RETO 2

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.

1.1 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 = {'videos': None,
               'category_id': None,
               'category': None,
               'country': None,
               'days': None}
```

Creamos una lista con todos los videos cargados para facilitar la consulta del tamaño total de datos cargados del archivo CSV.

```
      catalog['videos'] = lt.newList('ARRAY_LIST')
```

La llave es el id y el valor de esta es el nombre de la categoría.

```
    catalog['category_id'] = mp.newMap(33,
                                       maptype='PROBING',
                                       loadfactor=0.5,
                                       comparefunction=compareCategor
ies)
```

La llave es el nombre de la categoría y el valor de esta es una lista con todos los videos que pertenecen a dicha categoría.

```
    catalog["category"] = mp.newMap(33,
                                    maptype='PROBING',
                                    loadfactor=0.5,
                                    comparefunction=compareCategories
)
```

La llave es el país y el valor de esta es una lista con todos los videos que pertenecen a dicha país.

```
    catalog["country"] = mp.newMap(11,
                                   maptype='PROBING',
                                   loadfactor=0.5,
                                   comparefunction=compareCategories)
```

Después de crear el catálogo lo que hacemos es agregar toda la información del archivo en una lista de tipo ARRAY_LIST, esta la usamos para conocer el size que lo necesitamos en varias funciones.

```
def addVideoToCat(catalog, video):

    lt.addLast(catalog['videos'], video)
```

Después de crear el catálogo lo que hacemos es relacionar las categorías con su id en el catálogo en la llave "category_id.

```
def addCategoryInfo(catalog, category):

    mp.put(catalog['category_id'], category['id'], category['name'].l
ower())
```

Después de crear el catálogo lo que hacemos es agregar la información de los videos en el catálogo en la llave "category" según a la categoria a la que pertenecen. La llave mencionada es un mapa por lo que la categoría va a ser la llave y el valor va a ser una lista con todos los videos y su información que pertenecen a esa categoría

```
def addVideo(catalog, video):

   categories = catalog['category']
    catName = convertIdtoCat(catalog, video['category_id'])
    present = mp.contains(categories, catName)
    if present:
        entry = mp.get(categories, catName)
        cat = me.getValue(entry)
    else:
        cat = videoCatalog()
        mp.put(categories, catName, cat)
    lt.addLast(cat['videos'], video)
```

Después de crear el catálogo lo que hacemos es agregar la información de los videos en el catálogo en la llave "country" según al país al que pertenecen. La llave mencionada es un mapa por lo que el nombre del país va a ser la llave y el valor va a ser una lista con todos los videos y su información que pertenecen a ese país.

```

def addCountry(catalog, video):

    country = catalog['country']
    countryName = video['country'].lower()
    present = mp.contains(country, countryName)
    if present:
        entry = mp.get(country, countryName)
        cat = me.getValue(entry)
    else:
        cat = videoCatalog()
        mp.put(country, countryName, cat)
    lt.addLast(cat['videos'], video)
```

1.2 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 todos los videos en una lista en específico. Esto nos serviría para conocer el size del archivo a leer y en primera medida todos los datos del csv.

La complejidad de nuestro catálogo es de O(n logn).

2. **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.

```
def getVideosCat(catalog, category, country):
     # Tiene un espacio porque los nombres de las categorias se guard
aron asi
    parameter = ' ' + category
    # Obtenemos la pareja llave valor de la categoria
    pair = mp.get(catalog['category'], parameter)
    if pair is None:
        newlist = 1
    else:
    # Sacamos la informacion de la pareja y es la lista con los video
s
        category_list = me.getValue(pair)
        countryList = lt.newList('ARRAY_LIST')
        iterator = ite.newIterator(category_list['videos'])
        while ite.hasNext(iterator):
            info = ite.next(iterator)
            if info['country'] == country:
                lt.addLast(countryList, info)
        # Organizamos la lista de los videos por la cantidad de likes
        newlist = sortVideos(countryList, cmpVideosByViews)
    return newlist
```

   Al final de esta función 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 del requerimiento 1 es de O(n logn).

3. **Requerimiento 2 (Estudiante A)**
   El requerimiento 2 nos exigía encontrar el video con más días en tendencia, en un país.

```
def mostTrendingVideoCountry(catalog, country):

    # Obtenemos la pareja llave valor de la categoria
    pair = mp.get(catalog['country'], country)
    if pair is None:
        newlist = 1
        days = 0
    else:
        # Sacamos la informacion de la pareja y es la lista con los v
ideos
        country_list = me.getValue(pair)
        dictionary = {}
        iterator = ite.newIterator(country_list['videos'])
        while ite.hasNext(iterator):
            info = ite.next(iterator)
            # Seleccionamos los valores que son utiles para la busque
da
            newinfo = (info['title'], info['channel_title'],
                       info['country'])

            llist = [1]
            if newinfo in dictionary:
                dictionary[newinfo].append(1)
            else:
                dictionary[newinfo] = llist
        # Sacamos la llave del dicionario cuyo valor es el mayor
        newlist = max(dictionary, key=dictionary.get)
        days = dictionary[newlist]

    return newlist, days
```

   Al final de esta función obtenemos una tupla, donde en la primera posición se encuentra la llave con la información del video más tendencia y en la segunda posición se encuentra una lista con la cantidad de veces que el video fue tendencia.
   La complejidad de nuestro requerimiento 2 es de O(n).

4. **Requerimiento 3**

   El requerimiento 3 nos exigía encontrar el video con más días en tendencia, en una categoría.

```
def mostTrendingVideoCat(catalog, category):
    # Tiene un espacio porque los nombres de las categorias se guarda
ron asi
    parameter = ' ' + category
    # Obtenemos la pareja llave valor de la categoria
    pair = mp.get(catalog['category'], parameter)
    if pair is None:
        newlist = 1
        days = 0
    else:
        # Sacamos la informacion de la pareja y es la lista con los v
ideos
        category_list = me.getValue(pair)
        dictionary = {}
        iterator = ite.newIterator(category_list['videos'])
        while ite.hasNext(iterator):
            info = ite.next(iterator)
            # Seleccionamos los valores que son utiles para la busque
da
            newinfo = (info['title'], info['channel_title'],
                       info['category_id'])

            llist = [1]
            if newinfo in dictionary:
                dictionary[newinfo].append(1)
            else:
                dictionary[newinfo] = llist
        # Sacamos la llave del dicionario cuyo valor es el mayor
        newlist = max(dictionary, key=dictionary.get)
        days = dictionary[newlist]

    return newlist, days
```

Al final de esta función obtenemos una tupla, donde en la primera posición se encuentra la llave con la información del video más tendencia y en la segunda posición se encuentra una lista con la cantidad de veces que el video fue tendencia.

La complejidad de nuestro requerimiento 3 es de O(n).

5. **Requerimiento 4**
   El requerimiento 4 nos exigía encontrar los videos con más likes en un país y con una etiqueta en específico.

```
def mostLikedVideosCountryTag(catalog, country, tag):
    # Obtenemos la pareja llave valor de la categoria
    pair = mp.get(catalog['country'], country)
    if pair is None:
        newList = 1
    else:
        # Obtenemos la lista con los videos de ese pais
        countryList = me.getValue(pair)

        finalList = lt.newList('ARRAY_LIST')
        iterator = ite.newIterator(countryList['videos'])
        while ite.hasNext(iterator):
            info = ite.next(iterator)
            # Buscamos los tags que desea el usuario
            if tag in info['tags']:
                # Verificamos que no hayan repeticiones de videos
                lt.addLast(finalList, info)
            # Le hacemos un sort a la lista dependiendo de los likes
        newList = sortVideos(finalList, cmpVideosByLikes)
        return newList
```

   Al final de esta función obtenemos una lista, donde se encuentran los videos con más likes del país dado por parámetro y que tienen la etiqueta dada.
   La complejidad de nuestro requerimiento 4 es de O(n)

- Comparar la complejidad de los requerimientos implementados en el Reto No. 1 con los implementados en este reto.

  Requerimiento 1:
  En nuestro reto 1 la complejidad era de O(n logn) y en el reto 2 es de O(n logn).
  Requerimiento 2:
  En nuestro reto 1 la complejidad era de O(2n^2) y en el reto 2 es de O(n).
  Requerimiento 3:
  En nuestro reto 1 la complejidad era de O(2n^2) y en el reto 2 es de O(n).
  Requerimiento 4:
  En nuestro reto 1 la complejidad era de O(2n) y en el reto 2 es de O(n).

Estas complejidades fueron calculadas teniendo en cuenta el peor de los casos.

Dicho esto, se puede concluir que los diferentes algoritmos utilizados para el reto 2 implementando los maps (tablas de Hash) son claramente más eficientes tanto en la búsqueda como también para agregar elementos, permitiéndonos tener búsquedas con complejidades logarítmicas. Conllevando, a que los tiempos fueran menores a los del reto 1.