**RETO 1**

**Estructuras de Datos y Algoritmos**

***Estudiante A:*** *Daniel Andrés Bernal Cáceres 202020706*

***Estudiante B:*** *Juan Martín Santos Ayala 202013610*

***Correos:*** *da.bernalc1@uniandes.edu.co*

*j.santosa* *@uniandes.edu.co*

1. **Preparación del ambiente de trabajo:**

Para la preparación del ambiente de trabajo, creamos los archivos correspondientes al modelo “MVC” para independizar los cambios hechos en código y que no se vieran reflejados en el producto entregado al usuario.

* 1. **Carga de archivos**

Para cargar los archivos primero creamos un catálogo para guardar la información de un archivo de forma organizada. Tal organización fue:

def newCatalog():

    catalog = {'trending\_date': None,

               'title': None,

               'channel\_title': None,

               'category\_id': None,

               'publish\_time': None,

               'tags': None,

               'views': None,

               'likes': 0,

               'dislikes': 0,

               'country': None,

               'days': 0}

    return catalog

Al tener este catálogo encontramos que había problemas iniciales cuando queríamos recorrer posiciones y organizar los títulos de los videos con alguna función de comparación, por lo que decidimos organizar el catálogo en una lista agrupada. Es decir que en el valor de un diccionario “elements” tenemos una lista con el catalogo de cada video. Esto quiere decir que en cada posición de la lista teníamos un diccionario que contenía la información detallada de un video en específico.

Esto lo logramos de la siguiente forma

def newCatalogList():

    catalogList = lt.newList("ARRAY\_LIST")

    return catalogList

Crea una lista inicial.

def addVideoInfo(catalog, video):

    cg = catalog.copy()

    cg['trending\_date'] = video['trending\_date']

    cg['title'] = video['title']

    cg['channel\_title'] = video['channel\_title']

    cg['category\_id'] = video['category\_id']

    cg['publish\_time'] = video['publish\_time']

    cg['tags'] = video['tags']

    cg['views'] = float(video['views'])

    cg['likes'] = int(video['likes'])

    cg['dislikes'] = int(video['dislikes'])

    cg['country'] = video['country']

    return cg

se agrega información a la lista del catálogo de un video determinado.

def addVideo(catalogList, cg):

    lt.addLast(catalogList, cg)

añade la lista al que contiene la información del video a nuestro catálogo o diccionario “elements”.

Después de crear el catalogo lo que hacemos es cargar en un diccionario de “elements” una lista con un diccionario de id y category\_name. Como resultado obtenemos algo parecido a lo que apareció en el catálogo inicial.

def newCategoryList():

    categoryList = lt.newList("ARRAY\_LIST")

    return categoryList

Crea la lista en la que se van a guardar la categoría y su id respectivo.

def addCategoryInfo(catalog, category):

    cl = catalog.copy()

    cl['id'] = category['id']

    cl['category'] = category['name']

    return cl

Agregaremos la información al diccionario dentro de la primera posición de la lista

def addCategory(catalog, cl):

    lt.addLast(catalog, cl)

Agregamos la lista al diccionario “elements” que contiene los id y category\_id

* 1. **Mostrar la información al usuario**

Por último, antes de comenzar a realizar los requerimientos, imprimimos la información al usuario. Acá fue donde nos dimos cuenta de que era mejor tener los datos de un video y la categoría en una lista en específico a sólo tener las posiciones. Esto nos ahorraría tiempo de ejecución en el algoritmo después.

La complejidad de nuestros catálogos es de O(n)

1. **Requerimiento 1**

El requerimiento 1 nos exigía encontrar la n cantidad de videos con mejores vistas o que fueron tendencia, en un país y una categoría específica. Por lo que lo primero que tenemos que hacer es traducir la categoría que nos da el usuario en su id correspondiente para poder hacer la comparación a la hora de hacer el sort.

def traduceCategoryToId(categoryList, category\_name):

    iterator = 0

    while iterator < (lt.size(categoryList)):

        category = categoryList['elements'][iterator]['category']

        if category\_name in category:

            return categoryList['elements'][iterator]['id']

        iterator += 1

Ya que obtuvimos el id de la categoría deseada, lo que hacemos es que creamos una función de ordenamiento que me agrega a la lista req 1 los valores que sean pertenezcan a la categoría y al país deseado y al final los organizamos por visitas. Usamos el método Shell Sort debido a que es un algoritmo de ordenamiento veloz y podíamos deducir fácilmente su complejidad.

def sortVideos(categoryList, catalogList, category\_name, country, size):

    req1 = lt.newList("ARRAY\_LIST")

    iD = traduceCategoryToId(categoryList, category\_name)

    iterator = 0

    while iterator < (lt.size(catalogList)):

        catalogCountry = catalogList['elements'][iterator]['country']

        catalogId = catalogList['elements'][iterator]['category\_id']

        if country.lower() in catalogCountry.lower() and iD in catalogId:

            lt.addLast(req1, catalogList['elements'][iterator])

        iterator += 1

    viewsList = shs.sort(req1, cmpVideosByViews)

    return viewsList

Al final de este ordenamiento obtenemos una lista con todos los elementos que cumplen los argumentos ordenados, y en el view sólo imprimimos los que nos pide el usuario.

La complejidad de nuestro requerimiento 1 es de O(n logn).

1. **Requerimiento 2 (Estudiante A)**

Para este requerimiento lo primero que hicimos fue encontrar una forma de poder sumar la cantidad de veces que un video aparecía en el documento, lo que significa que estuvo en tendencias en días, la misma cantidad de veces que aparecía en el documento.

def loadDays(catalogList):

    catalogDays = lt.newList("ARRAY\_LIST")

    i = 0

    while i < lt.size(catalogList):

        lt.addLast(catalogDays, catalogList['elements'][i]['title'])

        i += 1

    return catalogDays

Esto crea un catalgo con solo los títulos para poder ver en la otra función la cantidad de veces que se repite y así calcular los días.

def addDays(catalogList, days):

    i = 0

    while i < lt.size(catalogList):

        if catalogList['elements'][i]['title'] in days:

            catalogList['elements'][i]['days'] += 1

        i += 1

Acá va sumando 1 cada vez que encuentra un título repetido

Después de lograr hallar los días, hicimos una función de ordenamiento para días

def cmpVideosByDays(video1, video2):

    return (float(video1['days']) > float(video2['days']))

Por último, creamos una función que agrega a una lista los videos que cumplen la función del país determinado y por último los organiza por likes.

def sortVideosCountry(catalogList, country):

    req2 = lt.newList("ARRAY\_LIST")

    iterator = 0

    while iterator < (lt.size(catalogList)):

        catalogCountry = catalogList['elements'][iterator]['country']

        if country.lower() in catalogCountry.lower():

            lt.addLast(req2, catalogList['elements'][iterator])

        iterator += 1

    daysList = shs.sort(req2, cmpVideosByDays)

    return daysList

Por último, el view imprime sólo la información del primer elemento.

La función es muy compleja y no pudimos encontrar mejor solución.

La complejidad de nuestro requerimiento 2 es de O(2n^2)

1. **Requerimiento 3**

Para este requerimiento lo primero que hicimos fue encontrar una forma de poder sumar la cantidad de veces que un video aparecía en el documento, lo que significa que estuvo en tendencias en días, la misma cantidad de veces que aparecía en el documento.

def loadDays(catalogList):

    catalogDays = lt.newList("ARRAY\_LIST")

    i = 0

    while i < lt.size(catalogList):

        lt.addLast(catalogDays, catalogList['elements'][i]['title'])

        i += 1

    return catalogDays

Esto crea un catalgo con solo los títulos para poder ver en la otra función la cantidad de veces que se repite y así calcular los días.

def addDays(catalogList, days):

    i = 0

    while i < lt.size(catalogList):

        if catalogList['elements'][i]['title'] in days:

            catalogList['elements'][i]['days'] += 1

        i += 1

Acá va sumando 1 cada vez que encuentra un título repetido

Después de lograr hallar los días, hicimos una función de ordenamiento para días

def cmpVideosByDays(video1, video2):

    return (float(video1['days']) > float(video2['days']))

Por último, creamos una función que agrega a una lista los videos que cumplen la función del país y categoría determinados. por último, los organiza por likes.

def sortVideosCategory(categoryList, catalogList, category\_name):

    req3 = lt.newList('ARRAY\_LIST')

    iD = traduceCategoryToId(categoryList, category\_name)

    iterator = 0

    while iterator < (lt.size(catalogList)):

        catalogId = catalogList['elements'][iterator]['category\_id']

        if iD in catalogId:

            lt.addLast(req3, catalogList['elements'][iterator])

        iterator += 1

    daysList = shs.sort(req3, cmpVideosByDays)

    return daysList

Por último, el view imprime sólo la información del primer elemento.

La función es muy compleja y no pudimos encontrar mejor solución

La complejidad de nuestro requerimiento 3 es de O(2n^2)

1. **Requerimiento 4**

Para este último requerimiento sólo tuvimos que crear una función de comparación con la cantidad de likes que tenía un video.

def cmpVideosByLikes(video1, video2):

    return (float(video1['likes']) > float(video2['likes']))

Ahora como debemos tener los países de una categoría y país determinados, usamos las funciones que ya habíamos creado para tal motivo.

def traduceCategoryToId(categoryList, category\_name):

    iterator = 0

    while iterator < (lt.size(categoryList)):

        category = categoryList['elements'][iterator]['category']

        if category\_name in category:

            return categoryList['elements'][iterator]['id']

        iterator += 1

y comparamos si el país y la categoría ingresado por el usuario está en el catálogo, si está, agregaremos la información a una lista llamada req4 y la ordenaremos con nuestra función de comparación.

def sortVideosTags(catalogList, tag, country):

    req4 = lt.newList("ARRAY\_LIST")

    iterator = 0

    while iterator < (lt.size(catalogList)):

        catalogCountry = catalogList['elements'][iterator]['country']

        catalogtag = catalogList['elements'][iterator]['tags']

        if country.lower() in catalogCountry.lower() and tag in catalogtag:

            lt.addLast(req4, catalogList['elements'][iterator])

        iterator += 1

    tagList = shs.sort(req4, cmpVideosByLikes)

    return tagList

Por último, imprimimos la cantidad de videos que el usuario desea consultar.