Documento de Análisis Reto 3

Grupo 3

- Requerimiento 2: Ana Sofía Villa Benavides, 201923361, as.villa@uniandes.edu.co
- Requerimiento 3: Daniela Alejandra Camacho Molano, 202110974, d.camachom@uniandes.edu.co

Requerimiento 0 (carga de datos)

-NewCatalog (): Esta función es O (1) debido a que se crea a un diccionario con 6 llaves principales:

1. Registros: en donde se crea otro diccionario en el cual las llaves son la información administrada por la base de datos.
2. IndiceCiudad: Se crea una tabla de hash en el que las llaves serán el nombre de la ciudad correspondiente. Como método se utilizó separate chaining.
3. IndiceDuración: Se crea un mapa ordenado tipo RBT el cual tendrá como llave la duración en segundos.
4. IndiceHoraMinuto: Se crea un mapa ordenado tipo RBT el cual tendrá como llave la duración en horas y minutos
5. IndiceFechas: Se crea un mapa ordenado tipo RBT el cual tendrá como llave las fechas de los avistamientos correspondientes
6. IndiceLatitud: Se crea un mapa ordenado tipo RBT el cual tendrá como llave la latitud.}

```
def newCatalog():
    """ Inicializa el analizador

    Crea una lista vacia para guardar todos los registros
    Se crean indices (Maps) por los siguientes criterios:
    -Ciudad

    Retorna el analizador inicializado.
    """
    catalogo = {'registros': None,
                'indiceCiudad': None,
                'indiceDuracion': None,
                'indiceHoraMinuto':None,
                'indiceFechas':None,
                'indiceLatitud':None,
                'indiceLongitud':None
                }

    catalogo['registros'] = lt.newList('ARRAY_LIST')
    catalogo['indiceCiudad'] = mp.newMap(50000,maptype="CHAINING",loadfactor=4)
    catalogo['indiceDuracion'] = om.newMap(omaptype='RBT',
                                           comparefunction=cmpDuracion)
    catalogo['indiceHoraMinuto'] = om.newMap(omaptype='RBT',
                                           comparefunction=cmpHoraMinuto)
    catalogo['indiceFechas'] = om.newMap(omaptype='RBT',
                                           comparefunction=cmpFechas)
    catalogo['indiceLatitud'] = om.newMap(omaptype='RBT',
                                           comparefunction=cmpCoordenada)
    return catalogo
```

-**AddRegistro (catalogo, registro):** Esta función se divide en dos partes importantes:

1. Adiciona al catálogo en la llave de registros toda la información de uno de los avistamientos con las llaves de: fechahora, ciudad, estado, país, país-ciudad, forma, duracionsegundos, duracionvariable, date posted, latitud y longitud. Esta operación.

2. Se actualizan los mapas de índices creados en la función anterior. Para cada índice se usa una función diferente, estas son:

updateIndiceCiudad (map, ciudad, fecha, registro): Se accede al valor de la ciudad por su llave en la tabla de hash lo que resulta O (1). Si esta llave no existe entonces se crea en la tabla de hash.El valor de la llave va a corresponder a un árbol RBT ordenado por FechaHora, entonces cada avistamiento que llegue desde addregistro va a ir almacenándose en un árbol por fecha que se encuentra dentro de un valor de una tabla de hash organizada por ciudad. Adicionar un valor a este arbol tendría en el peor caso la complejidad de su altura O(h) cómo se trata de un RBT está relativamente balanceado y su altura se aproxima a log2n por lo tanto la complejidad sería alrededor de ~ O(lognC) donde nC corresponde a los avistamientos de una ciudad en específico.

updateIndiceDuracion (map, registro): Esta función actualiza el mapa RBT para el índice de duración en segundos. Primero busca el nodo que corresponde a la duración que tiene el registro dentro del árbol RBT que almacena todos los registros, eso resulta O(logND) donde ND es el número de duraciones diferentes entre los registros. Luego cuando ya llega a este nodo, este tiene como valor otro árbol que se encuentra organizado por fechahora, agregar a este árbol dentro del árbol resulta O(lognD) donde nD es el número de registros que tienen esa misma duración. Por lo tanto, tiene complejidad de O(LogND) +O(LognD)= ~O(LogND)

updateHoraMinuto: Primero, toma la fecha y hora del registro y por medio de data time de la fecha inicial solo se toma en cuenta las horas y los minutos. Luego, se actualiza el mapa RBT para el índice de tiempo en horas y minutos. Para ello, si la llave no existe en el mapa se crea una lista vacía y se añade al mapa con la llave correspondiente, también se crea un nuevo mapa tipo RBT y se añade el registro por medio de la fecha. Si ya existe esta llave se obtiene la pareja llave valor y añade el valor por medio de una lista en la llave existente (esta actualización es por medio del índice de la fecha completa). Al hacer esta actualización su complejidad es de O(logFC), donde FC es la fecha completa del registro. Posteriormente a esto se actualiza el subárbol por medio de la fecha en horas y minutos, esto tiene complejidad de O(logHM), donde HM es los diferentes tiempos que se encuentran en los registros.

updateFechas: A partir de la entrada de fechaHora, se toma únicamente la fecha, es decir el año, el mes y el día del avistamiento y se verifica si ya existe un nodo del árbol de fechas con dicha fecha esto sería O(lognFechas) en el peor caso. Luego, se accede al valor que corresponde a otro mapa ordenado por Hora Minuto, en este subárbol se adiciona el registro, para esto la complejidad sería la altura del subárbol de hora minuto de una fecha en específico lo que sería O(logn) donde n corresponde a los registros con esa fecha.

updateLatitud: Esta función actualiza el mapa RBT para el índice de latitud. Primero busca el nodo que corresponde a la latitud que tiene el registro dentro del árbol RBT que almacena todos los registros, eso resulta O(logNLAT) donde NLAT es el número de latitudes diferentes entre los registros. Este nodo tiene como valor otro árbol que se encuentra organizado por longitud, agregar a este árbol dentro del árbol resulta O(lognLON) donde nLON es el número de registros que tienen esa misma latitud. Por lo tanto, tiene complejidad de O(LogNLAT) +O(LognLON)= ~O(LogNLAT)

Requerimiento 1

registrosPorCiudad

Primero se toma la ciudad dada como llave y se accede a su valor en la tabla de hash, esto es O (1). Su valor corresponde a un árbol de nC registros que son de esa ciudad, este árbol es un RBT ordenado por FechaHora. Luego para tener una lista de todos los registros de la ciudad ordenado por fecha simplemente se obtiene con om.ValueSet todos los valores del árbol esto sería O(nC). Estos valores deberían corresponder a un solo registro, sin embargo, considerando que puede haber más de un registro con misma ciudad y fecha entonces se recorre de cada valor y se agrega a la lista final de respuesta.

Total: O(nC) [nC= registros de la ciudad seleccionada]

```
def registrosPorCiudad(catalogo,nombreCiudad):
    respuesta= lt.newList("ARRAY_LIST")
    par= mp.get(catalogo['indiceCiudad'], nombreCiudad)
    if par== None:
        registros=None
    else:
        mapaFechaCiudad= me.getValue(par)
        registros= om.valueSet(mapaFechaCiudad)
        for registro in lt.iterator(registros):
                lt.addLast(respuesta,registro)
    return(respuesta)
```

Requerimiento 2 (Ana Sofia Villa)

RegistrosEnRangoDuracion:

Primero se toma los límites de duración en segundos dada como llaves y se accede a los valores en este rango del árbol RBT organizado con este índice con om.values. Esto sería en el peor caso O(h) de este árbol que sería aproximadamente O(logND) donde ND corresponde a el número de diferentes duraciones que se tienen como nodos en el árbol, en el peor caso si todos los registros tienen duración distinta este N se aproxima al NT total de registros.

Luego, de la respuesta de valores obtenidas en el rango de duración corresponde a una lista de mapas ordenados por fechahora. Se recorre esta lista de mapas y de cada mapa se obtiene la lista de sus valores, cada uno de estos valores se agrega a la lista de respuesta.

Total: O(logND) [ND= número de nodos de duraciones]

```
#REQ 2#
def registrosEnRangoDuracion(catalogo,limiteMaximo,limiteMinimo):
    listaEnRango= lt.newList("ARRAY_LIST")
    listaDeMapas = om.values(catalogo['indiceDuracion'],limiteMinimo,limiteMaximo)
    if lt.isEmpty(listaDeMapas)==False:
        for Mapa in lt.iterator(listaDeMapas):
            registrosDuracion= om.valueSet(Mapa)
            for registros in lt.iterator(registrosDuracion):
                for registro in lt.iterator(registros):
                    lt.addLast(listaEnRango,registro)
    return listaEnRango
```

Requerimiento 3 (Daniela Camacho)

NumAvistamientosPorHoraMinuto

Primero, se toman el límite inferior y superior indicado por el usuario y por medio de ellos se accede a los valores encontrados en este rango por medio de om.values; esto tiene complejidad de O(logHM), donde HM es el número de diferentes tiempos (en horas y minutos) que se encuentra en el registro. A partir de esto, se recorre el mapa ordenado por medio de la fecha para ordenar la información por medio de este índice y se agregan los valores a lista final.

Total= O(logHM)

```
def NumAvistamientosPorHoraMinuto (catalogo,inferior,superior):
    inferior= datetime.datetime.strptime(inferior,"%H:%M:%S")
    inferior=datetime.time(inferior.hour,inferior.minute)
    superior=datetime.datetime.strptime(superior,"%H:%M:%S")
    superior=datetime.time(superior.hour,superior.minute)
    mapMinutoHora=catalogo["indiceHoraMinuto"]
    rangoKey=om.values(mapMinutoHora,inferior,superior)
    numAvistamientos=lt.size(rangoKey)
    listaInfo=lt.newList("ARRAY_LIST")
    for i in lt.iterator(rangoKey):
        value=om.valueSet(i)
        for n in lt.iterator(value):
            for j in lt.iterator(n):
                lt.addLast(listaInfo,j)
    dicRta={'avistamientos':numAvistamientos,'info':listaInfo}
    return(dicRta)
```

Requerimiento 4

registrosenRangoFecha

Primero, se toman el límite inferior y superior indicado por el usuario y por medio de ellos se accede a los valores encontrados en este rango por medio de om.values; esto tiene complejidad de O(logYMD), donde YMD es el número de diferentes fechas en formato AA-MM-DD que se encuentran en el registro. A partir de esto, se recorre el mapa ordenado por medio de la fecha para ordenar la información por medio de este índice y se agregan los valores a lista final.

Total=O(logYMD)

```
def registrosenRangoFecha (catalogo,liminferior,limsuperior):
    inferior= datetime.datetime.strptime(liminferior,'%Y-%m-%d')
    inferior=datetime.date(inferior.year,inferior.month,inferior.day)
    superior= datetime.datetime.strptime(limsuperior,'%Y-%m-%d')
    superior=datetime.date(superior.year,superior.month,superior.day)
    mapFecha=catalogo["indiceFechas"]
    listaEnRangoFecha= lt.newList("ARRAY_LIST")
    listaDeMapasporHora = om.values(mapFecha,inferior,superior)
    if lt.isEmpty(listaDeMapasporHora)==False:
        for Mapa in lt.iterator(listaDeMapasporHora):
            registrosHora= om.valueSet(Mapa)
            for registros in lt.iterator(registrosHora):
                for registro in lt.iterator(registros):
                    lt.addLast(listaEnRangoFecha,registro)
    return listaEnRangoFecha
```

Requerimiento 5

AvistamientosPorZonaGeografica

Primero se toma los límites de latitud dados como llaves y se accede a los valores en este rango del árbol RBT organizado con este índice con om.values. Esto sería en el peor caso O(h) de este árbol que sería aproximadamente O(logNLAT) donde NLAT corresponde a el número de diferentes latitudes que se tienen como nodos en el árbol, en el peor caso si todos los registros tienen latitudes distintas este N se aproxima al NT total de registros.

Luego, la respuesta de valores obtenidas en el rango de latitudes corresponde a una lista de mapas ordenados por longitudes. Se recorre esta lista de mapas y de cada mapa se obtiene la lista de valores dentro del rango de longitudes con om.values cada uno de estos valores se agrega a la lista de respuesta, esto es O(lognLON) donde nLON corresponde al número de longitudes diferentes dentro del rango seleccionado de latitudes.

Total: O(logNLAT)+ O(lognLON)) = O(logNLAT)

```
def avistamientosPorZonaGeografica(catologo,longitudMin,longitudMax,latitudMin,latitudMax):
    mapLatitud=catologo["indiceLatitud"]
    ListadeMapasenRangoLatitud=om.values(mapLatitud,latitudMin,latitudMax)
    ListaRangoLatyLon= lt.newList("ARRAYLIST")
    for mapaLongitudes in lt.iterator(ListadeMapasenRangoLatitud):
        listaRegistros= om.values(mapaLongitudes,longitudMin,longitudMax)
        for registros in lt.iterator(listaRegistros):
            for registro in lt.iterator(registros):
                lt.addLast(ListaRangoLatyLon,registro)
    return(ListaRangoLatyLon)
```

Requerimiento 6 (BONO)

Para obtener los datos entre las coordenadas se utiliza la misma función que en el requerimiento 5. Luego para poder visualizarlo se utiliza folium y el plugin MarkerCluster.

Primero se crea un mapa vacío que empieza con un zoom de 4.5 centrado en el promedio de las longitudes y latitudes de entrada. Además, se usa el plugin de Marker Cluster para que dependiendo del zoom los avistamientos de agrupen en ciertos grupos por cercanía de manera visual. Luego, para cada registro de la lista obtenida (O NLL), donde NLL es el número de avistamientos en el rango de latitud y longitud, se agrega un marcador al mapa generado con folium.

```
import folium
from folium.plugins import MarkerCluster
```

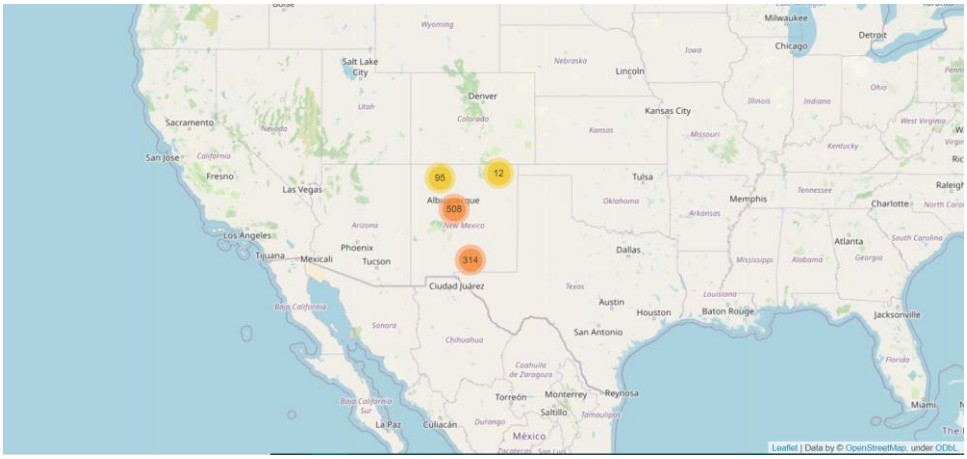
```
def mapaBONO(lista, lati,longi):
    m = folium.Map(location=[lati, longi], zoom_start=4.5)
    marker_cluster = MarkerCluster().add_to(m)
    for i in lt.iterator(lista):
        lat= i["latitud"]
        lon=i["longitud"]
        ciudadPais=str(i["ciudad"])+"-"+str(i["pais"])
        tooltipp="Click para mas información"
        folium.Marker(
            location=[lat, lon],
            popup='</p><strong>Fecha y Hora:<strong></p>'+str(i['fechahora'])+'<p>Ciudad-País:</p>'+ciudadPais+
                    '</p><p>Forma:</p>'+str(i["forma"]) +'<p>Duración:</p>'+str(i["duracionsegundos"])+
                    '<p>Longitud:</p>'+str(i["longitud"])+'</p><p>Latitud:</p>'+str(i["latitud"])+'</p>',
            tooltip=tooltipp,
            icon=folium.Icon(color="green", icon="ok-sign"),
                    ).add_to(marker_cluster)
    m.save("mapa"+str(lati)+"-"+str(longi)+".html")
```

En el repositorio se encuentra ya guardado un mapa de ejemplo llamado “mapa34.065--106.025.html” este se realizó a partir de las coordenadas que se dan el a guía del reto 3: “los avistamientos reportados en la zona de Nuevo México que se encuentra en una longitud desde los -103.00 a -109.05 y una latitud desde 31.33 a 37.00”

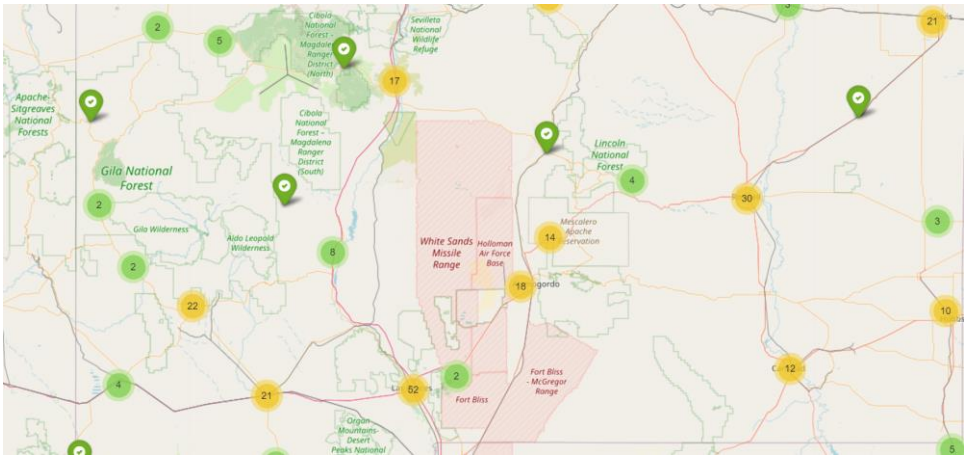
A continuación, se encuentra la salida en la terminal y luego imágenes de cómo se ven el mapa desde el browser.

```
Ingrese el limite máximo de latitud 37.00
Ingrese el limite minimo de latitud 31.13
Ingrese el limite máximo de longitud -103.00
Ingrese el limite minimo de longitud -109.05
El total de avistamientos en el área es: 929
Se ha guardado el mapa en el archivo:mapa34.065--106.025.html
```

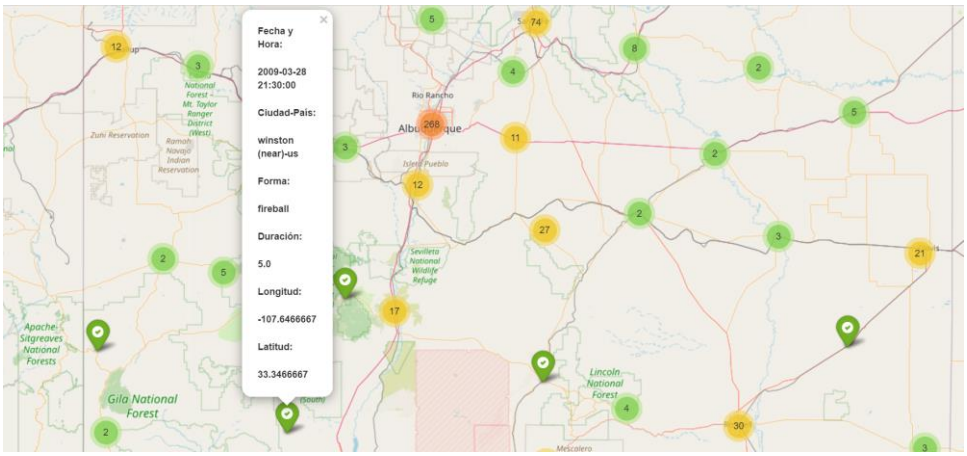
Al inicio sale alejado de la siguiente manera:



Luego se puede ir haciendo zoom en las diferentes zonas:



Si se hace click en cualquier marcador, se muestra la información requerida del avistamiento al cual corresponde:


Fecha y Hora:
2009-03-28 21:30:00
Ciudad-Pais:
winston (near)-us
Forma:
fireball
Duración:
5.0
Longitud:
-107.6466667
Latitud:
33.3466667
Rio Rancho
Albuquerque
Isleta Pueblo
Zuni Reservation
Ramah Navajo Indian Reservation
Sevilleta National Wildlife Refuge
Apache-Sitgreaves National Forests
Gila National Forest
Lincoln National Forest
Mescalero