Advanced Algorithms and Data Structures

Assignment 3

Note: **V** represents the Vertices in the graph. **E** represents the Edges in the graph

Exercise 1 – The time complexity observed in the algorithm for the isAcyclic method is thought to be O(V+E). The DFS method has the complexity of O(V+E) and, even though it is called within a for loop, the overall time complexity of the whole method remains O(V+E) because the DFS is not run for every single vertex in the for loop. The logic of how this method works follows a DFS principle. Every vertex is examined and we check whether a cycle is possible to be formed by checking the recursion and the visited arrays. If the vertex has not been visited or is not in the recursion array, then it is added to both of them, and a recursive call of the method is done on the edges. Otherwise, if the recursion if condition is met, the method returns true because this implies that a cycle was, indeed, found. If the visited condition is met, the method returns false instead because this implies that this vertex has been previously visited though no cycle containing it was found, as can be seen by examining the recursion array if condition (which will have not been met).

The isConnected method is understood to have an O(V+E) because of the method structure. All loops within the method have a time complexity of O(V). There are three of such loops and the other important factor in determining the time complexity of the method relates to the visitCheck method, which is also inside the isConnected one. This method, in its turn, has a while loop which contains a recursive call to itself. The operations carried inside are in real time, thus only the loop itself contributes to the time complexity calculation. The combination of the while loop and the recursiveness represents the worst-case in terms of time complexity, putting it at O(E). Altogether, this means that the method itself has the time complexity of O(V+E).

In the case of the undirected graph, the time complexity does not vary for each methods since we continue to use DFS methods for the traversals.

Exercise 2

The hasEuelerPath method is understood to possess the following time complexities:

In the best-case scenario, just like the worst-case, the big oh is O(V+E). The first if condition calls the isConnected method, which has a time complexity of O(V+E) as demonstrated in the above paragraph. Other than the call, there is a single for loop, which runs with the complexity of O(V). All other operations are carried out in real-time. Thus, the total time complexity for this method equals the one for isConnected.

The other method, eulerPath, which returns a list of integers, is understood to have the running time complexity of $O((V+E)^2)$. The function addEulerVertex, which is called inside eulerPath, acts similarly to a DFS function and calls the nextEdge function, which acts as yet another DFS.

The time complexity of a DFS for adjacency list representation is O(V+E). In face of the fact that we are running two sequential DFS's, the time complexity is O((V+E)*(V+E)). This can be rewritten mathematically as $O((V+E)^2)$. In the case of a connected graph, the time complexity becomes $O(E^2)$ instead.

Exercise 3

The time complexity of the DFS in the numberOfPeopleAtFriendshipDistance method is O(E+V). This is due to the fact that the DFS method called inside it has a worst-case time complexity of O(E). The following for loop has a big oh of O(V) since it runs for the amount of times as vertices. This means that the method has a worst-case time complexity of O(V+E).

The second method, furthestDistanceInFriendshipRelationships, has a similar time complexity as the previous method mentioned. The logic is pretty similar due to the usage of the same DFS search method and the subsequent for loop, which runs V times, V being the amount of vertices. The consequence is, then, that the worst case time complexity of the whole method amounts to O(V+E).

The third method, named possibleFriends returns another method called DFSPossible. This method starts by calling the method hasCycle, which has the time complexity of O(E). A for loop is, then, initiated and runs for V times, V being the amount of vertices. Two for loops then run subsequently for the amount of vertices that are situated at a distance of two edges from the index vertex. Finally, the last for loop called runs for the amount of vertices that are at a distance of 2 from the index vertex and have mutual friends. The worst-case for this situation is that the time complexity amounts to O(V+E).