Space and Time complexity analysis

For every function used, we provide an analysis of complexity:

Consider first :

Y_r = 143 years (the range of years)

n= total number of Players

m= total number of Teams

- Reader: *downloads and parses data from the url*

We cross the data for every year (loop), so the time complexity is O(Y_r*nb_players*nb_teams).

With our notations:

-Spacial complexity= $O(n*m)$

-Time complexity= $O(n*m)$

- playerTeams: *crosses the dataframe: provides the teams of each player*

We have a single loop, and we create a dictionary:

-Spacial complexity= $O(n)$

-Time complexity= $O(n)$

- Cleaner: *puts the teams of a player into a list with no repetition*

We have a single loop, and we simply modify the dictionary.

-Spacial complexity= $O(n)$

-Time complexity= $O(n)$

- C_{mp}: *calculates p-uplets among m elements and returns a list of p-uplets*

The complexity in time of this recursive program can be written :

$$C(m)= 1 + 2C(m-1) \text{ so } O(2^m)$$

This program also generates C_{mp} lists of p elements :

the spatial complexity is, after simplifications and taking p=3 (for the problem given) : $p*C_{mp} = O(m^{(m-3)})$.

- Calculator: *calculates the number of players that have played for any triple of teams*

There is a loop over the n players, and for each we apply C_{mp} .

- Time complexity=$O(n*2^m)$

- Space complexity= $O(m^{(m-3)})$

C_{mp} is the most costly function in our program. However, having a list of teams a player has played for, it’s compulsory to consider each triplet and add 1 to its count. This fact implies taking every « 3 teams among m ». The recursive way is efficient for doing it.