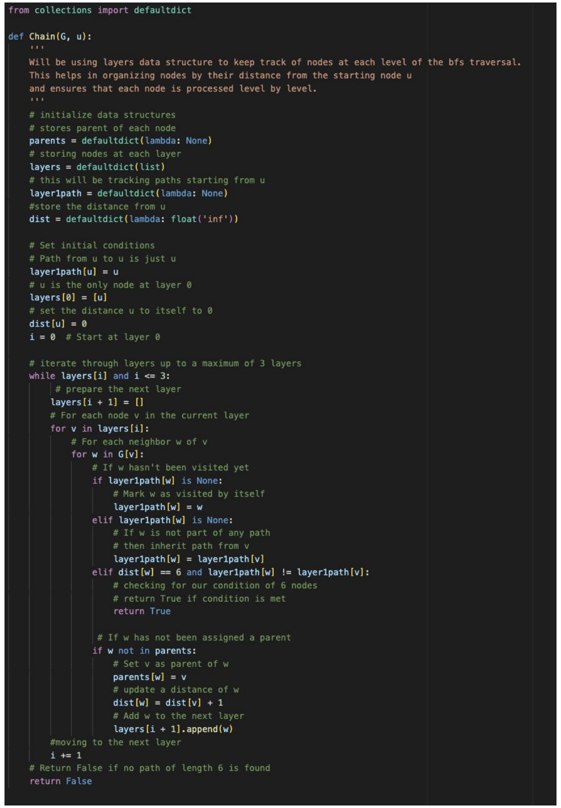
- Yu Ham Ng

Background Information

“Six degrees of separation is the theory that any person on the planet can be connected to any other person on the planet through a chain of acquaintances that has no more than five intermediaries. The concept of six degrees of separation is often represented by a graph database, a type of NoSQL database that uses graph theory to store, map and query relationships. Real-world applications of the theory include power grid mapping and analysis, disease transmission mapping and analysis, computer circuitry design and search engine ranking.” - TechTarget

I will be solving this question by BFS.

Brief Explanation of The Code



```python
from collections import defaultdict

def Chain(G, u):
    '''
    Will be using layers data structure to keep track of nodes at each level of the bfs traversal.
    This helps in organizing nodes by their distance from the starting node u
    and ensures that each node is processed level by level.
    '''
    # initialize data structures
    # stores parent of each node
    parents = defaultdict(lambda: None)
    # storing nodes at each layer
    layers = defaultdict(list)
    # this will be tracking paths starting from u
    layer1path = defaultdict(lambda: None)
    #store the distance from u
    dist = defaultdict(lambda: float('inf'))

    # Set initial conditions
    # Path from u to u is just u
    layer1path[u] = u
    # u is the only node at layer 0
    layers[0] = [u]
    # set the distance u to itself to 0
    dist[u] = 0
    i = 0  # Start at layer 0

    # iterate through layers up to a maximum of 3 layers
    while layers[i] and i <= 3:
         # prepare the next layer
        layers[i + 1] = []
        # For each node v in the current layer
        for v in layers[i]:
            # For each neighbor w of v
            for w in G[v]:
                # If w hasn't been visited yet
                if layer1path[w] is None:
                    # Mark w as visited by itself
                    layer1path[w] = w
                elif layer1path[w] is None:
                    # If w is not part of any path
                    # then inherit path from v
                    layer1path[w] = layer1path[v]
                elif dist[w] == 6 and layer1path[w] != layer1path[v]:
                    # checking for our condition of 6 nodes
                    # return True if condition is met
                    return True

                 # If w has not been assigned a parent
                if w not in parents:
                    # Set v as parent of w
                    parents[w] = v
                    # update a distance of w
                    dist[w] = dist[v] + 1
                    # Add w to the next layer
                    layers[i + 1].append(w)
        #moving to the next layer
        i += 1
    # Return False if no path of length 6 is found
    return False
```

The algorithm uses breadth-first search (BFS) principles, expanding outwards from the starting node u, and tracking vertices and their distances from u. When the graph G consists of only the vertex u (or a set of vertices with no connections to u within 3 layers), the algorithm returns False, correctly indicating that there is no qualifying path meeting the specified condition (distance of 6 from u with distinct layer1paths).If v is within a distance of 6 from u and has a different initial path (layer1path) compared to its parent vertex, the algorithm returns True, correctly identifying the specific path condition. This logic follows for each vertex added to the graph.

Detailed Explanation of The Code

The algorithm design begins with the initialization phase where a BFS is started from the starting node, denoted as u . A dictionary or list (dist) is used to keep track of the distance from u to each node. This is initialized to infinity for all nodes except u , which is initialized to 0. In the BFS execution phase, a queue is used to manage the nodes to explore, initially containing only u . For each dequeued node v , all its neighbors are explored. For each neighbor w , if it has not been visited (for example, if dist[w] is infinity), its distance is updated (dist[w] = dist[v] + 1) and w is enqueued. The BFS is stopped once the distance exceeds 6, as we are only interested in chains of at most 6 friends. In the chain checking phase, during the BFS, for each node v at a distance of 6, it is checked if there exists an edge between v and u . If such an edge exists, the algorithm returns True. If the BFS completes without finding such a chain, the algorithm returns False. Hence, it will find if someone is connected within a chain of 6 in this algorithm.

Time and Space Complexity

Space Complexity: O(|V | + |E|). The provided algorithm contain hash table of parents, layers, dist and layer1path which contribute O(v) spaces. The graph contributes O(E) spaces, hence the total space complexity of given code is O(|V | + |E|).

Time Complexity: O(|V |+|E|). The outer while loop will operate no more than 6 times in the worst-case scenario. The inner loop for v in layer[i] will iterate each node in layer[i]. Then the second inner loop for w in G[v] will track each neighbor of G[v]. Hence in the worst-case scenario, the given code will run all nodes and all edges from G, which contribute to the run time complexity of O(|V |+|E|), which strickly follows the rule O(n+m).