**Análisis de complejidad**

**Nicolás Díaz Montaña – 202021006 –** [**n.diaz9@uniandes.edu.co**](mailto:n.diaz9@uniandes.edu.co)

**Daniel R Alonso A - 201419873 -** [**dr.alonso10@uniandes.edu.co**](mailto:dr.alonso10@uniandes.edu.co)

Resumen Ejecutivo:

|  |  |
| --- | --- |
| Requerimiento | Algoritmo |
| 1 | O(log n) |
| 2 | O(n) |
| 3 | O(n) |
| 4 | O(n) |
| 5 | O(n) |

Requerimiento 1:

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

print("-" \* 50 + "Requerimeinto 1 Inputs" + ("-" \* 50))

ciudad = input("UFO Sightings in the city of: ")

Requerimiento1(catalog, ciudad)

*Imagen 1 - Input Req 1*

El requerimiento empieza una vez el usuario ingresa el numero 3, lo primero que hace el programa es pedirle al usuario la ciudad que se quiere investigar. Después de eso se llama a la función Requerimiento1(), utilizando como parámetros el catalogo y la ciudad ingresada.

def Requerimiento1(catalog, ciudad):

avist\_city = controller.getAvistamientoporCiudad(catalog, ciudad.lower())

first = controller.firstThreeN(avist\_city)

last = controller.lastThreeN(avist\_city)

total = controller.indexSize(catalog, "cityIndex")

print("-" \* 50)

print("-" \* 50 + "Requerimeinto 1 Answers" + ("-" \* 50))

print("There are " + str(total) + " differrent cities with UFO sightings..." + "\n")

print("There are " + str(lt.size(avist\_city)) + " at the: " + ciudad + " city.")

print("-" \* 50)

print('Los 3 primeros avistamientos: ')

print("-" \* 50)

printData(first)

print("-" \* 50)

print('Los 3 ultimos avistamientos: ')

print("-" \* 50)

printData(last)

print("-" \* 50 + "\n")

*Imagen 2 - Requerimiento 1*

De aquí salen otras 4 funciones: getAvistamiento(), firstThreeN(), lastThreeN() y indexSize(). Las más importante, en terminos de tiempo, de las 4 es getAvistamiento() y indexSize(), por que son las que interactuan directamente con catalogo. Aún así el algoritmo de tanto firstThreeN() y lastThreeN() es de O(m), m siendo la cantidad de elementos de la lista avist\_city entregada por getAvistamiento().

def getAvistamientoporCiudad(catalog, ciudad\_entry):

"""

Esta función se encarga de encontrar los avistamientos de una ciudad dada por el usuario.

"""

exist = om.contains(catalog["cityIndex"], ciudad\_entry)

if exist:

avist = om.get(catalog["cityIndex"], ciudad\_entry)["value"]

ms.sort(avist, cmpfunction= compareDates2)

return avist

*Imagen 3 - model.getAvistamientoporCiudad()*

Como se puede evidenciar en la imagen 3, la función getAvistamiento() recibe como parámetros la ciudad ingresada por el usuario y el catalogo, entonces lo que se hace es revisar si la ciudad hace parte de los índices del catalogo de “cityIndex” por medio de la función om.contains(), el cual tiene un algoritmo O(log n). Luego se busca y se obtiene la lista de los avistamientos de esa ciudad con la función om.get(), la cual es O(1). Luego se hace un merge sort a esa lista, el cual tiene un algoritmo de O(m log m), siendo otra vez el tamaño de elementos de esa lista. Lo cual nos lleva a concluir que toda la función es de O(log n)

Por otro lado la función de indexSize() se encarga por medio de la función om.size() ver el tamaño de elementos del catalogo “cityIndex”, el cual tiene un algoritmo de O(log n).

Finalmente, todo este análisis nos lleva a la conclusión del que el Requerimiento 1 resulta ser de O(log n).

Requerimiento 2:

El requerimiento inicia cuando el usuario ingresa la opción 4. Se piden un par de inputs por parte del usuario y se va a la función req2:

Text

Description automatically generated

Text

Description automatically generated

Las dos funciones principales de este requerimiento son entonces maxDurationD() y req2(). Examinemos cada una:

Text

Description automatically generated

Esta función solo usa funciones propias del ordered map así que su complejidad sería constante O(1).

Text

Description automatically generated

La función req2() itera sobre las llaves del rango en cuestión para sacar el número de avistamientos en el rango. Esto crece a medida que los datos crecen así que tiene complejidad O(n). Luego, se hace 6 veces la operación de remover una llave y hallar el dato correspondiente para agregarlo a una lista que, a su vez, se ordena por medio de un mergesort() para finalmente sacar agregar estos datos a otra lista. Esto tendría complejidad O(log(n)) por mucho.

Así, concluimos que el requerimiento 2 tiene una complejidad O(n).

Requerimiento 3:

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

hora\_inicial = input("Límite inferior en formato HH: MM.: ")

hora\_final = input("Límite superior en formato HH: MM.: ")

Requerimiento3(catalog, hora\_inicial, hora\_final)

*Imagen # - input Requerimiento 3*

El requerimiento inicia una vez que el usuario ingrese el numero 5. Una vez iniciado el programa, este le pedir al usuario que ingrese una hora inicial y una hora final. Despues se llamara a la función Requerimiento3().

def Requerimiento3(catalog, hora\_inicial, hora\_final):

late = controller.maxKey(catalog, "hourIndex")

size\_late = lt.size(controller.getAvistamientoPorHora(catalog, late))

org = controller.organizarAvistamientoPorRangoHora(catalog, hora\_inicial, hora\_final)

first = controller.firstThreeN(org)

last = controller.lastThreeN(org)

print("There are " + str(controller.indexSize(catalog, "hourIndex")) + " UFO sightings with different times [hh:mm:ss]...")

print("The latest UFO sightings time is: ")

print(str(late) + ": " + str(size\_late) + "\n")

print("There are " + str(lt.size(org)) + " sightings " + hora\_inicial + " and " + hora\_final)

print('Los 3 primeros avistamientos: ')

print("-" \* 50)

printData(first)

print("-" \* 50)

print('Los 3 ultimos avistamientos: ')

print("-" \* 50)

printData(last)

print("-" \* 50 + "\n")

*Imagen # - Requerimiento 3*

Aquí se debe encontrar primero la hora más tardía, o índice más grande, en el catalogo “hourIndex” y devolver su tamaño. Para eso se utiliza la función controller.maxKey(), la cual recorre todo el catalogo lo cual significa que tiene un algoritmo O(n). Y luego se utiliza la función getAvistamientoPorHora() dentro de una función lista size() para determinar el tamaño de la lista que devuelva la primera función.

def getAvistamientoPorHora(catalog, hora\_entry):

"""

Esta función se encarga de encontrar los avistamientos de una hora dada por el usuario.

"""

exist = om.contains(catalog["hourIndex"], hora\_entry)

if exist:

avist = om.get(catalog["hourIndex"], hora\_entry)["value"]

return avist

*Imagen # - model.getAvistamientoPorHora()*

En esta función simplemente se revisa si se encuentra la llave dada dentro del catalogo “hourIndex” por medio om.contains(), el cual tiene un algoritmo de O(1). Y si se este se encuentra dentro del catalogo, fácilmente obtiene los avistamientos por medio om.get() el cual es O(1).

def organizarAvistamientoPorRangoHora(catalog, hora\_inicial, hora\_final):

"""

Organiza y retorna los avistamientos que esten en un rango de

una hora inicial y una hora final.

"""

org = lt.newList()

time\_1 = datetime.datetime.strptime(hora\_inicial,"%H:%M")

time\_2 = datetime.datetime.strptime(hora\_final,"%H:%M")

time\_interval = time\_2 - time\_1

h = str(time\_interval)

hh, mm, ss = h.split(':')

delta = int(hh) \* 60 + int(mm) + int(ss) \* 0

for minute in range(delta + 1):

new\_day = time\_1 + datetime.timedelta(minutes=minute)

new\_hour = new\_day.strftime("%H:%M:%S")

getAvistamientoPorHora2(catalog, str(new\_hour), org)

return org

*Imagen # - model.organizarAvistamientoPorRangoHora()*

Después de haber completado la primera parte del Requerimiento 3, ahora toca buscar los avistamientos por un rango de horas. Esto empieza llamando a la función organizarAvistamientosPorRangoHora() tomando como parámetros el catalogo y las horas ingresadas por el usuario. Aquí lo que se hace es recorrer un delta entre la hora final e inicial lo cual seria O(n).

def getAvistamientoPorHora2(catalog, hora\_entry, org):

"""

Esta función se encarga de encontrar los avistamientos de una hora dada por el usuario.

"""

exist = om.contains(catalog["hourIndex"], hora\_entry)

if exist:

avist = om.get(catalog["hourIndex"], hora\_entry)["value"]

for avit in lt.iterator(avist):

lt.addLast(org, avit)

*Imagen # - model.getAvistamientoPorHora2 ()*

Al recorrer ese Delta, se va buscando por cada minuto si una hora esta en el catalogo “hourIndex” por medio de la función getAvistamientoPorHora2() que hace exactamente lo mismo que getAvistamientoPorHora() con la diferencia de que recorre la lista de avistamientos que encontró y las mete dentro de la lista “org” que se creo en la función organizarAvistamientosPorRangoHora(), lo cual lo haría O(a).

Lo cual lleva a que finalmente que la función organizarAvistamientoPorRangoHora() fuera O(n + a).

Con todo esto se puede concluir que el Requerimiento 3 tiene una complejidad de O(n).

Requerimiento 4:

El requerimiento inicia cuando el usuario ingresa la opción 4. Se piden un par de inputs por parte del usuario y se va a la función req2:

Text

Description automatically generated

Text

Description automatically generated

Las dos funciones principales de este requerimiento son entonces maxDurationD() y req2(). Examinemos cada una:

Text

Description automatically generated

Esta función solo usa funciones propias del ordered map así que su complejidad sería constante O(1).

Text

Description automatically generated

La función req4() itera sobre las llaves del rango en cuestión para sacar el número de avistamientos en el rango. Esto crece a medida que los datos crecen así que tiene complejidad O(n). Luego, se hace 6 veces la operación de remover una llave y hallar el dato correspondiente para agregarlo a una lista que, a su vez, se ordena por medio de un mergesort() para finalmente sacar agregar estos datos a otra lista. Esto tendría complejidad O(log(n)) por mucho.

Así, concluimos que el requerimiento 4 tiene una complejidad O(n).

Requerimiento 5:

elif int(inputs[0]) == 7:

long0 = input("Ingrese la longitud inicial: ")

long1 = input("Ingrese la longitud final: ")

lat0 = input("Ingrese la latitud inicial: ")

lat1 = input("Ingrese la latitud final: ")

Requerimiento5(catalog, long0, long1, lat0, lat1)

El requerimiento empieza una vez el usuario ingrese el numero 7, luego el programa le pedirá que ingrese las longitudes y latitudes iniciales y finales, después llamara a la función Requerimiento5().

def Requerimiento5(catalog, long0, long1, lat0, lat1):

lst = controller.avistamientosPorGeografia(catalog, long0, long1, lat0, lat1)

tamaño = lt.size(lst)

last = controller.lastFiveD(lst)

firts = controller.firstFiveD(lst)

print("=" \* 50 + " Req No. 5 Answers " + "=" \* 50)

print("There are " + str(tamaño) + " different UFO sightings in the current area")

print("-" \* 50)

print('Los 5 primeros avistamientos: ')

print("-" \* 50)

printData(firts)

print('Los 5 primeros avistamientos: ')

print("-" \* 50)

printData(last)

En la función Requerimiento la más importante y que produce más impacto en términos de tiempo es avistamientosPorGeografía()

def avistamientosPorGeografia(catalog, long0, long1, lat0, lat1):

"""

Esta función devuelve una lista con los avistamientos que

se encuentran en un rango de latitudes y longitudes.

"""

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

lst = om.values(catalog["coordIndex"], long0, long1) #O(n)

for avist in lt.iterator(lst):

for a in lt.iterator(avist):

if float(a["latitude"]) >= float(lat0) and float(a["latitude"]) <= float(lat1):

lt.addLast(final, a)

return final

En avistamientosPorGeografía() lo que se hace es utilizar la función om.values() para buscar los avistamientos en el rango de longitudes, devolviendo una lista con ellos. Después se recorre esa lista para añadir a una lista final los avistamientos que corresponden al rango de latitudes. Haciendo que todo este proceso sea O(n + a) que al final de cuentas es O(n).

En conclusión, tomando este análisis en cuenta el Requerimiento 5 es O(n).