Problem analysis

The proposed problem can be reduced to calculate the maximum island size in an undirected Graph. While the data provided is in the form of directed graph, the nodes that are not bi-directional can be removed as they cannot be included in any solution.

The biggest difficulty of this problem is to keep the complexity of the solution to O(n * m) where n is the number of players (nodes) and m is the maximum number of other players an individual player can see (maximum number of edges). This limitation (in execution time) means that we cannot use either just connectivity matrix or just linked list, because the operations using only one of them would be O(n^2). Thus, both representations are used at the same time.

Proposed solution

First, we read the file and create both the connectivity matrix and a list of all the connections defined in the file. This process happens in the function *create_graph* in *utils.py*.

Then, for each defined connection (this is setting the complexity to O(n*m), so all the following operations must be O(1)), we check if the connection is bi-directional [line 66 in *utils.py*]. This is done using the connectivity matrix with complexity O(1), if we were to use the linked list, would be of complexity O(m), which would increase the complexity of the solution. If is a bi-directional connection, it means that is a connection to be considered. Then, we check if any of the nodes belong already to an island (O(1)) [line 67 in *utils.py*]. If one of them already belongs to one, the other is added to that island. Otherwise, we create a new island using as an index, the lowest node number. [lines 68-79 in *utils.py*].

If instead, we were to use only the connectivity matrix to search for the connections, it would be O(n^2) since we would need to check all the upper / lower half matrix.

Since we are visiting the nodes in order, i.e. $\nexists s_i, s_j \mid s_j < s_i$ where s_i and s_j correspond to the source nodes in a two different connections, and s_i appears before in the file than s_j, we can be sure that the islands will be created properly. This behaviour is ensured by the structure of the file, where each player is defined in an individual row.

While we assign the island number to each node, we keep track of the size of each island (increasing it or decreasing it). This has to be done at the same time, because if we were only to calculate the islands and, then, calculate the size of each island, that calculation would be O(n^2).

While this solution has a higher memory requirement than other possible ones (we are saving redundant information) it ensures that the complexity of the solution is O(n*m).

How to run

To run this program is necessary to have installed Python 3 with the numpy library. To run the tests, the pytest library is also needed. To run the program, in a console type:

```
python main.py path_to_file
```

Where path_to_file is the file where the players are defined.

The tests can be executed by running the command:

```
python -m pytest
```