ANÁLISIS DEL RETO

Manuel Caro, 202020303, m.caror@uniandes.edu.co

Santiago Flórez Castañeda, 201912420, s.florezc@uniandes.edu.co

Elkin Rafael Cuello Romero, 202215037, e.cuello@uniandes.edu.co

Requerimiento 1

Descripción

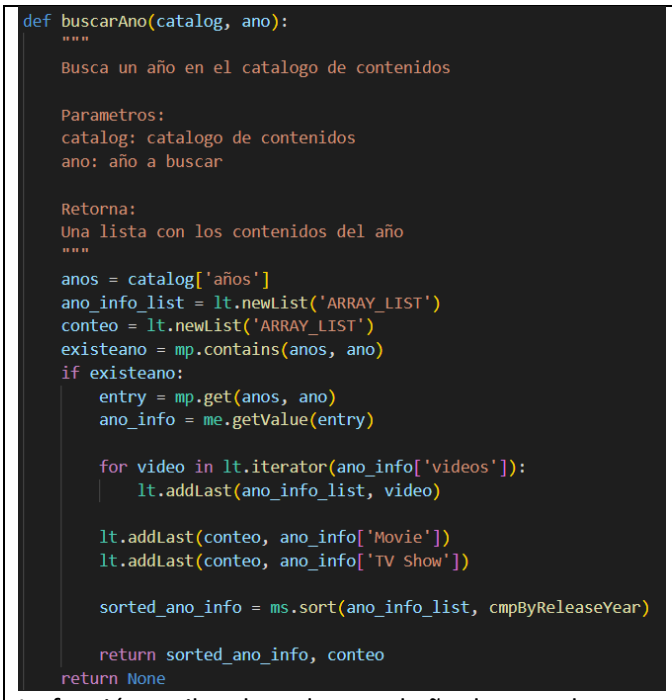
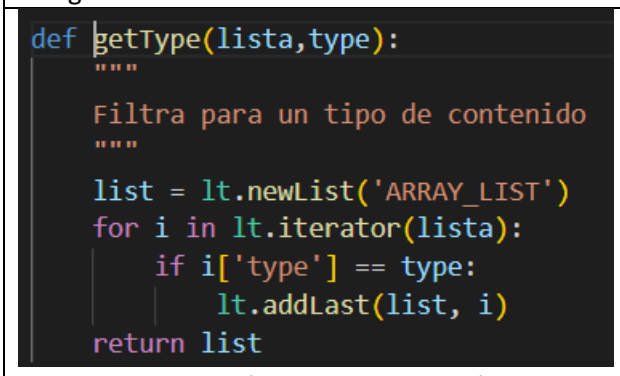
El código hace un recorrido por todo el contenido almacenado en un mapa con los años como llaves. Al iterar, este revisa si una película dada cumple la condición de que su año de lanzamiento es igual al año ingresado por el usuario. Si dicha condición se cumple, esta película se agrega a una lista con formato ARRAY_LIST. Para ordenar los datos de dicha lista se decidió usar Mergesort, pues según las pruebas del laboratorio era el más estable, y más rápido en general para todas las máquinas. Adicionalmente, en todos los requisitos se optó por manejar mapas de tipo PROBING con un factor de carga de 0.5 que se encontró ser el más eficiente de acuerdo con las pruebas llevadas a cabo. Finalmente se recorre esta lista comprobando que contenido pertenece a la categoría Movies, y se almacena en una nueva lista que cumple el requerimiento.

Entrada	Año de consulta
Salidas	Tabla con las 3 primeras y 3 últimas películas estrenadas en ese año, y el número total de películas estrenadas en ese año
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

 La función recibe el catalogo y el año de consulta, comprueba si el año ingresado existe en el mapa previamente creado con todos los años, en caso de que no exista, no devuelve nada. En caso de que, si existe, se extrae la llave con todos sus valores y se almacenan en una lista que es organizada con Mergesort.	O(N)
 Posteriormente se filtra para obtener únicamente las películas en una lista	O(n)
TOTAL	***O(N)***

Pruebas Realizadas

Se escogió el año 1999 como año de consulta para realizar las distintas pruebas.

Para la medición del tiempo y de la memoria usada se usaron las librerías “time” y “tracemalloc” respectivamente.

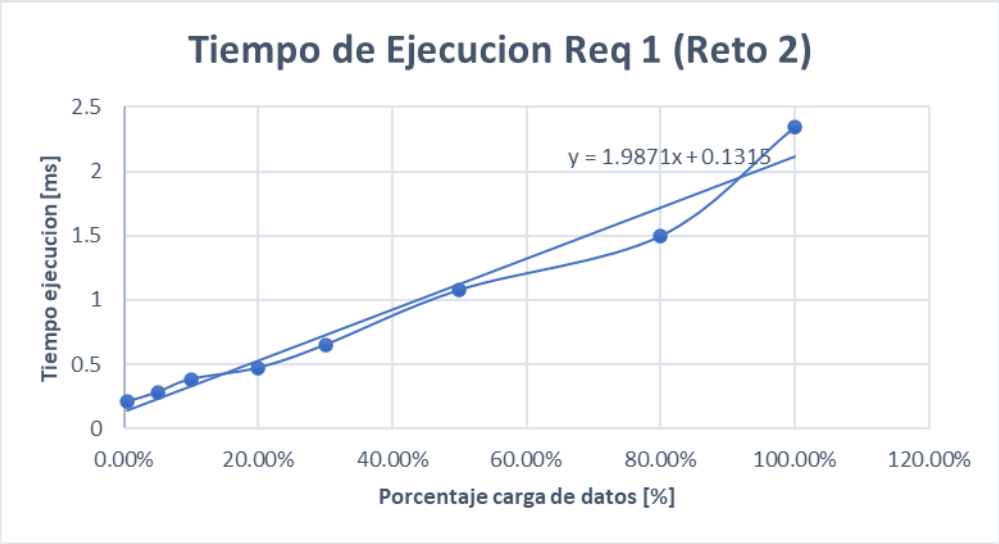
Procesador Intel Core i7 11th Gen Memoria RAM (GB) 16.0 GB Sistema Operativo Windows 10 Pro.

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño del archivo	Tiempo de Ejecución Real @LP [ms]	Consumo de Datos [kB]
0,5%	0.209	3.156
5%	0.283	5.188
10%	0.385	5.25
20%	0.473	6.094
30%	0.651	6.969
50%	1.079	7.938
80%	1.498	8.281
100%	2.346	9.719

Graficas



Análisis

Los resultados muestran una complejidad parcialmente lineal, lo que es congruente con la complejidad esperada, las variaciones se pueden deber a que el numero de años totales en los que se estreno al menos una película no se distribuye uniformemente entre los distintos tamaños de archivos.

Requerimiento 2

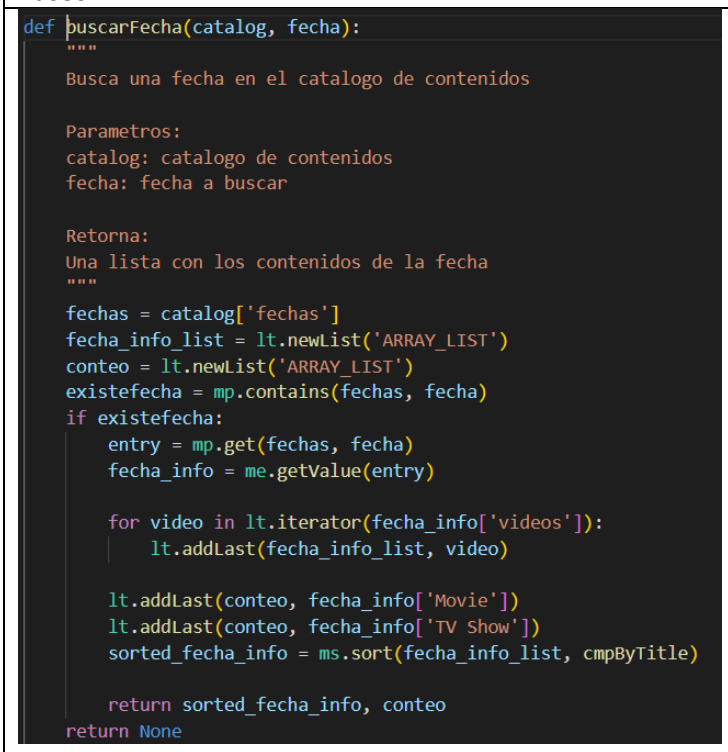
Descripción

Para este requerimiento se decidió usar la librería datetime con el fin de convertir en este formato todos los strings de fechas en las cuales las series fueron agregadas a la plataforma. Básicamente el código itera sobre todo el contenido añadido al mapa de fechas. Al iterar, revisa si un contenido especifico dado cumple la condición de que su fecha de lanzamiento es igual a la fecha ingresada por el usuario. Si la condicione se cumple, el contenido se va agregando a una lista, la que posteriormente se filtra para encontrar únicamente las series y se le retorna al usuario. Nuevamente se usa Mergesort para organizar los datos.

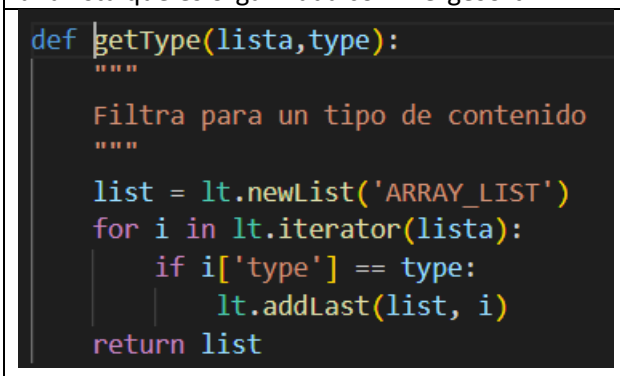
Entrada	Fecha de consulta
Salidas	Tabla con las 3 primeras y 3 últimas series agregadas en la fecha de consulta, y el número total de series agregadas en esa fecha.
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
 La función recibe el catálogo y la fecha de consulta, comprueba si la fecha ya transformada al formato datetime en el archivo de vista ingresado existe en el mapa previamente creado con todos los años, en caso de que no exista, no devuelve nada. En caso de que, si existe,	O(N)

```
def buscarFecha(catalog, fecha):
    """
    Busca una fecha en el catalogo de contenidos

    Parametros:
    catalog: catalogo de contenidos
    fecha: fecha a buscar

    Retorna:
    Una lista con los contenidos de la fecha
    """
    fechas = catalog['fechas']
    fecha_info_list = lt.newList('ARRAY_LIST')
    conteo = lt.newList('ARRAY_LIST')
    existefecha = mp.contains(fechas, fecha)
    if existefecha:
        entry = mp.get(fechas, fecha)
        fecha_info = me.getValue(entry)

        for video in lt.iterator(fecha_info['videos']):
            lt.addLast(fecha_info_list, video)

        lt.addLast(conteo, fecha_info['Movie'])
        lt.addLast(conteo, fecha_info['TV Show'])
        sorted_fecha_info = ms.sort(fecha_info_list, cmpByTitle)

        return sorted_fecha_info, conteo
    return None
```

se extrae la llave con todos sus valores y se almacenan en una lista que es organizada con Mergesort.	
 Posteriormente se filtra para obtener únicamente las series en una lista.	O(n)
TOTAL	***O(N)***

```
def getType(lista,type):
    """
    Filtra para un tipo de contenido
    """
    list = lt.newList('ARRAY_LIST')
    for i in lt.iterator(lista):
        if i['type'] == type:
            lt.addLast(list, i)
    return list
```

Pruebas Realizadas

Se escogió November 12, 2019 como la fecha de consulta para realizar las distintas pruebas.

Para la medición del tiempo y de la memoria usada se usaron las librerías “time” y “tracemalloc” respectivamente.

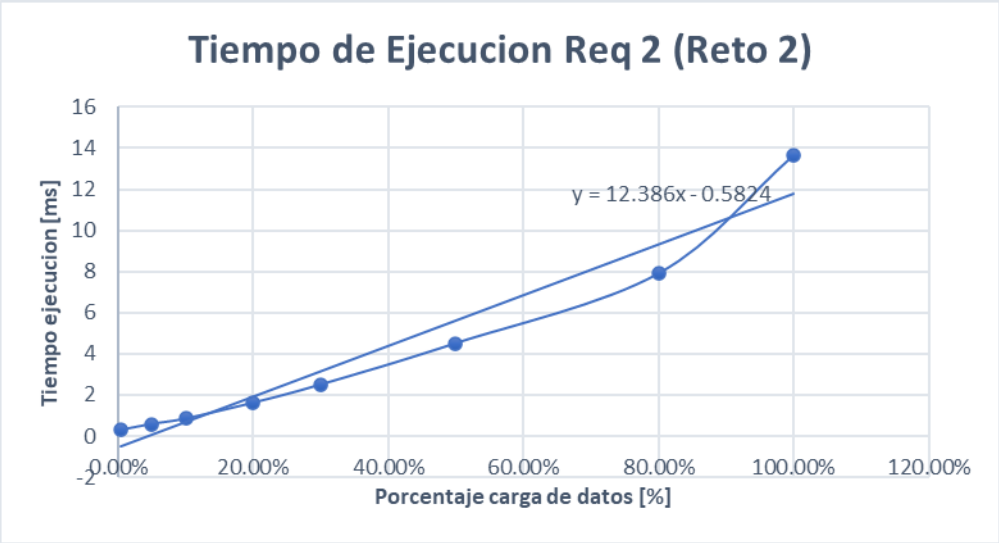
Procesador Intel Core i7 11th Gen Memoria RAM (GB) 16.0 GB Sistema Operativo Windows 10 Pro.

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño del archivo	Tiempo de Ejecución Real @LP [ms]	Consumo de Datos [kB]
0,5%	0.295	5.148
5%	0.581	5.93
10%	0.844	7.68
20%	1.615	8.961
30%	2.501	3.195
50%	4.513	11.426
80%	7.914	14.176
100%	13.677	15.645

Graficas



Análisis

La grafica muestra un comportamiento lineal acorde con la teoría. La desviación que se ve al final de los datos, mas que deberse a otro tipo de complejidad, puede ser causada por diversos factores como podrían ser, momentos en los que se realizaron otras tareas con el computador que comprometieron parte de la memoria asignada a correr el código.

Requerimiento 3

Descripción

Para este requerimiento se recibe una entrada de actor a buscar. Se busca dentro del catálogo cargado y se filtran los datos para añadirlos al mapa correspondiente. Para este requerimiento se emplean contadores en el mapa que se retornan al final. Finalmente, se imprime la información en las tablas. Cabe resaltar que aparte de la función buscarActor() que se describió anteriormente, para la carga de datos se emplean otras dos. Las funciones addActor y newActor cargan las funciones en el mapa al inicio de todo.

Entrada	Nombre del actor a buscar.
Salidas	Contador con el número diferenciado de Movies o TV Show. Además, del catálogo filtrado y ordenado.
Implementado (Sí/No)	Si se implementó, lo realizo Santiago Flórez Castañeda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Paso 1:
Se crean las listas a retornar.
`actores = catalog['actores']`
`actor_info_list = lt.newList('ARRAY_LIST')`
`conteo = lt.newList('ARRAY_LIST')`
`existeactor = mp.contains(actores, actor)`
	O(n+3)

Paso 2:
Se consulta el mapa y se procede a hacer el conteo y agregar la información.
`existeactor = mp.contains(actores, actor)`
`if existeactor:`
`  entry = mp.get(actores, actor)`
`  actor_info = me.getValue(entry)`
`  for video in lt.iterator(actor_info['videos']):`
`  lt.addLast(actor_info_list, video)`
`  lt.addLast(conteo, actor_info['Movie'])`
`  lt.addLast(conteo, actor_info['TV Show'])`
	O(n)

Paso 3:
Se organiza la información (según el criterio) y se retorna.
`  sorted_actor_info = ms.sort(actor_info_list, cmpGenresByReleaseYear)`
`  return sorted_actor_info, conteo`
`return None`
	O(n log(n)) (peor caso)

TOTAL	***O(n)***

Pruebas Realizadas

Para las pruebas se escogió el actor Adam Sandler y se midió el tiempo y el espacio de memoria empleado, este nada más en la función de buscar el actor y el requerimiento.

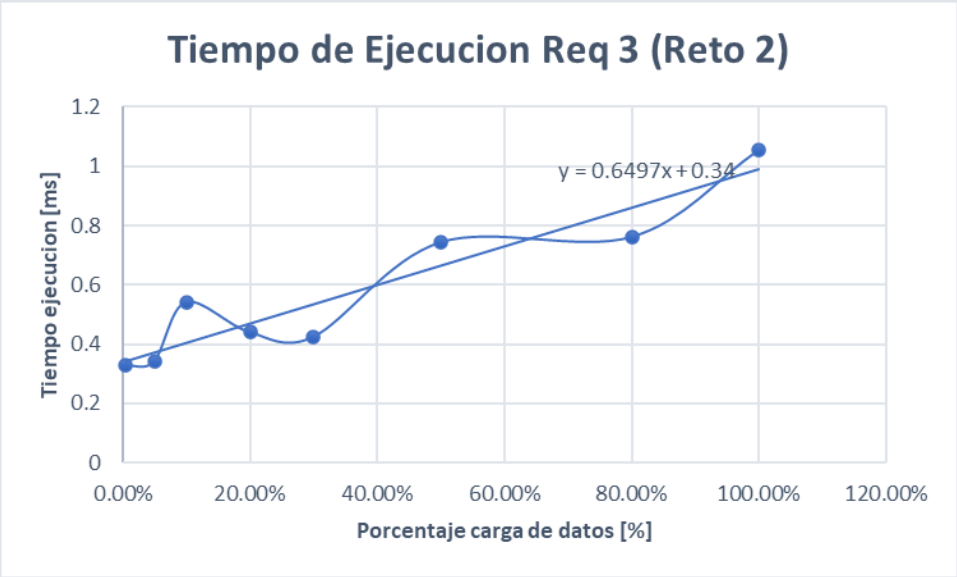
	Especificaciones
Procesadores	Intel CORE I5 10 TH GEN.
Memoria RAM (GB)	8.0 GB
Sistema Operativo	Windows 11 64 bits

Tablas de datos

Tamaño del archivo	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0,5%	0.727	0.328
5%	0.727	0.343
10%	1.844	0.542
20%	1.844	0.441
30%	3.047	0.435

50%	4.344	0.744
80%	4.953	0.762
100%	4.953	1.055

Graficas



Análisis

Se puede observar que la tendencia de la función da lineal, con una pendiente aproximada a 1. Esto quiere decir que el análisis de notación Big O que se hizo tiene sentido. Se intuye que algunas diferencias se deben a distintos procesos que se llevaba a cabo durante las pruebas (tener abierto el explorador, entre otras cosas).

Esta grafica se compara con los datos obtenidos en el mismo requerimiento en el reto 1:

Porcentaje de la Muestra	Tiempo (ms)
Small	0.39
5%	2.98
10%	3.62
20%	5.91
30%	6.81
50%	9.67
80%	15.62
Large	17.43

Donde las pruebas de tiempo eran mucho más largas con el mejor algoritmo (notación O(n^2)). Se concluye que para búsqueda de información una vez ya cargados los datos, la estructura de mapas representa una mayor eficiencia.

Requerimiento 4

Descripción

Para este requerimiento se recibe por parámetro un género que se desea buscar y se imprimen los primeros y 3 últimos datos ordenados que pertenezcan a este género. Para abordar este requerimiento, se creó la función buscarGenero() que recibe por parámetros el mapa de géneros del catálogo general y el género que se está buscando. Para filtrar los datos por género, la función verifica que el género pasado por parámetro exista en el mapa de géneros, y si esto ocurre, itera en la lista de videos del mapa y agrega los datos a una ARRAY LIST para que sea más eficiente de manipular. Luego se agregan los datos del conteo de géneros que también se encuentran en el mapa a otra lista, después se ordenan los datos y finalmente se retorna la lista ordenada de géneros y el conteo de géneros.

Entrada	Género que se desea buscar en la lista
Salidas	La cantidad total de películas y series del género, y una tabla con los primeros y últimos 3 registros de la lista de contenidos del género especificado.
Implementado (Sí/No)	Sí. Implementado por Elkin Rafael Cuello - 202215037

Análisis de complejidad

Pasos	**Complejidad**

Paso 1: Este paso tiene complejidad O(3), porque se ejecutan 3 instrucciones de complejidad O(1)
`generos = catalog['generos']`
`genero_info_list = lt.newList('ARRAY_LIST')`
`conteo = lt.newList('ARRAY_LIST')`
	O(3)

Paso 2: Este paso tiene complejidad O(n) en el peor de los casos. `existegenero = mp.contains(generos, genero)`	O(n)

Paso 3: Este paso tiene complejidad O(n+2) porque se ejecutan 2 instrucciones con complejidad O(1), que son el if y el me.getValue(), y se ejecuta una instrucción que tiene complejidad O(n) en el peor de los casos que es el mp.get().
`if existegenero:`
`  entry = mp.get(generos, genero)`
`  genero_info = me.getValue(entry)`
	O(n+2)

Paso 4: Este paso tiene complejidad O(n), ya que debe iterar por todos los datos.	O(n)

`for video in lt.iterator(genero_info['videos']):` `  lt.addLast(genero_info_list, video)`	

Paso 5: Este paso tiene complejidad O(2) porque se ejecutan 2 instrucciones de complejidad O(1)
`lt.addLast(conteo, genero_info['Movie'])`
`lt.addLast(conteo, genero_info['TV Show'])`
	O(2)

Paso 6: Para este paso se ordena la lista con los datos obtenidos mediante el ordenamiento recursivo merge sort, que en el peor de los casos tiene una complejidad de O(n log n).
`sorted_genero_info = ms.sort(genero_info_list, cmpGenresByReleaseYear)`
`return sorted_genero_info, conteo`
`return None`
	O(n log(n))

TOTAL	***O(n)***

Debido a que se usa la notación Big O, se toma como complejidad del algoritmo O(n) que representa el peor caso.

Pruebas Realizadas

Para las pruebas de tiempos de ejecución, se ejecutó el código correspondiente al requerimiento 4, teniendo en cuenta cada tamaño de archivo. Al ejecutar el requerimiento se utilizó como parámetro para todas las pruebas el género “Drama”, ya que es el género que cuenta con mayor cantidad de contenido, y por lo tanto representa el peor caso que puede ejecutar el programa con los archivos dados. El ordenamiento utilizado fue merge sort. Para la toma de datos de tiempos de ejecución y memoria se utilizaron las librerías “time” y “tracemalloc” respectivamente, y se tomó el tiempo de ejecución del algoritmo del model.py, sin tener en cuenta lo que tardó en imprimir por pantalla.

Especificaciones del equipo:

	Especificaciones
Procesadores	Intel(R) Core(TM) i5-3320M CPU @ 2.60 GHz (4 CPUs), ~2.60 GHz
Memoria RAM (GB)	8.0 GB
Sistema Operativo	Windows 10 Pro 64 bits

Tablas de datos

Tamaño del archivo	**Consumo de Datos [kB]**	**Tiempo de Ejecución Real @LP [ms]**

0,5%	6,30	2,82
5%	9,02	8,00
10%	11,17	18,16
20%	15,86	47,82
30%	20,01	74,21
50%	29,58	136,51
80%	41,67	229,7
100%	50,95	301,55

Graficas

Análisis

Para hallar la complejidad del requerimiento 4 se debe tener en cuenta la función buscarGenero(). El análisis de la complejidad de este algoritmo se realizó mediante la notación Big O, que define el peor caso de ejecución del programa, dando como resultado O(n). En la gráfica obtenida, se puede observar un comportamiento lineal, el cual corresponde con esperado en el análisis de complejidad.

En el reto 1 al realizar este mismo requerimiento con ARRAY LIST se obtuvo una complejidad O(n^2) y la siguiente tabla de tiempos de ejecución.

Porcentaje de la Muestra	Tamaño de la Muestra	Tiempo (ms)
Small	228	1,60
5%	1148	8,92
10%	2298	19,24
20%	4598	40,94
30%	6898	75,48
50%	11498	104,18
80%	18397	143,34
Large	22998	184,44

Comparando los resultados del reto 1 con los obtenidos en el reto 2, se puede concluir que la implementación de ARRAY LIST utilizada en el reto 1 fue más eficiente en tiempos de ejecución a pesar

de tener una complejidad mayor. Esto puede deberse a que en el reto anterior se utilizaba la información directamente del catálogo de lista, y en este caso se debía extraer la información de un mapa de contenido para poder manipular los datos.

Requerimiento 5

Descripción

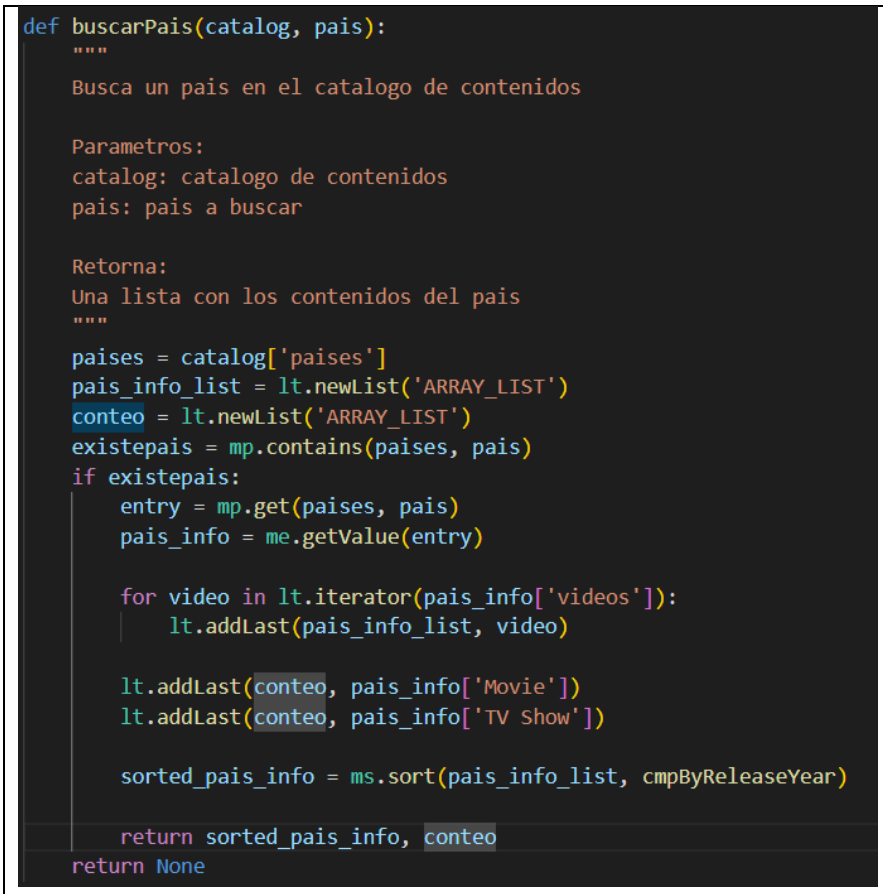
El código hace un recorrido sobre todo el mapa de países de todas las plataformas, y posteriormente busca si existe una llave con el mismo nombre del país. Si el elemento cumple con dicha condición, lo extrae y se colocan todos los valores de dicha llave en una lista. Posteriormente se realiza un ordenamiento por Mergesort, y se usa una función de comparación que lo hace por el año de estreno. Todas las listas en este requerimiento también se manejan con un tipo de lista ARRAYLIST.

Entrada	País de Consulta
Salidas	El número total de películas y programas producidas en dicho país. Y los primeros y últimos 3 elementos de dicha lista.
Implementado (Sí/No)	Si. Manuel José Caro, 202020303, m.caror@uniandes.edu.co

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

	O(N)
TOTAL	***O(N)***

Pruebas Realizadas

Se escogió Germany como país de consulta para realizar las distintas pruebas.

Para la medición del tiempo y de la memoria usada se usaron las librerías “time” y “tracemalloc” respectivamente.

Procesador Intel Core i7 11th Gen Memoria RAM (GB) 16.0 GB Sistema Operativo Windows 10 Pro.

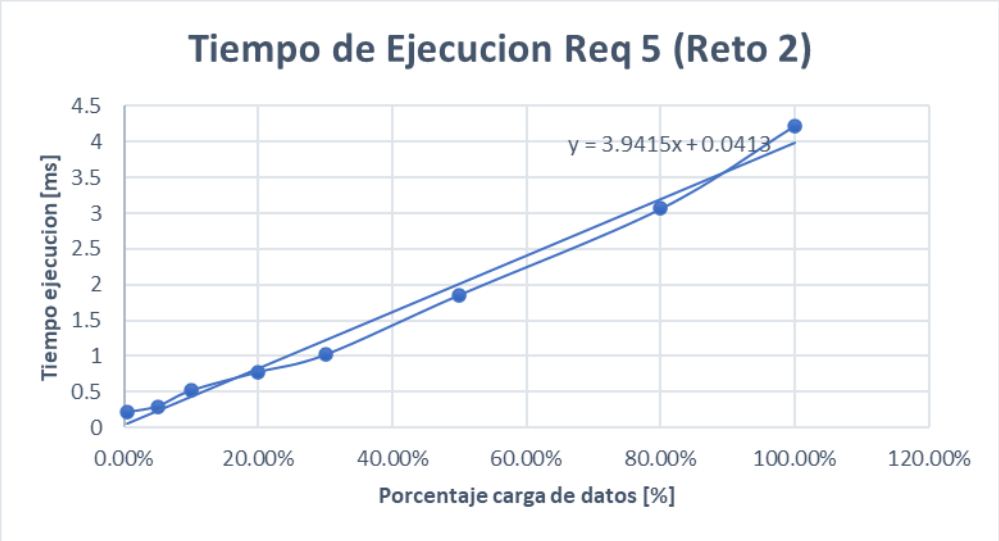
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño del archivo	Tiempo de Ejecución Real @LP [ms]	Consumo de Datos [kB]
0,5%	0.217	1.375
5%	0.299	5.555
10%	0.519	5.617
20%	0.778	6.523

30%	1.02	7.398
50%	1.857	8.367
80%	3.063	9.383
100%	4.225	10.379

Graficas



Análisis

Se puede observar el comportamiento lineal esperado de la complejidad O(N). Es importante agregar que el mergesort, se realiza sobre una lista más pequeña que la que incluye el mapa de todos los paises, por lo que prima la complejidad del recorrido del mapa sobre la del mergesort.

Requerimiento 6

Para este requerimiento se recibe una entrada de director a buscar. Se busca dentro del catálogo cargado y se filtran los datos para añadirlos al mapa correspondiente. Para este requerimiento se emplean contadores en el mapa que se retornan al final. Finalmente, se imprime la información en las tablas. Cabe resaltar que aparte de la función buscarDirector() que se describió anteriormente, para la carga de datos se emplean otras dos. Las funciones addDirector y newDirector cargan las funciones en el mapa al inicio de todo.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Nombre del director a buscar.

Salidas	Contador con el número diferenciado de Movies y TV Show y por otro lado el contador de las diferentes plataformas. También, retorna los tipos de géneros en los que participa el director. Además, del catálogo filtrado y ordenado.
Implementado (Sí/No)	Si se implementó y se hizo grupal.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Paso 1:
Se crean las listas a retornar.
`directores = catalog['directores']`
`director_info_list = lt.newList('ARRAY_LIST')`
`conteoType = lt.newList('ARRAY_LIST')`
`conteoPlat=lt.newList('ARRAY_LIST')`
	O(4)

Paso 2:
Se consulta el mapa y se procede a hacer el conteo y agregar la información.
`existedirector = mp.contains(directores, director)`
`if existedirector:`
`  entry = mp.get(directores, director)`
`  director_info = me.getValue(entry)`
`  for video in lt.iterator(director_info['videos']):`
`  lt.addLast(director_info_list, video)`
`  lt.addLast(conteoType, director_info['Movie'])`
`  lt.addLast(conteoType, director_info['TV Show'])`
`  lt.addLast(conteoPlat, director_info['Amazon Prime'])`
`  lt.addLast(conteoPlat, director_info['Disney Plus'])`
`  lt.addLast(conteoPlat, director_info['Hulu'])`
`  lt.addLast(conteoPlat, director_info['Netflix'])`
	O(n+n)

Paso 3: Se organiza la información según el criterio. `sorted_director_info = ms.sort(director_info_list, cmpGenresByReleaseYear)`	O(n log(n)) (peor caso)
TOTAL	***O(n)***

Pruebas Realizadas

Se uso el director Woody Allen para esta prueba, ya que es de los que más contenido ha de tener. En la prueba se mide específicamente el proceso de búsqueda del requerimiento; función buscarDirector().

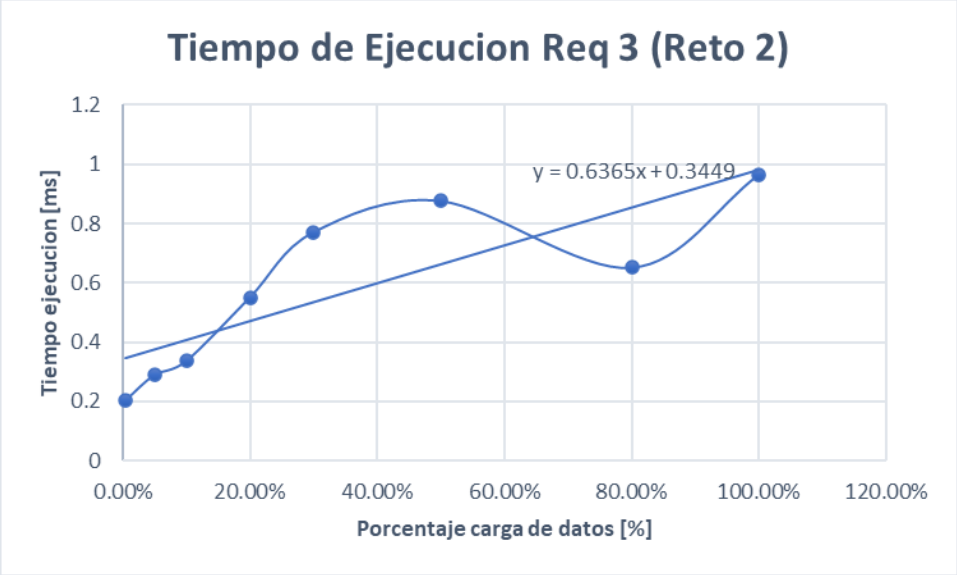
	Especificaciones
Procesadores	Intel CORE I5 10th GEN
Memoria RAM (GB)	8.0 GB
Sistema Operativo	Windows 11 64 bits

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño del archivo	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0,5%	0.852	0.540
5%	0.852	0.289
10%	0.852	0.855
20%	2.956	1.210
30%	2.956	1.667
50%	4.148	0.876
80%	4.863	0.651
100%	4.863	0.964

Graficas



Análisis

Se puede observar que la tendencia de la función da lineal, con una pendiente aproximada a 1. Esto quiere decir que el análisis de notación Big O que se hizo tiene sentido. Se intuye que algunas diferencias se deben a distintos procesos que se llevaba a cabo durante las pruebas (tener abierto el explorador, entre otras cosas).

Esta grafica se compara con los datos obtenidos en el mismo requerimiento en el reto 1:

Porcentaje de la Muestra	Tiempo (ms)
Small	0.52
5%	0.86

10%	1.11
20%	1.47
30%	2.5
50%	3.44
80%	4.50
Large	6.39

Donde las pruebas de tiempo eran mucho más largas con el mejor algoritmo (notación O(n^2)). Se concluye que para búsqueda de información una vez ya cargados los datos, la estructura de mapas representa una mayor eficiencia.

Requerimiento 7

Descripción

Para este requerimiento se recibe por parámetro un entero que es la cantidad de géneros que se quiere ver por orden de contenidos. Para abordar este requerimiento se creó la función topGeneros() que es la encargada de ordenar los elementos del índice de géneros de acuerdo a la cantidad de contenidos y retornarlos en una lista. En esta función no es necesario realizar el conteo de los géneros, porque esto se realiza en la carga de datos del índice de géneros.

Entrada	El número N de géneros a identificar
Salidas	Una tabla ordenada de mayor a menor según la cantidad de películas y programas de cada género y una tabla ordenada de mayor a menor según la cantidad de películas y programas de cada género que contenga el nombre del género, el número total de películas y programas por plataforma, el número de películas y el número de programas.
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Pasos	**Complejidad**

Paso 1: Este paso tiene complejidad O(2), porque se ejecutan dos instrucciones de complejidad O(1)
`gen_top = lt.newList('ARRAY_LIST')`
`generos = catalog['generos']`
	O(2)

Paso 2: Este paso tiene complejidad O(3n+1), ya que se ejecuta un ciclo que recorre todas las llaves de el	O(3n+1)

indice de géneros lo cual tiene complejidad O(n). Además se agregan elementos al final de un ARRAY LIST lo cual tiene complejidad O(n) y se obtiene el valor de una pareja llave valor, que tiene complejidad O(1)
`for genero in lt.iterator(mp.keySet(generos)):`
`  entry = mp.get(generos, genero)`
`  genero_info = me.getValue(entry)`
`  lt.addLast(gen_top, genero_info)`	

Paso 3:
Este paso tiene una complejidad de O(n log(n)+1)
`sorted_list = ms.sort(gen_top, cmpGenresByCount)`
`top = lt.subList(sorted_list, 1, num_top)`
	O(n log(n)+1)

TOTAL	***O(n)***

Pruebas Realizadas

Para las pruebas de tiempos de ejecución, se ejecutó el código correspondiente al requerimiento 7, teniendo en cuenta cada tamaño de archivo. Al ejecutar el requerimiento se utilizó como parámetro para todas las pruebas el top "20". El ordenamiento utilizado fue merge sort. Para la toma de datos de tiempos de ejecución y memoria se utilizaron las librerías "time" y "tracemalloc" respectivamente, y se tomó el tiempo de ejecución del algoritmo del model.py, sin tener en cuenta lo que tardó en imprimir por pantalla.

Especificaciones del equipo:

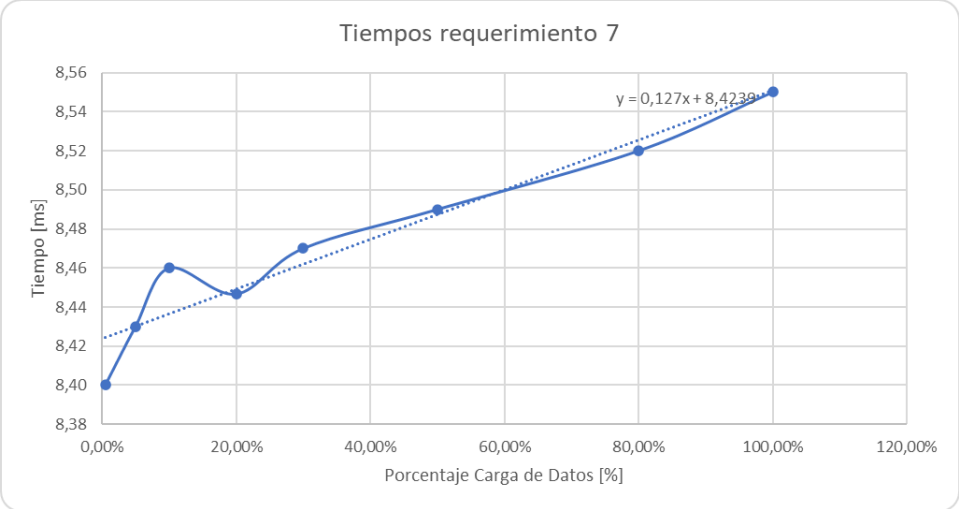
	Especificaciones
Procesadores	Intel(R) Core(TM) i5-3320M CPU @ 2.60 GHz (4 CPUs), ~2.60 GHz
Memoria RAM (GB)	8.0 GB
Sistema Operativo	Windows 10 Pro 64 bits

Tablas de datos

Tamaño del archivo	**Consumo de Datos [kB]**	**Tiempo de Ejecución Real @LP [ms]**
0,5%	15,56	8,40
5%	17,11	8,43
10%	24,12	8,46
20%	24,12	8,45
30%	24,12	8,47
50%	24,01	8,49

80%	23,11	8,52
100%	23,87	8,55

Graficas



Análisis

Para hallar la complejidad del requerimiento 7 se debe tener en cuenta la función topGeneros(). El análisis de la complejidad de este algoritmo se realizó mediante la notación Big O, que define el peor caso de ejecución del programa, dando como resultado O(n). En la gráfica obtenida, se puede observar un comportamiento lineal, el cual corresponde con esperado en el análisis de complejidad.

En el reto 1 al realizar este mismo requerimiento con ARRAY LIST se obtuvo una complejidad O(n^2) y la siguiente tabla de tiempos de ejecución.

Porcentaje de la Muestra	Tamaño de la Muestra	Tiempo (ms)
Small	228	2,43
5%	1148	7,37
10%	2298	15,96
20%	4598	29,01
30%	6898	39,17
50%	11498	59,08
80%	18397	79,85
Large	22998	93,29

Comparando los resultados del reto 1 con los obtenidos en el reto 2, se puede concluir que la implementación de ARRAY LIST utilizada en el reto 1 fue más eficiente en tiempos de ejecución con pocos datos. Sin embargo, la implementación con mapas de el reto 2 es más eficiente para tamaños más grandes de datos, ya que la diferencia de tiempos de ejecución entre los diferentes tamaños de datos, aunque aumenta linealmente, es muy poca. Al comparar estos resultados con los comportamientos de las diferentes complejidades algorítmicas, se puede concluir que concuerdan, ya que la complejidad O(n^2) es más eficiente cuando la cantidad de datos es pequeña y O(n) es más eficiente cuando la cantidad de datos es muy grande.