Gestor de Vídeos

PART 1 del projecte ED del curs 2024/25

Pràctica Estructures de Dades

Avui en dia existeixen moltes utilitats per a gestionar biblioteques de vídeos. Una característica apreciada pels usuaris és la capacitat de fer recomanacions basades en les seves preferències.

En aquesta pràctica s'implementarà un sistema senzill de reproducció i recomanació de vídeos.

Una característica important d'aquesta pràctica serà la utilització d'estructures de dades adients per a optimitzar tant l'ús de la memòria com l'execució de les operacions del gestor de la col·lecció de vídeos.

1. Introducció

Aquesta pràctica es pot catalogar dintre del conjunt d'aplicatius anomenats *Gestors de Vídeos* o *Video Managers*. Aquestes aplicacions tenen com a objectiu principal emmagatzemar i gestionar les col·leccions de vídeos dels usuaris, proporcionant determinades funcionalitats auxiliars com ara la creació de llistes de reproducció i recomanacions de seqüències de vídeos. Òbviament, la pràctica no consisteix en construir una aplicació comercial d'aquest tipus. Però sí que s'implementaran algunes de les funcionalitats típiques d'aquestes aplicacions, entre elles:

- Recuperació de metadades (títol, data, gènere, etc.) d'arxius mp4.
- Gestió de llistes de reproducció M3U dels vídeos d'una determinada col·lecció.
- Reproducció dels vídeos en format mp4.
- Recomanacions basades en cerques i creuament de metadades.

Dintre d'aquest domini la pràctica desenvolupada haurà de realitzar amb la suficient eficiència les tasques que s'aniran descrivint. Mostrant així la utilitat de fer servir estructures de dades complexes per augmentar el rendiment del processament de dades. A més, no només el rendiment haurà de ser suficientment bo, sinó que la implementació a baix nivell de les estructures de dades haurà de ser correcta i robusta. Això es comprovarà realitzant una sèrie de tests que avaluaran l'execució —incloent casos extrems— dintre d'un corpus de vídeos donat (a banda de l'òbvia depuració i les proves que vosaltres ja heu de fer habitualment durant el desenvolupament).

1.1 Els arxius MP4 i M3U

En aquesta pràctica els vídeos utilitzats seran arxius existents en format MP4. I encara que es podria fer extensiu a d'altres formats (MOV, AVI, MPEG-TS, etc.), com que l'objectiu de la pràctica no és pas crear una aplicació real, podem fer aquesta simplificació. Respecte als arxius MP4 cal tenir present que es tracta d'un contenidor estandarditzat que pot emmagatzemar tant seqüències de vídeo com d'àudio (encara que enforcat en el vídeo). Aquests seqüències es guarden internament en un stream comprimit que utilitza un còdec concret (mpeg-2, h.264, hevc, vc1, etc.) per a cadascun. Per tant, es pot utilitzar per una àmplia veritat d'usos, com ara captures de càmera, vídeos curts per Internet, pel·lícules, gravacions, etc. I dintre del contenidor a més de les mostres de cada stream, també hi poden haver una sèrie de metadades que s'emmagatzemen en una capçalera. Aquestes metadades reben el nom de tags, els quals hi existeixen en diferents versions. De fet amb

el contenidor MP4 existeixen diferents subformats de metadades. Nosaltres utilitzarem els més habituals de QuickTime i XMP que actualment la majoria d'aplicacions poden llegir, i que són les mateixes metadades que generen les càmeres dels smartphones.

El conjunt de metadades QT/XMP present en un arxiu MP4 és completament opcional. Així doncs en cada arxiu hi poden haver només certs camps amb un contingut arbitrari. Aquests camps poden ser molt variats, alguns dels quals son: *Title* (títol del vídeo), *Album* (àlbum o col·lecció), *Artist* (intèrpret, autor o propietari), *Composer* (compositor o distribuidor), *Genre* (gènere/s), *Date* (any de publicació), *Comment* (comentari general, per exemple URL), *ISCR* (identificador internacional de material audiovisual), etc. La llista ha crescut segons les necessitats, i actualment poden incloure fins i tot diferents imatges associades a l'arxiu, com ara un pòster o similar. Però encara que el conjunt d'aquestes metadades és molt obert, dins aquesta pràctica només hi considerarem un subconjunt reduït d'elles per a simplificar. Cal mencionar que existeixen utilitats que permeten editar (tant manualment com de forma automàtica) aquestes metadades. Dins la pràctica NO és necessari utilitzar cap d'elles, encara que hi podeu utilitzar qualsevol recurs disponible que vulgueu per a visualitzar i/o editar les que hi puguin haver dins els arxius MP4 que vulgueu utilitzar.

Un altre format rellevant d'arxiu relacionat és el **format M3U**. Aquests arxius s'anomenen llistes de reproducció. El seu contingut bàsic és simplement una llista ordenada d'arxius de mitjans, descrita amb un format de text senzill, els quals estan referenciades per un URI. Un **URI (*Uniform Resource Identifier*)** és una generalització de les URL que habitualment fem servir. Fonamentalment és simplement una cadena de text que identifica un recurs, i que dins el format M3U pot ser un simple nom d'arxiu, tant amb path absolut com relatiu. Vegem aquí un exemple senzill d'una llista de reproducció M3U:

```
#EXTM3U
#EXTINF:-1, Autor 1 - Vídeo 1
video1.mp4
#EXTINF:-1, Autor 2 – El meu primer vídeo
subdirectory1/video01.mp4
```

Fixeu-vos que el format és molt simple de processar: La primera línia es sempre la mateixa seqüència que identifica el tipus d'arxiu: #EXTM3U. Després per a cada arxiu hi han dues línies de text, la primera de les quals comença amb el caràcter #, i que als efectes es pot tractar com un comentari; seguida d'una segona que és el path del vídeo.

Dintre d'aquesta pràctica la utilitat específica d'aquests arxius serà representar una seqüència de vídeos que es reprodueixen seguint l'ordre de la llista. Per tant, no serà necessari tenir en compte totes les opcions que proporciona d'aquest format. El fet d'utilitzar un format establert és per que així es poden utilitzar eines externes per a reproduir i crear aquests arxius. A més d'exemplificar l'ús recomanable de formats oberts quan volem intercanviar dades entre sistemes.

1.2 Recursos necessaris

Degut a que l'objectiu és assolir les competències necessàries per a implementar i utilitzar estructures de dades complexes que permeten fer un processament eficient d'un conjunt d'informació, determinats aspectes de la programació es veuran simplificats artificialment. Això té importants implicacions, com ara la interfície d'ús o les funcionalitats de la mateixa. Per tant, per a determinats aspectes es donaran instruccions concretes, com ara fer servir algunes llibreries externes, utilitzar codi d'exemple o resoldre certes parts d'una determinada forma.

S'ha de tenir en consideració que el model de desenvolupament que s'haurà de seguir serà el model basat en fites, en les quals es presentaran una sèrie d'objectius que s'aniran afegint segons el curs vagi avançant. L'entorn de desenvolupament serà amb el llenguatge de programació Python (versió 3.11) dins l'entorn Anaconda+Spyder, que els alumnes ja hi coneixeu. La interfície del programari a desenvolupar serà per línia de comandes, no essent necessari incorporar una interfície d'usuari gràfica.

Cal mencionar a més que el present enunciant no és una documentació completa. Per tant caldrà consultar documentació externa, com ara els apunts de l'assignatura, la bibliografia recomanada, així com la documentació del llenguatge Python i de les llibreries utilitzades. Així i tot podeu consultar certs aspectes amb els professors o compartir experiències en els grups de l'assignatura.

Seguidament teniu una llista dels recursos necessaris per desenvolupar la pràctica:

- Anaconda: Spyder 5.4.x + Python 3.11.x.
- Plataforma: Windows, MacOS o Linux.
- Corpus de vídeos: conjunt free de vídeos proporcionat dins d'un enllaç ZIP[1].
- Reproductor de vídeo: VLC[2] amb suport MP4 i M3U.
- tinytag[3]: llibreria per accedir als tags de les metadades d'arxius MP4.
- python-vlc: binding[4] de la llibreria "libvlc" per reproduir MP4 (cal instal·lar VideoLAN).

Tingueu present que per a posar en marxa l'entorn de desenvolupament només cal fer una instal·lació del programari Anaconda i del VideoLAN (VLC), i després afegir els paquets de les llibreries esmentades. Per a fer això darrer cal obrir "Anaconda Prompt" i executar les següents dues comandes:

```
$ pip install tinytag
$ pip install python-vlc
```

[1] *Free Stock Video Footage*: https://www.videvo.net

[2] https://www.videolan.org

[3] https://pypi.org/project/tinytag/

[4] Aquesta llibreria i el VLC són opcionals

Si el vostre computador està connectat a Internet es descarregaran els components de Python que permeten fer servir aquestes llibreries, i es configurarà l'entorn. Llavors ja es podran fer els imports corresponents. Si voleu reproduir els vídeos des del vostre codi tal tenir prèviament instal·lat l'executable VLC (VideoLAN) en el vostre sistema, o llavors la llibreria "python-vlc" donarà lògicament errors en temps d'execució. A més, resulta imprescindible que l'arquitectura utilitzada de Python i VLC siguin idèntiques. Això vol dir que encara que el vostre S.O. pugui executar binaris de múltiples arquitectures (ARM64, X86-64, X86, etc.) si l'arquitectura de "python-vlc" no és la mateixa que la DLL ó el Shared-Object de VLC, llavors no es podran reproduir els vídeos. Per exemple, un VideoLAN de 32bits no es podrà utilitzar amb un Python de 64bit. Ni tampoc un VideoLAN per x86-64 amb un Python ARM64. Però en qualsevol cas, simplement desactivant la reproducció, que és l'únic requisit per utilitzar VLC i python-vlc, es pot realitzar la resta de la pràctica només tenint els vídeos MP4 i la llibreria tinytag.

Finalment, tingueu en compte les següents restriccions dins la pràctica:

- Encara el corpus proporcionat, hi podeu fer servir qualsevol conjunt de vídeos en format MP4, però exclusivament en aquest format[5].

- El corpus d'arxius de vídeo s'organitza a partir d'un subdirectori del vostre computador. Anomenarem aquest subdirectori **ROOT_DIR**. Dintre d'aquest hi podran haver tants subdirectoris amb d'altres subdirectoris com es vulgui.

- A la pràctica serà necessari fer un *import* de l'arxiu "cfg.py". Aquest arxiu conté la configuració del ROOT_DIR, i algunes funcions que podeu fer servir. Podeu modificar les constants (variables amb noms en majúscules) d'aquest arxiu segons les vostres necessitats, però no l'haureu de lliurar, doncs la configuració és pròpia de cada equip a on s'executa la pràctica. Per tant no afegiu ni modifiqueu cap codi dins d'aquest arxiu.

- El conjunt de vídeos proporcionat pertany al *Free Stock Video Footage*[6], i es pot fer servir lliurement. Encara que vos animen a que vosaltres feu servir els arxius MP4 que vos siguin més convenients, obtinguts d'Internet, gravats per vosaltres, o de qualsevol altra font disponible de la qual hi tingueu drets.

- Les llistes de reproducció en format M3U només hi podran ser al directori ROOT_DIR. I les referències incloses en elles hauran de ser relatives a aquest path.

- Dintre de la pràctica, un vídeo es considerarà únic només des del punt de vista de l'arxiu. És a dir, que un arxiu repetit dues vegades serà considerat com a dos vídeos

[5] Si voleu veure les metadades dels vostres arxius MP4 podeu utilitzar per exemple les opcions "Propietats" dels arxius tant en Windows com en MacOS. També podeu fer servir eines multiplataforma com MediaInfo (https://mediaarea.net/es/MediaInfo) o FFmpeg (https://ffmpeg.org). Però si voleu editar també teniu disponibles eines com MP3TAG (https://www.mp3tag.de/en/).

[6] https://www.videvo.net

diferents. Igualment, dos arxius amb les mateixes metadades tampoc es considerarà com el mateix vídeo. Però cal tenir present que dues llistes de reproducció podran contenir el mateix vídeo, la qual cosa serà certa només si apunten al mateix arxiu.

Un darrer comentari important que heu de tenir-ne en compte respecte al projecte: És imprescindible respectar totes les instruccions d'aquesta guia per a poder avaluar correctament el resultat. Això és especialment important en quant a l'execució del vostre codi dins de l'entorn d'execució i proves de Caronte. No s'acceptarà cap lliurament que no es pugui executar, i per tant, es consideraran invàlids els lliuraments que només es puguin executar dins el vostre entorn. Tingueu doncs cura de no actualitzar els elements de l'entorn de desenvolupament, i de fer servir exclusivament les versions indicades.

2. Codi d'exemple

2.1 Descripció

Per tal de fer una primera aproximació, aquí teniu un exemple de codi que implementa algunes de les funcionalitats necessàries de forma simple:

```
# -*- coding: utf-8 -*-
"""
test-mp4.py : Script de proves per reproduïr MP4
"""

import cfg      # Necessari per a la pràctica !!
                # Mireu el contingut de l'arxiu
import os.path
import sys
import numpy    #  installed in anaconda by default
import time
#               Només per fer play dels media
import vlc      #  $ pip install python-vlc
#               Especific per obtenir les metadades
import tinytag  #  $ pip install tinytag


# STEP 1: Cerca dels arxius al filesystem
print("Cercant arxius dins [" + cfg.get_root() + "]\n")
uri_file = cfg.get_one_file(0)  # Aquesta funció és només de proves!
if not os.path.isfile(uri_file):
    print("ERROR: Arxiu MP4 inexistent!")
    sys.exit(1)


# STEP 2: Obtenció de les metadades
metadata = tinytag.TinyTag.get(uri_file)
print(metadata)
print('')

if metadata is None:
    print("ERROR: Arxiu MP4 erroni!")
    sys.exit(1)

try:
    duration = int(numpy.ceil(metadata.duration))
except AttributeError:
    duration = -1
try:
    title    = metadata.title
except AttributeError:
    title    = "None"
try:
    album    = metadata.album
except AttributeError:
    album    = "None"
try:
```

```
    artist   = metadata.artist
except AttributeError:
    artist   = "None"
try:
    composer = metadata.composer
except AttributeError:
    composer = "None"
try:
    genre    = metadata.genre
except AttributeError:
    genre    = "None"
try:
    date     = metadata.year
except AttributeError:
    date     = "None"
try:
    comment  = metadata.comment
except AttributeError:
    comment  = "None"


# STEP 3: Generació del identificador únic
name_file = cfg.get_canonical_pathfile(uri_file)
mp4_uuid  = cfg.get_uuid(name_file)


# STEP 4: Reproducció
if (cfg.PLAY_MODE < 2):
    print("Reproduïnt [{}]".format(uri_file))
    print(" Duració:  {} segons".format(duration))
    print(" Títol:    {}".format(title))
    print(" Àlbum:    {}".format(album))
    print(" Artista:  {}".format(artist))
    print(" Composer: {}".format(composer))
    print(" Gènere:   {}".format(genre))
    print(" Data:     {}".format(date))
    print(" Comments: {}".format(comment))
    print(" UUID:     {}".format(mp4_uuid))
    print(" Arxiu:    {}".format(name_file))

if (cfg.PLAY_MODE > 0):
    player = vlc.MediaPlayer(uri_file)

    player.play()     # Nota: Crida ASYNC !!

    # Poolling loop pel control de la reproducció
    timeout = time.time() + duration
    while True:
        if time.time() < timeout:
            try:
                time.sleep(1)
            except KeyboardInterrupt:  # STOP amb <CTRL>+<C> a la consola
                break
        else:
            break

    player.stop()


# END
print("\nFinal!")
```

En aquest exemple podeu veure les següents funcions implementades:

- Com comprovar l'existència d'un arxiu.
- Com obtenir algunes metadades d'un arxiu MP4.
- Com generar un identificar únic d'un arxiu basat en el seu URI complet.
- Com reproduir el vídeo d'un arxiu MP4.

Tingueu present que cal importar sempre l'arxiu "cfg.py"[7] en la vostra pràctica. Com s'indica a la secció 1.2 aquest arxiu conté la configuració del vostre entorn. Reviseu el seu contingut i feu els canvis de les constants per adaptar-les al vostre entorn. A més tingueu present que hi podeu fer servir les funcions declarades dins aquest arxiu, excepte que s'indiqui el contrari.

[7] Aquest arxiu el podeu trobar juntament amb el "test-mp4.py" dins el paquet amb el codi d'exemple de la pràctica juntament amb un petit vídeo de prova.

3. Primer lliurament (de 2)

3.1 Objectius

L'objectiu de la primera part de la pràctica és construir l'esquelet de l'aplicació. Dintre d'aquest esquelet hi haurà una implementació simple d'algunes de les funcionalitats demanades. Principalment es vol aconseguir:

- Familiaritzar-se amb l'entorn de les llibreries i els formats dels arxius utilitzats.
- Tenir-ne una representació a la memòria de les dades d'un arxiu de vídeo.

A partir d'aquí definim una sèrie de funcionalitats que cal implementar:

- Func1: /* llistat d'arxius mp4 */
- Func2: /* generar ID de vídeos */
- Func3: /* consultar metadades vídeos */
- Func4: /* reproduir vídeos amb metadades */
- Func5: /* reproduir llistes de reproducció m3u */
- Func6: /* cercar vídeos segons certs criteris */
- Func7: /* generar llista basada en una cerca */

Anem a descriure en detall aquestes funcionalitats i com implementar-les, tenint present que s'indiquen els tipus generals dels paràmetres de les funcions per a que vos resulti més fàcil d'entendre:

3.1.1. Func1 /* llistat d'arxius mp4 */

Aquesta funcionalitat implica obtenir una llista completa de tots els arxius MP4 presents dins la biblioteca de vídeos. Això vol dir recórrer tots els subdirectoris a partir de ROOT_DIR per a obtenir la llista de tots els MP4 existents al filesystem. Cal tenir-ne en compte que aquesta funció es podrà cridar en qualsevol moment per a obtenir una llista «actualitzada» dels arxius presents en la biblioteca.

Tingueu present a més que, internament i dins el codi de la pràctica, el que representa un arxiu és el path relatiu a on es troba dins el filesystem. Per tant, un arxiu queda identificat pel nom de l'arxiu i els subdirectoris des del ROOT_DIR que el contenen. D'aquesta forma un exemple d'arxiu amb path relatiu hi seria "subdir1/carpeta-A/video01.mp4"; i aquest seria diferent de l'arxiu "subdir2/video01.mp4", encara que l'arxiu físic fos una còpia idèntica.

Per a implementar aquesta funcionalitat hi caldrà crear una classe anomenada "VideoFiles". Aquesta classe en una primera fase haurà d'implementar aquests mètodes:

- `VideoFiles.reload_fs(path: str)`
- `VideoFiles.files_added() -> list`
- `VideoFiles.files_removed() -> list`

Respecte a quests mètodes, fixeu-vos que la funció “VideoFiles.reload_fs()” s’encarrega de mantenir a memòria una representació dels arxius que hi han al disc. Per tant, els mètodes “VideoFiles.files_added()” i “VideoFiles.files_removed()” faran referència només als canvis des de la darrera vegada que s’ha cridat al mètode que llegeix el disc.

També fixeu-vos que aquesta classe sempre treballarà amb Strings que representen directoris o arxius: i per tant les llistes que hi retornaran els mètodes seran llistes de Strings.

3.1.2. Func2 /* generar ID de vídeos */

Dintre del programa per a cada arxiu MP4 es generarà un identificador únic de vídeo. Aquest identificador seguirà el format estàndard UUID ó GUID de 128 bits. Això vol dir que per a identificar arxius de vídeo no hi farem servir el path de l’arxiu, sinó un altre valor més compacte. L’avantatge és que podrem fer servir un identificador de mida fixa envers de l’adreça al disc de l’arxiu. Per a fer això disposarem d’una funció que generarà a partir del *path canònic* de l’arxiu un valor únic, el qual serà el ID del vídeo. Un exemple d’aquest UUID seria el següent String “e2960755-c2b7-56bc-b0b1-5d206e899647”.

Ara bé, cal tenir-ne en compte que aquesta funció, encara que té una probabilitat extremadament baixa, podria generar una col·lisió retornant el mateix identificador a partir de dos arxius diferents. Aquest cas extrem aquí no el tindrem en compte (suposarem que els UUID són realment únics dins el nostre sistema), però sí que haureu de comprovar que efectivament l’identificador és lliure quan es generi un de nou. En el cas de que no fos així i ja estigui utilitzat, el que caldrà fer és no tenir-ne en compte el nou arxiu trobat. És a dir, que el segon arxiu seria com si no hi fos. Caldrà però avisar d’aquest fet traient un missatge per pantalla que indiqui que aquest arxiu no s’utilitzarà, i continuar després amb l’execució de forma habitual.

Per a implementar aquesta funcionalitat hi caldrà crear una classe anomenada “VideoID”. Aquesta classe en una primera fase haurà d’implementar aquests mètodes:

- `VideoID.generate_uuid(file: str) -> str`
- `VideoID.get_uuid(file: str) -> str`
- `VideoID.remove_uuid(uuid: str)`

Fixeu-vos que aquesta classe també utilitzarà Strings per a representar els identificadors UUID. I que internament haurà de guardar els identificadors utilitzats. Per tant, només el mètode “VideoID.generate_uuid()” haurà de comprovar que l’identificador no està essent utilitzat. Així una vegada un UUID ha estat generat una vegada, només es podria tornar a utilitzar si s’ha cridat a “VideoID.remove_uuid()”. Finalment, el mètode “VideoID.get_uuid()” retornarà el valor del UUID sense cap més comprovació que veure si ha estat generat i és actiu (és a dir, que no hi ha estat esborrat).

3.1.3. Func3 /* consultar metadades vídeos */

Donat un arxiu MP4 qualsevol (de fet, donat el path relatiu de l'arxiu), cal poder llegir aquest arxiu i consultar les seves metadades. Aquestes metadades podran existir o no, i per tant, caldrà utilitzar alguna convenció predeterminada quan un camp no hi estigui definit. Així doncs, internament les metadades es representaran a la pràctica sempre per Strings, i el valor "None" indicarà que el camp està buit. La llista de metadades que obligatòriament heu de considerar són: *duration*, *title*, *album*, *artist*, *composer*, *genre*, *date* i *comment*. Tot seguint l'exemple del codi descrit a la secció 2.1 d'aquest document, doncs els nomes no són idèntics per totes les aplicacions. I de fet no són exactament els mateixos dins la llibreria tinytag. Tingueu present que el camp de gènere, pot tenir més d'un valor pel mateix vídeo, i llavors es llisten tots al mateix String separats per comes. Així doncs un vídeo pot ser "computer" i "digital" al mateix moment, indicant llavors que té assignades aquestes dues etiquetes, i llavors tindria com a gènere "computer,digital".

A més de les metadades caldrà guardar també de cada vídeo el path de l'arxiu. Això vol dir que si utilitzem com a identificador el UUID de l'arxiu, llavors serà necessari emmagatzemar també no només els camps de les metadades, sinó també el path de l'arxiu físic original. D'aquesta forma hi serà possible obtenir l'adreça de l'arxiu al disc a partir del identificador únic de vídeo que s'utilitza dins el codi de la pràctica.

Com podeu veure, existeix doncs una relació 1:1 entre els arxius i els vídeos. Els arxius hi queden identificats pels paths relatius, i els vídeos pels identificadors UUID.

Per a implementar aquesta funcionalitat hi caldrà crear una classe anomenada "VideoData". Aquesta classe en una primera fase haurà d'implementar aquests mètodes:

- `VideoData.add_video(uuid: str, file: str)`
- `VideoData.remove_video(uuid: str)`
- `VideoData.load_metadata(uuid: str)`
- `VideoData.get_duration(uuid: str) -> int`
- `VideoData.get_title(uuid: str) -> str`
- `VideoData.get_album(uuid: str) -> str`
- `VideoData.get_artist(uuid: str) -> str`
- `VideoData.get_composer(uuid: str) -> str`
- `VideoData.get_genre(uuid: str) -> str`
- `VideoData.get_date(uuid: str) -> str`
- `VideoData.get_comment(uuid: str) -> str`

És important tenir-ne en compte que el mètode "VideoData.add_video()" només crea l'entrada pel vídeo en qüestió, sense afegir els camps de metadades (però sí que guarda el path de l'arxiu). Per tant, inicialment totes les metadades estaran buides fins que es crida al mètode "VideoData.load_metadata()". A més, s'ha de tenir present que aquest mètode es podria cridar múltiples vegades, especialment si l'arxiu s'ha modificat.

3.1.4. Func4 /* reproduir vídeos amb metadades */

Per a reproduir un vídeo hi han dues accions que resulten necessàries. La primera és imprimir per pantalla les metadades, i la segona és cridar a un tercer per a reproduir el vídeo. Dins la pràctica, la primera acció és farà consultant les metadades ja presents dins la memòria. Mentre que la segona es farà passant a un player l'adreça de l'arxiu MP4. La raó de fer aquesta dicotomia és que el procés de llegir les metadades des de l'arxiu és redundant, i per tant resulta més òptim fer-ho així. En conseqüència l'acció de reproduir utilitzarà com a paràmetre el path de l'arxiu, i l'acció d'imprimir les metadades utilitzarà el id del vídeo.

Aquestes dues accions hi seran per tant independents. Però també caldrà una funció que pugui ajuntar-les de tal forma que es facin alhora. A més, artificialment definirem que aquesta funció es pugui configurar permetent així que es pugui sel·leccionar quines accions s'executaran realment. Això es farà utilitzant el paràmetre definit a la configuració per aquesta tasaca: "cfg.PLAY_MODE". La raó d'això serà poder comprovar l'execució de la pràctica dins un test de proves sense haver de reproduir sempre els vídeos.

Per a implementar aquesta funcionalitat hi caldrà crear una classe anomenada "VideoPlayer". Aquesta classe en una primera fase haurà d'implementar aquests mètodes:

- `VideoPlayer.print_video(uuid: str)`
- `VideoPlayer.play_file(file: str)`
- `VideoPlayer.play_video(uuid: str, mode: int)`

Cal tenir present que "VideoPlayer.play_file()" és una funció asíncrona, i per tant finalitza (retorna) mentre es reprodueix el vídeo en background. En canvi "VideoPlayer.play_video()" és una funció síncrona, i no retorna fins que el vídeo no ha finalitzat, o també si l'usuari interromp la reproducció (reviseu el codi d'exemple per a veure com estar esperant a que un vídeo finalitzi, i pugueu aturar aquesta reproducció).

3.1.5. Func5 /* reproduir llistes de reproducció m3u */

Una llista de reproducció en format M3U defineix una llista ordenada d'arxius MP4. Així doncs dins la pràctica caldrà poder llegir aquests arxius simples, i fer una representació interna del seu contingut. Per tal de ser eficients, la representació interna estarà basada en els identificadors dels vídeos, els quals s'obtenen a partir dels paths dels arxius. Així doncs una primera acció que s'ha de fer és llegir un arxiu M3U; a més de comprovar per a cada arxiu MP4 llistat que efectivament hi existeix dins la col·lecció de vídeos. Llavors si aquest hi és es podrà emmagatzemar el seu identificador de vídeo dins una llista. Però per no complicar el vostre codi, l'algorisme de càrrega d'arxius M3U es simplifica al mínim fent només això: línia a línia es llegeix l'arxiu, i si el primer caràcter no és un "#" i el final de la línia és ".mp4", llavors s'agafa tota la línia com el path d'un arxiu MP4. Després només resta comprovar que efectivament l'arxiu existeix, i llavors s'afegeix a la llista i es continua llegint la resta de l'arxiu. Si no es compleix qualque d'aquestes condicions simplement es continua processant obviant aquella línia.

Una segona acció a realitzar ha de ser que a partir d'una col·lecció de vídeos (és a dir, una llista amb identificadors UUID) es puguin reproduir en ordre els vídeos de la llista (utilitzant el mètode "VideoPlayer.play_video()"). En tot cas, és important veure que per a reproduir un vídeo que apareixen dins un arxiu M3U cal primer generar els identificadors de vídeo (utilitzant les classes anteriors), i que només llavors es podrà cridar al reproductor per a visualitzar-los, doncs cal tenir-ne els UUID corresponents per poder utilitzar-los.

Per a implementar aquesta funcionalitat hi caldrà crear una classe anomenada "PlayList". Aquesta classe de moment haurà només d'implementar aquests mètodes:

- `PlayList.load_file(file: str)`
- `PlayList.play()`

No oblideu manipular correctament les instàncies d'aquesta classe. Cada llista de reproducció serà un nou objecte PlayList. Per tant, guardeu dins alguna estructura simple (un array/vector ó similar) les diferents llistes que pugueu anar creant.

3.1.6. Func6 /* cercar vídeos segons certs criteris */

Una funcionalitat intrínseca d'un gestor de col·leccions de vídeos és poder establir cerques dins la biblioteca. Per tant, l'acció de retornar els vídeos que compleixen determinats criteris de cerca és fonamental. Dins la pràctica, resultarà necessari implementar alguns mètodes per a poder fer cerques d'arxius, encara que siguin senzilles. Per exemple, retornar la llista dels vídeos que hi puguin ser d'un determinat tipus, o les que pertanyen a un autor, o que s'han gravat a una determinada localització, etc. Per a ser suficientment útil, i a la vegada no massa complex, es demana que com a mínim es puguin cercar subcadenes de text dins els camps de les metadades. És a dir, que cal comprovar la presència d'una cadena de text (substring) dins un String; i això ho podeu fer utilitzant les funcions bàsiques de la classe "str", com ara "str.find()".

D'altra banda, també s'hauran de poder realitzar operacions lògiques bàsiques entre llistes de resultats. Així, per exemple, cal poder trobar vídeos que compleixen diferents criteris. Respecte a això, només caldrà implementar els operadors AND i OR de les llistes retornades.

Per a implementar aquesta funcionalitat hi caldrà crear una classe anomenada "SearchMetadata". Aquesta classe en una primera fase haurà d'implementar aquests mètodes:

- `SearchMetadata.duration(min: int, max: int) -> list`
- `SearchMetadata.title(sub: str) -> list`
- `SearchMetadata.album(sub: str) -> list`
- `SearchMetadata.artist(sub: str) -> list`
- `SearchMetadata.composer(sub: str) -> list`
- `SearchMetadata.genre(sub: str) -> list`
- `SearchMetadata.date(sub: str) -> list`
- `SearchMetadata.comment(sub: str) -> list`

- `SearchMetadata.and_operator(list1: list, list2: list) -> list`
- `SearchMetadata.or_operator(list1: list, list2: list) -> list`

Cal prestar atenció a que les llistes retornades hi poden ser buides. I també, que els valors han de ser sempre identificadors de vídeos (UUID). Fixeu-vos també que aquestes "lists" **no són** en cap moment un objecte PlayList sinó llistes normals de Python.

3.1.7. Func7 /* generar llista basada en una cerca */

Com que la funcionalitat anterior és en realitat independent de les llistes de reproducció, però a la vegada a nivell semàntic no hi ha diferencia entre una llista d'identificadors de vídeos (resultat d'una cerca) i una llista de reproducció (objecte PlayList), cal implementar una darrera funcionalitat que solucioni aquest fet. Per tant, una nova funcionalitat a implementar dins la primera fase de la pràctica serà afegir a la classe "PlayList" els mètodes necessaris per a poder afegir i treure elements, de tal forma que es pugui emmagatzemar el resultat d'una cerca en una llista de reproducció. S'entén que això pugui semblar artificial a priori, però en la segona part de la pràctica això tindrà més sentit.

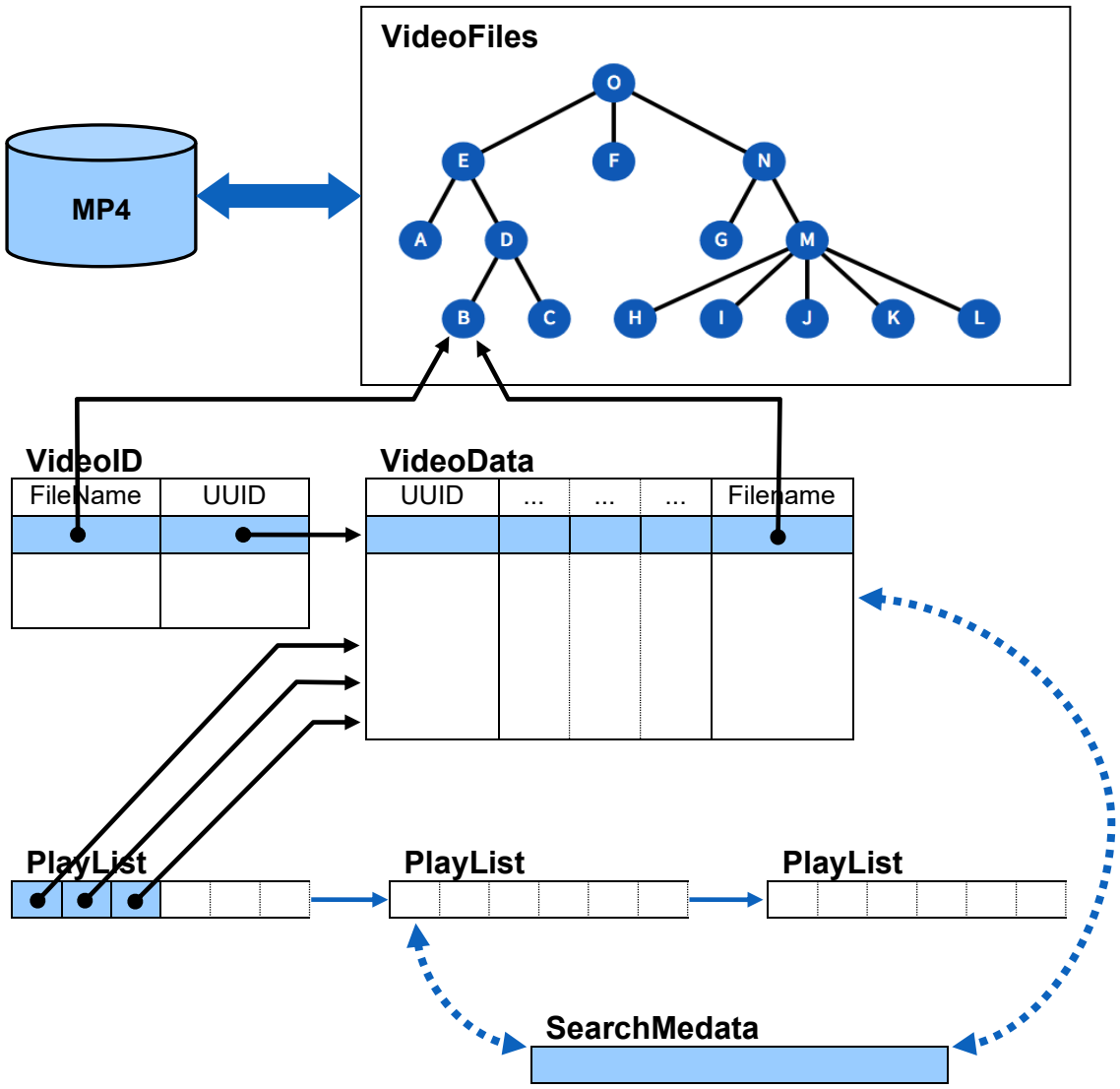
De moment, simplement caldrà ampliar la classe "PlayList" amb els següents mètodes:

- `PlayList.add_video_at_end(uuid: str)`
- `PlayList.remove_first_video()`
- `PlayList.remove_last_video()`

Com es pot veure, aquests mètodes no consumeixen temps fent cerques dels elements que hi guarden, i per tant són ràpids en la seva execució. Així doncs, les úniques operacions que es podran fer seran les més simples possibles: afegir un element al final de la llista, i eliminar un element, que podrà ser el darrer o el primer de la llista.

3.2 Diagrama

Seguidament teniu un diagrama no formal que descriu els elements rellevants de la primera part de la pràctica per a que hi pugueu veure gràficament les relacions entre les diferents estructures de dades:



3.3 Lliurament

Per a fer el lliurament del projecte caldrà fer un upload dels fitxers de la vostra pràctica dins el Caronte. La llista dels arxius necessaris son (fixeu-vos que “cfg.py” no està inclòs):

- `p1_main.py`
- `VideoFiles.py ; VideoID.py ; VideoData.py ; PlayList.py ; SearchMedata.py`
- `Docum.PDF`

El darrer document consisteix en omplir el següent template i generar un arxiu PDF amb les vostres respostes. Això servirà com a documentació bàsica del vostre treball.

Documentació Projecte E.D. - Fase I	
Autors	*Noms i NIUs*
Corpus MP4 proves	*Descripció de tots els Corpus diferents que heu utilitzat, indicant per a cadascun el nº d'arxius i la font (pot ser 'personal')*
Qualificació "grade"	*Valor obtingut amb l'execució del test de proves*
Implementació utilitzada per a les estructures	*Per a cada Classe implementada, explicar el conjunt de dades utilitzat per a emmagatzemar la informació*
Cerques realitzades	*Llista de les cerques de prova realitzades*
Anotacions	*Expliqueu aquí qualsevol comentari rellevant respecte a la vostra implementació*
Diagrama	*Feu in diagrama explicant l'arquitectura de la vostra implementació*