ANALISIS DE RESULTADOS RETO 2

Estudiante 1: ANDRÉS FELIPE PÉREZ MARTÍNEZ - 202215659
Estudiante 2: MATEO PARRA OCHOA - 202213933
Estudiante 3: CARLOS DAVID CASADIEGO PÉREZ - 202212187

Carga De Datos

Descripción de las pruebas de tiempos de ejecución y memoria utilizada.

Entrada	**Tiempo (ms)**	**Consumo de Datos (kB)**
small	75.06	617.24
5pct	116.88	621.54
10pct	42.15	2218.05
20pct	44.11	672.59
30pct	66.01	594.43
50pct	75.62	594.29
80pct	76.77	619.69
large	37.09	2218.05

Requerimiento 1

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Anio: El usuario inserta una fecha específica de formato AAAA, y se espera que busque contenido para esa fecha y la retorne. Catalog: Aquí se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, Directores, Country, entre otros)
Salidas	NoPeliculaspresentes: El contador que indica el número de películas sobre la fecha dada por parámetro. Pelispresent: La lista con las peliculas filtradas por la fecha que dio el usuario. Ordenadas alfabéticamente de forma ascendente

Implementado (Sí/No)	Si se implementó y se hizo conjuntamente por los integrantes

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```python
# REQUERIMIENTO 1
def peliculasbyanio(catalog, anio):
    pelispresent=lt.newList("ARRAY_LIST")
    NoPeliculaspresentes= 0
    exist=mp.contains(catalog["Years"], anio)
    if exist:
        year=mp.get(catalog["Years"], anio)
        shows_year= me.getValue(year)["shows"]
        NoPeliculaspresentes=lt.size(shows_year)
        pelispresent= sortPeliculasbyanio(shows_year)
    return NoPeliculaspresentes, pelispresent
```

Pasos	**Complejidad**
Paso 1: Por medio de la función mp.contains() se comprueba que el año consultado se encuentra en el map catalog[“Years”].	O(1)
Paso 2: Se accede a la pareja llave-valor correspondiente al año consultado, por medio de la función get. Se accede al valor de la pareja llave-valor extraída del mapa por medio de la función getValue	O(1)
Paso 3: Se hace un size del valor extraído (que es una lista con la información de las películas correspondientes al año buscado), para conocer cuantas películas corresponden al año consultado	O(1)
Paso 4: Se hace un ordenamiento alfabético de la lista extraída por medio del algoritmo Merge Sort	O(NlogN)
TOTAL: 4 pasos	***O(NlogN)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Pruebas realizadas con inputs: 1999

Entrada	**Tiempo (ms)**	**Consumo de Datos (kB)**

small	0.148	0.188
5pct	0.196	0.734
10pct	0.174	0.188
20pct	0.263	0.188
30pct	0.285	0.188
50pct	0.219	0.188
80pct	0.293	0.188
large	0.385	3.586

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	0.148	2.188
5pct	0.196	0.734
10pct	0.174	0.188
20pct	0.263	0.188
30pct	0.285	0.188
50pct	0.219	0.188
80pct	0.293	0.188
large	0.385	3.586

Análisis

Comparación De Complejidad Temporal Obtenida en Reto 1 y Reto 2

COMPLEJIDAD TEMPORAL RETO 1: ***O(NlogN)***

COMPLEJIDAD TEMPORAL RETO 2: ***O(NlogN)***

Análisis

Tanto en Reto 1 como en Reto 2, obtenemos una complejidad temporal de O(NlogN) (Donde N en reto 2 representa el tamaño de la lista que contiene la información de las películas donde se encuentra el año consultado y en Reto 1, el tamaño de la lista en donde están cargadas las películas de cada plataforma) debido a que, en ambas implementaciones, se realiza un ordenamiento de los datos por medio del algoritmo Merge Sort, que tiene una complejidad temporal de O(NlogN). Igualmente, podemos concluir que, en ambas implementaciones, ninguna operación tiene una complejidad temporal superior a la calculada que represente un cambio significativo en los tiempos de ejecución obtenidos.

Requerimiento 2

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Date_1: Es la fecha que indica el usuario por parámetro en "%B %d, %Y" y por la cual se va a trabajar para cumplir el objetivo del requerimiento sobre buscar contenido (Shows de Televisión) agregado en la fecha ingresada. Catalog: Aquí se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, Directores, Country, entre otros)
Salidas	List: La lista que contiene el contenido filtrado por la fecha dada por el usuario y ordenada de forma alfabéticamente según lo requerido.
Implementado (Sí/No)	Si se implementó y se hizo conjuntamente por los integrantes del grupo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```
# REQUERIMIENTO 2
def Show_by_time(catalog,date1):
    date1 = time.strptime(date1,"%B %d, %Y")
    list1 = lt.newList("ARRAY_LIST")
    exist=mp.contains(catalog["Dates"], date1)
    if exist:
        date=mp.get(catalog["Dates"], date1)
        shows_date= me.getValue(date)["shows"]
        list1= sortPeliculasbyanio(shows_date)
    return list1
```

Pasos	**Complejidad**
Paso 1: Se convierte la fecha ingresada a formato "%B %d, %Y" para hacer las comparaciones.	O(1)
Parte 2: Por medio de la función mp.contains() se comprueba que la fecha consultada se encuentra en el map catalog["Dates"].	O(1)
Paso 3: Se accede a la pareja llave-valor correspondiente a la fecha consultada, por medio de la función get. Se accede al valor (lista con todos los Shows TV correspondientes a la fecha consultada) de la pareja llave-valor extraída del mapa por medio de la función getValue	O(1)

Paso 4: Se hace un ordenamiento alfabético de la lista extraída por medio del algoritmo Merge Sort	O(NlogN)
TOTAL: 4 pasos	

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Pruebas realizadas con el input: November 12, 2019

Entrada	**Tiempo (ms)**	**Consumo de Datos (kB)**
small	0.406	2.117
5pct	0.439	2.883
10pct	0.678	4.211
20pct	0.692	4.875
30pct	0.851	5.539
50pct	1.657	5.539
80pct	2.085	6.203
large	2.730	6.203

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	0.406	2.117
5pct	0.439	2.883
10pct	0.678	4.211
20pct	0.692	4.875
30pct	0.851	5.539
50pct	1.657	5.539
80pct	2.085	6.203
large	2.730	6.203

Análisis

Comparación De Complejidad Temporal Obtenida en Reto 1 y Reto 2

COMPLEJIDAD TEMPORAL RETO 1: ***O(NlogN)***

COMPLEJIDAD TEMPORAL RETO 2: ***O(NlogN)***

Análisis

Tanto en Reto 1 como en Reto 2, obtenemos una complejidad temporal de O(NlogN) (Donde N representa el tamaño de la lista que contiene la información de los TV Shows donde se encuentra la

fecha consultada consultado) debido a que, en ambas implementaciones, se realiza un ordenamiento de los datos por medio del algoritmo Merge Sort, que tiene una complejidad temporal de O(NlogN). Igualmente, podemos concluir que, en ambas implementaciones, ninguna operación tiene una complejidad temporal superior a la calculada que represente un cambio significativo en los tiempos de ejecución obtenidos.

Requerimiento 3

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Actor_name: El usuario digita el nombre de un actor que desee buscar y el contenido que esté relacionado con este. Catalog: Aqui se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, Directores, Country, entre otros)
Salidas	Shows_act: Lista que contiene la información relacionada a los shows de televisión y las películas en las que se encuentra el actor dado por parámetro por el usuario. Num_peliculas: Contador que indica cuantas películas tienen a este actor entre su reparto. Num_programas: Contador que indica cuantos shows tienen a este actor entre su reparto.
Implementado (Sí/No)	Si se implementó y fue hechos por el Estudiante 1: Andrés Felipe Pérez Martínez.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```python
# REQUERIMIENTO 3
def Shows_by_Actor(catalog, actor_name):
    shows_act= lt.newList("ARRAY_LIST")
    Num_peliculas= 0
    Num_programas= 0
    exist=mp.contains(catalog["Actores"], actor_name)
    if exist:
        actor=mp.get(catalog["Actores"], actor_name)
        shows_actor= me.getValue(actor)["shows"]
        for show in lt.iterator(shows_actor):
            if show["type"]=="Movie":
                Num_peliculas+=1
            else:
                Num_programas+=1
        shows_act= sortShowsbyTitle(shows_actor)
    return shows_act, Num_peliculas, Num_programas
```

Pasos	Complejidad
Paso 1: Verifica si el nombre del actor existe en el mapa catalog [“Actores”] por medio de la función mp.contains()	***O(1)***
Paso 2: Se accede a la pareja llave-valor correspondiente al actor consultado, por medio de la función get. Se accede al valor (lista con todos los Shows TV y películas correspondientes al actor consultado) de la pareja llave-valor extraída del mapa por medio de la función getValue	***O(1)***
Paso 3: Recorre por medio de un for la lista del contenido donde aparece el actor, y con base a eso ejecuta un conteo de películas y shows por aparte.	***O(N)***
Paso 4: Se hace un ordenamiento por año de lanzamiento de la lista extraída por medio del algoritmo Merge Sort	***O(NlogN)***
TOTAL 4 pasos	***O(NlogN)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Pruebas realizadas con el input: Sonia Agarwal

Entrada	Tiempo (ms)	Consumo de Datos (kB)

small	0.124	1.062
5pct	0.144	1.086
10pct	0.168	1.086
20pct	0.194	1.250
30pct	0.189	3.109
50pct	0.170	3.773
80pct	0.190	3.773
large	0.183	3.773

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	0.124	1.062
5pct	0.144	1.086
10pct	0.168	1.086
20pct	0.194	1.250
30pct	0.189	3.109
50pct	0.170	3.773
80pct	0.190	3.773
large	0.183	3.773

Análisis

Comparación Tiempos de Ejecución Reto 1 y Reto 2

Tiempos Obtenidos En Reto 1

Pruebas realizadas con el input: Sonia Agarwal

Entrada	Tiempo (ms)
small	0,38
5pct	1,74
10pct	2,87
20pct	2,80
30pct	3,66
50pct	8,94
80pct	9,74
large	10,64

Tiempos obtenidos en Reto 2

Pruebas realizadas con el input: Sonia Agarwal

Entrada	Tiempo (ms)	Consumo de Datos (kB)
small	0.124	1.062
5pct	0.144	1.086
10pct	0.168	1.086
20pct	0.194	1.250
30pct	0.189	3.109
50pct	0.170	3.773
80pct	0.190	3.773
large	0.183	3.773

Análisis

Teniendo en cuenta que tanto en Reto 1 como Reto 2 realizamos pruebas con la misma entrada en diferentes tamaños de archivos, podemos evidenciar como en Reto 2 obtenemos tiempos considerablemente inferiores, puesto que ni siquiera llegan a un milisegundo, lo que, si pasa con los tiempos obtenidos en reto 1, en donde se llegan a tiempos de hasta aproximadamente 10 milisegundos.

Podríamos justificar estos cambios de tiempos, debido a que en reto 1 se debe hacer un recorrido sobre el catálogo "Total" (un ARRAY_LIST en donde se encuentra la información de cada show cargado por cada plataforma), lo que implica multitud de comparaciones tanto para conocer si el actor buscado se encuentra dentro del "cast" del show como para saber si este es de "type" "Movie" o "TV Show". Caso contrario, es lo que pasa en reto 2, en donde la información al estar cargada en una TABLA DE HASH, en la que las llaves era los nombres de los actores y los valores todos los shows correspondientes a ese actor, para acceder a la información y hacer las comparaciones correspondientes, no se necesitó hacer ningún recorrido, ya que al tener el nombre del actor a buscar se accedía directamente a la llave y al valor correspondiente a ese nombre por medio de funciones como el get y el getValue. Esta última función del TAD map, nos permitió acceder sin hacer ningún recorrido a la lista en donde estaban todos los shows relacionados al actor buscado, que solo debió ser recorrida para saber el "type" de cada show presente. Esto significó un ahorro inmenso de tiempo, pues se trabajó sobre tamaños de lista mucho más pequeños que ya estaban filtrados por el nombre del actor buscado.v

Comparación De Complejidad Temporal Obtenida en Reto 1 y Reto 2

COMPLEJIDAD TEMPORAL RETO 1: ***O(NlogN)***

COMPLEJIDAD TEMPORAL RETO 2: ***O(NlogN)***

Análisis

Tanto en Reto 1 como en Reto 2, obtenemos una complejidad temporal de O(NlogN) (Donde N representa el tamaño de la lista que contiene la información de los shows donde se encuentra el actor consultado) debido a que, en ambas implementaciones, se realiza un ordenamiento de los datos por medio del algoritmo Merge Sort, que tiene una complejidad temporal de O(NlogN). Igualmente, podemos concluir que, en ambas implementaciones,

ninguna operación tiene una complejidad temporal superior a la calculada que represente un cambio significativo en los tiempos de ejecución obtenidos.

Requerimiento 4

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Género: El género dado por el usuario por parámetro uy por el cual se va a trabajar para filtrar el contenido relacionado con este género. Catalog: Aquí se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, Directores, Country, entre otros)
Salidas	GenContent: La lista con el contenido relacionado al género dado por parámetro. Este contenido esta ordenado alfabéticamente por el titulo según lo requerido. Películas: Un contador el cual almacena las apariciones del contenido de tipo película que coincida con el género dado por parámetro. Shows: Un contador el cual almacena las apariciones del contenido de tipo shows de TV que coincida con el género dado por parámetro.
Implementado (Sí/No)	Si se implementó y lo hizo el estudiante 2 (Mateo Parra Ochoa)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```python
# REQUERIMIENTO 4
def ContentbyGenero(catalog,genero):
    """
    Filtra el contenido por shows y peliculas que sean sobre un genero en especifico.
    """

    genContent= lt.newList("ARRAY_LIST")
    peliculas= 0
    shows= 0
    exist=mp.contains(catalog["Generos"], genero)
    if exist:
        g=mp.get(catalog["Generos"], genero)
        shows_generos= me.getValue(g)
        for show in lt.iterator(shows_generos):
            if show["type"]=="Movie":
                peliculas +=1
            else:
                shows +=1
        genContent= SortGeneros(shows_generos)
    return genContent, peliculas, shows
```

Pasos	Complejidad
Paso 1: Se crea la lista la cual va a almacenar el contenido relacionado con el género dado por parámetro.	**O(1)**
Paso 2: Se crea una variable donde por medio de la función contains se va a evaluar las apariciones del género.	**O(1)**
Paso 3: Por medio de la función get toma los géneros y los compara con el género dado por parámetro. Después los almacena en una variable.	**O(1)**
Paso 4: Por medio de un for recorre la variable para realizar el conteo sobre el tipo de contenido. Finalmente se aplica un ordenamiento Merge Sort.	**O(NlogN)**
TOTAL 4 pasos	**(NlogN)**

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Inputs utilizados para pruebas: Fantasy

Entrada	Tiempo (ms)	Consumo de Datos (kB)

small	78.98	0.242
5pct	70.42	0.266
10pct	85.17	0.266
20pct	21.78	0.238
30pct	29.72	0.215
50pct	38.60	0.215
80pct	38.81	0.242
large	53.42	0.242

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	78.98	0.242
5pct	70.42	0.266
10pct	85.17	0.266
20pct	21.78	0.238
30pct	29.72	0.215
50pct	38.60	0.215
80pct	38.81	0.242
large	53.42	0.242

Análisis

Comparación Tiempos de Ejecución Reto1 y Reto 2

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Tiempos Obtenidos En Reto 1

Entrada	Tiempo (ms)
small	12,87
5pct	40,32
10pct	57,49
20pct	128,13
30pct	223,03
50pct	470,22
80pct	984,61
large	1485,22

Tiempos Obtenidos En Reto 2

Entrada	Tiempo (ms)	Consumo de Datos (kB)
small	78.98	0.242
5pct	70.42	0.266
10pct	85.17	0.266
20pct	21.78	0.238
30pct	29.72	0.215
50pct	38.60	0.215
80pct	38.81	0.242
large	53.42	0.242

Análisis

Al hacer la comparación de los tiempos obtenidos en las ejecuciones de los Retos 1 y 2, y teniendo en cuenta que se aplicaron las mismas entradas con los mismos tamaños de archivos, podemos asegurar una clara disminución en el tiempo del Reto 2. Esta disminución es atribuida a que la implementación en el reto uno era realizada sobre una estructura del tipo “Array_List”, y este tipo de lista requería la realización de múltiples comparaciones para sacar en este caso el género y el tipo de contenido (TV o Película). Ahora bien, para el Reto numero dos se utilizaron Tablas de Hash, las cuales toman por llave los géneros y los valores las películas y shows relacionados a ese género. Al ser Tablas de Hash permite utilizar funciones como get y getValue las cuales facilitan las búsquedas.

Comparación De Complejidad Temporal Obtenida en Reto 1 y Reto 2

COMPLEJIDAD TEMPORAL RETO 1: ***O(NlogN)***

COMPLEJIDAD TEMPORAL RETO 2: ***O(NlogN)***

Análisis

Para el requerimiento 4, tanto en el reto 1 con en el reto 2 se obtuvo una complejidad de O(NlogN), donde N representa el tamaño de la lista con la información donde se encuentran los géneros. Esta complejidad es atribuida al ordenamiento Merge sort que aplicamos en este requerimiento. Por lo tanto, no represento ningún cambio abrupto en los tiempos de ejecución.

Requerimiento 5

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catalog: Aquí se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, directores, Country, entre otros) Country: El país por el cual se van a filtrar los datos.
Salidas	Country_list: la lista de cada producción filtrada por país y ordenada por título. Movies: número de películas dentro de Country_list TV_Shows: número de series dentro de Country_list
Implementado (Sí/No)	Se implementó y fue hecho por el estudiante 3 (Carlos Casadiego)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```
# REQUERIMIENTO 5

def MoviesByCountry(catalog,country):
    """
    Filtra el contenido por shows y peliculas del mismo pais que el ingresado por parametro.
    """
    country_list= lt.newList("ARRAY_LIST")
    movies= 0
    tv_show= 0
    exist=mp.contains(catalog["Country"], country)
    if exist:
        c=mp.get(catalog["Country"], country)
        shows_country= me.getValue(c)
        for show in lt.iterator(shows_country):
            if show["type"]=="Movie":
                movies +=1
            else:
                tv_show +=1
        country_list= SortCountry(shows_country)
    return country_list, movies, tv_show
```

Pasos	Complejidad
Paso 1: Se crea la lista la cual va a almacenar el contenido relacionado con el país dado por parámetro.	**O(1)**
Paso 2: Se crea una variable donde por medio de la función contains se va a evaluar las apariciones del país.	**O(1)**
Paso 3: Por medio de la función get toma los géneros y los compara con el país dado por	***O(1)***

parámetro. Después los almacena en una variable.	
Paso 4: Por medio de un for recorre la variable para realizar el conteo sobre el tipo de contenido.	***O(N)***
Paso 5 : Finalmente se aplica un ordenamiento Merge Sort.	***O(NlogN)***
Total 5 pasos	***O(NlogN)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (ms)**	**Consumo de Datos (kB)**
small	0.24	3.648
5pct	0.35	5.023
10pct	0.62	5.023
20pct	0.84	5.711
30pct	1.22	6.398
50pct	2.40	7.086
80pct	3.72	7.086
large	4.89	7.773

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	0.24	3.648
5pct	0.35	5.023
10pct	0.62	5.023
20pct	0.84	5.711
30pct	1.22	6.398
50pct	2.40	7.086
80pct	3.72	7.086
large	4.89	7.773

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Reto 1

Entrada	Tiempo (s)
small	3,59
5pct	2,98
10pct	2,35
20pct	3,53
30pct	3,86
50pct	4,96
80pct	7,43
large	9,64

Reto 2

Entrada	Tiempo (ms)	Consumo de Datos (kB)
small	0.24	3.648
5pct	0.35	5.023
10pct	0.62	5.023
20pct	0.84	5.711
30pct	1.22	6.398
50pct	2.40	7.086
80pct	3.72	7.086
large	4.89	7.773

Requerimiento 6

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Director_name: Es el nombre del director de cine, del cual el usuario quiere averiguar el contenido que es dirigido por este director en específico. Catalog: Aquí se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, Directores, Country, entre otros)
Salidas	titles_list: Es la lista que contiene el contenido en el participa el director dado por parámetro con el usuario. Esta lista se ordena de acuerdo al recorrido del catálogo. type_dict: Un diccionario que indica el tipo de contenido en el que aparece el director y si show de Tv o Película.

	service_dict: Un diccionario que indica a que servicio pertenece el contenido con el cual está relacionado el director dado por parametro. listed_in_dict: Un diccionario que indica si el contenido de un género en específico ya a coincidido con el director dado por parámetro, e inicia un contador para tener un orden sobre este.
Implementado (Sí/No)	Si se implementó y se hizo grupalmente.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```python
# REQUERIMIENTO 6
def Shows_by_director(catalog, director_name):
    director_map = catalog["Directors"]
    type_dict = {}
    service_dict = {}
    listed_in_dict = {}
    titles_list = lt.newList()
    exist = mp.contains(director_map, director_name)
    if exist:
        titles_list = me.getValue(mp.get(director_map,director_name))
        for i in lt.iterator(titles_list):
            if i["type"] not in type_dict:
                type_dict[i["type"]] = 1
            else:
                type_dict[i["type"]] += 1
            if i["stream_service"] not in service_dict:
                service_dict[i["stream_service"]] = {"Movie":0,"TV Show":0}
            service_dict[i["stream_service"]][i["type"]] += 1
            for e in i["listed_in"]:
                if e not in listed_in_dict:
                    listed_in_dict[e] = 1
                else:
                    listed_in_dict[e] += 1
        sortShowsbydirector(titles_list)
    return titles_list,type_dict,service_dict,listed_in_dict
```

Pasos	**Complejidad**
Paso 1: Se crea un mapa donde se van a guardar los directores del catálogo en general.	**O(1)**
Paso 2: Por medio de la función contains se compara el nombre del director dado por parámetro y el mapa de directores, buscando coincidencias.	**O(NlogN)**
Paso 3: Crea una lista donde se almacena el contenido filtrado por el director y por medio de un for recorre dicha lista contando el tipo de servicio y el tipo de contenido.	**O(N)**
TOTAL 3 pasos	***O(NLogN)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (ms)	Consumo de Datos (kB)
small	0.42	0.188
5pct	0.328	1.422
10pct	0.344	0.734
20pct	0.360	0.188
30pct	0.358	0.188
50pct	0.379	0.188
80pct	0.283	0.188
large	0.293	0.188

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	0.42	0.188
5pct	0.328	1.422
10pct	0.344	0.734
20pct	0.360	0.188
30pct	0.358	0.188
50pct	0.379	0.188
80pct	0.283	0.188
large	0.293	0.188

Análisis

Comparación Tiempos de Ejecución Reto1 y Reto 2

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Tiempos Obtenidos En Reto 1

Entrada	Tiempo (s)
small	2,30
5pct	2,90

10pct	2,14
20pct	2,14
30pct	2,16
50pct	3,0
80pct	3,40
large	4,10

Tiempos Obtenidos En Reto 2

Entrada	Tiempo (ms)	Consumo de Datos (kB)
small	0.42	0.188
5pct	0.328	1.422
10pct	0.344	0.734
20pct	0.360	0.188
30pct	0.358	0.188
50pct	0.379	0.188
80pct	0.283	0.188
large	0.293	0.188

Análisis

Para el caso del requerimiento número seis podemos decir que la comparación de los tiempos obtenidos en las ejecuciones de los Retos 1 y 2(tomando las mismas entradas con los mismos tamaños de archivos), podemos observar una gran disminución en el tiempo del Reto 2. Esta disminución se da gracias a que la implementación en el reto uno era realizada sobre una estructura del tipo “Array_List”, y este tipo de lista consume una gran cantidad de tiempo para poder realizar múltiples comparaciones para sacar en este caso el director y el tipo de contenido (TV o Película). Ahora bien, para el Reto numero dos se utilizaron Tablas de Hash, las cuales toman por llave el nombre de los directores y los valores son las películas y shows relacionados con dicho director. Al ser Tablas de Hash permite utilizar funciones como get y getValue, contain, entre otras que hacen más eficiente el código y también facilitan las búsquedas.

Comparación De Complejidad Temporal Obtenida en Reto 1 y Reto 2

COMPLEJIDAD TEMPORAL RETO 1: ***O(NlogN)***

COMPLEJIDAD TEMPORAL RETO 2: ***O(NlogN)***

Análisis

Al tomar la complejidad temporal del requerimiento número seis del reto 2 y recordando la complejidad arrojada en el reto 1, observamos que ambos coinciden en el valor de NlogN (Donde N representa el tamaño de la lista que contiene las películas y programas correspondientes al actor consultado). Esta complejidad en ambos casos se debe al ordenamiento aplicado para filtrar los datos del catálogo. El algoritmo de ordenamiento que utilizamos en ambos casos fue MergeSort, y teniendo en cuenta que en el peor de los casos su complejidad equivale a O(NlogN) se desprecia las complejidades de otras funciones del programa como la de get que es O(1) y for que es O(N). Es así como, se puede decir que la no variación en la complejidad temporal implica que los tiempos de ejecución tampoco hayan variado significativamente.

Requerimiento 7

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	N: El número de TV shows y Películas por el cual se quiere hacer el top. Catalog: Aquí se encuentra la información cargada de cada uno de los servicios streaming, además de los mapas creados para cada criterio de búsqueda (Years, Dates, Actores, Géneros, Directores, Country, entre otros)
Salidas	Sublist : Lista de los top N géneros con más apariciones, ordenados por cantidad, adicionalmente con un contador por cada tipo de contenido y tipo de servicio..
Implementado (Sí/No)	Si, se hizo grupalmente

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```python
# REQUERIMIENTO 7
def TOP_genero(catalog,N):
    list = lt.newList("ARRAY_LIST")
    keys = mp.keySet(catalog["Generos"])
    for i in lt.iterator(keys):
        lt.addLast(list,{"genre":i,"titles":me.getValue(mp.get(catalog["Generos"],i))})
    mer.sort(list,cmpreq7)
    sublist = lt.subList(list,1,N)
    for a in lt.iterator(sublist):
        Movie = 0
        shows = 0
        amazon = 0
        netflix = 0
        hulu = 0
        disney = 0
        for i in lt.iterator(a["titles"]):
            if i["type"] == "Movie":
                Movie += 1
            else:
                shows += 1
            if i["stream_service"] == "Hulu":
                hulu += 1
            elif i["stream_service"] == "Disney plus":
                disney += 1
            elif i["stream_service"] == "Netflix":
                netflix += 1
            else:
                amazon += 1
        a["count"] = (Movie,shows,amazon,netflix,hulu,disney)
    return sublist
```

Pasos	Complejidad
Paso1: Se crea una lista donde se almacenarán los géneros presentes en el map catalog[“Géneros”] y la información de los shows asociados.	**O(1)**
Paso 2: Se recorre la lista de llaves del map catalog[“Géneros”] por medio de un for y se almacena, en la lista creada anteriormente, el contenido asociado a cada género.	**O(N)**
Paso 3: Se aplica un ordenamiento de la lista creada en el primer paso y cargada en el segundo, utilizando el algoritmo MergeSort.	**O(MlogM)**
Paso 4: Se crea una sublista de la lista mencionada anteriormente con tamaño correspondiente al Top N ingresado por parámetro.	**O(1)**
Paso 5: Se hace un recorrido for por la sublista creada para extraer la información asociada a cada género (Número de películas y programas asociados, números asociados por plataforma, etc.)	**O(S)**
Paso 6: Se hace un recorrido for por los shows asociados a cada género recorrido por el for anterior. Esto se hace con el objetivo de extraer	**O(T)**

información asociada al “type” y al “stream_service” de cada show	
TOTAL 6 pasos	***O(S.T)***

Nota: El tamaño asociado a T (tamaño de cada lista en la que se encuentra la información de los programas y películas asociadas a un género específico) es considerablemente superior a los asociados a las otras variables. Es por esto, que es importante y relevante en el análisis de complejidad del código.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (ms)**	**Consumo de Datos (kB)**
small	1.456	27.711
5pct	2.175	27.734
10pct	2.824	27.871
20pct	4.031	28.008
30pct	4.491	28.062
50pct	6.671	28.172
80pct	9.515	26.301
large	11.154	28.199

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tabla De Datos

Entrada	Tiempo de Ejecución Real @SC [ms]	Consumo de Datos [kB]?
small	104.35	21.51
5pct	3.33	27.73
10pct	3.62	21.89
20pct	3.45	20.75
30pct	3.79	20.80
50pct	4.53	20.83
80pct	4.76	20.98
large	5.66	20.91

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Comparación Tiempos de Ejecución Reto 1 y Reto 2

Tiempos Obtenidos En Reto 1

Entrada	Tiempo (s)
small	8,21
5pct	24,91
10pct	52,87
20pct	113,18
30pct	165,90
50pct	266,37
80pct	437,91
large	544,55

Tiempos obtenidos en Reto 2

Entrada	Tiempo (ms)	Consumo de Datos (kB)
small	1.456	27.711
5pct	2.175	27.734
10pct	2.824	27.871
20pct	4.031	28.008
30pct	4.491	28.062
50pct	6.671	28.172
80pct	9.515	26.301
large	11.154	28.199

Análisis

Observando los tiempos del requerimiento 7 en cada uno de los retos, se concluye que hubo una evidente disminución en el reto número dos. Esta conclusión es posible de formular partiendo de la idea de que utilizamos las mismas entradas y los mismos tamaños de archivos en ambos retos, por lo que la diferencia radica en la composición del código y los requerimientos específicos, que entre sus amplias diferencias está el tipo de estructura que paso de ARRAY_LIST a TABLAS DE HASH. Esta última se caracteriza por tener algunas funciones tales como get, getvalue, contains, entre algunas otras que facilitan búsquedas y el acceso directo a información sin hacer recorridos con múltiples comparaciones que implican excesos de tiempo que pueden llegar a hacer ineficiente un algoritmo. Es así como, las estructuras de datos utilizadas del primer reto tomaban más trabajo para buscar información, pues realizaban recorridos completos específicamente sobre una lista de gran tamaño en la que se encontraba la información de todas las películas y programas cargadas por cada plataforma, lo que implicaba hacer multitud de comparaciones para buscar el número de apariciones de cada género y su información numérica asociada a cada plataforma y a cada tipo de show.

Comparación De Complejidad Temporal Obtenida en Reto 1 y Reto 2

COMPLEJIDAD TEMPORAL RETO 1: ***O(N^3)***

COMPLEJIDAD TEMPORAL RETO 2: ***O(S.T)***

Al sacar la complejidad temporal del requerimiento número siete en el reto 2, y recordando la complejidad de este mismo requerimiento en el reto 1 podemos analizar que para este caso en particular hubo pequeñas diferencias. Esto se debe a que en ambos casos es necesario el uso de for anidados para hacer el conteo preciso del número de apariciones de cada género y su información numérica asociada al “type” y “stream_service” de cada show relacionado. Las pequeñas diferencias, se resumen en el número y tamaños de las listas que se recorren, ya que, en reto 2 se manejan cuatros listas (N: Tamaño de la lista de las llaves del map “Generos”; M: Tamaño de la lista en la que se almacena la información asociada a cada género; S: Tamaño de la sublista y, T: Tamaño de la lista en donde estaban los shows asociados a un género) cuyo tamaño es inferior al de las listas manejadas en Reto 1, ya que en este caso el contenido en cada una de ellas ya está filtrado por los géneros y la información asociada a cada uno de ellos.