OBSERVACIONES DE LA PRACTICA

Guillermo Antonio Villalba Escamilla - 202114000 - g.villalba@uniandes.edu.co

Nicolás Ruiz Pérez-202123608- n.ruizp2@uniandes.edu.co

Gabriel Francisco González Estrada – 201912668 – gf.gonzalez@uniandes.edu.co

	Máquina 1	**Máquina 2**	**Máquina 3**
Procesadores	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Memoria RAM (GB)	16,0 GB	8,0 GB	12,0 GB
Sistema Operativo	Windows 11 Home – 64 bits	Windows 10 Home Single Language - 64 bits	Windows 10 Home Single Language – 64 bits

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Descripción:

Requerimiento 1

```
def MoviesInYear(catalog,year): #Función Principal Requerimiento 1
    yearMap = catalog["MapReleaseYear"]
    yearlist = me.getValue(mp.get(yearMap,year))
    merg.sort(yearlist,CMPMoviesInYear)
    return yearlist
```

Entrada	Mapa de hash (PROBING), año
Salidas	Mapa de hash (PROBING)
Implementado (Sí/No)	Si

Requerimiento 2:

```
def ShowsInDate(catalog,date): #Función Principal Requerimiento 2
    if mp.contains(catalog["MapDateAdded"],date) == True:
        DateList = me.getValue(mp.get(catalog["MapDateAdded"],date))
        merg.sort(DateList,CMPShowsInDate)
    else:
        DateList = lt.newList("ARRAY_LIST")
    return DateList
```

Entrada	Mapa de hash (PROBING), fecha
Salidas	Lista (ARRAY)
Implementado (Sí/No)	Si

Requerimiento 3 (Nicolás):

```
def ContentByActor(catalog,actor): #Función Principal Requerimiento 3
    ActorMap = catalog["MapActor"]
    movies = 0
    shows = 0
    if mp.contains(ActorMap,actor) == True:
        ActorList = me.getValue(mp.get(ActorMap,actor))
    else:
        ActorList = lt.newList("ARRAY_LIST")
    for i in lt.iterator(ActorList):
        if i["type"] == "Movie":
            movies += 1
        else:
            shows += 1
    merg.sort(ActorList,CMPContentByActor)
    return ActorList,movies,shows
```

Entrada	Mapa de hash (PROBING), nombre actor (str)
Salidas	Tupla (Mapa de hash (PROBING), número (int), número (int))
Implementado (Sí/No)	Si

Requerimiento 4 (Gabriel):

```
def contentByGenre(catalog, genre): #Función Principal Requerimiento 4
    GenreMap = catalog['MapListedIn']
    movies = 0
    shows = 0
    if mp.contains(GenreMap, genre) == True:
        GenreList = me.getValue(mp.get(GenreMap,genre))
    else:
        GenreList = lt.newList('ARRAY_LIST')
    for video in lt.iterator(GenreList):
        if video['type'] == 'Movie':
            movies+=1
        else:
            shows+=1
    merg.sort(GenreList, CMPContentByActor)
    return GenreList, movies, shows
```

Entrada	Mapa de hash (PROBING), género (str)
Salidas	Tupla (Lista (ARRAY), número (int), número (int))
Implementado (Sí/No)	Si

Requerimiento 5 (Guillermo):

```
def ContentbyCountry(catalog,country): #Función Principal Requerimiento 5
    CountryMap = catalog["MapCountry"]
    movies = 0
    shows = 0
    if mp.contains(CountryMap,country):
        CountryList = me.getValue(mp.get(CountryMap,country))
    else:
        CountryList = lt.newList("ARRAY_LIST")
    for i in lt.iterator(CountryList):
        if i["type"] == "Movie":
            movies += 1
        else:
            shows += 1
    merg.sort(CountryList,CMPContentByCountry)
    return CountryList,movies,shows
```

Entrada	Mapa de hash (PROBING), nombre país (str)
Salidas	Tupla (Lista (ARRAY), número (int), número (int))
Implementado (Sí/No)	Si

Requerimiento 6:

```
def TitlesByDirector(catalog,director): #Función Principal Requerimiento 6
    type = {"Movie":0,"TV Show":0}
    service_name = {"amazon_prime":{"Movie":0,"TV Show":0},"disney_plus":{"Movie":0,"TV Show":0},
        "hulu":{"Movie":0,"TV Show":0},"netflix":{"Movie":0,"TV Show":0}}
    listed_in = {}
    if mp.contains(catalog["MapDirector"],director) == True:
        DirectorList = me.getValue(mp.get(catalog["MapDirector"],director))
        for i in lt.iterator(DirectorList):
            type[i["type"]] += 1
            service_name[i["streaming_service"]][i["type"]] += 1
            for e in i["listed_in"].split(","):
                e = e.strip()
                if e not in listed_in:
                    listed_in[e] = 1
                else:
                    listed_in[e] += 1
        merg.sort(DirectorList,CMPTitlesByDirector)
        return DirectorList,type,service_name,listed_in
    else:
        return lt.newList("ARRAY_LIST"),type,service_name,listed_in
```

Entrada	Mapa de hash (PROBING), nombre director (str)
Salidas	Tupla (Lista (ARRAY), Diccionario {películas: número (int), series: número (int))
Implementado (Sí/No)	Si

Requerimiento 7:

```
def TopNGenres(catalog,N): #Función Principal Requerimiento 7
    GenresMap = catalog["MapListedIn"]
    GenresList = mp.keySet(GenresMap)
    GenreSizeList = lt.newList("ARRAY_LIST")
    for genre in lt.iterator(GenresList):
        type_dict = {"Movie":0,"TV Show":0}
        stream_dict = {"amazon_prime":0,"disney_plus":0,"hulu":0,"netflix":0}
        title = me.getValue(mp.get(GenresMap,genre))
        for i in lt.iterator(title):
            type_dict[i["type"]] += 1
            stream_dict[i["streaming_service"]] += 1
        lt.addLast(GenreSizeList,{"genre":genre,"size":lt.size(title),"type":type_dict,"stream":stream_dict})
    merg.sort(GenreSizeList,CMPTopGenres)
    return lt.subList(GenreSizeList,1,N)
```

Entrada	Mapa de hash (PROBING), número
Salidas	Lista (ARRAY)
Implementado (Sí/No)	Si

Requerimiento 8:

```
def TopNActorsByGenre(catalog, genre, N): #Función Principal Requerimiento 8
    dictActores = {}
    GenreMap = catalog["MapListedIn"]
    Titles = me.getValue(mp.get(GenreMap,genre))
    ActorList = lt.newList("ARRAY_LIST")
    for video in lt.iterator(Titles):
        for actor in video["cast"].split(","):
            actor = actor.strip()
            if actor not in dictActores:
                dictActores[actor] = lt.newList("ARRAY_LIST")
            lt.addLast(dictActores[actor],video)
    count = 0
    while count < N:
        max_ = None
        name = None
        for key in dictActores:
            if max_ == None:
                name = key
                max_ = dictActores[key]
            elif lt.size(dictActores[key]) > lt.size(max_):
                name = key
                max_ = dictActores[key]
            elif lt.size(dictActores[key]) == lt.size(max_):
                if key < name:
                    name = key
                    max_ = dictActores[key]
        lt.addLast(ActorList,{"name":name,"titles":max_})
        dictActores.pop(name)
        count += 1
    Info_list = lt.newList("ARRAY_LIST")
    for i in lt.iterator(ActorList):
        lt.addLast(Info_list,infoActor(i))
    return Info_list
```

Entrada	Mapa de hash (PROBING), nombre género (str), número (int)
Salidas	Lista (ARRAY)
Implementado (Sí/No)	Sí

Análisis de complejidad:

- Requerimiento 1:

Pasos	**Complejidad**
Asignación	O(k)
Me.getValue	O (1)
Mp.get	O(1)
Merge Sort	O(NlogN)
TOTAL	***O(NlogN)***

- Requerimiento 2:

Pasos	**Complejidad**

Contains	O(n)
Comparación	O(1)
get	O(1)
getValue	O(1)
Merge Sort	O(NlogN)
TOTAL	***O(NlogN)***

- Requerimiento 3 (Nicolás):

Pasos	**Complejidad**
Contains	**O(n)**
Asignación	O(k)
Comparación	O(n)
get	O (1)
getValue	O(1)
incremento	O(n)
Merge Sort	O(nlogn)
TOTAL	***O(nlogn)***

- Requerimiento 4 (Gabriel):

Pasos	**Complejidad**
Asignación	O(1)
Comparación	O(n)
Contains	O (n)
get	O(1)
GetValue	O(1)
Merge Sort	O(nlogn)
TOTAL	***O(nlogn)***

- Requerimiento 5 (Guillermo):

Pasos	**Complejidad**
Asignación	O(1)
Contains	O(n)
get	O (1)
getValue	O(1)
Merge Sort	O(nlogn)
Incremento	O(n)
TOTAL	***O(nlogn)***

- Requerimiento 6:

Pasos	**Complejidad**

Asignación	O(n)
Comparación	O(1)
Not in	O (n)
contains	O(n)
get	O(1)
Merge Sort	O(nlogn)
getValue	O(1)
TOTAL	***O(nlogn)***

- Requerimiento 7:

Pasos	**Complejidad**
Asignación	O(n)
Addlast	O(n)
Not in	O (n)
contains	O(n)
get	O(n)
Merge Sort	O(nlogn)
getValue	O(n)
TOTAL	***O(nlogn)***

- Requerimiento 8:

Pasos	**Complejidad**
Asignación	O(n)
Comparación not in	O(n)
Addlast	O (n)
Comparación ==	O(n)
get	O(1)
InfoActor()	O(nlogn)
getValue	O(1)
TOTAL	***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).

Para la realización de las pruebas de tiempo se utilizó la librería time y para las pruebas de memoria se utilizó la librería tracemalloc. En el controlador se crearon las funciones de getTime(), deltaTime(), getMemory() y deltaMemory() y en la Vista se imprimieron los resultados de tiempo y memoria para cada requerimiento, de manera que se obtuvo el tiempo de ejecución en una línea de código con la función perf_counter() de time y la memoria en la línea de código con take_snapshot(). Las pruebas

se ejecutaron en la máquina 3 con procesador Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz, 12,0 GB de RAM y Windows 10 Home Single Language – 64 bits. Se ejecutaron los diferentes requerimientos para distintos porcentajes de los datos (0.5%, 5,% 10%, 20%, 50%, 80%, 100%) tres veces y se registraron los promedios de los resultados en las tablas. Para los requerimientos se utilizaron las entradas que se observan en la siguiente tabla. Posteriormente, se realizaron las gráficas de tiempo vs porcentaje de datos y de memoria vs porcentaje de datos para visualizar la complejidad de los algoritmos en ambos dominios. Cabe resaltar que en todos los casos se utilizó la estructura de datos PROBING con un factor de carga de 0.5.

Requerimiento	Entrada	Tiempo [ms]	Memoria [kB]
Carga de datos	-large	17227.73	153741.28
1	1999	744.40	6.72
2	2019-11-12	906.57	3.37
3 (Nicolás)	Bing Crosby	980.59	6.00
4 (Gabriel)	Fantasy	975.47	6.21
5 (Guillermo)	Germany	984.29	2.67
6	John Hughes	919.12	4.43
7	5	990.02	14.93
8	Drama, 10	1935.81	22.54

Tablas de datos

Requerimiento 1:

Datos	Tiempo (ms)	Memoria (kB)
Prueba 1 (0,5%)	33.43	2.82
Prueba 2 (5%)	56.57	3.88
Prueba 3 (10%)	108.62	1.74
Prueba 4 (20%)	234.45	1.58
Prueba 5 (30%)	273.51	5.24
Prueba 6 (50%)	346.96	6.33
Prueba 7 (80%)	575.28	6.83
Prueba 8 (100%)	744.40	6.72

Requerimiento 2:

Datos	Tiempo (ms)	Memoria (kB)
Prueba 1 (0,5%)	20.10	5.84
Prueba 2 (5%)	64.02	2.22
Prueba 3 (10%)	130.08	4.70
Prueba 4 (20%)	292.13	5.73
Prueba 5 (30%)	295.53	3.89
Prueba 6 (50%)	433.86	3.50
Prueba 7 (80%)	710.5	3.18

Prueba 8 (100%)	906.57	3.37

Requerimiento 3 (Nicolás):

Datos	**Tiempo (ms)**	**Memoria (kB)**
Prueba 1 (0,5%)	32.59	5.66
Prueba 2 (5%)	94.14	7.59
Prueba 3 (10%)	113.74	10.04
Prueba 4 (20%)	284.89	11.67
Prueba 5 (30%)	312.80	5.49
Prueba 6 (50%)	435.83	7.09
Prueba 7 (80%)	670.34	20.01
Prueba 8 (100%)	980.59	6.00

Requerimiento 4 (Gabriel):

Datos	**Tiempo (ms)**	**Memoria (kB)**
Prueba 1 (0,5%)	58.69	27.13
Prueba 2 (5%)	86.90	18.26
Prueba 3 (10%)	133.78	18.62
Prueba 4 (20%)	202.24	16.35
Prueba 5 (30%)	326.83	12.32
Prueba 6 (50%)	459.92	16.19
Prueba 7 (80%)	920.05	20.73
Prueba 8 (100%)	975.47	6.21

Requerimiento 5 (Guillermo):

Datos	**Tiempo (ms)**	**Memoria (kB)**
Prueba 1 (0,5%)	55.87	4.59
Prueba 2 (5%)	94.15	6.20
Prueba 3 (10%)	134.50	2.55
Prueba 4 (20%)	238.95	2.55
Prueba 5 (30%)	321.45	3.49
Prueba 6 (50%)	438.92	5.35
Prueba 7 (80%)	680.87	4.43
Prueba 8 (100%)	984.29	2.67

Requerimiento 6:

Datos	Tiempo (ms)	Memoria (kB)
Prueba 1 (0,5%)	34.96	12.09
Prueba 2 (5%)	70.46	0.67
Prueba 3 (10%)	162.75	0.20
Prueba 4 (20%)	195.70	4.35
Prueba 5 (30%)	293.88	3.66
Prueba 6 (50%)	409.56	0.87
Prueba 7 (80%)	700.90	1.19
Prueba 8 (100%)	919.12	4.43

Requerimiento 7:

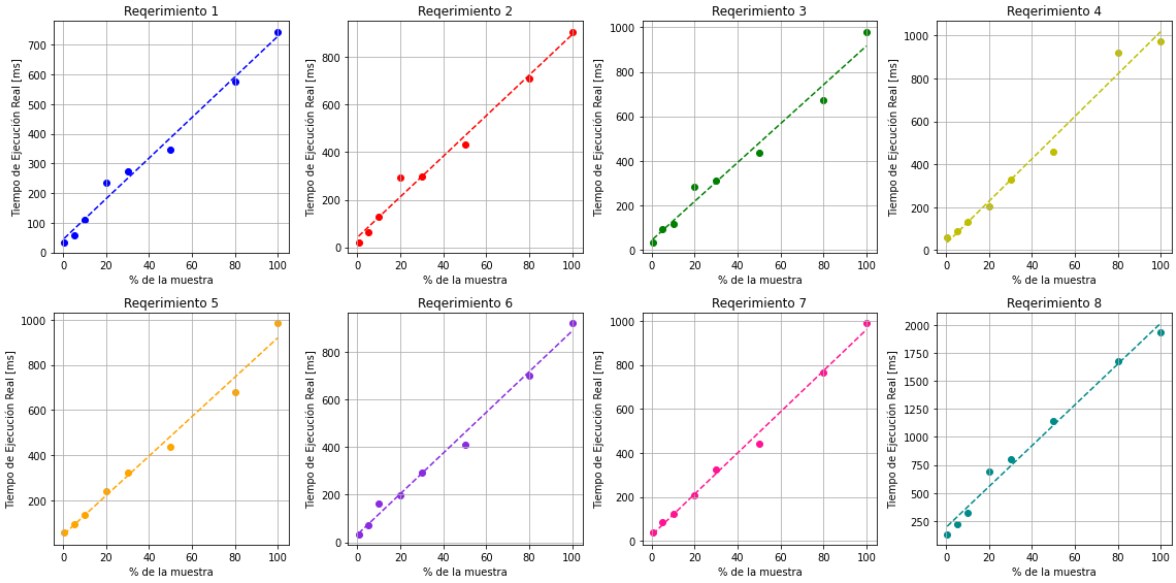
Datos	Tiempo (ms)	Memoria (kB)
Prueba 1 (0,5%)	35.77	18.54
Prueba 2 (5%)	84.98	6.63
Prueba 3 (10%)	123.59	0.30
Prueba 4 (20%)	209.38	15.04
Prueba 5 (30%)	322.37	15.18
Prueba 6 (50%)	439.60	0.42
Prueba 7 (80%)	763.78	4.66
Prueba 8 (100%)	990.02	14.93

Requerimiento 8:

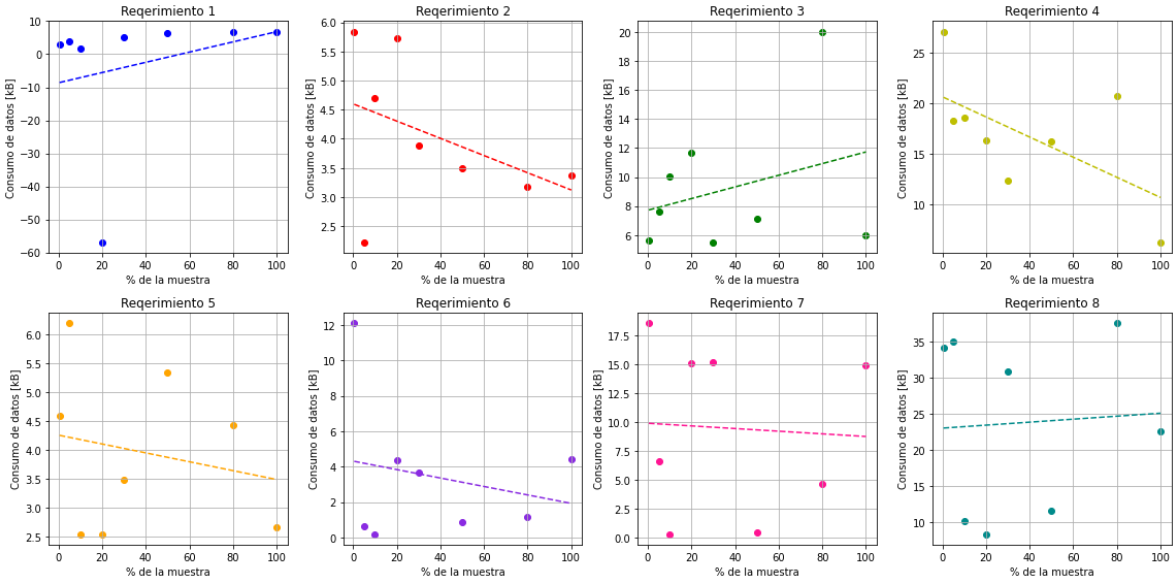
Datos	Tiempo (ms)	Memoria (kB)
Prueba 1 (0,5%)	132.98	34.17
Prueba 2 (5%)	217.05	34.99
Prueba 3 (10%)	325.89	10.21
Prueba 4 (20%)	692.47	8.42
Prueba 5 (30%)	800.57	30.87
Prueba 6 (50%)	1144.62	11.62
Prueba 7 (80%)	1675.62	37.50
Prueba 8 (100%)	1935.81	22.54

Graficas

- Tiempos de ejecución en ms por porcentaje de muestra para cada requerimiento:



- Memoria consumida en kB por porcentaje de muestra para cada requerimiento:



Análisis

Se evidenció que el uso de tablas de hash permite reducir los tiempos de ejecución de manera significativa en comparación con el uso del TAD lista cuando el número de datos a procesar es muy grande. Por otro lado, en términos de uso de memoria se observó que las tablas de Hash presentan resultados de complejidad espacial muy dispersos. Esto puede explicarse debido a que las tablas de Hash organizan los datos de manera que asignan los datos en los índices de un mapa según la función de hash definida, de manera que, si hay colisiones los datos, en este caso, al utilizar PROBING, se guardarán en el índice siguiente; por lo tanto, por la naturaleza cambiante de las colisiones cuando se eliminan valores de los índices de los mapas se pueden generar diferentes resultados de complejidad espacial. En comparación con los resultados de cada requerimiento del Reto 1 se observó que en el Reto 2, si bien se tiene un mismo tipo de complejidad para todos los

requerimientos (NlogN), la pendiente de la recta dada por esta complejidad es mucho menor al utilizar tablas de Hash, de manera que hace que el proceso de funcionamiento de los requerimientos sea mucho más eficiente temporalmente cuando la cantidad de datos de entrada es muy alta.