Estructuras de Datos y Algoritmos

Jose Luis Tavera - 201821999
Juan Diego Yepes - 202022391

Reto II

Índice:

1 Construcción del Catálogo

Para dar solución a los requerimientos, fue necesario replantear la organización del catálogo del reto 1, para organizar la información en mapas en vez de listas.

1.1 Funciones del View

Para empezar a inicializar el catálogo y cargar la información, el usuario debe digitar las siguientes opciones 1 y 2, las cuales empiezan las siguientes instrucciones:

```
    if int(inputs[0]) == 1:
        x = controller.initCatalog()

    elif str(inputs[0]) == "2":
        print("\nCargando informacion de los archivos ....")
        print('------------------------------------------------------------\n')
        answer = controller.loadData(x)
        print('Videos cargados: ' + str(controller.videosSize(x)))
        print("Tiempo [ms]: ", f"{answer[0]:.3f}", "  ||  ",
              "Memoria [kB]: ", f"{answer[1]:.3f}")
        print('\n')
        loaded_categories = controller.categoriesSize(x)
        print('Categorias cargadas: ' + str(loaded_categories))
        print_table(x)
        print('\n')
        loaded_countries = controller.countriesSize(x)
        print('Pa ses cargados: ' + str(controller.countriesSize(x)))
        print_list(x)
```

Estas funciones hacen referencia a las funciones en el controller que se exponen a continuación. A su vez, imprimen la información retorno de lo que se cargó en el catálogo; es decir cuántas y cuáles categorías se cargaron, cuantos y cuáles países, etc.

1.2 Funciones del Controller

Las funciones utilizadas en el controller se detallan a continuación.

```
def initCatalog():
    """
    Llama la funcion de inicializacion del catalogo del modelo.
    """
    catalog = model.newCatalog()
    return catalog
```

Esta función sirve de intermediaria con la función del model llamada newCatalog().

```
def loadData(catalog):
    """
    Carga los datos de los archivos y cargar los datos en la
    estructura de datos, a su vez mide el tiempo de ejecuci n
    """
    delta_time = -1.0
    delta_memory = -1.0

    tracemalloc.start()
    start_time = getTime()
    start_memory = getMemory()

    loadVideos(catalog)
    loadCategories(catalog)

    stop_memory = getMemory()
    stop_time = getTime()
```

```
    tracemalloc.stop()

    delta_time = stop_time - start_time
    delta_memory = deltaMemory(start_memory, stop_memory)

    return delta_time, delta_memory
```

Esta función carga los datos de los archivos y mide el tiempo de ejecución de las operaciones utilizando la librería time. Para cargar los datos, referencia las funciones loadVideos() y loadCategories(), las cuales se exponen a continuación:

```
def loadVideos(catalog):
    """
    Carga los videos del archivo.
    """
    videosfile = cf.data_dir + 'videos/videos-large.csv'
    input_file = csv.DictReader(open(videosfile, encoding='utf-8'))
    for video in input_file:
        model.addVideo(catalog, video)
```

```
def loadCategories(catalog):
    """
    Carga todas las categor as del archivo y los agrega a la lista de
    categorias
    """
    categoriesfile = cf.data_dir + 'videos/category-id.csv'
    input_file = csv.DictReader(
        open(categoriesfile, encoding='utf-8'), delimiter='\t')
    for category in input_file:
        model.addCategory(catalog, category)
```

Estas funciones son las que leen el archivo .csv y van llamando a las otras funciones que agregan la información en el catálogo, las cuales se encuentran en el model.

1.3 Funciones del Model

Esta primera función inicializa el catálogo vacío, especificando cuáles índices van a ser mapas y cuáles van a ser listas. Una vez inicializado el catálogo, se agrega la información mediante las siguientes funciones:

```
def newCatalog():
    catalog = {'videos': None,
               'category_id': None,
               'category': None,
               'country': None}


    catalog['videos'] = lt.newList('ARRAY_LIST')
    catalog['category_id'] = lt.newList('ARRAY_LIST')
    catalog['category'] = mp.newMap(20, maptype='PROBING', loadfactor=0.5)
    catalog['country'] = mp.newMap(20, maptype='PROBING', loadfactor=0.5)
    return catalog
```

Luego, las funciones de carga de datos, llamadas para cada video individual en el controller, se exponen a continuación:

```
def addVideo(catalog, video):
    """
    La funcion de addVideo() adiciona los videos a una lista de videos,
    a su vez los adiciona a los mapas de 'category' y 'country'
    """
    lt.addLast(catalog['videos'], video)
    addVideoOnMap(catalog, video['category_id'], video, 'category')
    addVideoOnMap(catalog, video['country'], video, 'country')
```

La función addVideo() no tiene retorno y se encarga de agregar individualmente, video por video, las funciones al catálogo. Primero, lo agrega a la lista de videos catalog['videos']; luego los agrega a cada uno de los

mapas video['category_id'] y video['country']. Para hacer esto, se referencia a la función addVideoOnMap() la cual se muestra a continuación:

```
def addVideoOnMap(catalog, int_input, video, catalog_key):
    """
    La funcion de addVideoOnMap() adiciona el video al mapa
    que se ha seleccionado.
    Args:
        catalog: Catalogo de videos
        int_input: Llave a analizar
        video: Video a a adir
        catalog_key: Especifica cu l catalogo
    """
    selected_map = catalog[catalog_key]
    existkey = mp.contains(selected_map, int_input)
    if existkey:
        entry = mp.get(selected_map, int_input)
        value = me.getValue(entry)
    else:
        value = newSeparator(int_input, catalog_key)
        mp.put(selected_map, int_input, value)
    lt.addLast(value['videos'], video)
```

Esta función agrega el video individual al mapa seleccionado, que se define con el parámetro 'catalog_key'. Si la llave del video, que puede ser la categoría o el país, ya se encuentra en el mapa, lo que hace la función es agregarlo a esa llave expandiendo la lista de valores organizada en Array. Si no, crea una nueva lista y una nueva llave organizada en Array y agrega ese valor.

2 Primer Requerimiento

El equipo de análisis quiere conocer cuáles son los n videos con más views que son tendencia en un determinado país, dada una categoría específica. Para dar respuesta a este requerimiento el equipo de desarrollo debe recibir como entrada la siguiente información:

- category_name
- country
- Número de videos que quiere listar (n)

Y como respuesta debe presentar en consola la siguiente información:

- trending_date
- title
- channel_title
- publish_time
- views
- likes
- dislikes

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

2.1 Funciones del View

El view le pide al usuario ingresar el país y la categoría de referencia que servirán como parámetro para la función del controller que une con la función homónima

```
elif str(inputs[0]) == "3":
        pais = input("\nIngrese el pa s de referencia: ")
        categoria = int(
            input('Ingrese el c digo la categor a de referencia: '))
        n = input("Ingrese el n mero de videos que desea imprimir: ")
        print("\nCargando ....")
        result = controller.getVideosByCategoryAndCountry(x, categoria, pais)
        printResults(result[1], int(n))
```

2.2 Funciones del Controller

Análogamente, el controller invoca la función homónima del model haciendo de puente entre el view y el model.

```
def getVideosByCategoryAndCountry(catalog, category, country):
    '''
    Retorna los videos dado un pa s y categor a espec ficos
    '''
    return model.getVideosByCategoryAndCountry(catalog, category, country)
```

2.3 Funciones del Model

La función general del model, similar a la del reto 1, invoca tres funciones: dos para crear las sublistas y otro para sortear los videos.

```
def getVideosByCategoryAndCountry(catalog, category, country):
    """
    La funci n de getVideosByCriteriaMap() filtra los videos por una
    llave y categor a espec ficas.
    """
    sublist = getVideosByCriteriaMap(
        catalog, 'category', category).get('videos')
    sublist2 = getVideosByCriteriaList(sublist, 'country', country)
    return sortVideos(sublist2, lt.size(sublist2), 'cmpVideosByViews')
```

La función de CriteriaMap filtra los vídeos por categoría usando el mapa del catálogo.

```
def getVideosByCriteriaMap(catalog, criteria, key):
    """
    La funci n de getVideosByCriteriaMap() filtra los videos por un
    criterio espec fico dado un key. El cat logo debe ser un mapa.
    """
    values = catalog[criteria]
    entry = mp.get(values, str(key))
    result = me.getValue(entry)
    return result
```

La función de CriteriaList itera la lista para filtrar por el segundo criterio que corresponde al país.

```
def getVideosByCriteriaList(catalog, criteria, x):
    """
    La funci n de getVideosByCriteriaList() filtra los videos por un
    criterio espec fico dado un x. El cat logo debe ser una lista.
    """
    listaretorno = lt.newList("ARRAY_LIST")
    for element in lt.iterator(catalog):
        nombre_pais = element.get(criteria)
        if nombre_pais == x:
            lt.addLast(listaretorno, element)

    return listaretorno
```

Finalmente, sortVideos organiza la sublista final por merge sort para obtener los videos con más views por medio de la función de cmpVideosbyViews.

```
def sortVideos(catalog, size, cmp):
    """
    La Funci n sortVideos() organiza los videos de acuerdo
    al par metro cmp.
    """
    sub_list = lt.subList(catalog, 0, size)
    sub_list = sub_list.copy()
    start_time = time.process_time()

    if cmp == 'cmpVideosByViews':
        sorted_list = ms.sort(sub_list, cmpVideosByViews)
    if cmp == 'comparetitles':
        sorted_list = ms.sort(sub_list, comparetitles)
    if cmp == 'comparelikes':
        sorted_list = ms.sort(sub_list, comparelikes)

    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return elapsed_time_mseg, sorted_list
```

3 Segundo Requerimiento (Juan Diego Yepes)

El equipo de análisis quiere conocer cuál es el video que más días ha sido trending para un país específico. Para dar respuesta a este requerimiento el equipo de desarrollo debe recibir como entrada la siguiente información:

- country

Y como respuesta debe presentar en consola la siguiente información:

- title
- channel_title
- country
- número de días

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

3.1 Funciones del View

El segundo requerimiento es invocado en el view cuando el usuario ingresa la opción número "4":

```
elif str(inputs[0]) == "4":
        pais = input("Ingrese el pa s de referencia: ")
        print("\nCargando ....")
        videos_filtrados = controller.getVideosByCountry(x, pais)
        result = controller.getMostTrendingDaysByID(videos_filtrados.get(
            'videos'))
        printResultsv3(result)
```

Para filtrar los videos, se invoca la función getVideosByCountry() del controller.

3.2 Funciones del Controller

Las funciones utilizadas en el controller se detallan a continuación.

```
def getVideosByCountry(catalog, country):
    '''
    Retorna los videos de un pa s espec fico
    '''
    return model.getVideosByCriteriaMap(catalog, 'country', country)
```

Esta función a su vez invoca la función del model getVideosByCriteriaMap()

3.3 Funciones del Model

```
def getVideosByCriteriaMap(catalog, criteria, key):
    """
    La funcion de getVideosByCriteriaMap() filtra los videos por un
    criterio espec fico dado un key. El cat logo debe ser un mapa.
    """
    values = catalog[criteria]
    entry = mp.get(values, str(key))
    result = me.getValue(entry)
    return result
```

La función de CriteriaMap filtra los vídeos por categoría usando el mapa del catálogo.

```
def getMostTrendingDaysByID(videos):
    """
    La funci n de  getMostTrendingDaysByID() itera la lista ordenada y
    retorna el elemento que m s se repite.
    """
    elemento = lt.firstElement(videos)
    mayor_titulo = None
    mayor = 0
    i = 0

    for video in lt.iterator(videos):
        if video['video_id'] == elemento['video_id']:
            i += 1
        else:
            if i > mayor:
                mayor_titulo = elemento
                mayor = i
            i = 1
            elemento = video

    if i > mayor:
        mayor_titulo = elemento
        mayor = i
    return (mayor_titulo, mayor)
```

4 Tercer Requerimiento (Jose Luis Tavera)

El equipo de análisis quiere conocer cuál es el video que más días ha sido trending para una categoría específica.Para dar respuesta a este requerimiento el equipo de desarrollo debe recibir como entrada la siguiente información:

- country

Y como respuesta debe presentar en consola la siguiente información:

- title
- channel_title
- country
- número de días

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

4.1 Funciones del View

Al seleccionar la opción cinco el usuario solo necesita ingresar la categoría de referencia para invocar dos funciones del controller: uno que filtra usando el mapa y otra que obtiene el vídeo que más veces aparece

```
.elif str(inputs[0]) == "5":
        categoria = int(
            input('Ingrese el c digo la categor a de referencia: '))
        print("\nCargando ....")
        videos_filtrados = controller.getVideosByCategory(x, categoria)
        result = controller.getMostTrendingDaysByID(videos_filtrados.get(
            'videos'))
        printResultsv3(result)
```

4.2 Funciones del Controller

Análogamente, el controller cumple su función como puente entre el view y el model, invocando las funciones homónimas del model.

```
def getVideosByCategory(catalog, category):
    '''
    Retorna los videos de una categor a espec fica
    '''
    return model.getVideosByCriteriaMap(catalog, 'category', category)
```

```
def getMostTrendingDaysByID(catalog):
    '''
    Retorna los videos que m s tiempo fueron tendencia
    '''
    sorted_videos = model.sortVideos(
        catalog, lt.size(catalog), 'comparetitles')
    result = model.getMostTrendingDaysByID(sorted_videos[1])
    return result
```

4.3 Funciones del Model

La función de getVideosByCriteriaMap usa los mapas creados en el catálogo para filtrar usando la categoría ingresada.

```
def getVideosByCriteriaMap(catalog, criteria, key):
    """
    La funci n de getVideosByCriteriaMap() filtra los videos por un
    criterio espec fico dado un key. El cat logo debe ser un mapa.
    """
    values = catalog[criteria]
    entry = mp.get(values, str(key))
    result = me.getValue(entry)
    return result
```

Inicialmente, la función necesita recibir la lista filtrada sorteada alfabéticamente para poder iterarla e ir guardando el video que más veces se repite.

```
def getMostTrendingDaysByID(videos):
    """
    La funci n de  getMostTrendingDaysByID() itera la lista ordenada y
    retorna el elemento que m s se repite.
    """
    elemento = lt.firstElement(videos)
    mayor_titulo = None
    mayor = 0
    i = 0

    for video in lt.iterator(videos):
        if video['video_id'] == elemento['video_id']:
            i += 1
        else:
            if i > mayor:
                mayor_titulo = elemento
                mayor = i
            i = 1
            elemento = video

    if i > mayor:
        mayor_titulo = elemento
        mayor = i
    return (mayor_titulo, mayor)
```

5 Cuarto Requerimiento

El equipo de análisis quiere conocer cuáles son los n videos diferentes con más likes en un país con un tag específico. Recuerde que el campo tags está compuesto por múltiples palabras o frases entre comillas (””) y separadas por el siguiente caracter —, así que el tag requerido puede ser una sub-cadena en el campo.

- tag

Y como respuesta debe presentar en consola la siguiente información:

- title
- channel_title
- publish_time
- views
- likes
- dislikes
- tags

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

5.1 Funciones del View

La función del view le pide al usuario ingresar el país y el tag de referencia para posteriormente invocar la función de getVideosByCountryAndTag

```
elif str(inputs[0]) == "6":
        pais = input("Ingrese el pa s de referencia: ")
        tag = input('Ingrese el tag de referencia: ')
        n = int(input("Ingrese el n mero de videos que desea imprimir: "))
        print("\nCargando ....")

        result = controller.getVideosByCountryAndTag(
             x, tag, pais)
```

5.2 Funciones del Controller

Posteriormente, el controller actúa como puente entre el view y el controller invocando su función homónima de getVideosByCountryAndTag

```
def getVideosByCountryAndTag(catalog, tag, country):
    '''
    Retorna los videos de un pa s y tag espec ficos
    '''
    return model.getVideosByCountryAndTag(catalog, tag, country)
```

5.3 Funciones del Model

La función de getVideosByCountryAndTag usa tres funciones del model:la primera filtra usando los mapas por categoría, la segunda itera para filtrar los vídeos que incluyen un tag específico y la última sortea los vídeos por likes.

```
def getVideosByCountryAndTag(catalog, tag, country):
    """
    La funcion de getVideosByCountryAndTag() filtra los videos por un pa s
    y tag espec fico
    """
    sublist = getVideosByCriteriaMap(catalog, 'country', country).get('videos')
    sublist2 = getVideosByTag(sublist, tag)
    sorted_list = sortVideos(
        sublist2, int(lt.size(sublist2)), 'comparelikes')
    return sorted_list
```

```
def getVideosByTag(videos, tag):
    """
    La funcion de getVideosByTag() filtra los videos por un tag
    espec fico
    """
    lista = lt.newList('ARRAY_LIST')
    i = 1

    while i <= lt.size(videos):
        c_tags = lt.getElement(videos, i).get('tags')
        tagpresence = tag in c_tags

        if tagpresence:
            element = lt.getElement(videos, i)
            lt.addLast(lista, element)

        i += 1

    return lista
```

```
def sortVideos(catalog, size, cmp):
    """
    La Funci n sortVideos() organiza los videos de acuerdo
    al par metro cmp.
    """
    sub_list = lt.subList(catalog, 0, size)
    sub_list = sub_list.copy()
    start_time = time.process_time()

    if cmp == 'cmpVideosByViews':
        sorted_list = ms.sort(sub_list, cmpVideosByViews)
    if cmp == 'comparetitles':
        sorted_list = ms.sort(sub_list, comparetitles)
    if cmp == 'comparelikes':
        sorted_list = ms.sort(sub_list, comparelikes)

    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return elapsed_time_mseg, sorted_list
```

6 Funciones Alternas

Con el fin de aprovechar los beneficios de los mapas para disminuir los ordenes de crecimiento, propusimos dos versiones alternas a la función de getMostTrendingDaysByID, en esta versión solo es necesario hacer dos iteraciones y no hace falta sortear la lista lo que disminuiría en teoría el tiempo de ejecución. Sin embargo, teniendo en cuenta el uso de TAD listas, algunas funciones necesitan reiterar la lista o los mapas creados por lo que en la práctica no fue la función más rápida como la función isPresent, entre otras.

```
def getMostTrendingDaysByIDv1(videos):
    ids = mp.newMap(
        500,
        maptype='PROBING',
        loadfactor=0.8)
    pos = mp.newMap(
        500,
        maptype='PROBING',
        loadfactor=0.8)
    ids_list = lt.newList('ARRAY_LIST')

    i = 1

    while i <= lt.size(videos):
        video_id = lt.getElement(videos, i).get('video_id')
        presence = lt.isPresent(ids_list, video_id) != 0

        if presence:
            n = int(mp.get(ids, video_id).get('value'))
            n += 1
            mp.put(ids, video_id, n)
        else:
            mp.put(ids, video_id, 1)
            mp.put(pos, video_id, i)
            lt.addLast(ids_list, video_id)
        i += 1

    mayor = video_id
    repeticiones_mayor = int(mp.get(ids, mayor).get('value'))
    mayor = video_id
    repeticiones_mayor = int(mp.get(ids, mayor).get('value'))

    for each_id in ids_list:
        repetitions = int(mp.get(ids, each_id).get('value'))
        if repetitions > repeticiones_mayor:
            mayor = each_id
            repeticiones_mayor = repetitions

    position = mp.get(pos, mayor).get('value')
    repetitions = mp.get(ids, mayor).get('value')
    result = lt.getElement(videos, position)

    return (result, repetitions)
```

7 Contraste Reto 1

Es evidente la gran cantidad de similaridades que existen entre el Reto 1 y el Reto 2, no solo ambos Retos poseen los mismos requerimientos, la única adición es el uso de Tablas de Hash como TAD listas en el Reto 2. Sin embargo, la construcción inicial del Reto 1 , que seguía los lineamientos del ejemplo y recibió el aval de monitores y profesores, permitía el uso de un diccionario inicial para la carga del catálogo, el cual cumplía una función similar a la de las nuevas tablas de hash implementadas que permitían crear mapas de llaves: i) países y ii) cateogry_id , lo cual se profundiza en el primer apartado de Construcción de Catálogo. Ahora bien, haremos un análisis de las alternativas existentes en el reto 1 y el 2 para la construcción de algoritmos para cada requerimiento:

7.1 Primer Requerimiento

En primera instancia, el primer requerimiento tiene como objetivo devolver los n vídeos con más views que son tendencia en un determinado país, dada una categoría específica. Nótese que en este caso es necesario sortear la muestra final ya que no existe otra forma de obtener n vídeos con más likes. Por lo tanto, la respuesta más eficiente para resolver este problema es:

1. Filtrar por país
2. Filtrar por cateogoría
3. Sortear la sublista final (que es además la más pequeña posible)

Por lo tanto, podemos hacer uso de los mapas para hacer una filtración inicial más eficiente, el cual cumple el mismo papel del catálogo del primer reto, es decir, se selecciona la lista correspondiente al valor de la llave con el país a filtrar, una vez obtenida esta se itera para filtrar. Una alternativa usando mapas a este problema es incluir en la construcción del catálogo un mapa con las distintas combinaciones de país-categoría. Sin embargo, el tiempo en la construcción de este mapa, con 10 países y 18 categorías (180 espacios) era más demorado que la solución previa. Por lo tanto, solo decidimos mantener los mapas para la construcción del catálogo

7.2 Segundo y Tercer Requerimiento

Teniendo en cuenta que ambos problemas son análogos y solo cambian en su parámetro de comparación (país o categoría) decidimos usar funciones estandarizadas que puedan ser usadas para ambos casos. Nuestra solución fue:

1. Filtrar por parámetro (país o categoría)
2. Sortear la lista filtrada
3. Iterarla para obtener el mayor

En este caso, reconocemos que existe una alternativa más corta que no necesita sortear la lista filtrada. Este código lo presentamos en el apartado anterior, y explicamos las razones por las cuales los TAD listas no permiten sacar al máximo los beneficios de estas, ya que al hacer esta función con las listas y diccionarios propios de Python no se encuentran estos problemas.

7.3 Cuarto Requerimiento

Finalmente, el cuarto requerimiento presenta una situación muy similar al primero, su objetivo es conocer cuáles son los n videos diferentes con más likes en un país con un tag específico. Por lo tanto, su solución sigue un esquema similar a la primera:

1. Filtrar por país
2. Filtrar por tag

3. Sortear la sublista final (que es además la más pequeña posible)

Al igual que el primero, no existe ninguna alternativa que provean los mapas para disminuir el orden de crecimiento del problema ya que es necesario sortear la lista. Asimismo, no es posible filtrar por tag sin iterarla ni usar mapas que incluyan una combinación de país-tags porque la instrucciónn ni siquiera busca analizar el tag completo sino un string dentro de uno de los tags, por lo que existirían infinitas combinaciones y no es una solución viable.

7.4 Comparación de Tiempos y Espacio

Usando una MacBook Pro con procesador Apple M1 y 16 GB de RAM.

TIEMPO		
Porcentaje	**Reto 1**	**Reto 2**
0,25%	106,3	82,2
5%	1.268,5	791,8
10%	2.526,3	1.564,8
20%	4.973,3	3.099,1
30%	7.518,5	4.650,5
50%	13.253,9	7.819,2
80%	20.642,5	12.198,8
100%	25.233,7	15.468,1

Tabla 1: Comparación Temporal Reto 1 vs Reto 2

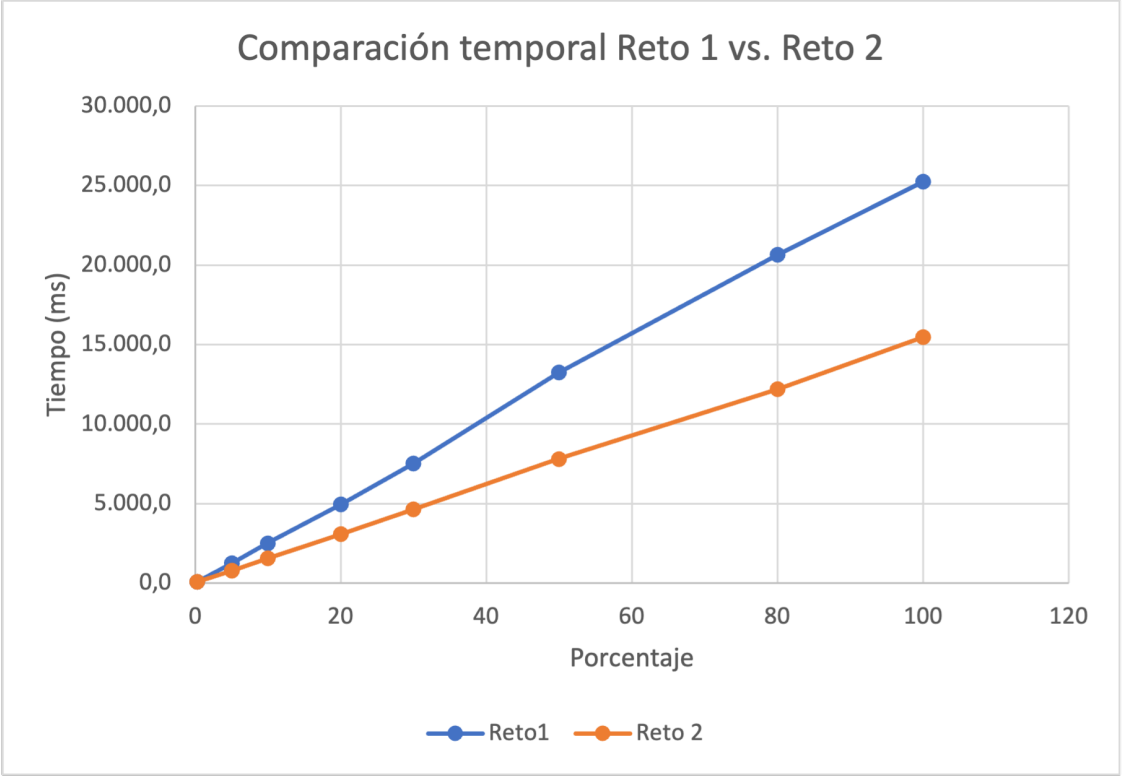

Gráfico 1: Comparación Temporal Reto 1 vs Reto 2

ESPACIO		
Porcentaje	**Reto 1**	**Reto 2**
0, 25%	3649,65	3687,76
5%	66314,66	66505,62
10%	132435,04	132790,31
20%	264515,30	265175,54
30%	396748,96	397617,48
50%	662505,23	664127,94
80%	1059307,26	1061893,97
100%	1324287,78	1327252,35

Tabla 2: Comparación Espacial Reto 1 vs Reto 2

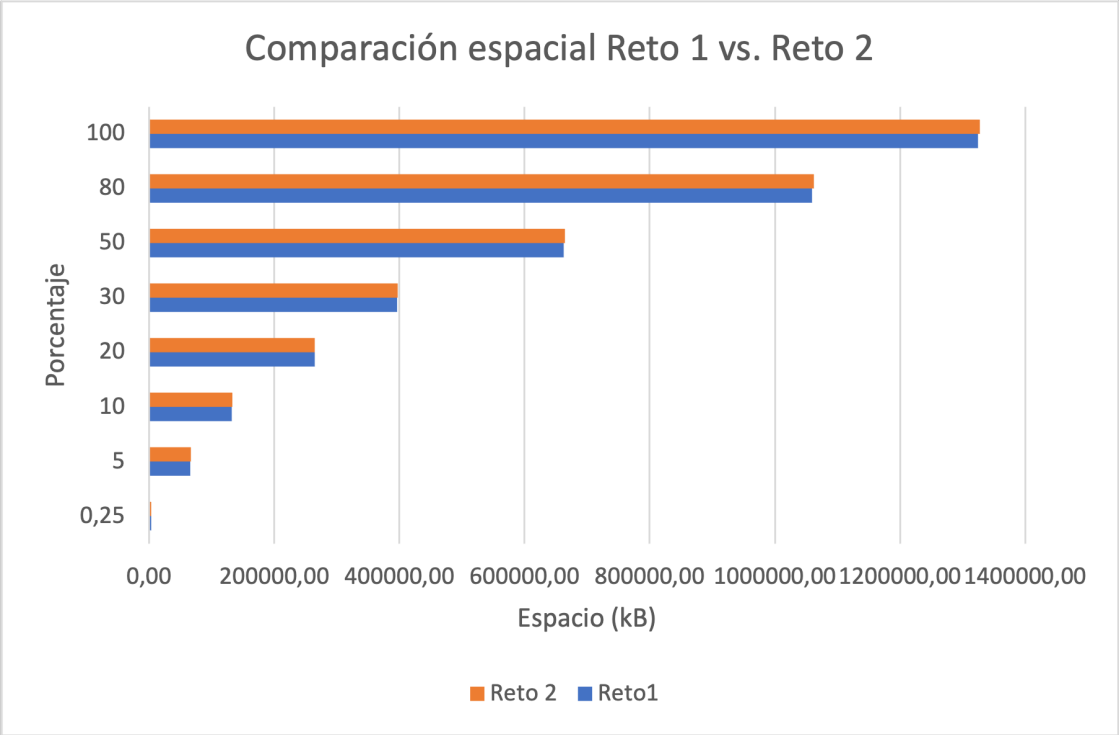

Gráfico 2: Comparación Espacial Reto 1 vs Reto 2

7.5 Conclusiones:

De nuestras comparaciones espaciales y temporales del código del reto 1 y el reto 2, es decir gracias a la adición de mapas de hash, podemos concluir lo siguiente:

- En primer lugar, decidimos comparar con diferentes tamaños de archivos en vez de variar las estructuras de datos, ya que gracias a los laboratorios sabíamos qué tipo de estructura de datos utilizar que fuera la más eficiente en términos de tiempo y espacio.
- Por otro lado, encontramos que no todas las categorías estaban relacionadas con un vídeo, ya que aunque en el reto 1 se cargaran todas, en el reto 2 solo se agregaban al mapa las que estuvieran en algún vídeo.
- Finalmente, encontramos que nuestro Reto 2 fue más eficiente en todos los casos de comparación. Esto debido a la utilización del TAD mapa en estructura Tabla de Hash; ya que su búsqueda es más eficiente.

8 Cálculos de Ordenes de Crecimiento

En este apartado haremos los cálculos de los ordenes de crecimiento de cada una de las funciones. Ya que sería negligente analizar las funciones en términos de "N" (siento este el número total de datos) definiremos las siguientes variables y haremos una aproximación en términos de N para su mejor y peor caso:

- P**:** Es el número total de países cargados por el catálogo que corresponde a 10 países.
- C**:** Es el número total de categorías cargadas por el catálogo que corresponde a 32 categorías aunque solo 18 categorías son cargadas ya que las otras no corresponden a ningún vídeo.
- NP: Es el tamaño promedio de datos filtrados por país. Su mejor caso es igual a: NP_B y su peor caso es: NP_W
- NC: Es el tamaño promedio de datos filtrados por categoría. Su mejor caso es igual a: NC_B y su peor caso es: NC_W
- NPC : Es el tamaño promedio de una filtración por país y categoría. Su mejor caso es igual a: NPC_B y su peor caso es: NPC_W
- NPT : Es el tamaño promedio de una filtración por país y tags. Su mejor caso es igual a: NPT_B y su peor caso es: NPT_W

Ahora bien, analizaremos paso a paso las funciones que de mayor orden de crecimiento por requerimiento

- **Requerimiento 1:**

  getVideosByCountry: Se demora (P), ya que revisa las llaves del catálogo en el apartado de catalog["countries"].

  getVideosByCategory: Se demora (NP), ya que filtra la lista anterior por categoría para lo cual necesita iterarla.

  sortVideos: Al usar merge sort se demora ($NPC \cdot log(NPC)$ en su mejor, peor y caso promedio.

  printResultsv1: Se demora en congruencia con el parámetro "n", es decir (n) porque esas son las veces que itera la lista.

- **Requerimiento 2:**

  getVideosByCountry: Se demora (P), ya que revisa las llaves del catálogo en el apartado de catalog["countries"].

  sortVideos: Al usar merge sort se demora ($NP \cdot log(NP)$ en su mejor, peor y caso promedio.

  getMostTrendingDaysByTitle: Se demora (NP) ya que debe iterar toda la lista sorteada para buscar el mayor.

- **Requerimiento 3:**

  getVideosByCategory: Se demora (C), ya que revisa las llaves del catálogo en el apartado de catalog["category_id"].

  sortVideos: Al usar merge sort se demora ($NC \cdot log(NC)$ en su mejor, peor y caso promedio.

  getMostTrendingDaysByTitle: Se demora (NC) ya que debe iterar toda la lista sorteada.

- **Requerimiento 4:**

  getVideosByCountry: Se demora (P), ya que revisa las llaves del catálogo en el apartado de catalog["countries"]

  getVideosByTag: Se demora (NPT),ya que filtra la lista anterior por tags para lo cual necesita iterarla.

sortVideos: Al usar merge sort se demora $(NPT \cdot log(NPT)$ en su mejor, peor y caso promedio.

printResultsv2: Se demora (NPT) en el peor caso ya que debe escoger los mejores videos y asegurarse de no repetirlos iterando la lista de tamaño (NPT).

9 Resultados

En congruencia con lo anterior, haciendo los cálculos pertinentes para cada una de las notaciones en relación a las variables de tamaño definidas anteriormente, obtuvimos los siguientes ordenes de crecimiento para cada uno de los requerimientos:

	Requerimiento 1	**Requerimiento 2**
Notacion Tilda	$\sim N(15 + P + NP + (NPC \cdot log(NPC) + n)$	$\sim N(10 + P + NP + (NP \cdot log(NP))$
Big Theta	$\Theta(NP)$	$\Theta(NP \cdot log(NP))$
Big O	$O(NP_W))$	$O(NP_W \cdot log(NP_W))$
Big Omega	$\Omega(NP_B))$	$\Omega(NP_B \cdot log(NP_B))$

	Requerimiento 3	**Requerimiento 4**
Notacion Tilda	$\sim N(10 + NC + (NC \cdot log(NC))$	$\sim N(20 + P + 2NPT + (NPT \cdot log(NPT))$
Big Theta	$\Theta(NC \cdot log(NC))$	$\Theta(NPT \cdot log(NPT))$
Big O	$O(NC_W \cdot log(NC_W))$	$O(NPT_W \cdot log(NPT_W))$
Big Omega	$\Omega(NC_B \cdot log(NC_B))$	$\Omega(NPT_B \cdot log(NPT_B))$