ANÁLISIS DEL RETO

Lina Muñoz Martínez-202310172 -lm.munoz23@uniandes.edu.co
Daniel Bolívar Bernal -202310329 -d.bolivarb@uniandes.edu.co
Tomás Velásquez -202311016 -t.velasquezd@uniandes.edu.co

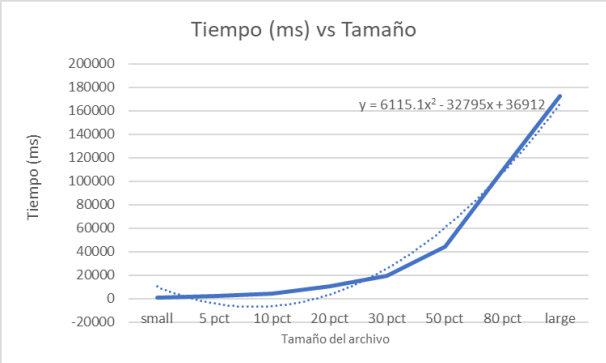
Carga de datos

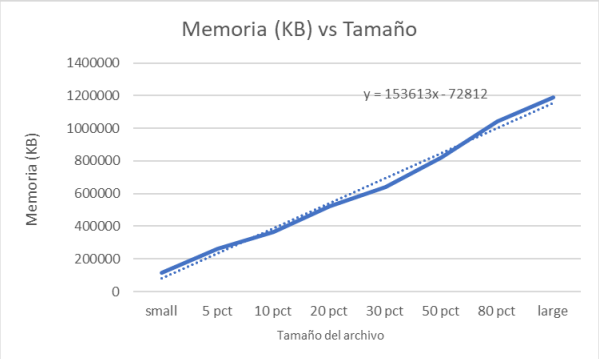
Para la carga de datos se cargó Results, Goalscorers y Shootouts como listas (ARRAY_LIST), y a su vez se crearon 7 mapas con los índices "scorer"," mapResults", "equipos", "anio_torneo", "goals", "torneo_goles" y "equipo_scorer":

```
53
54  def new_data_structs():
55      """
56      Inicializa las estructuras de datos del modelo. Las crea de
57      manera vacía para posteriormente almacenar la información.
58      """
59      catalog = {
60          'goalscorers': None,
61          'results': None,
62          'shootouts': None,
63          "scorer": None,
64          'mapResults':None,
65          'equipos':None,
66          "anio_torneo":None,
67          'goals':None,
68          'torneos_goles': None,
69          "equipo_scorer":None
70      }
71
72      catalog['goalscorers'] = lt.newList("ARRAY_LIST")
73      catalog['results'] = lt.newList("ARRAY_LIST")
74      catalog['shootouts'] = lt.newList("ARRAY_LIST")
75      catalog["scorer"] = mp.newMap(13000, prime=109345121, loadfactor=4, maptype="CHAINING")
76      catalog['mapResults'] = mp.newMap(40000, prime=109345121, loadfactor=4, maptype="CHAINING")
77      catalog['equipos'] = mp.newMap(300, prime=109345121, loadfactor=0.5, maptype="PROBING")
78      catalog["anio_torneo"] = mp.newMap(160, prime=109345121, loadfactor=0.5, maptype="PROBING")
79      catalog["goals"] = mp.newMap(10000, prime=109345121, loadfactor=4, maptype="CHAINING")
80      catalog["torneos_goles"] = mp.newMap(10000, prime=109345121, loadfactor=4, maptype="CHAINING")
81      catalog["equipo_scorer"] = mp.newMap(300, prime=109345121, loadfactor=0.5, maptype="PROBING")
82
83      return catalog
```

Tablas de datos:

Tamaño de datos	Tiempo (ms)	Memoria (KB)
small	583.663	113319.300
5 pct	1912.182	260593.747
10 pct	4027.421	366274.547
20 pct	10408.924	520361.172
30 pct	19425.144	638592.109
50 pct	44491.703	819398.307
80 pct	108853.801	1040593.704
large	172459.307	1188452.060





Requerimiento 1

```python
def req_1(data_structs, cantidad, nombre, condicion):
    """
    Función que soluciona el requerimiento 1: Listar los ultimos n partidos de un equipo segun condición.
    Inputs:
        data_structs = estructuras de datos
        nombre = nombre del jugador
        cantidad = cantidad de partidos que queremos que nos devuelva

    Devuelve:
        respuesta = lista con los partidos con la cantidad (n) que nos interesa.
        totalpartidos = lleva el conteo del total de partidos que cumplen las condiciones.
    """
    c = int(cantidad)
    totalcondicion=0
    totalequipos=mp.size(data_structs['equipos'])
    respuesta = lt.newList("ARRAY_LIST")
    nombre=nombre.title()
    condicion=condicion.lower()

    entry=mp.get(data_structs['equipos'],nombre)
    partidos=me.getValue(entry)
    totalpartidos=lt.size(partidos)

    for dic in lt.iterator(partidos):
        if condicion== "local":
            if dic["home_team"] == nombre:
                totalcondicion+=1
                if totalcondicion <= c:
                    lt.addLast(respuesta, dic.copy())
        elif condicion == "visitante":
            if dic["away_team"] == nombre:
                totalcondicion+=1
                if totalcondicion <= c:
                    lt.addLast(respuesta, dic.copy())
        else:
            if dic["away_team"] == nombre or dic["home_team"] == nombre:
                totalcondicion+=1
                if totalcondicion <= c:
                    lt.addLast(respuesta, dic.copy())


    return totalequipos,totalpartidos,totalcondicion,respuesta
```

Descripción

Este requerimiento se encarga de listar la cantidad n (entra por parámetro) de últimos partidos de un equipo según la condición (Local, Visitante o Indiferente), así como dar el total de equipos con info disponible, la cantidad de partidos en las que participo el equipo, y un total de la cantidad de partidos que cumplen la condición.

Para esto se obtiene el valor (lista de partidos del equipo sin importar condición) que le corresponde a la llave (equipo) a través del mapa con índice "equipos".

Luego se recorre esta lista, y por cada partido que cumpla las condiciones se le agrega 1 al contador, y se agrega a lista respuesta si no se ha alcanzado el límite de n partidos.

Entrada	Estructuras de datos del modelo, cantidad de partidos que se desea consultar, nombre del equipo, condición (Local, Visitante o Indiferente)
Salidas	Total de equipos, Total de partidos encontrados (int), Total de partidos que cumplen condición, lista de diccionarios con la cantidad de partidos que se desean consultar(list)
Implementado (Sí/No)	Si. Implementado por Lina Muñoz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso1: Definición de distintas variables	O (1)
pPaso 2: acceder al valor de una llave	O (1)
Paso 3: Recorrido de la lista de partidos que le corresponden al equipo n=cantidad de partidos que le corresponde al equipo	O(n)
Paso 4 (dentro del for): Añadir 1 al contador de condición por cada partido que encuentre, y añadir a la lista el dic si no ha alcanzado el c entrado por parámetro	O(n)
TOTAL	O(n)

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron cantidad de partidos a consultar: 10, nombre del equipo: Italy, Condición: Indiferente.

Procesadores	**AMD Ryzen 7 5800H with Radeon Graphics**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms) RETO 1**	**Tiempo (ms) RETO 2**
small	0.434	0.106
5 pct	1.509	0.169

10 pct	3.204	0.232
20 pct	5.951	0.297
30 pct	7.928	0.386
50 pct	11.371	0.495
80 pct	18.318	0.623
large	21.978	0.803

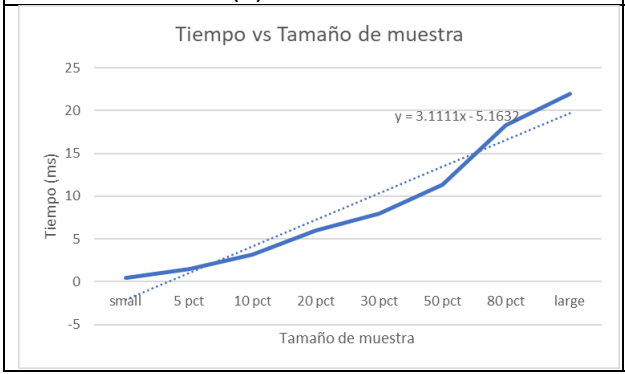
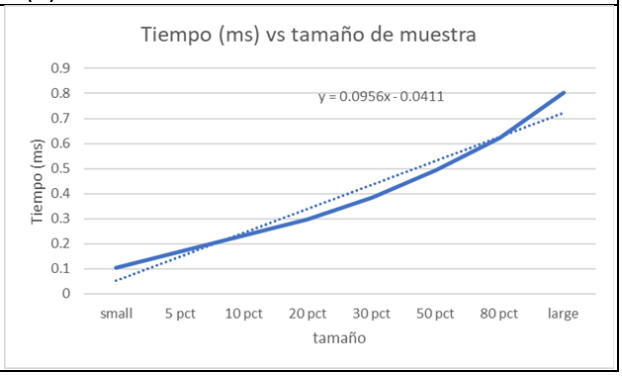
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Partidos totales encontrados	Tiempo (ms)
small	18	0.106
5 pct	45	0.169
10 pct	72	0.232
20 pct	126	0.297
30 pct	174	0.386
50 pct	266	0.495
80 pct	374	0.623
large	467	0.803

Graficas

Las gráficas con la representación de las pruebas realizadas.

RETO 1 COMPLEJIDAD= O(n)	RETO 2 O(n)
	

Análisis

Este requerimiento teóricamente es complejidad temporal O(n) debido a que hay un loop en el código, puesto que en la línea 347 existe un for que recorre la lista de los partidos que le corresponden al equipo consultado. Este comportamiento se puede evidenciar experimentalmente en la gráfica. Dado que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.

A pesar de que tanto en reto 1 como 2 la complejidad “es la misma”, la razón de que haya una enorme diferencia en los tiempos de ejecución es que en el reto 1 n representa la totalidad de partidos en Results, mientras que en este reto n representa la cantidad de partidos que le corresponde solo al equipo consultado gracias al uso de mapas.

Requerimiento 2

Descripción

```
def req_2(data_structs, name, cantidad):
    """
    Función que soluciona el requerimiento 2: Listar los primeros N goles anotados por un jugador (G).

    Inputs:
        data_structs = estructuras de datos
        name = nombre del jugador
        cantidad = cantidad de goles que queremos que nos devuelva

    Devuelve:
        respuesta = lista con los goles anotados por el la cantidad (n) que nos interesa
        num_partidos = lleva el conteo de goles que ha anotado el jugador (deberia ser num_goles pero aja)
    """
    c = int(cantidad)
    respuesta = lt.newList("ARRAY_LIST")
    name=name.title()
    entry=mp.get(data_structs['scorer'],name)
    goles=me.getValue(entry)
    num_goles=lt.size(goles)
    total__scorers=  lt.size(data_structs['scorer'])
    penalty=0

    for dic in lt.iterator(goles):
        if dic["penalty"] == "True":
            penalty+=1

    if c<num_goles:
        respuesta=lt.subList(goles,lt.size(goles)-c+1,c)
    else:
        respuesta=goles

    return respuesta , num_goles , total__scorers, penalty
```

Este requerimiento se encarga de listar los n(cantidad) goles anotados por un jugador(name)

Entrada	Estructuras de datos del modelo, el nombre del jugador(name) y la n(cantidad) cantidad de goles a consultar.
Salidas	Respuesta (Lista de los n goles), num_partidos (total de goles jugador), total_scorers y penalty
Implementado (Sí/No)	Si. Implementado por Tomás Velásquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
1. Asignación de variables, encontrar información del map y filtrar de manera adecuada los parametros entregados.	O(1)
2. Crear contador para penales	O(1)
3. For que itera dentro de la información de los valores encontrados en el map e ir sumando a penales	O(n)
4. Condicional para poder filtrar la n cantidad de goles que se entregaran	O(1)
5. Si la cantidad es menor al numero de goles entonces se realiza sub list para printear los n primeros, de lo contrario se printean todos	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron

Nombre: Lionel Messi

Cantidad: 6

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	MacOs Ventura

Entrada	Tiempo (ms) (Reto 1)	Tiempo (ms) (Reto 2)
small	1.904	0.067
5 pct	7.154	0.077
10 pct	14.393	0.082
20 pct	28.150	0.086
30 pct	41.611	0.095
50 pct	68.652	0.104
80 pct	109.710	0.118
large	136.334	0.23

Tablas de datos

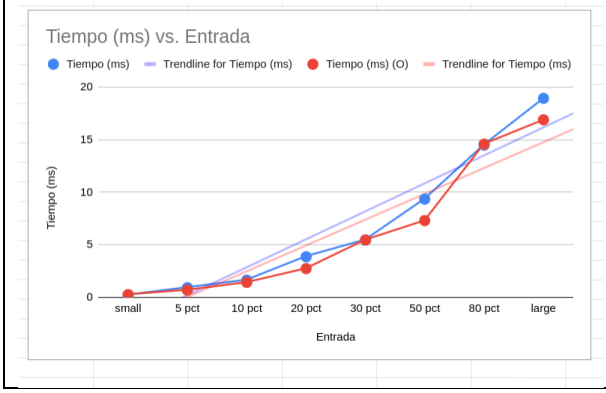
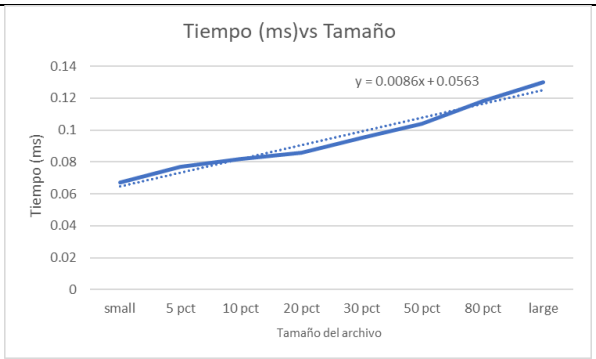
Las tablas con la recopilación de datos de las pruebas.

Muestra	Cantidad de goles	Tiempo (ms)
small	1	0.067
5 pct	3	0.077
10 pct	9	0.082
20 pct	11	0.086

30 pct	16	0.095
50 pct	19	0.104
80 pct	37	0.118
large	54	0.23

Graficas

Las gráficas con la representación de las pruebas realizadas.

Reto 1	Reto 2
	

Análisis

La complejidad temporal teórica de la función es O(n). Gracias a la gráfica en un principio podemos corroborar que se encuentra en una complejidad temporal de O(n) y esto se fortalece gracias a la descomposición de la función, en primera instancia se realizan operaciones con complejidad constante por lo cual no influye en el general, y todo recae en el ciclo for que permite filtrar los goles del jugador para ver si fue o no de penal lo que permite entregar la información completa. Y para finalizar se realiza un filtrado de la cantidad de datos que se desean entregar lo cual es constante debido a que no se itera, sencillamente se realiza una sublist o se entrega como estaba desde el paso anterior.

Requerimiento 3

Descripción

```python
def req_3(equipo,fecha_inicial,fecha_final,data_structs):
    """
    Función que soluciona el requerimiento 3

    Consultar los partidos que disputó un equipo durante un periodo especifico. (I)

    Parametros:

        equipo: Nombre del equipo
        fecha_inicial: Fecha inicial del periodo
        fecha_final: Fecha final del periodo

    Devuelve:

        Número total de partidos disputados.
        Número total de partidos disputados como local.
        Número total de partidos disputados como visitante.
        El listado de los partidos disputados ordenados cronologicamente.
        obtener si el partido tuvo autogol o penal a partir de goalscorer


    """

    equipo=equipo.title()
    fecha_inicial = datetime.datetime.strptime(fecha_inicial, "%Y-%m-%d")
    fecha_final = datetime.datetime.strptime(fecha_final, "%Y-%m-%d")

    partidos_finales = lt.newList("ARRAY_LIST")

    mapa = data_structs["equipo_scorer"]
    partidos = me.getValue(mp.get(mapa, equipo))
    local=0
    visitante=0

    for partido in lt.iterator(partidos):
        if fecha_inicial <= partido["date"] and partido["date"] <= fecha_final:
            if partido["home_team"] == equipo:
                local+=1
            elif partido["away_team"] == equipo:
                visitante+=1

            lt.addLast(partidos_finales, partido)


    return lt.size(data_structs["equipo_scorer"]), local, visitante, partidos_finales
```

Esta función se encarga de encontrar los partidos de una selección dentro de un rango especificoconsiderando agregar la información presente en el datastruct de “results” asi mismo como información presente en el datastruct de “goalscorer” por lo tanto la idea es acceder a un datastruct que contiene la información y solo iterar para confirmar aquellos datos que cumplen las fechas.

Este requerimiento se encarga de

Entrada	Equipo(str): La selección de la cual se desea usar la función. Fecha_inicial(str): El límite inferior del rango a considerar. Fecha_final(str): El límite superior del rango a considerar. Estructura de dato del modelo
Salidas	lt.size(data_structs[“equipo_scorer”])(int): Total de partidos que se consideran. Local(int): Número total de partidos como local. Visitante(int): Número total de partidos como visitante. Partidos_finales(data_struct): El array list donde está contenida la nueva información que se pide en el view, esta proximamente se

	desempaqueta y permite mostrar al usuario la información completa.
Implementado (Sí/No)	Si. Implementado por Tomás Velásquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
1) Se entregan las fechas a las cuales se convierten en formato datetime con formato %Y -%m-%d, a su vez se inicializa un nuevo datastruct donde se almacena la información que será entregada y se crean contadores para cantidad de partidos de la selección como locales y visitantes. Asi mismo se accede a la informacion o entry propio de los datos pedidos.	O(1)
2) Se implementa un for o ciclo que recorre una cantidad especifica de datos para comprobar si están dentro del rango de fechas indicado, si es asi se agrega este partido al array final y se agrega al contador de local o visitante	O(n) en el peor caso donde se deban filtrar una gran cantidad de datos por lo que se comparan todas las fechas de todos los partidos, sin embargo, si las fechas a considerar son muy específicas esto se reduce a un n menor.
TOTAL	O(N) donde n depende del rango que entrega el usuario.

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron

PRUEBAS:

Equipo: Argentina

Fecha_inicial= 2000-01-01

Fecha_final= 2020-12-31

Procesadores	**Chip M1 macbook pro**
Memoria RAM	8 GB
Sistema Operativo	MacOs Ventura

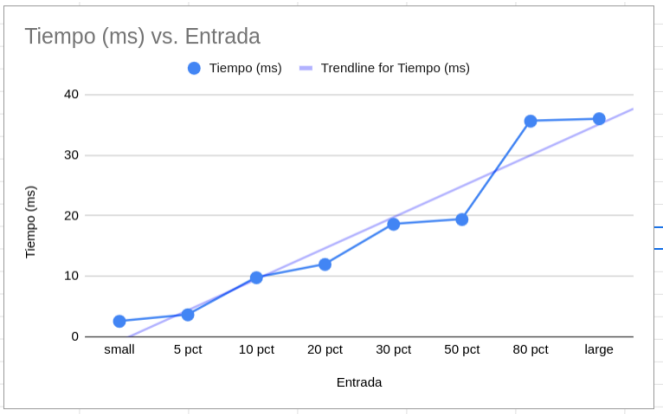
Entrada	**Tiempo (ms)**
small	0.336

5 pct	0.470
10 pct	0.965
20 pct	1.073
30 pct	1.387
50 pct	3.870
80 pct	4.871
large	8.254

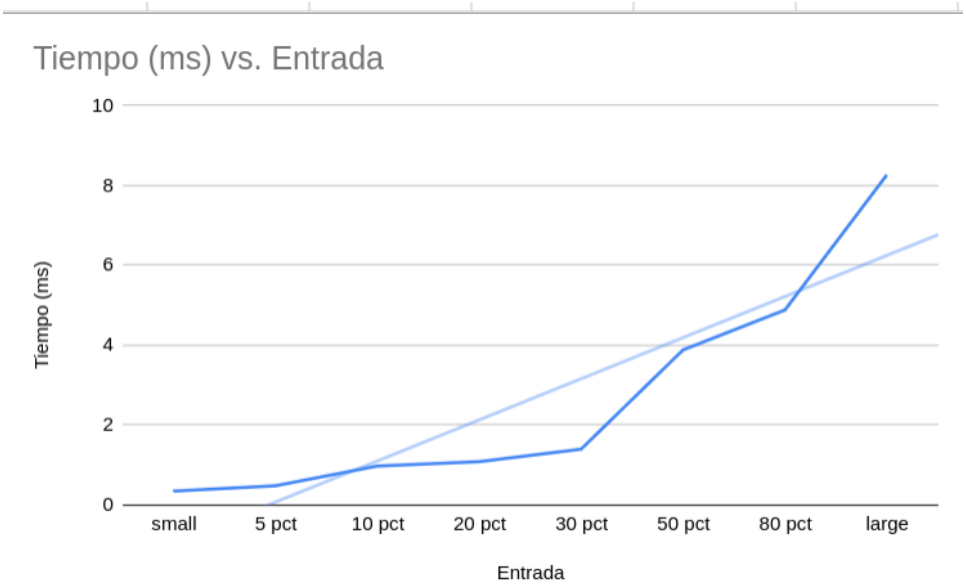
Graficas

Las gráficas con la representación de las pruebas realizadas.

Reto 1:



Reto 2:



Análisis

Después de realizar el proceso del requerimiento 3, se puede analizar varias particularidades con las que nos encontramos. Por un lado, se empezó con una implementación la cual consistía en recorrer todo el datastruct de results para filtrar a través de iteración los partidos que estaban entre el rango definido, y próximo a eso otro for anidado donde se revisaba el datastruct de goalscorer para extraer la información de penales y own_goal. Sin embargo, después de realizar un proceso de análisis de complejidad se logró pensar un método alterno usando la creación de un datastruct externo que a manera de mapa permitiera acceder a toda la información con un solo ciclo de complejidad maxima O(n) ya que al permitir que en la carga de datos se incluya la información tanto de reulst como goalscorer, logramos reducir la complejidad drásticamente y se nota en los resultados comparados al reto 1.

Requerimiento 4

Descripción

```
def req_4(tournament, fecha_inicial, fecha_final, data_structs):
    """
    Función que soluciona el requerimiento 4

    Consultar los partidos de un torneo durante un periodo especifico.

    Parámetros:

            tournament: Nombre del torneo
            fecha_inicial: Fecha inicial del periodo
            fecha_final: Fecha final del periodo
            data_structs: Estructuras de datos

    Devuelve:

            partidos_lista: Lista con los partidos del torneo
            num_paises: Número de paises diferentes que se jugo el torneo
            num_ciudades: Número de ciudades diferentes donde se jugaron los partidos del torneo
            penalties: Número de penalties que hubo
            num_partidos: Número de partidos que se jugaron en el torneo
            num_torneos: Número de torneos diferentes que se jugaron en el periodo

    """
    tournament=tournament.title()

    fecha_inicial = datetime.datetime.strptime(fecha_inicial, "%Y-%m-%d")
    fecha_final = datetime.datetime.strptime(fecha_final, "%Y-%m-%d")

    paises = mp.newMap(210)
    ciudades = mp.newMap(1000)
    penalties = 0
    torneos = mp.newMap(30)
    partidos = mp.newMap(1000)
    partidos_lista = lt.newList("ARRAY_LIST")
```

```
rango_binaria = crear_rango_busqueda_binaria(data_structs["results"], fecha_inicial, fecha_final)

for partido in lt.iterator(rango_binaria):
    # Obtengo la key del torneo que me interesa
    torneo = mp.get(torneos, partido["tournament"].strip().title())
    # Si no encuentra el torneo entonces añade el torneo al mapa
    if not torneo:
        mp.put(torneos, partido["tournament"].strip().title(), 1)
    # Si el torneo es el que me interesa entonces añade los datos a los mapas
    if partido["tournament"].title() == tournament:

        pais = mp.get(paises, partido["country"].title().strip())
        if not pais:
            mp.put(paises, partido["country"].title().strip(), 1)

        ciudad = mp.get(ciudades, partido["city"].title().strip())
        if not ciudad:
            mp.put(ciudades, partido["city"].title().strip(), 1)

        penalty, winner = penalty_check(data_structs, partido)
        penalties += penalty

        date=datetime.datetime.strftime(partido['date'], "%Y-%m-%d")

        partido_get = mp.get(partidos, date + partido["home_team"].title() + partido["away_team"].title())
        if not partido_get:
            mp.put(partidos, date + partido["home_team"].title() + partido["away_team"].title(),1)
            lt.addLast(partidos_lista, {
                        "date": partido["date"],
                        "tournament": partido["tournament"],
                        "country": partido["country"],
                         "city": partido["city"],
                         "home_team": partido["home_team"],
                         "away_team": partido["away_team"],
                         "home_score": partido["home_score"],
                         "away_score": partido["away_score"],
                         "winner": winner,
                    })

num_paises = mp.size(paises)
num_ciudades = mp.size(ciudades)
num_partidos = mp.size(partidos)
num_torneos = mp.size(torneos)

return partidos_lista, num_paises, num_ciudades, penalties, num_partidos, num_torneos
```

Este requerimiento se encarga de consultar los partidos relacionados con un torneo en un periodo especifico. Toma como parámetros: el nombre del torneo a consultar, la fecha inicial y la fecha final del periodo que se desea consultar y la estructura de datos sobre la que se va a operar.

Está función hace búsqueda binaria para establecer el rango de fecha sobre el que va a iterar. Después por cada ítem de la lista resultante. Es importante recalcar que intenta entrar a varias llaves, si no encuentra dicha llave va a crearla. Revisa si el torneo ya se contó, si no lo agrega. Por cada ítem de la búsqueda binaria revisará si lo está va ahora a revisar si el país y la ciudad donde se disputa el partido ya se contó, sí no lo agrega. También verifica si este partido en específico ya se había contado, sí no, añade la información respecto.

Finalmente añade a una lista llamada partidos la información de dicho partido especifico. Devuelve la lista de los partidos con toda su información, saca el número de países, ciudades, penaltis, partidos y torneos y los devuelve.

Entrada	Nombre del torneo, fecha inicial, fecha final, estructura de datos
Salidas	Lista de partidos jugados en ese intervalo, número de países, ciudades, penaltis, partidos y torneos.
Implementado (Sí/No)	Si. Implementado por Daniel Bolivar

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Se cambia el formato de las fechas a datetime y se crean las variables necesarias.	O(1)
Paso 2: Hace búsqueda binaria y obtiene una lista de los elementos que nos interesa, aquellos que están dentro del rango de las fechas.	O(n)
Se itera por cada partido dentro de la lista resultante de la búsqueda binaria.	O(n)
Se verifican llaves de torneos. Y si hace parte del torneo que nos importa.	O(1)
Se añade el país, ciudades si estos no estaban añadidos con anterioridad.	O(1)
Verifica en la función penalty_check (qué es O(1)) si hubo penaltis. Si hubo obtiene un 1 y el ganador del partido, si no devuelve un 0 y el ganador de dicho partido.	O(1)
Pasa la fecha a string y revisa si el resultado del partido existe en mapa anteriormente creado. Si no existe entonces lo agrega con la información adicional del ganador del partido. A su vez lo añade a una lista que lleva el recuento de todos los partidos que nos interesan.	O(1)

Finalmente obtiene el valor de los mapas de paises, ciudades, partidos y torneos y los guarda a cada uno en una variable. Devuelve una lista con los partidos que nos interesa y el número de países, ciudades, penaltis, partidos y torneos.	O(1)
TOTAL	***O(n) debido a que el peor caso es cuando la busqueda binaria es de todos los elementos. Es decir que la busqueda binaria termina siendo igual a la cantidad t***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron

Torneo: FIFA World Cup

Desde: 1900-01-01

Hasta: 2023-12-12

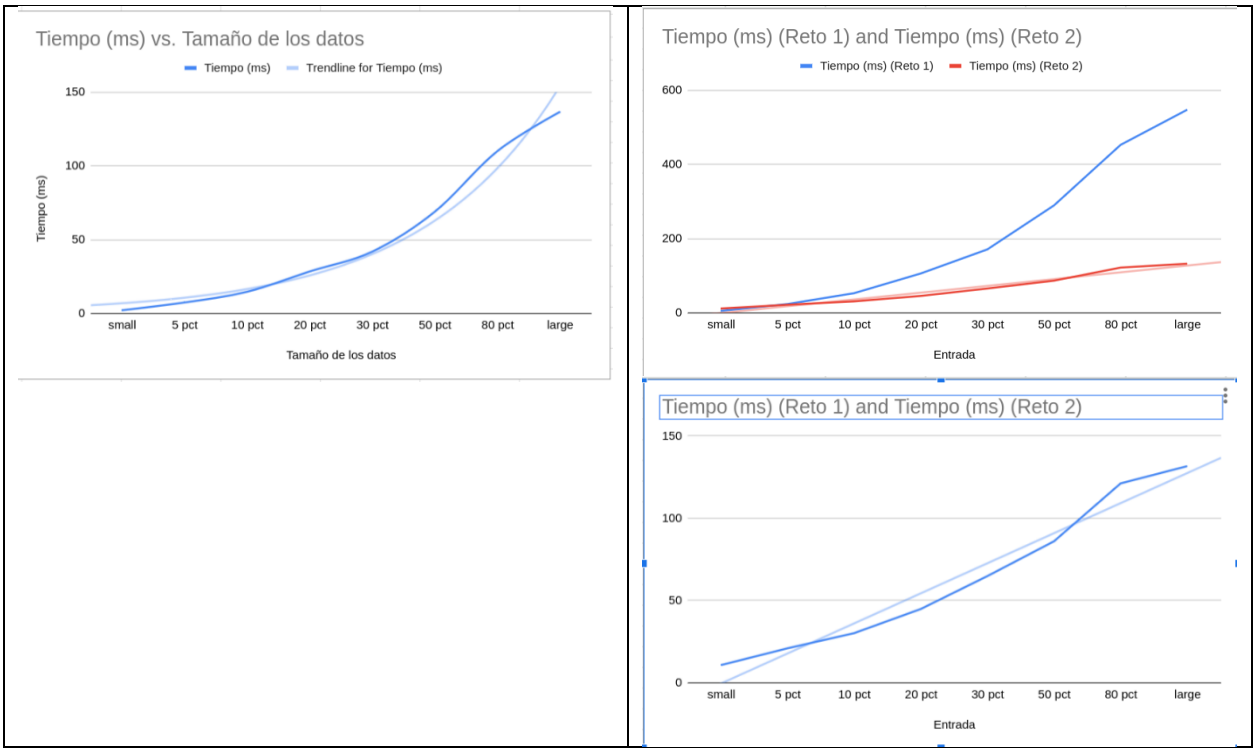
Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	16 GB
Sistema Operativo	Fedora Linux 38 (Workstation Edition)

Entrada	**Tiempo (ms) (Reto 1)**	**Tiempo (ms) (Reto 2)**
small	4.297	10.569
5 pct	22.551	20.791
10 pct	51.923	29.975
20 pct	104.964	44.645
30 pct	169.873	64.652
50 pct	288.450	85.794
80 pct	451.666	120.966
large	546.441	131.460

Graficas

Las gráficas con la representación de las pruebas realizadas.

Reto 1	Reto 2

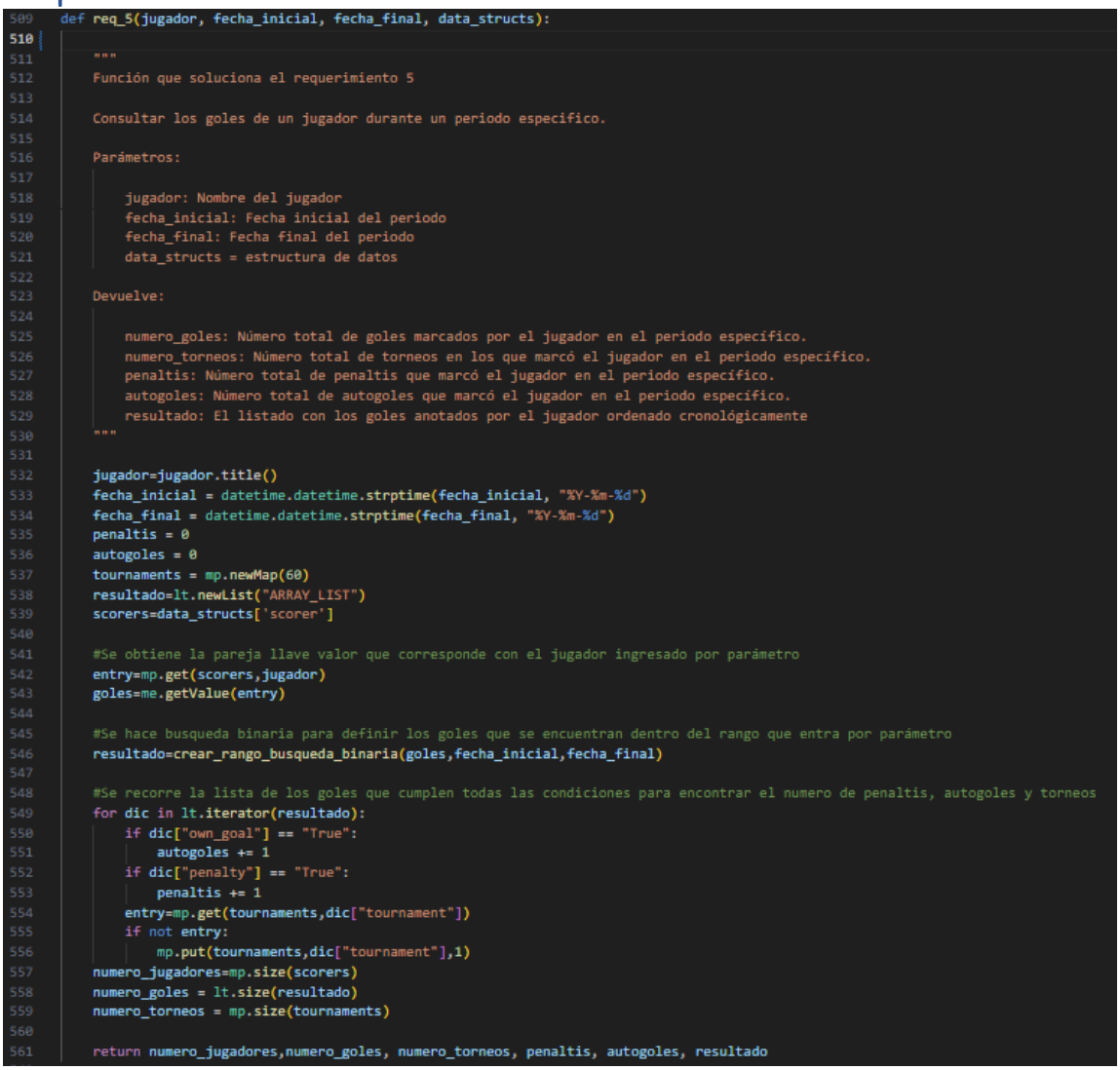


Análisis

La complejidad temporal de la función es O(n) en el peor caso. Esta complejidad depende del rango recibido por la búsqueda binaria que puede ser igual a n cuando todos los elementos están adentro del rango de la búsqueda binaria. Respecto al reto 1, nos encontramos una complejidad temporal lineal en el reto 2 frente a una exponencial que obtuvimos en el reto 1. Encontramos la diferencia de tiempo de ejecución es alrededor de 4 veces más rápido. En tilda la función es alrededor de de ~6 +2n en complejidad, relativamente baja y en las gráficas se puede ver que fue eficiente.

Requerimiento 5

Descripción



```
def req_5(jugador, fecha_inicial, fecha_final, data_structs):

    """
    Función que soluciona el requerimiento 5

    Consultar los goles de un jugador durante un periodo especifico.

    Parámetros:

        jugador: Nombre del jugador
        fecha_inicial: Fecha inicial del periodo
        fecha_final: Fecha final del periodo
        data_structs = estructura de datos

    Devuelve:

        numero_goles: Número total de goles marcados por el jugador en el periodo especifico.
        numero_torneos: Número total de torneos en los que marcó el jugador en el periodo especifico.
        penaltis: Número total de penaltis que marcó el jugador en el periodo especifico.
        autogoles: Número total de autogoles que marcó el jugador en el periodo especifico.
        resultado: El listado con los goles anotados por el jugador ordenado cronológicamente
    """

    jugador=jugador.title()
    fecha_inicial = datetime.datetime.strptime(fecha_inicial, "%Y-%m-%d")
    fecha_final = datetime.datetime.strptime(fecha_final, "%Y-%m-%d")
    penaltis = 0
    autogoles = 0
    tournaments = mp.newMap(60)
    resultado=lt.newList("ARRAY_LIST")
    scorers=data_structs['scorer']

    #Se obtiene la pareja llave valor que corresponde con el jugador ingresado por parámetro
    entry=mp.get(scorers,jugador)
    goles=me.getValue(entry)

    #Se hace busqueda binaria para definir los goles que se encuentran dentro del rango que entra por parámetro
    resultado=crear_rango_busqueda_binaria(goles,fecha_inicial,fecha_final)

    #Se recorre la lista de los goles que cumplen todas las condiciones para encontrar el numero de penaltis, autogoles y torneos
    for dic in lt.iterator(resultado):
        if dic["own_goal"] == "True":
            autogoles += 1
        if dic["penalty"] == "True":
            penaltis += 1
        entry=mp.get(tournaments,dic["tournament"])
        if not entry:
            mp.put(tournaments,dic["tournament"],1)
    numero_jugadores=mp.size(scorers)
    numero_goles = lt.size(resultado)
    numero_torneos = mp.size(tournaments)

    return numero_jugadores,numero_goles, numero_torneos, penaltis, autogoles, resultado
```

Este requerimiento se encarga de consultar las anotaciones obtenidas por un jugador utilizando su nombre y un periodo entre dos fechas especificadas.

Para esto se obtiene el valor (lista de goles del jugador) que le corresponde a la llave (jugador) a través del mapa con índice “scorer”.

Se recorre la lista de goles del jugador para encontrar los goles marcados únicamente dentro del periodo especifico (este recorrido se hace con un binary serch). Luego, se recorren únicamente los goles que cumplen con todas las condiciones, y por cada gol se revisa si fue penalti o autogol y se revisa el torneo, posteriormente se suma a los contadores (penaltis y autogoles) y a al mapa torneos si no es

repetido. Finalmente se encuentra el número total de jugadores, torneos y partidos con el tamaño de los mapas y la lista. correspondientes.

La función retorna: el número de jugadores con info disponible, el número de goles marcados por el jugador, el número de torneos en los que anoto, cuantos penaltis y autogoles marco, y la lista de diccionarios con todos los goles del jugador entre las fechas recibidas.

Entrada	Estructuras de datos del modelo, nombre del jugador, fecha inicial y fecha final.
Salidas	El número de jugadores con info disponible (int), el número de goles marcados por el jugador (int), el número de torneos en los que anoto (int), cuantos penaltis (int) y autogoles marco (int), y la lista de diccionarios con todos los goles del jugador entre las fechas recibidas (list).
Implementado (Sí/No)	Si. Implementado por Lina Muñoz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Definición de distintas variables y creación de un mapa y una lista vacía.	O (1)
Paso 2: Acceder al valor de una llave en el mapa “scorer”	O (1)
Paso 3: Recorrer la lista goles (todos los goles del jugador) para encontrar aquellos que se encuentran en el rango de fechas. Función crear_rango_búsqueda binaria. n’= es el rango del binary search (tamaño de resultado), en el peor caso las fechas no ayudan a delimitar y el rango del bynary search es el mismo tamaño que goles.	O(n’)
Paso 3: Recorrer la lista resultado (lista de los goles del jugador en el periodo especifico) para determinar número de penaltis, autogoles y torneos. n’=Tamaño de resultado	O (n’)
Paso 4: Encontrar el tamaño de dos mapas y una lista	O (1)
TOTAL	***O(n’)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron:

¿Cuál es el nombre del jugador? Michael Ballack

¿Desde qué fecha desea consultar? (con formato %Y-%m-%d) 1992-10-16

¿Hasta qué fecha desea consultar? (con formato %Y-%m-%d) 2006-10-16

Procesadores	**AMD Ryzen 7 5800H with Radeon Graphics**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms) RETO 1**	**Tiempo (ms) RETO 2**
small	0.377	0.485
5 pct	1.447	0.491
10 pct	3.168	0.502
20 pct	5.333	0.519
30 pct	8.882	0.547
50 pct	18.738	0.586
80 pct	35.940	0.627
large	53.281	0.689

Tablas de datos

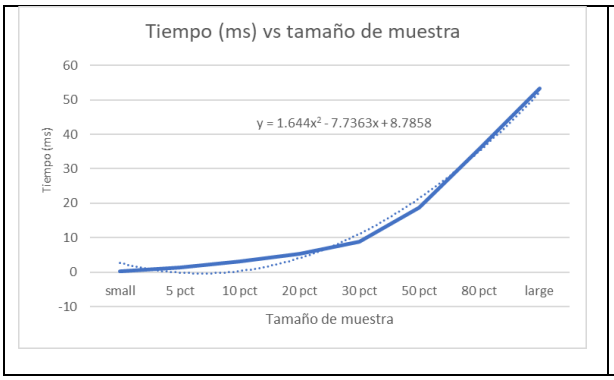
Las tablas con la recopilación de datos de las pruebas.

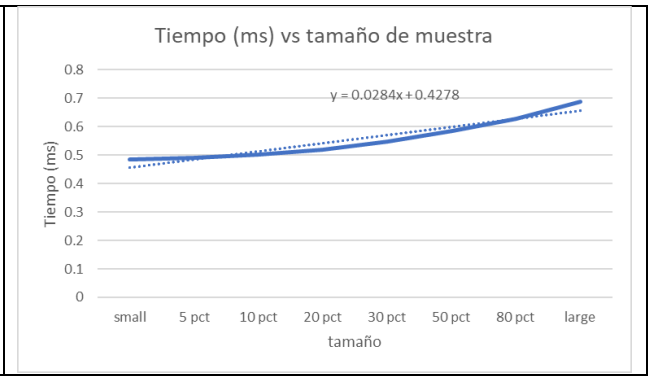
Muestra	**Salida (Numero de goles encontrados)**	**Tiempo (ms)**
small	2	0.485
5 pct	2	0.491
10 pct	5	0.502
20 pct	6	0.519
30 pct	6	0.547
50 pct	9	0.586
80 pct	14	0.627
large	20	0.689

Graficas

Las gráficas con la representación de las pruebas realizadas.

RETO 1 O(n*m)	RETO 2 ***O(n')***





Análisis

Este requerimiento teóricamente es complejidad temporal ***O(n')*** en el peor caso debido a que se recorrer la lista resultado (lista de los goles del jugador en el periodo especifico) y es el peor caso cuando el rango del binary search (n') tiene el tamaño total de los goles del jugador (pues las fechas no delimitaron la lista de goles pues están son muy amplias). Por lo que en un caso promedio seria log n. Este comportamiento se puede evidenciar experimentalmente en la gráfica. Dado que, gracias a que los tiempos obtenidos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.

En el Reto 1 el requerimiento tenía una complejidad de O(n*m), mientras que en este Reto 2 tiene una complejidad de O(***n'***), esta diferencia se puede evidenciar en la considerable disminución en los tiempos de ejecución y en la diferencia de la línea de tendencia, puesto que n en el reto 1 es n es el tamaño de todo Results y m el tamaño de todo Goalscorers, mientras que para este reto n' es la cantidad de goles que le corresponde únicamente al jugador consultado, lo que disminuye considerablemente los elementos que recorre el for..

Requerimiento 6

Descripción

```python
def req_6(cantidad, torneo,anio,data_structs):
    """
    Función que soluciona el requerimiento 6

    Consultar a los n mejores equipos de un torneo durante un año especifico.

    Parámetros:

        cantidad: cantidad de equipos que queremos que nos devuelva
        torneo: Nombre del torneo a consultar
        anio: Año de consulta
        data_structs = estructura de datos

    Devuelve:

        c_anios: Total de años con info disponible.
        c_torneos: Total de torneos disputados en el año.
        c_equipos: Total de equipos involucrados en el torneo.
        c_partidos: Total de partido disputados en el torneo.
        total_paises: Total de países en los que se disputo el torneo durante el año.
        total_ciudades: Total de ciudades en las que se disputo el torneo durante el año.
        max_ciudad : Nombre de la ciudad en la que más partidos se disputaron.
        estadisticas: Listado de las estadisticas ordenadas de mejor a peor equipo.

    """
    torneo=torneo.title()
    c=int(cantidad)
    anio=int(anio)
    anios=data_structs['anio_torneo']
    c_anios=mp.size(anios)-1

    #Se obtiene el valor(otra mapa de torneos) que corresponde con el año ingresado por parámetro
    torneos=me.getValue(mp.get(anios,anio))
    c_torneos=mp.size(torneos)-1

    #Se obtiene el valor(otra mapa de equipos) que corresponde con el torneo ingresado por parámetro
    equipos=me.getValue(mp.get(torneos,torneo))
    partidos_torneo=me.getValue(mp.get(equipos, "encuentros_torneo"))

    c_partidos=lt.size(partidos_torneo)
    mp.remove(equipos,"encuentros_torneo")
    lista_equipos=mp.keySet(equipos)
    mp.put(equipos,"encuentros_torneo",partidos_torneo)
    c_equipos=mp.size(lista_equipos)
    paises=lt.newList("ARRAY_LIST")
    ciudades={}

    #Se recorre la lista de los partidos en el año y torneo específicos
    for dic in lt.iterator(partidos_torneo):
            if lt.isPresent(paises, dic["country"]) == 0:
                lt.addLast(paises, dic["country"])
            if dic["city"] not in ciudades:
                ciudades[dic["city"]]=1
            else:
                ciudades[dic["city"]]+=1

    total_paises=lt.size(paises)
    total_ciudades=len(ciudades)
    max_ciudad=max_dic(ciudades)
    estadisticas=lt.newList("ARRAY_LIST")

    #Se recorre la lista de todos los equipos que participaron en el torneo para ir llenar sus estadisticas uno a uno
    for equipo in lt.iterator(lista_equipos):
        team={"team":equipo,
                "total_points":0,
                "goal_difference":0,
                "penalty_points":0,
                "matches":0,
                "own_goal_points":0,
                "wins":0,
                "draws":0,
                "loses":0,
                "goals_for":0,
                "goals_against":0,
                "top_scorer":"No hay"}
```

```
    partidos_equipo=me.getValue(mp.get(equipos,equipo))
    for partido in lt.iterator(partidos_equipo):

        team["matches"]=lt.size(partidos_equipo)
        if partido["home_team"]==equipo:
            team["goals_for"]+=int(partido["home_score"])
            team["goals_against"]+=int(partido["away_score"])
        elif partido["away_team"]==equipo:
            team["goals_for"]+=int(partido["away_score"])
            team["goals_against"]+=int(partido["home_score"])

        if partido["winner"]==equipo:
            team["total_points"]+=3
            team["wins"]+=1
        elif partido["winner"]=="empate":
            team["total_points"]+=1
            team["draws"]+=1
        else:
            team["loses"]+=1
    team["goal_difference"]=team["goals_for"]-team["goals_against"]

    #Se busca si existen goles que teangan que ver con equipo (anotados y recibidos) en el año y torneo, si encuentra añade la información de "top_socer", si no lo deja como "No hay"
    r=equipo+str(anio)+torneo
    entry=mp.get(data_structs["goals"],r)
    if entry:
        goles_equipo=me.getValue(entry)
        jugadores={}
        for gol in lt.iterator(goles_equipo):
            if gol["penalty"]=="True":
                    team["penalty_points"]+=1
            if gol["own_goal"]=="True":
                team["own_goal_points"]+=1
            if gol["team"]==equipo:
                if gol["scorer"] not in jugadores:
                    jugadores[gol["scorer"]]=1
                elif gol["scorer"] in jugadores:
                    jugadores[gol["scorer"]]+=1

        if jugadores:
            goleador=max_dic(jugadores)
            estadisticas_goleador={"scorer":goleador, "goals":0, "matches":0, "avg_time":0}
            dates=mp.newMap(100)
            minutos=0
            for gol in lt.iterator(goles_equipo):
                if gol["scorer"]==goleador:
                    estadisticas_goleador["goals"]+=1
                    minutos+=float(gol["minute"])
                    en=mp.get(dates,gol["date"])
                    if not en:
                        mp.put(dates,gol["date"],1)

            estadisticas_goleador["matches"]=mp.size(dates)
            estadisticas_goleador["avg_time"]=minutos/estadisticas_goleador["goals"]
            goleado=lt.newList("ARRAY_LIST")
            lt.addLast(goleado,estadisticas_goleador)
            team["top_scorer"]=goleado
        else:
            estadisticas_goleador={"scorer":"No hay", "goals":0, "matches":0, "avg_time":0}
            goleado=lt.newList("ARRAY_LIST")
            lt.addLast(goleado,estadisticas_goleador)
            team["top_scorer"]=goleado
    else:
        estadisticas_goleador={"scorer":"No hay", "goals":0, "matches":0, "avg_time":0}
        goleado=lt.newList("ARRAY_LIST")
        lt.addLast(goleado,estadisticas_goleador)
        team["top_scorer"]=goleado

    lt.addLast(estadisticas, team)

#Se ordena estadisticas del mejor al peor equipo, y se recorta la lista teniendo en cuenta la cantidad entrada por parámetro
merg.sort(estadisticas, cmpReq6)
if lt.size(estadisticas)>c:
    estadisticas=lt.subList(estadisticas,1,c)

return c_anios,c_torneos,c_equipos,c_partidos,total_paises,total_ciudades,max_ciudad, estadisticas
```

Este requerimiento se encarga de buscar el top n de mejores equipos en un torneo y un año (lista de n diccionarios con las estadísticas de cada equipo), y a su vez entrega la cantidad de años con información disponible, la cantidad de torneos en el año consultado, la cantidad partidos totales que cumplen el año y torneo, la cantidad de equipos que participaron en el torneo, la cantidad de ciudades y países en los que se jugaron, así como el nombre de la ciudad en la que más partidos se jugaron.

Para esto se obtiene el valor (mapa de torneos) que le corresponde a la llave (anio) a través del mapa con índice “anio_torneo”.

Luego se obtiene el valor (mapa de equipo) que le corresponde a la llave (torneo) a través del mapa obtenido anteriormente.

Seguidamente se obtiene el valor (lista de goles del torneo) que le corresponde a la llave (“encuentros_torneo”) a través del mapa anterior.

A continuación, se recorre la lista de goles del torneo para obtener el numero de países, ciudades y el nombre de la ciudad con más partidos disputados.

Luego se recorre la lista de los equipo (keyset del mapa equipos) y sus goles para ir llenando toda la información de sus estadísticas, en caso de haber un goleador se llena estadisticas goleador y se agrega a estadisticas, y al finalizar de completar el diccionario por equipo se agrega el diccionario a la lista estadísticas.Finalmente, cuando se tenga estadísticas lleno, se hace un mergesort para ordenar del mejor al peor equipo con la ayuda de la cmpreq6 y dependiendo del n entregado por parámetros se hace una sublista

Entrada	Estructuras de datos del modelo, top n equipos que se desea consultar, nombre del torneo y año.
Salidas	c_anios (int) ,c_torneos (int),c_equipos (int) ,c_partidos (int), total_paises (int), total_ciudades (int), max_ciudad (str), estadisticas (list)
Implementado (Sí/No)	Si. Implementado por Lina Muñoz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Definición de distintas variables	O (1)
Paso 2: Acceder al valor de una llave de tres mapas	O (1)
Paso 3: Recorre la lista de partidos torneo h=número de elementos de la lista de partidos torneo	O(h)
Paso 4: Definición de distintas variables usando lt.size	O (1)
Recorridos con doble for Paso 5: Por cada elemento de la lista de equipos se recorren los partidos que les corresponden, los cuales se encuentra accediendo con la llave (equipo) al mapa equipos (l 611) m=Numero de equipos en el keyset, los cuales participaron en el torneo j=Numero de partidos que le corresponden al equipo en el año y torneo especifico.	O(m*j)
Paso 6: Acceder con la llave (equipo+anio+torneo) al listado de goles (Que se encuentran en goalscorers) que le corresponden al equipo del que se están llenando las estadisticas y recorrerlo. m=Numero de equipos en el keyset, los cuales participaron en el torneo n=número de goles del equipo	O(m*n)

Paso 7: Hacer un merge sort de estadísticas para ordenar de mejor a peor	O (n log (n))
TOTAL	(m*n) + (m*j) O(m*(n+j)) ***O(m*k)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron:

¿Desea consultar el top de cuantos equipos? 11

¿Qué torneo desea consultar? FIFA World Cup qualification

¿Qué año desea consultar? 2021

Procesadores	**AMD Ryzen 7 5800H**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms) RETO 1**	**Tiempo (ms) RETO 2**
small	2.915	7.85
5 pct	18.405	13.05
10 pct	49.390	23.17
20 pct	84.393	35.15
30 pct	149.692	46.03
50 pct	428.976	55.84
80 pct	693.917	65.22
large	861.400	79

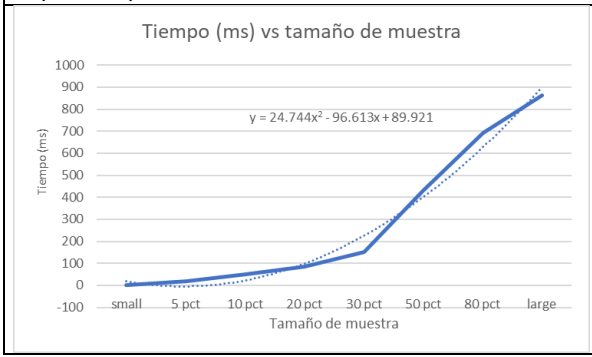
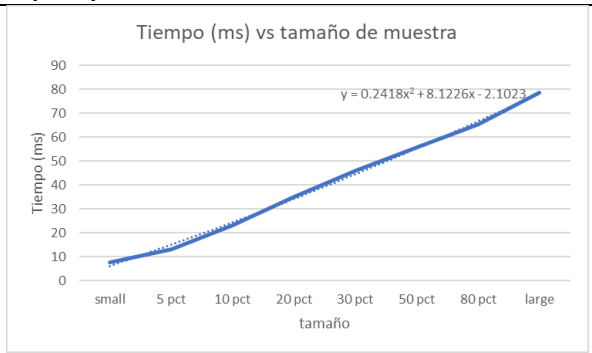
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Partidos totales encontrados**	**Tiempo (ms)**
small	43	7.85
5 pct	169	13.05
10 pct	335	23.17
20 pct	572	35.15
30 pct	745	46.03
50 pct	998	55.84
80 pct	1218	65.22
large	1293	79

Graficas

Las gráficas con la representación de las pruebas realizadas.

RETO 1 O(n*m*k)	RETO 2 ***O(m*k)***
	

Análisis

Este requerimiento teóricamente es complejidad temporal ***O(m*k)*** debido a que hay un loop doble en el código, puesto que en la línea 800 existe un for que recorre la lista de equipos y en la línea 841 existe un for anidado que recorre los goles del equipo. Este comportamiento se puede evidenciar experimentalmente de manera parcial en la gráfica. Dado que, la tendencia puede llegar a confundirse con una complejidad lineal, esto se debe a que m (lista de equipos) nunca va a superar a 200 (número de países totales) mientras que k siempre serán los goles que le corresponden únicamente al equipo. Por lo que m y k nunca serán número muy grandes, lo que ´permite que los tiempos de ejecución no superen los 900 ms en ningún caso.

En el Reto 1 el requerimiento tenía una complejidad de O(n*m*k), mientras que en este Reto 2 tiene una complejidad de ***O(m*k)***, esta diferencia se puede evidenciar en la considerable disminución en los tiempos de ejecución, puesto que n en el reto 1 es n es el tamaño de todo Results y m el tamaño de Goalscorers y k el tamaño de Shootouts, mientras que para este reto m es el número de equipos en el keyset (nunca mayor a 200) y k es la suma de goles y partidos que le corresponde al equipo que se revisa de la lista del keyset.

Requerimiento 7

Descripción

```python
def req_7(n_puntos, torneo, data_structs):
    """
    Encontrar los anotadores con N puntos dentro un torneo especifico
    """

    jugadores = lt.newList("ARRAY_LIST")
    tournament_key = me.getValue(mp.get(data_structs["torneos_goles"], torneo))
    tournament_scorers = mp.keySet(me.getValue(mp.get(tournament_key, "scorers")))

    num_torneos = mp.size(data_structs["torneos_goles"])
    num_scorers = lt.size(tournament_scorers)
    num_partidos = me.getValue(mp.get(tournament_key, "results"))
    num_goals = me.getValue(mp.get(tournament_key, "goals"))
    num_penalties = me.getValue(mp.get(tournament_key, "penalties"))
    num_own_goals = me.getValue(mp.get(tournament_key, "own_goals"))


    scorers_map = me.getValue(mp.get(tournament_key, "scorers"))

    for scorer in lt.iterator(tournament_scorers):
        scorer_info = me.getValue(mp.get(scorers_map, scorer))
        points = me.getValue(mp.get(scorer_info, "total_points"))

        if points == int(n_puntos):
            lt.addLast(jugadores, scorer_info)

    merg.sort(jugadores, cmpReq7)

    return num_torneos, num_scorers, num_partidos, num_goals, num_penalties, num_own_goals, jugadores
```

Este requerimiento se encarga de encontrar a los anotadores con N puntos en un partido especifico.

Entrada	Los n_puntos que nos interesan, torneo que queremos buscar y la estructura de datos.
Salidas	*num_torneos: Número de torneos diferentes que se jugaron en el periodo* *num_scorers: Número de jugadores diferentes que anotaron en el torneo* *num_partidos: Número de partidos que se jugaron en el torneo* *num_goals: Número de goles que se anotaron en el torneo* *num_penalties: Número de goles que se anotaron en el torneo* *num_own_goals: Número de goles que se anotaron en el torneo* *jugadores: Lista de jugadores con N puntos en el torneo*
Implementado (Sí/No)	Si. Implementado por Daniel Bolívar y Tomás Velásquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de lista para almacenar a los jugadores. Creación de variables. (tournament_key, tournament_scorers) El valor tourna lo _key obtiene yendo a la llave “torneos_goles” de modelo. El valor de tournament_scorers lo obtiene pidiendo el valor de	O(1)

todas las llaves de los scorers en el torneo que necesitamos.	
Busca el número de torneos que se encontraron. Esto hace yendo a la estructura de la llave “torneos_goles” del modelo y viendo su tamaño. Después revisa el tamaño de los scorers obtienendo el sieze de tournament_scorers. Adentro de la key de torneos accede al valor de las keys results, goals, penalties y own_goals. Estas keys le da los valores de num_partidos, num_goals, num_penalties, num_own_goals respectivamente.	O(1)
Obtiene el valor de todos los scorers yendo a la key “scorers” dentro del torneo de interes.	O(1)
Por cáda scorer en la lista tournament_scoreres va a obtener la información del scorer,y va a revisar sus total_points. Después va a revisar si tiene los puntos que nos interesan. Si es asi los va a añadir a la lista de jugadores.	O(K) N := numero de jugadores totales K := numero de scorers dentro de un torneo En el peor caso N = K
Se va a ordenar la lista de jugdores usando merge sort.	O(Klog(K))
TOTAL	***O(Klog(K))***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron

Inputs Reto 1

¿Desea consultar el top de cuantos jugadores? **8**

¿Desde qué fecha desea consultar? (con formato %Y-%m-%d) **2018-11-11**

¿Hasta qué fecha desea consultar? (con formato %Y-%m-%d) **2023-11-11**

Inputs Reto 2

¿Qué torneo desea consultar? Copa América

Desea consultar los jugadores con cuantos puntos: 2

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	16 GB
Sistema Operativo	Fedora Linux 38 (Workstation Edition)

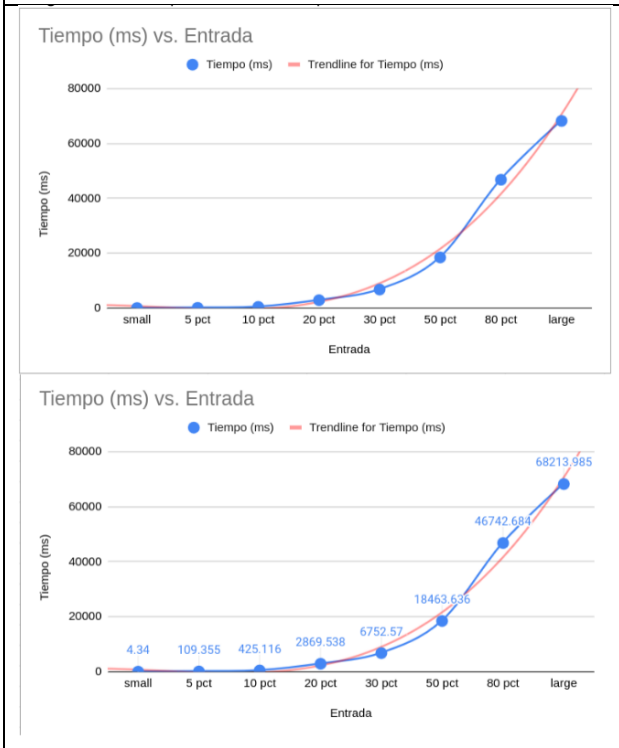
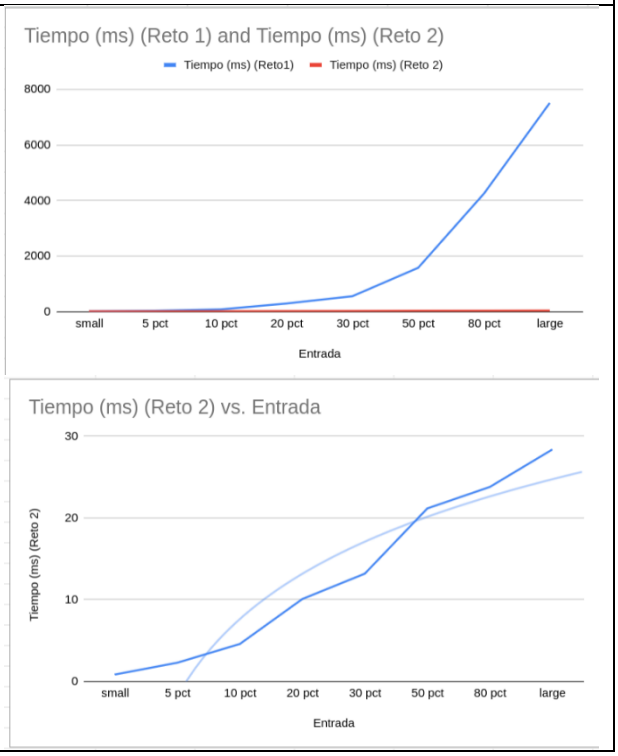
Entrada	**Tiempo (ms) (Reto1)**	**Tiempo (ms) (Reto 2)**

small	1.445	0.811
5 pct	22.198	2.257
10 pct	72.428	4.547
20 pct	283.898	10.043
30 pct	545.936	13.146
50 pct	1567.299	21.123
80 pct	4233.794	23.747
large	7492.475	28.328

Graficas

Las gráficas con la representación de las pruebas realizadas.

Grafica (1)

Reto 1	Reto 2
	

Análisis

La función tiene una complejidad teórica de O(nlog(n)) en el peor caso. Como en google docs no encontré forma de poner la línea de tendencia nlog(n) puse la de log(n). También vemos una considerable reducción en el tiempo de ejecución respecto al reto 1. El archivo large se carga más de 250 veces más rápido en reto 2. La complejidad del peor caso de daría si todos los scorers y además

todos tienen los puntos que nos interesan, es decir, es poco probable. Esta complejidad terminaría siendo un n que finalmente se volvería un nlog(n) debido al merge sort.