**Análisis de complejidad**

**Nicolás Díaz Montaña – 202021006 – n.diaz9@uniandes.edu.co**

**Daniel R Alonso A - 201419873 - dr.alonso10@uniandes.edu.co**

Carga de datos:

En el view.py la carga de datos se ejecuta cuando el usuario escoge la opción 1 del menú:

if int(inputs[0]) == 1:

start\_time = time.process\_time()

print("Cargando información de los archivos ....")

catalog = initCatalog()

loadData(catalog)

print("-" \* 74)

print('Obras cargadas: ' + str(lt.size(catalog['artworks']))+ "\n")

print('Artistas cargados: ' + str(lt.size(catalog['artists'])) + "\n")

print("-" \* 74)

print(("-" \* 21) + "Ultimos 3 elementos de Artistas " + ("-" \* 21))

printArtistData(controller.lastThree(catalog["artists"]))

print("-" \* 73)

print(("-" \* 22) + "Ultimos 3 elementos de Obras " + ("-" \* 22))

printArtworkData(catalog, controller.lastThree(catalog["artworks"]))

print("-" \* 73)

stop\_time = time.process\_time()

elapsed\_time\_mseg = (stop\_time - start\_time)\*1000

print("Tiempo de ejecución: " + str(elapsed\_time\_mseg))

Aquí la mayoría de operaciones son operaciones print que tienen una complejidad de O(1) mientras que el initCatalog() tiene complejidad O(n) ya que esta asignando llaves a un diccionario vacio. Las que tiene más complejidad son las funciones loadData(), printArtistData() y printArtworkData().

loadData() se encarga de llamar al controller.loadData para que las siguientes funciones hagan su trabajo.

def loadArtworks(catalog):

"""

Carga las obras del archivo.

"""

booksfile = cf.data\_dir + 'MoMA/Artworks-utf8-small.csv'

input\_file = csv.DictReader(open(booksfile, encoding='utf-8'))

for artwork in input\_file:

model.addArtwork(catalog, artwork)

def loadArtists(catalog):

"""

Carga los artistas.

"""

booksfile = cf.data\_dir + 'MoMA/Artists-utf8-small.csv'

input\_file = csv.DictReader(open(booksfile, encoding='utf-8'))

for artist in input\_file:

model.addArtist(catalog, artist)

Las cuales llaman a sus respectivas funciones en model.py

def addArtwork(catalog, artwork):

if artwork["DateAcquired"] == "" or artwork["DateAcquired"] == "Unknown":

hoy = date.today()

artwork["DateAcquired"] = hoy.strftime("%Y-%m-%d")

lt.addLast(catalog["artworks"], artwork)

addNation(catalog, artwork)

def addNation(catalog, artwork):

"""

Adiciona una nacionalidad a la lista de de nacionalidades,

la cual hace referencia a las obras que provengan de esa nacionalidad.

"""

codes = artwork["ConstituentID"]

nations = catalog["nations"]

countries = ArtworkNationality(catalog, codes)

for country in lt.iterator(countries):

pos\_country = lt.isPresent(nations, country)

if pos\_country > 0:

nation = lt.getElement(nations, pos\_country)

else:

nation = newCountry(country)

lt.addLast(nations, nation)

lt.addLast(nation["artworks"], artwork)

def addArtist(catalog, artist):

lt.addLast(catalog["artists"], artist)

Por default los elementos se guardan como estructuras de datos tipo “Linked\_List”, el primero en actuar es addArtist() el cual guarda al final de la lista del catalogo de artists por cada elemento iterado del csv, lo cual significa que tiene complejidad O(n).

El segundo en actuar es el addArtwork() el cual tiene una función muy similar en al addArtist() pero con dos añadidos, el primero siendo que si en el artwork en el que se encuentra el ciclo no tiene fecha le agregue la fecha actual (O(n)); el segundo siendo añadir ese artwork al catalogo de nacionalidades según su nacionalidad el cual tiene una complejidad de O(n) debido a que las funciones utilizadas realizan varios ciclos for-in. Pero debido al uso de isPresent() dentro de un ciclo se vuelve O(n^2).

def ArtworkNationality(catalog, codes):

"""

Consigue la nacionalidad de una obra utilizando los codigos de la misma.

"""

artists = getArtistbyCode(catalog, codes)

nations = lt.newList()

for artist in lt.iterator(artists):

if artist["Nationality"] == "":

artist["Nationality"] = "Nationality unknown"

lt.addLast(nations, artist["Nationality"])

return nations

def getArtistbyCode(catalog, codes):

"""

Consigue a un Artista por su codigo.

"""

artists = catalog["artists"]

group = lt.newList()

cIDSs = codes.replace("[", "").replace("]", "").split(",")

for cID in cIDSs:

cID =cID.strip()

pos = lt.isPresent(artists, cID)

if pos>0:

artist = lt.getElement(artists, pos)

lt.addLast(group, artist)

return group

A su vez debido a que los addArtwork() y addArtists() del model.py están siendo llamados por los addArtwork() y addArtists() del controller.py cada vez que itera por un artwork significa estos dos últimos tienen una complejidad O(n^2) debido a que son ciclos for-in anidados. Por ende la función loadData() también es O(n^2).

Ahora para printArtistData() y printArtworkData(), la complejidad seria O(n) y O(O^2) respectivamente. Ya que printArtworkData() utiliza la función getArtistArtwork() que se encarga de traer el nombre de los artistas de esa obra, pero esa función utilza un isPresent()

def printArtworkData(catalog, artworks):

size = lt.size(artworks)

if size:

for artwork in lt.iterator(artworks):

artists = controller.getArtistsofArtwork(catalog, artwork["ConstituentID"])

print ("Título: " + artwork["Title"] + " Artista(s): " + artists + "Fecha de Adquisición: "

+ artwork["DateAcquired"] + " Medio: " + artwork["Medium"] + " Dimensiones: " + artwork["Dimensions"] + "\n")

else:

print ("No se encontraron obras")

def printArtistData(artists):

size = lt.size(artists)

if size:

for artist in lt.iterator(artists):

print ("Nombre: " + artist["DisplayName"] + " Contituent ID: "

+ artist["ConstituentID"] + " \n")

else:

print ("No se encontraron artistas")

Lo cual significa que la complejidad de la carga de Datos es O(n^2).

Requerimiento 1:

En el view.py el requerimiento 1 se ejecuta cuando el usuario escoge la opción 2 del menú:

A screenshot of a computer

Description automatically generated with medium confidence

Se piden dos datos al usuario lo cual son operaciones básicas de tiempo t. Falta ver la complejidad de la función requerimiento1. Esta función hace lo siguiente:  
Text

Description automatically generated

Los “print()” son operaciones básicas de tiempo t. Veamos la complejidad de las funciones controller.artistbyAnio(), controller.lastThree(), controller.firstThree() y printArtistData\_Req1().

La función controller.artistbyAnio() corresponde en el model con la función organizeArtistsbyanio() que hace lo siguiente:

Text

Description automatically generated

Aparte de operaciones básicas de asignación (de tiempo t) hay un par de ciclos que debemos examinar. El ciclo for depende de la cantidad de artistas así que tiene complejidad O(n). La cantidad de veces que se ejecute el ciclo while depende de la función getElementbyparameterE(). Esta función hace lo siguiente:

Graphical user interface, text

Description automatically generated

La función lt.getElement() tiene complejidad O(n) en el caso que se implemente con una lista encadenada. La otra función que se usa es lt.addLast() la cual tiene complejidad O(1). Así que podemos decir que el ciclo while tiene complejidad O(n).

Por otro lado, las controller.lastThree(), controller.firstThree() son muy sencillas:

Text

Description automatically generated

Así que no aumentan la complejidad del requerimiento.

Por último la función printArtistData\_Req1() tiene complejidad O(n) pues depende del tamaño de la lista de artistas que cumplan la condición de estar en el rango de años en cuestión:

Text

Description automatically generated

De esta manera, concluimos que el requerimiento 1 tiene complejidad O(n) pues ninguna función que use tiene complejidad mayor.

Requerimiento 2 ( Nicolás Díaz Montaña):

elif int(inputs[0]) == 3:

size = input("Indique tamaño de la muestra: ")

printOptionSort()

option = input('Seleccione una opción para continuar\n')

startDate = input("Fecha de Inicio (YYYY-MM-DD): ")

finishDate = input("Fecha Final (YYYY-MM-DD): ")

start\_time = time.process\_time()

artworksBydate(catalog, startDate, finishDate, size, option)

stop\_time = time.process\_time()

elapsed\_time\_mseg = (stop\_time - start\_time)\*1000

print("Tiempo de ejecución: " + str(elapsed\_time\_mseg))

El requerimiento dos se encarga de buscar las obras que estén en un rango de fechas determinado por el usuario. Como tal la mayoría de la funciones son O(t) exceptuando artworksBydate() la cual llama a organizeArtworksBydate() en el controller.py trasnfireindole los datos a la función del mismo nombre en el model.py. lastThree() y firstThree() son O(n) por que solo sacan los primeros y ultimo elementos de una lista, en este caso org\_dates() que son las obras del rango determinado.

def artworksBydate(catalog, startDate, finishDate, size, option):

"""

Genera una lista cronológicamente ordenada de las obras adquiridas

por el museo en un rango de fecha. Retorna el total de obras en el rango cronológico,

total de obras adquiridas por compra y las primeras 3 y utimas 3 obras del rango.

"""

org\_dates = controller.organizeArtworksbyDate(catalog, startDate, finishDate, size, option)

last = controller.lastThree(org\_dates[1])

first = controller.firstThree(org\_dates[1])

print("\n")

print("Total de obras en el rango " + str(startDate) + " - " + str(finishDate) + ": " + str(lt.size(org\_dates[1]))+ "\n")

print("-" \* 84 + "\n")

print("Total de obras compradas en el rango: " + str(controller.countPurchase(org\_dates[1])) + "\n")

print("-" \* 84)

print(("-" \* 21) + "Estos son las 3 primeras Obras encontradas" + ("-" \* 21) + "\n")

printArtworkData(catalog, first)

print("-" \* 84)

print(("-" \* 22) + "Estos son las 3 ultimas Obras encontradas" + ("-" \* 21) + "\n")

printArtworkData(catalog, last)

print("-" \* 84)

print("Para la muestra de elementos " + size + " el tiempo (mseg) es: ", str(org\_dates[0]))

organizeArtworksbyDate() se encarga de conseguir las obras según el rango primero sacando las diferencia de días entre la fecha final y la fecha inicial, luego organiza la lista de las obras por fecha utilizando el tipo de sort y el tamaño de datos que requiera el usuario (pero en este caso tomaremos el merge sort haciendo que esta parte tenga O(n log n) y el tamaño completo del archivo) para luego ir por un rango a partir de la diferencia obtenida.

def organizeArtworkbyDate(catalog, startDate, finishDate, size, option):

"""

Organiza y retorna las obras que esten en un rango de

una fecha inicial y final. ()()()

"""

org = lt.newList()

d\_0 = date.fromisoformat(startDate)

d\_f = date.fromisoformat(finishDate)

delta = d\_f - d\_0

start\_time = time.process\_time()

artworks = sortArtworksbyDate(catalog, int(size), option)

stop\_time = time.process\_time()

elapsed\_time\_mseg = (stop\_time - start\_time)\*1000

for day in range(delta.days + 1):

new\_day = d\_0 + timedelta(days=day)

new\_date = new\_day.strftime("%Y-%m-%d")

getArtworkbydate(artworks, new\_date, org)

return elapsed\_time\_mseg, org

def getArtworkbydate(artworks, date, org):

"""

Retorna una obra por su fecha de adquisición.

"""

for artwork in lt.iterator(artworks):

if date == artwork["DateAcquired"]:

lt.addLast(org, artwork)

Debido a que getArtworkbydate() utiliza un for-in para buscar la fecha en el catalogo para luego integrarlo a una lista dada, hace que organizeArtworksbyDate() tenga una complejidad de O(n^2) haciendo que todo el req 2 tambien sea O(n^2).

Requerimiento 3 (Daniel R Alonso A):

En el view.py el requerimiento 3 se ejecuta cuando el usuario escoge la opción 4 del menú:

Text

Description automatically generated

Se pide un dato al usuario lo cual es una operación básica de tiempo t. Falta ver la complejidad de la función requerimiento3. Esta función hace lo siguiente:

Text

Description automatically generated

Los “print()” son operaciones básicas de tiempo t. Veamos la complejidad de las otras funciones involucradas. Podemos asumir que la función printArtistData\_Req3() tiene complejidad O(n) pues es muy similar a la función printArtistData\_Req1() que ya analizamos. La pregunta es entonces si alguna de las 4 funciones que van hasta el controller tiene complejidad mayor. Veamos lo que hacen. Con un indicador morado señalamos lo que es más relevante examinar (en términos de complejidad):

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

La función artworksbyArtists tiene complejidad O(n) pues usa tres ciclos que son, a lo sumo, de complejidad O(n). La función artworksbyMedium tiene complejidad O(n^2) pues usa dos ciclos dobles de complejidad O(n^2), también usa un ordenamiento pero su complejidad es menor O(nlog(n)). La función contarMedios() usa la función listaMedios() que usa un ciclo doble así que tiene complejidad O(n^2). La función medioMax() tiene complejidad constante.

Así, pues, concluimos que el requerimiento 3 tiene complejidad O(n^2).

Requerimiento 4:

El requerimiento 4 devuelve el top 10 de las naciones con más obras y luego devuelve las primeras tres y las ultimas tres obras del top 1.

elif int(inputs[0]) == 5:

start\_time = time.process\_time()

top = (controller.organizeTopNationaliy(catalog))

topArtworksbyNationality(top)

first = str(list(top.keys())[0])

artworks = controller.getArtworkbyNationality(catalog, first)

print("-" \* 84)

print(("-" \* 21) + "Estos son las 3 primeras Obras encontradas" + ("-" \* 21) + "\n")

printArtworkData(catalog, controller.firstThree(artworks))

print("-" \* 84)

print(("-" \* 22) + "Estos son las 3 ultimas Obras encontradas" + ("-" \* 21) + "\n")

printArtworkData(catalog, controller.lastThree(artworks))

stop\_time = time.process\_time()

elapsed\_time\_mseg = (stop\_time - start\_time)\*1000

print("Tiempo de ejecución: " + str(elapsed\_time\_mseg))

Las funciones más importantes son organizeTopNationality() que se encarga de organizar el catalogo por el tamaño de la lista de obras de cada nación, de mayor a menor, utilizando el método de merge sort haciendo que la complejidad sea O(n log n), luego integra a un diccionario el nombre de las naciones y el numero de obras que contiene por medio de un ciclo y organiza alfabéticamente las obras de esa nación (O(n log n)) dentro de un ciclo for-in que es de complejidad O(n). Haciendo que la complejidad de organizeTopNationality() sea O(n log n).

def organizeTopNationaly(catalog):

"""

Organiza el Top de Nacionalidades con más obras y tambien organiza alfabeticamente las obras de una nacionalidad.

"""

ms.sort(catalog["nations"], cmpfunction=compareSizes)

top = {}

for nation in lt.iterator(catalog["nations"]):

top[nation["name"]] = lt.size(nation["artworks"])

ms.sort(nation["artworks"], cmpfunction=compareAlphabet)

return top

Mientras que la segunda función más importante de este requerimiento es getArtworkbyNationality() la cual devuelve todas las obras de la nación top 1. Debido a que las naciones dentro del catalgo de naciones están guardadas desde la carga de datos de la siguiente manera:

ef newCountry(name):

"""

Crea un nuevo diccionario de pais para agregar su

nombre y obras relacionadas a ese pais.

"""

country = {"name": "", "artworks": None}

country["name"] = name

country["artworks"] = lt.newList()

return country

Es fácil obtener las obras por medio de la función del model.py getArtworksbyNationality() utilizando el nombre de la nacionalidad que se pide y un isPresent() que tiene complejidad O(n).

def getArtworkbyNationality(catalog, country):

"""

Devuelve todos las obras que pertenezcan a la nacionalidad

solicitada.

"""

nations = catalog["nations"]

pos\_country = lt.isPresent(nations, country)

if pos\_country > 0:

nation = lt.getElement(nations, pos\_country)

return nation["artworks"]

Haciendo que el requerimiento 4 tenga una complejidad de O(n).

Requerimiento 5:

En el view.py el requerimiento 3 se ejecuta cuando el usuario escoge la opción 6 del menú:

Text

Description automatically generated

Se pide un dato al usuario lo cual es una operación básica de tiempo t. Falta ver la complejidad de la función requerimiento3. Esta función hace lo siguiente:

Text

Description automatically generated

La función printArtworkData\_Req5() es muy similar a la función printArtworkData\_Req1() que ya analizamos, así que esta función tiene complejidad O(n). Veamos lo que hacen las otras funciones que van hasta el controller a ver si tienen complejidad mayor. . Con un indicador morado señalamos lo que es más relevante examinar (en términos de complejidad):

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Pues bien, la complejidad de la función artworksbyDepartment() es de O(n) pues usa dos ciclos de complejidad O(n). La función costoTotal() usa un ciclo de complejidad O(n) y usa la función costoTransporte() que aunque usa un ordenamiento, la complejidad de este es constante pues la lista a ordenar siempre tiene 4 elementos. En definitiva la función costoTotal() tiene complejidad O(n). La función pesoTotal usa solo un ciclo por lo que tiene complejidad O(n). Las funciones masAntiguas() y masCostosas() usan ciclos de complejidad O(n) (hay un ciclo doble en ambos casos pero su complejidad sigue siendo O(n) pues el primer ciclo tiene complejidad constante 5t), en particular, ambas funciones usan un ordenamiento de complejidad O(nlog(n)). En efecto, ambas funciones tienen complejidad O(nlog(n)).

Concluimos, pues, que el requerimiento 5 tiene complejidad O(nlog(n)) pues sus funciones más complejas tienen esta complejidad.

En conclusión:

|  |  |
| --- | --- |
| Requerimiento | Complejidad |
| Carga | O(n^2) |
| 1 | O(n) |
| 2 | O(n^2) |
| 3 | O(n^2) |
| 4 | O(n) |
| 5 | O(n log n) |

**Análisis de tiempo de ejecución:**

# **Ambientes de pruebas**

|  |  |  |
| --- | --- | --- |
|  | Máquina 1 | Máquina 2 |
| Procesadores | Intel® Core™ i5- 8250U CPU @ 1,60 GHz, 1.8 Ghz | 2,7 GHz Intel Core i5 de cuatro núcleos |
| Memoria RAM (GB) | 8GB | 16 GB 1600 MHz DDR3 |
| Sistema Operativo | Windowa 10 Home 64-bits | macOS Catalina Version. 10.15.7 |

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

# **Promedio de tiempo de ejecución entre Maquina 1 y Maquina 2**

Datos en milisegundos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tamaño de Datos | Carga de Datos | Req. 1 | Req. 2 | Req. 3 | Req. 4 | Req 5. |
| -10pct | 43873.51 | 5601.40 | 74999.26 | 3561.70 | 26934.91 | 36254.41 |
| -20pct | 102715.95 | 9976.10 | 298371 | 14207 | 101192.36 | 140737.40 |
| -50pct | 310026.015 | 19562.22 | 947765 | 87791.8 | 567074.65 | 825329.32 |
| Large | 693926.50 | 30043.71 | 2300472.88 | 334712.47 | 2093100.97 | 1850678.43 |

# **Graficas de análisis de tiempo de ejecución.**

# 









