Análisis de los resultados: Reto 2

Estudiante A: Santiago Ballén Cod 202023544

Estudiante B: Paola Campiño Cod 202020785

Reto 2:

https://github.com/EDA2021-1-SEC07-G-2Grupo/Reto2-S07-G02.git

El documento a continuación presenta el análisis en los tiempos de ejecución y el consumo datos para este reto 2. Cabe resaltar que para el conteo del tiempo de respuesta se usó la función time.perf_couter y que las pruebas de los requerimientos 1,3,4 y la carga de datos se realizaron con la maquina 1 y las pruebas del requerimiento 2 se realizó con la maquina 2.

	Máquina 1	**Máquina 2**
Procesadores	AMD Ryzen 5 2500U with radeon Vega Mobile Gfx, 2.00GHz	Intel Core i3-3220 CPU @ 3.30GHz
Memoria RAM (GB)	6,00GB (5,65GB utilizable)	4,00 GB (3,89 GB utilizable)
Sistema Operativo	Sistema operativo de 64 bits, procesador x64	Sistema operativo de 64 bits, procesador x64

Carga de Datos:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
1327342.80	88252.18 (1.47 m)

Es importante resaltar que para este reto 2 fue necesaria la creación de 3 mapas y una arreglo de tipo “ARRAY LIST” ya que en los laboratorios pasados se había visto tiempos menores en este tipo de lista a comparación del tipo “SINGLE_LINKED”.

```
catalog['video'] = lt.newList('ARRAY_LIST', comparevideoIds)
```

```
    catalog['category_id'] = mp.newMap(100,
                                       maptype='PROBING',
                                       loadfactor=0.5,
                                       comparefunction=comparebyName)
    catalog['country'] = mp.newMap(34,
                                       maptype='PROBING',
                                       loadfactor=0.5,
                                       comparefunction=comparebyName)
    catalog['videos_by_category_id'] = mp.newMap(80,
                                       maptype='PROBING',
                                       loadfactor=0.5,
                                       comparefunction=comparebyINt)
```

Algo muy importante a resaltar de está creación del catalogo es que se decidió usar el tiempo de mapa linear probing con un factor de carga de 0.5. Pues se encontró que los tiempos de respuesta con este tipo de mapa tienen tiempos de espera menores a con los de chaining. Cabe resaltar que el tamaño de los mapas es mayor a la cantidad de elementos de espera ya que se quiere disminuir la cantidad de veces que se haga rehash pues esto puede llegar a aumentar el tiempo de respuesta del programa, así como la complejidad.

Requerimiento 1: Consultar n número de videos más vistos por país y en una categoría especifica.

Tiempo total de respuesta y el consumo de datos de este requerimiento es:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
30	24216.43

Para hacer el conteo del tiempo de ejecución se decidió eliminar los inputs, por lo que para hacer las pruebas optó por usar los siguientes datos:

País de búsqueda	Categoría	Numero de videos
japan	Music	10

```
if    .contains(catalog["country"], pei)== False
            print_separador()
            print("No se ha encontrado ningun video del país "+ str(pei))
            print_separador()
```

```
else:
            videos_por_pais=(mp.get(catalog["country"],pei))

            #categ=str(input("Escriba la categoria de los videos que desea consultar: \n"
))
            categ=" "+"Music"
            if mp.contains(catalog["category_id"],str(categ))==False:
                print_separador()
                print("No se ha encontrado ninguna categoría "+ str (categ))
                print_separador()
            else:
                num=mp.get(catalog["category_id"],categ)
                Vids_por_pais_categ=controller.Getvideosbycateg(videos_por_pais,num["valu
e"]["category_id"])
                if lt.isEmpty(Vids_por_pais_categ)==True:
                    print(lt.size(Vids_por_pais_categ))
                    print("No se ha encontrado videos de la categoría "+str(categ)+ " del
 país "+ str(pei))
                else:
                    print("Se ha encontrado un total de "+ str(lt.size(Vids_por_pais_cate
g))+ " videos.")
                    #n=int(input("Escriba la cantidad de videos que desea consultar\n"))
                    n=10
                    lista_organizada=controller.videos_por_algo(Vids_por_pais_categ,n,"vi
ews")
                    imprime_toda_lista_econtrada_req1(lista_organizada,n)
```

Este requerimiento está dividido en 3 partes:

- La primera es consultar el mapa que tenga como llave el país a consultar:

```
-   if mp.contains(catalog["country"], pei)== False:
-               print_separador()
-               print("No se ha encontrado ningun video del país "+ str(pei))
-               print_separador()
-   else:
-                videos_por_pais=(mp.get(catalog["country"],pei))
```

En este punto se hiso uso de la función mp.get(map,key) función que recibe un mapa y una llave para devolver un arreglo con todos los datos asociados a una llave. La complejidad en esta parte del el codigo es de O(k) ya que solo se está llamando a una llave que se encuentra en un mapa.

Hasta este punto la complejidad total es de: O(k)

El tiempo de respuesta y el consumo de datos tiene los siguientes valores:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
1.02	0.69

- Los segundo es filtrar los videos por categorías:

```
-    categ=" "+"Music"
-                if mp.contains(catalog["category_id"],str(categ))==False:
-                    print_separador()
-                    print("No se ha encontrado ninguna categoría "+ str (categ))
-                    print_separador()
-                else:
-                    num=mp.get(catalog["category_id"],categ)
-                    Vids_por_pais_categ=controller.Getvideosbycateg(videos_por_pais,nu
    m["value"]["category_id"])
```

En está segunda parte lo primero que se hace es obtener del usuario la categoría a consultar. Se hace uso de la función get para obtener un arreglo con todos los datos asociados a está llave, ya que nos interesa obtener numero asociado a esa categoría.

Después de conseguir esos datos se hace uso de la función `Getvideosbycateg(array,category)`, función que permitirá separar a todos los videos con la categoría de interés.

```
def Getvideosbycateg(catalog,numero):
    lista_filtrada=lt.newList("ARRAY_LIST")
    i = 1
    while i <= (lt.size(catalog["value"]["video"])):
        wow=lt.getElement(catalog["value"]["video"],i)
        if numero==wow["category_id"] or str(numero)== wow ["category_id"]:
            lt.addLast(lista_filtrada, wow)
        i+=1

    return lista_filtrada
```

Esta función lo que hace es ir por cada uno de los videos y acceder a su category_id si el valor del elemento es igual a la categoría que se busca, entonces el video se añade a una lista nueva. Este codigo tiene una complejidad de O(n) siendo n el total de elementos en el mapa consultado

La complejidad total hasta este punto es de: O(N)

El tiempo de respuesta y el consumo de datos tiene los siguientes valores:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
12.01	1627.30

- Lo tercero es ordenar la lista por "views":

Para organizar la lista por views se hizo uso de la siguiente función, la cual pide el n numero de videos que el usuario desea consultar, la lista que hay que ordenar y en base a que organiza.

```
-    lista_organizada=controller.videos_por_algo(Vids_por_pais_categ,n,"vi
    ews")
```

En el controller se establece cual función se va usar para comparar:

```
def videos_por_algo(catalog,size,comparacion):
    if comparacion=="views":
        def cmpVideosByviews(video1, video2):

            return (float(video1["views"]) > float(video2["views"]))
        comparacion=cmpVideosByviews
    elif comparacion=="dias":
         def cmpVideosBydias(video1, video2):
             return (float(video1["dias"]) > float(video2["dias"]))
         comparacion=cmpVideosBydias

    elif comparacion=="likes":
         def cmpVideosBylikes(video1, video2):

             return (float(video1["likes"]) > float(video2["likes"]))
         comparacion=cmpVideosBylikes
```

En este caso como es por views se va a establecer que la función debe ser `cmpVideosByviews`.Lo siguiente que va a pasar es que que se va a usar la función merge sort para organizar los datos por views.

```
def videos_por_algo(catalog,size,comparacion):
    sub_list = lt.subList(catalog,0, size)
    sub_list = catalog.copy()
    sorted_list=merg.sort(sub_list, comparacion)
    return  sorted_list
```

Hacer un merge sort tiene una complejidad de: L log (L) en este caso L es numero menor al n mencionado anteriormente la lista fue filtrada.

El tiempo y los datos necesarios fueron los siguientes:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
16.09	143.65

Complejidad total:

La complejidad total es de: O(N + N log(L)) siendo N el total de Objetos encontrados en un mapa y L ≤ N.

Esta complejidad es mucho menor a la que se dio en el reto 1 pues este primer requerimiento tenía una complejidad de: O(3N + N(logN))

Requerimiento 2: Consultar el video más trending por país

Requerimiento 3: Consultar el video más trending por categoría

Tiempo total de respuesta y el consumo de datos de este requerimiento es:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
2492.91	666741.13

Para hacer el conteo del tiempo de ejecución se decidió eliminar los inputs, por lo que para hacer las pruebas optó por usar los siguientes datos:

Categoria
Music

```
categoria=" "+ "Music"
        num=mp.get(catalog["category_id"],categoria)
        num=num["value"]["category_id"]
        print_separador()
        if mp.contains(catalog["videos_by_category_id"],str(num))==False:
            print_separador()
            print("No se ha encontrado la catgoría "+str(categoria))
            print_separador()
        else:

            Solo_por_categoría=mp.get(catalog["videos_by_category_id"],num)
            print_separador()
```

```
            agrupacion=controller.agrupacion_id(Solo_por_categoría["value"][
"video"])

            print_separador()
            print("Cargando información de los archivos ....")
            lista_organizada=controller.videos_por_algo(agrupacion,5,"dias")
            print_separador()
            print_req3(lista_organizada)
            print_separador()
```

complejidad total es de: O (N log N +N)

Este requerimiento está dividido 3 partes:

- La primera es consultar el mapa que tenga como llave el país a consultar:

```
-   categoria=" "+ "Music"
-           num=mp.get(catalog["category_id"],categoria)
-           num=num["value"]["category_id"]
-           print_separador()
-           if mp.contains(catalog["videos_by_category_id"],str(num))==Fal
    se:
-               print_separador()
-               print("No se ha encontrado la catgoría "+str(categoria))
-               print_separador()
-           else:
-
-               Solo_por_categoría=mp.get(catalog["videos_by_category_id"]
    ,num)
```

En esta primera parte se está haciendo uso de la función mp.get(map,key) para obtener todos los datos asociados a la llave que has ido pedida por el usuario que es la el numero de la categoría. Los siguientes datos son el tiempo de ejecución y el consumo de datos:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
0.17	1.26

Hasta este punto la complejidad total del programa es de: O(K)

- La segunda parte del programa está en que se debe agrupar los videos repetidos

Para este punto se hizo uso de la función
`controller.agrupacion_id(Solo_por_categoría["value"]["video"]:`

```
def agrupacion_id(catalog):
        lista_prohibido=lt.newList(datastructure="ARRAY_LIST")
        lista_organizada=lt.newList(datastructure="ARRAY_LIST")
        i=1
        while i <= lt.size(catalog):
            videos=lt.getElement(catalog,i)
            ID=videos["video_id"]
            dato={"ID": ID,"title":videos["title"], "Channel title": videos[
"channel_title"], "country":videos["country"],"dias":1}
            precencia=lt.isPresent(lista_prohibido,ID)
            if precencia>0:
                precencia_en_la_main=lt.getElement(lista_prohibido,precencia
+1)
                el_cambio=lt.getElement(lista_organizada,int(precencia_en_la
_main))
                el_cambio["dias"]+=1
            else:
                lt.addLast(lista_prohibido,ID)
                lt.addLast(lista_organizada,dato)
                a=str(lt.size(lista_organizada))
                lt.addLast(lista_prohibido,a)
            i+=1
        return lista_organizada
```

Los tiempos de repuesta y consumo de datos de la función son los siguientes:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
2473.67	690458.03

La complejidad de este código es de O(N) siendo N el número de videos encontrados en el mapa.

Hasta este punto la complejidad total es de: O(N) siendo N el número de videos encontrados en el mapa

- Lo tercero es ordenar la lista por “dias”:

En este punto se usa la función merge sort y con un compare función en la que se acceda a los likes.

El tiempo de ejecución y consumo de datos es el siguiente:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
6.02	1182.29

Un merge sort tiene una complejidad de: O (N log N)

Complejidad total:

La complejidad total es de: O (N log N +N)

Esta complejidad es mucho menor a la que se dio en el reto 1 pues este tercer requerimiento tenía una complejidad de: O (2N+ N(logN)).

Requerimiento 4: Consultar los n videos con más “likes” por 'tag' de acuerdo con un país especifico

Tiempo total de respuesta y el consumo de datos de este requerimiento es:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
19.65	2375.44

Para hacer el conteo del tiempo de ejecución se decidió eliminar los inputs, por lo que para hacer las pruebas optó por usar los siguientes datos:

País de búsqueda	Tag	Numero de videos
japan	2018	10

```
        if mp.contains(catalog["country"],pais)==False:
            print_separador()
            print("No se ha encontrado el pais que está cosultado ")
            print_separador()
        else:
            lista_por_pais=mp.get(catalog["country"], pais)
            #categoria_cosultar=str(input("Escriba el tag que desea consulta
r "))
            categoria_cosultar="2018"
```

```
            lista_filtrada=controller.get_video_by_tag(lista_por_pais,catego
ria_cosultar)
            if lt.size(lista_filtrada)==0:
                print_separador()
                print("No se a encontrado ningún video con el tag "+str(cate
goria_cosultar) )
                print_separador()
            else:
                print("Se ha encontrado un total de "+str(lt.size(lista_filt
rada)))
                #n=int(input("Escriba la cantidad de videos que desea consul
tar "))
                n=10
                lista_organizada=controller.videos_por_algo(lista_filtrada,n
,"likes")
                print_req4(lista_organizada,n)
```

Este requerimiento está dividido en 3 partes:

- La primera es consultar el mapa que tenga como llave el país a consultar:

Esto se hace con la función mp.get de la librería map. El tiempo de ejecución y consumo de datos es el siguiente:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
0.17	6.63

El conseguir una información del mapa tienen una complejidad de O(K)

- La segunda parte es revisar cada uno de los video con el fin de encontrar el tag

Para lograr esto se hace uso de la función get_diveo_by_tag que lo que hace es consultar en cada uno de los elementos de la lista siniestrada ver si hay algún tag similar o igual al que ha dsido suministrado por el usurio.

```
-   def get_video_by_tag(catalog,tag):
-       lista_filtrada=lt.newList("ARRAY_LIST")
-       i=1
-       while i <= lt.size(catalog["value"]["video"]):
-           a =lt.getElement(catalog["value"]["video"],i)
-           if tag in a["tags"]:
-               lt.addLast(lista_filtrada,a)
-           i+=1
-       return lista_filtrada
```

El tiempo de ejecución y consumo de datos es el siguiente:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
11.17	3939.34

Esta segunda parte del condigo individualmente tiene una complejidad de: O(N) siendo N el número de elementos encontrados en el mapa especificado por el usuario

- Lo tercero es ordenar la lista por “likes”:

En este punto se usa la función merge sort y con un compare función en la que se acceda a los likes.

El tiempo de ejecución y consumo de datos es el siguiente:

Consumo de Datos[kB]	Tiempo de Ejecución [ms]
6.02	1.49

En esta última parte se da un merge sort el cual tiene una complejidad de: O(L log L) siendo $L \leq N$ (N es el número de elementos encontrados asociados a una llave en un mapa especificado por el usuario)

Complejidad total:

La complejidad total de este requerimiento es de: O(N +L log L) siendo $L \leq N$ (N es el número de elementos encontrados asociados a una llave en un mapa especificado por el usuario)

Esta complejidad es mucho menor a la que se dio en el reto 1 pues este cuarto requerimiento tenía una complejidad de: O (3N + NlogN).