Análisis de Jugadores FIFA

Estudiante 1
Nombre: Pablo Martinez
Código : 202122937
Correo: p.martinezm2@uniandes.edu.co
Requerimiento: 3

Estudiante 2
Nombre: Daniel pedroza
Código : 202123283
Correo: d.pedroza@uniandes.edu.co
Requerimiento: 4

Análisis de complejidad

Requerimiento 1(G); Para el primer requerimiento tenemos una complejidad de log(2 log x log (n)) esto se basa porque se utiliza un contains y un get en un árbol binario balanceado como lo es un RBT aparte de esto utilizamos un merge sort al final de la función para organizar los datos por esto el resultado resulta en log(2log(n) x nlog(n)).

```
# Requerimiento 1----------------------------------------------------------------------------------------------------


def getRecentPlayersByClub(catalog, club):
    club_index = catalog["club_index"]

    if om.contains(club_index, club):
        player_list = om.get(club_index, club)["value"]

        return organizarDate(player_list)

    else:
        return "No existe ese club ( ´･･)/(._.`)"
```

Requerimiento 2(G): Para el segundo requerimiento empezamos con una complejidad de: 2log(n) donde n es la cantidad en el postion index basados en el get() y el contains() en RBTs que resultan en una suma de log n + log n= 2log n, despues sumamos el resultado previo a la cantidad de objetos en el position index (esta variable la llamaremos P), podemos notar que a nuestra variable p se le realiza un merge sort con complejidad e Plog P y se le suma a la lista.Justo ahorita la complejidad resultaria en O(2*log(n) + P*log(P) + P) durante este proceso se ve la mayoria de la complejidad temporal del algoritmo, por lo cual podriamos llamar este resultado como la complejidad temporal aproximada. Pero despues se realizan las operaciones en las listas. Empezando con la lista_1, a la que le asignaremos la variable L1 que seria la cantidad de elementos que salidos del recorrido P, similarmente como se le hace un merge sort y se le suma a la lista_dos, se puede ver una complejidad de L1log L1 + L1. Este proceso se repite para la lista_2 con la varibale L2 que se refiere a los elementos salidos de L1, y por ultimo se vuelve repetir con la lista_final que tengra variable L3 donde L3 son los elementos salidos de L2. Finalmente la complejidad resulta en:
O(2*log(n) + P*log(P) + P + L1*log(L1) + L1 + L2*log(L2) + L2 + L3*log(L3))

```python
# Requerimiento 2----------------------------------------------------------------------------------------


def getPlayersByPositionPotentialWage(
    catalog,
    position,
    low_overall,
    high_overall,
    low_potential,
    high_potential,
    low_wage,
    high_wage,
):
    position_index = catalog["position_index"]

    if om.contains(position_index, position):
        player_list = om.get(position_index, position)["value"]

        player_list = organizarOverall(player_list)

        list_uno = lt.newList("ARRAY_LIST")
        for player in lt.iterator(player_list):
            overall = player["overall"]
            if low_overall <= overall and overall <= high_overall:
                lt.addLast(list_uno, player)
        organizarPotential(list_uno)

        list_dos = lt.newList("ARRAY_LIST")
        for player in lt.iterator(list_uno):
            potential = player["potential"]
            if low_potential <= potential and potential <= high_potential:
                lt.addLast(list_dos, player)
        organizarWage(list_dos)

        list_final = lt.newList("ARRAY_LIST")
        for player in lt.iterator(list_dos):
            wage = player["wage_eur"]
            if low_wage <= wage and wage <= high_wage:
                lt.addLast(list_final, player)

        if lt.size(list_final) == 0:
            return "No hay jugadores en esa posicion con esas especificaciones"
        else:

            return organizarOverallPotentialWage(list_final)

    else:
        return "Esa posicion no existe ಥ_ಥ"
```

Requerimiento 3(I): El requerimiento 3 es muy similar al segundo al tener un algoritmo muy similar pero con unos cambios pequeños que resultan en una menor complejidad temporal. Como el anterior el get() y el contains(), generan una complejidad de 2Log(N) donde N es la cantidad de elementos en el position index, despues como el anterior la variable P se relaciona con el for y resulta de los elementos encontrados en el position index, se resulta una suma de O(2Log N + P). Finalmente se vuelven a contar las complejidades en las listas que tienen variables L1 y L2, donde L1 son los elementos encontrados en P y L2 son los elementos encontrados en L1. la suma de complejidad de estos dos seria entonces de L1 + L2Log L2 + L2, por lo cual el resultado final de la complejidad seria de:
O(2Log N + P + L1 + L2Log L2 + L2)

```python
# Requerimiento 3----------------------------------------------------------------------------------------

def getPlayersByTagWage(catalog, player_tag, low_wage, high_wage):
    tag_index = catalog["tag_index"]

    if om.contains(tag_index, player_tag):
        player_list = om.get(tag_index, player_tag)["value"]

        list_uno = lt.newList("ARRAY_LIST")
        for player in lt.iterator(player_list):
            tags = player["player_tags"]
            for tag in tags:
                if player_tag == tag:
                    lt.addLast(list_uno, player)

        list_dos = lt.newList("ARRAY_LIST")
        for player in lt.iterator(list_uno):
            wage = player["wage_eur"]
            if low_wage <= wage and wage <= high_wage:
                lt.addLast(list_dos, player)
        if lt.size(list_dos) == 0:
            return "No hay jugadores con esta especificacion"
        else:
            return organizarWage(list_dos)
    else:
        return "Este player_tag no existe ¤_¤"
```

Requerimiento 4(I): Curiosamente el algoritomo implementado para el requerimiento 3 y el requerimiento 4 es exactamente el mismo, por lo cual se puede utilizar el proceso anterior para ubicar la complejida temporal, que seria entonces, donde N es la cantidad de elementos en el position index, P es la cantidad de elementos encontrados en el position index, L1 es la cantidad de elementos encontrados en P y L2 es la cantidad de elementos encontrados en L1:
O(2Log N + P + L1 + L2Log L2 + L2)

```python
# Requerimiento 4----------------------------------------------------------------------------------------

def getPlayerByTraitsDob(catalog, player_traits, low_dob, high_dob):
    low_dob = datetime.strptime(low_dob, "%Y-%m-%d").date()
    high_dob = datetime.strptime(high_dob, "%Y-%m-%d").date()
    trait_index = catalog["trait_index"]

    if om.contains(trait_index, player_traits):
        player_list = om.get(trait_index, player_traits)["value"]

        list_uno = lt.newList("ARRAY_LIST")
        for player in lt.iterator(player_list):
            traits = player["player_traits"]
            for trait in traits:
                if player_traits == trait:
                    lt.addLast(list_uno, player)
        list_dos = lt.newList("ARRAY_LIST")
        for player in lt.iterator(list_uno):
            dob = player["dob"]
            if low_dob <= dob and dob <= high_dob:
                lt.addLast(list_dos, player)
        if lt.size(list_dos) == 0:
            return "No hay jugadores con esta especificaion"
        else:
            return organizarDob(list_dos)
    else:
        return "Ese trait no existe"
```

Requerimiento 5(G): Para el ultimo requerimiento podemos empezar los calculos de complejidad temporal con el contains() y el ger() pero al ser un mapa tiene una complejidad de O(1) por esto no se considera. Las operaciones minKey() y maxKey() tienen una complejidad de O(Log N) pero al ser dos la complejidad seria de O(2Log N) donde N es el numero de diferentes valores indicada con el low_key y big_key, despues se requiere sumar la cantidad segmentos a la complejidad con su propiedad(con variables S para la cantidad de segmentos y P para la propiedad elegida) que generan una complejidad de O(S*P+P) por lo cual la complejidad temporal final seria:
O(2Log N + S*P+P)

```python
# Requerimiento 5----------------------------------------------------------------------------------------


def getCantidadJugadoresPorPropiedadRango(catalog, propiedad, segmentos_rango, niveles):

    if mp.contains(catalog["player_info_index"], propiedad):
        propiedad_index = mp.get(catalog["player_info_index"], propiedad)["value"]
        low_key = om.minKey(propiedad_index)
        big_key = om.maxKey(propiedad_index)
        # math
        diff = (big_key - low_key) / segmentos_rango
        respuesta = lt.newList("ARRAY_LIST")
        low_key_solo = low_key - 0.004

        low_range = low_key_solo
        big_range = (low_key + diff) + 0.0004
        total = 0

        for _ in range(segmentos_rango):
            players_lists = om.values(
                propiedad_index, math.ceil(low_range), math.floor(big_range)
            )
            count = 0
            info_bin = {}
            for player in lt.iterator(players_lists):
                count += lt.size(player)

            lv = count // niveles
            info_bin = {
                "bin": (
                    "("
                    + str(round(low_range, 3))
                    + ", "
                    + str(round(big_range, 3))
                    + "]"
                ),
                "count": count,
                "lvl": lv,
                "mark": lv * "*",
            }
            lt.addLast(respuesta, info_bin)
            low_range = big_range
            big_range = low_range + diff
            total += count

        return respuesta, total
    else:
        return "Esa propiedad no existe"
```