Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

Sara Ramos Ramos, 202321247, s.ramosr2@uniandes.edu.co

Juan Pablo Garcia Osorio, 202421543,jp.garciao1@uniandes.edu.co

Sebastian Lara Urquijo, 202420591, s.larau@uniandes.edu.co

Requerimiento <<1>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

```
def req_1(catalog, anioB):

    """

    Retorna el resultado del requerimiento 1

    """

    # TODO: Modificar el requerimiento 1

    inicio = get_time()

    fechaUltimoRegistro = None

    indexUR = -1

    for i in range(0, lt.size(catalog['registros'])):

        anio = int(lt.get_element(catalog["registros"], i)["year_collection"])

        if anio == int(anioB):

            fechaR = lt.get_element(catalog["registros"], i)["load_time"]

            if fechaUltimoRegistro == None:

                fechaUltimoRegistro = fechaR

                indexUR = i

            else:

                fechaDT = datetime.strptime(fechaR, "%Y-%m-%d %H:%M:%S")

                fechaURDT = datetime.strptime(fechaUltimoRegistro, "%Y-%m-%d %H:%M:%S")
```

```
            if fechaDT > fechaURDT:
                fechaUltimoRegistro = fechaR
                indexUR = i
    fin = time.time()
    tiempo_ejecucion = (fin - inicio)  # Convertir a milisegundos
    return indexUR, tiempo_ejecucion
```

Entrada	Catalog: Diccionario que contiene la lista de datos anioB: Año de recopilación que se busca en el catalogo
Salidas	Tiempo de la ejecución del requerimiento en milisegundos. Número total de registros que cumplieron el filtro por año de recopilación. Para el registro encontrado se debe presentar la siguiente información: Año de recopilación del registro.(ej.“2007”) Fecha de carga de registros en la plataforma. (con formato "%Y-%m-%d" .ej.“2012-05-15”). Tipo de fuente/origen del registro. (ej.: “CENSUS” o “SURVEY”) Frecuencia de la recopilación del registro. (ej.: “ANNUAL”, “WEEKLY”, etc.) Nombre del departamento del registro. Tipo del producto del registro (ej.: “HOGS”, “SHEEP”, etc.) Unidad de medición del registro. (ej.: “HEAD”, “$”, etc.) Valor de la medición del registro.
Implementado (Sí/No)	Si se implementó

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido del catalog (for)	O(n)
Comparación del año (if)	O(1)
Conversión de fecha	O(1)
Comparación de fechas (if)	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	Apple M3
Memoria RAM	8 GB
Sistema Operativo	macOS

2005

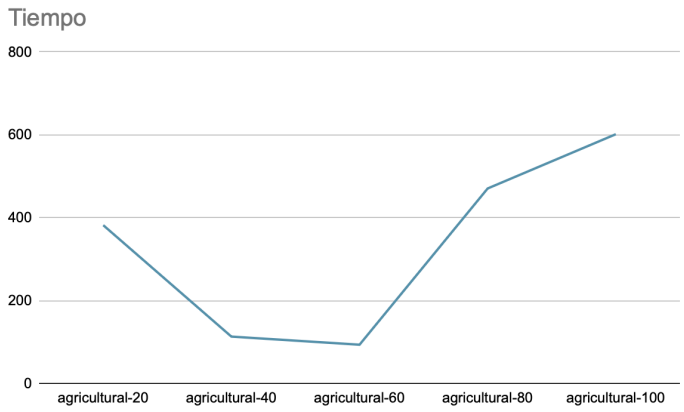
Entrada	Tiempo (ms)
agricultural-20	381.806
agricultural-40	112.861
agricultural-60	093.298
agricultural-80	470.454
agricultural-100	601.409

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
agricultural-20	Dato1	381.806
agricultural-40	Dato2	112.861
agricultural-60	Dato3	093.298
agricultural-80	Dato4	470.454
agricultural-100	Dato5	601.409

Gráficas



Análisis

Se observa en la grafica que el tiempo de ejecución no esta siguiendo una continuidad completamente lineal respecto al tamaño del conjunto de datos. El aumento en los datos de *agricultural-80* y *agricultural-100* es significativo, lo que nos da a entender que la implementación se vuelve menos eficiente con datos más grandes.

Requerimiento <<2>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

```
def req_2(catalog, depB):

    """

    Retorna el resultado del requerimiento 2

    """

    inicio = get_time()

    fechaUltimoRegistro = None

    indexUR = -1

    for i in range(0, lt.size(catalog['registros'])):

        departamento = lt.get_element(catalog["registros"], i)["location"]

        if departamento == depB:

            fechaR = lt.get_element(catalog["registros"], i)["load_time"]

            if fechaUltimoRegistro == None:

                fechaUltimoRegistro = fechaR

                indexUR = i

            else:

                fechaDT = datetime.strptime(fechaR, "%Y-%m-%d %H:%M:%S")
```

```
            fechaURDT = datetime.strptime(fechaUltimoRegistro, "%Y-%m-%d %H:%M:%S")
            if fechaDT > fechaURDT:
                fechaUltimoRegistro = fechaR
                indexUR = i
    fin = time.time()
    tiempo_ejecucion = (fin - inicio) # Convertir a milisegundos
    return indexUR, tiempo_ejecucion
```

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Catalog: Diccionario que contiene listas con los datos dep: Nombre del departamento buscar en la base de datos
Salidas	Tiempo de la ejecución del requerimiento en milisegundos. Número total de registros que cumplieron el filtro por nombre de departamento. Para el registro encontrado se debe presentar la siguiente información: Año de recopilación del registro. (ej.: “2007”) Fecha de carga del registro. (con formato "%Y-%m-%d".ej.: “2012-05-15”). Tipo de fuente/origen del registro. (ej.: “CENSUS” o “SURVEY”) Frecuencia de la recopilación del registro. (ej.: “ANNUAL”, “WEEKLY”, etc.) Nombre del departamento del registro. Tipo del producto del registro (ej.: “HOGS”, “SHEEP”, etc.) Unidad de medición del registro. (ej.: “HEAD”, “$”, etc.) Valor de la medición del registro
Implementado (Sí/No)	Si se implementó

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido del catalog (for)	O(n)
Comparación del departamento (if)	O(1)
Comparación de fechas	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	Intel i7 14700k
Memoria RAM	32GB
Sistema Operativo	Windows

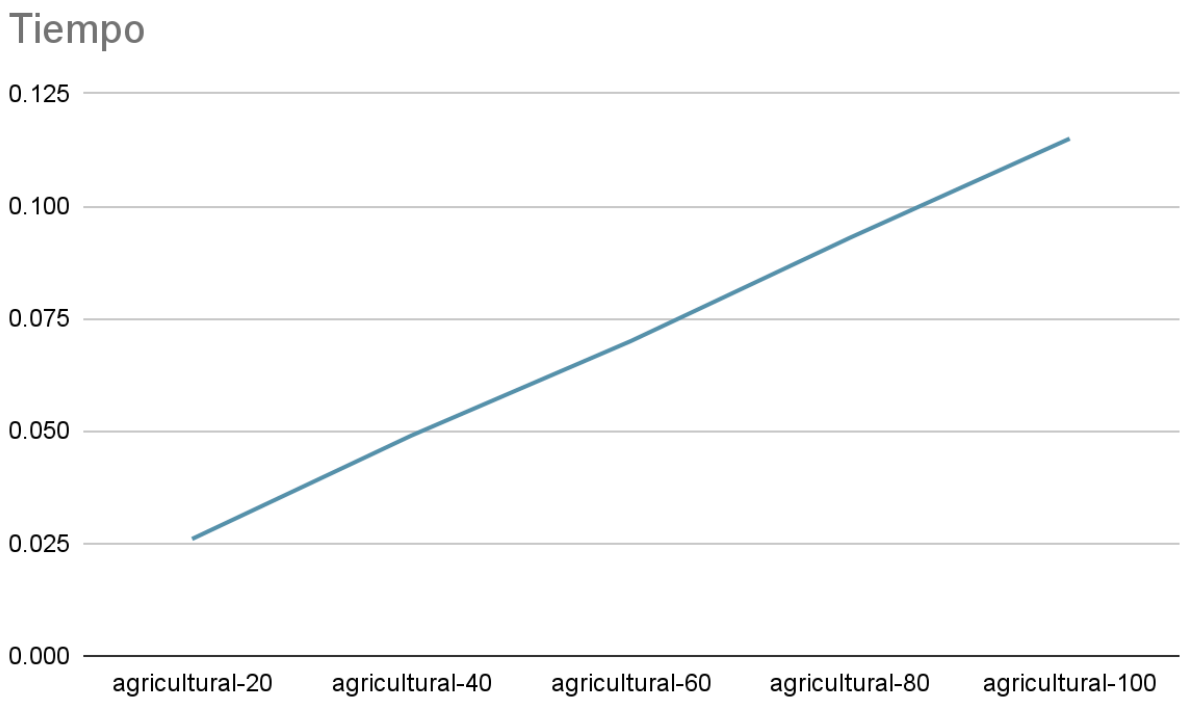
Entrada	Tiempo (ms)
agricultural-20	0.026 ms
agricultural-40	0.049 ms
agricultural-60	0.07 ms
agricultural-80	0.093 ms
agricultural-100	0.115 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
agricultural-20	Dato1	0.026 ms
agricultural-40	Dato2	0.049 ms
agricultural-60	Dato3	0.07 ms
agricultural-80	Dato4	0.093 ms
agricultural-100	Dato5	0.115 ms

Gráficas



Análisis

El código busca el último registro con una ubicación específica al comparar fechas de carga sin ordenar la lista, de forma parecida al requerimiento que usa el año de recopilación. Su eficiencia depende de la estructura de datos usada, lo que podría afectar el rendimiento con grandes volúmenes de información.

Requerimiento <<3>>

Descripción

```
def req_3(catalog, nombre_departamento, year_inicio, year_final):
    """
    Retorna el resultado del requerimiento 3
    """
    start_time = get_time()
    registros_copilados = []
    total_registros = 0
    total_survey = 0
```

```
    total_census = 0




    for i in range(lt.size(catalog["registros"])):
        registro = lt.get_element(catalog["registros"], i)
        departamento = registro["state_name"]
        if departamento == nombre_departamento:
            year = int(registro["year_collection"])


            if year_inicio <= year <= year_final:
                total_registros +=1


                if registro["source"] == "CENSUS":
                    tipo_fuente = "CENSUS"
                    total_census += 1
                elif registro["source"] == "SURVEY":
                    tipo_fuente = "SURVEY"
                    total_survey += 1
                fecha_carga = registro["load_time"]
                fecha_carga = datetime.strptime(fecha_carga, "%Y-%m-%d
%H:%M:%S").strftime("%Y-%m-%d")
                registros = {
                    "Fuente": tipo_fuente,
                    "Año_Recopilacion": year,
                    "Fecha_Carga": fecha_carga,
                    "Frecuencia": registro["freq_collection"],
                    "Tipo de Producto": registro["commodity"],
                    "Unidad": registro["unit_measurement"]
```

```
            }

            registros_copilados.append(registros)


if len(registros_copilados) > 20:
    registros_filtrados = registros_copilados[:5] + registros_copilados[-5:]
end_time = get_time()
tiempo_ejecucion = delta_time(start_time, end_time)


return {
    "Tiempo de ejecución": tiempo_ejecucion,
    "Total registros filtrados": total_registros,
    "Total registros (SURVEY)": total_survey,
    "Total registros (CENSUS)": total_census,
    "Registros": registros_filtrados
}
```

El requerimiento 3 hace una filtracion por el nombre de departamentos solicitado por el usuario, posterior a eso se filtra dentro de un rango de años, seleccionando los 5 primeros y 5 ultimos de los resultados propuestos.

Entrada	Catalog: Diccionario con los datos cargados nombre_departamento: Nombre del departamento a filtrar year_inicio: Año inicial del rango de busqueda year_final: Año final del rango de busqueda
Salidas	Tiempo de ejecución del requerimiento Número total de registros encontrados Número total de registros con fuente tipo “SURVEY” Número total de registros con fuente tipo “CENSUS” Lista con los registros encontrados, si hay mas de 20, retornar los primero 5 y los ultimos 5
Implementado (Sí/No)	Si se implementó, por: Sara Ramos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Recorrido de catalog (for)	O(n)
Comparación de departamento (if)	O(1)
Comparación de año (if)	O(1)
Conteo de registros por tipo de fuente (if - elif)	O(1)
Crear diccionario con todos los registros	O(1)
Filtracion de registros si hay mas de 20	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Apple M3**
Memoria RAM	8 GB
Sistema Operativo	macOS

Parametros: TEXAS, 2010 - 2020

Entrada	Tiempo (ms)
agricultural-20	60.874
agricultural-40	74.417
agricultural-60	70.746
agricultural-80	111.599
agricultural-100	146.032

Tablas de datos

Las tablas con la recopilación de datos de las pruebas/

Muestra	Salida	Tiempo (ms)
agricultural-20	Dato1	60.874
agricultural-40	Dato2	74.417
agricultural-60	Dato3	70.746
agricultural-80	Dato4	111.599
agricultural-100	Dato5	146.032

Gráficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

En la grafica se logra observar un comportamiento no completamente lineal, con un ligero descenso en *agricultural-60* antes de un incremento significativo en *agricultural-80* y *agricultural-100*. Los tiempos de ejecución aumentan a medida que crece el tamaño del conjunto de datos, lo que sugiere que la complejidad del algoritmo puede estar influenciada por la cantidad de registros procesados. La complejidad seria de **O(n)**, lo que significa que el tiempo de ejecución va aumentando proporcionalmente al tamaño del conjunto de datos.

Requerimiento <<4>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

```
def req_4(catalog, producto, anio_inicio, anio_fin):

    """

    Retorna el resultado del requerimiento 4

    """

    # TODO: Modificar el requerimiento 4

    inicio = get_time()



    registros_filtrados = []

    total_survey = 0

    total_census = 0
```

```
    # Iterate over dictionary values
    for i in range(lt.size(catalog["registros"])):
        registro = lt.get_element(catalog["registros"], i)
        if registro["commodity"] == producto and anio_inicio <=
int(registro["year_collection"]) <= anio_fin:
            registros_filtrados.append({
                "Fuente": registro["source"],
                "Año_Recopilacion": registro["year_collection"],
                "Fecha_Carga": registro["load_time"],
                "Frecuencia": registro["freq_collection"],
                "Departamento": registro["location"],
                "Unidad": registro["unit_measurement"]
            })

            if registro["source"] == "SURVEY":
                total_survey += 1
            elif registro["source"] == "CENSUS":

                total_census += 1

    total_registros = len(registros_filtrados)

    fin = time.time()
    tiempo_ejecucion = (fin - inicio)  # Convertir a milisegundos

    return {
```

```
        "Tiempo de ejecución (ms)": tiempo_ejecucion,
        "Total registros": total_registros,
        "Total SURVEY": total_survey,
        "Total CENSUS": total_census,
        "Registros": registros_filtrados
    }
```

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Tipo de producto a filtrar (ej.: “HOGS”, “SHEEP”, etc.) Año inicial del periodo a consultar (con formato "YYYY ", ej.: “2007”). Año final del periodo a consultar (con formato "YYYY", ej.: “2010”).
Salidas	Tiempo de la ejecución del requerimiento en milisegundos. Número total de registros que cumplieron el filtro. Número total de registros con tipo de fuente/origen “SURVEY” Número total de registros con tipo de fuente/origen “CENSUS” Para el listado de registros resultante, presentar por cada uno de los registros la siguiente información: Tipo de fuente/origendel registro. (ej.: “CENSUS” o “SURVEY”)oAño de recopilación del registro. (ej.: “2007”) Fecha de carga del registro. (con formato "%Y-%m-%d".ej.: “2012-05-15”). Frecuencia de la recopilación del registro. (ej.: “ANNUAL”, “WEEKLY”, etc.) Nombre del departamento del registro. oUnidad de medición del registro. (ej.: “HEAD”, “$”, etc.)
Implementado (Sí/No)	Si. Por: Juan Pablo Garcia

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido de catalog (for)	O(n)
Comparación de commodity (if)	O(1)
Comparación de año (if)	O(1)
Crear lista de registros	O(1)
Adicion a lista de registros(append)	O(1)
Revision de survey(if)	O(1)
Revision de census(if)	O(1)

Conteo de cuantos registros	O(1)
Filtracion de registros si hay mas de 20	O(1)
TOTAL	O(n)

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Intel i7 14700k**
Memoria RAM	32GB
Sistema Operativo	Windows

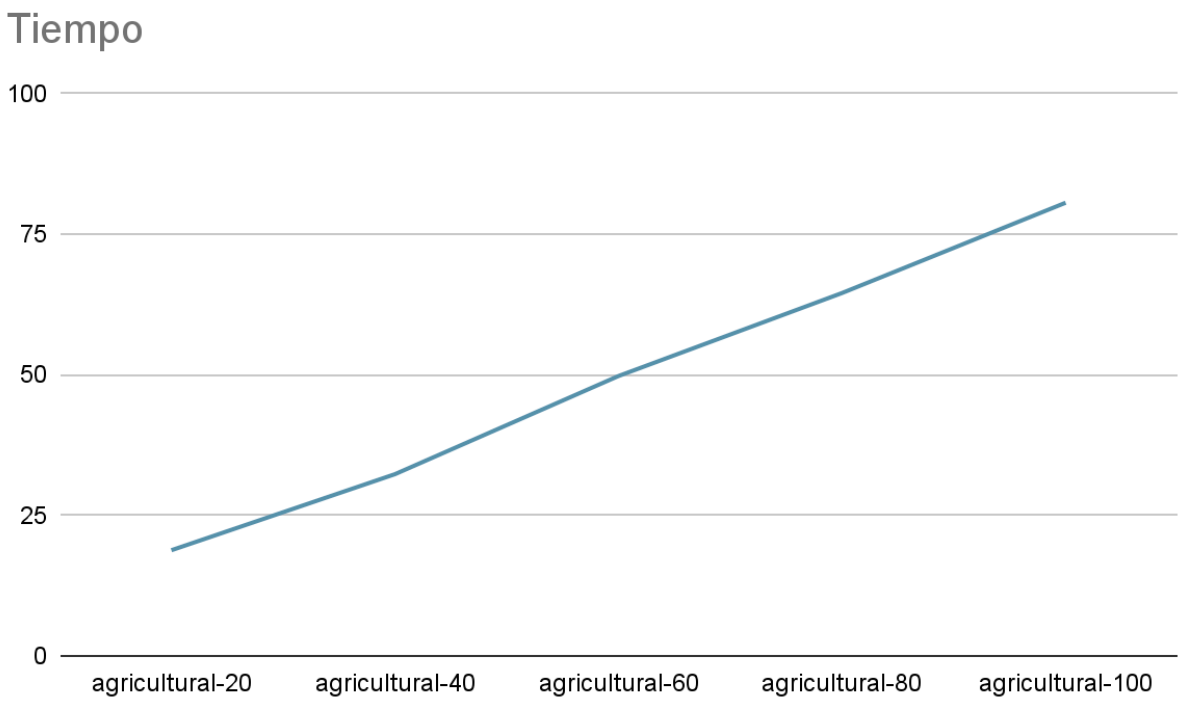
Entrada	**Tiempo (s)**
agricultural-20	18.793 ms
agricultural-40	32.319 ms
agricultural-60	49.779 ms
agricultural-80	64.545 ms
agricultural-100	80.604 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
agricultural-20	Dato1	18.793 ms
agricultural-40	Dato2	32.319 ms
agricultural-60	Dato3	49.779 ms
agricultural-80	Dato4	64.545 ms
agricultural-100	Dato5	80.604 ms

Gráficas



Análisis

El código filtra los registros por tipo de producto y año de recopilación dentro del rango entregado. Luego, revisa si los registros son un census o un survey y extrae la información relevante. Si la lista supera los 20 elementos, se limita la salida a los primeros y últimos 5 registros encontrados. La eficiencia depende de la estructura de datos utilizada y el tamaño del conjunto de datos.

Requerimiento <<5>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

```
def req_5(catalog, stat_category, anioI, anioF):

    """

    Retorna el resultado del requerimiento 5

    """

    # TODO: Modificar el requerimiento 5

    inicio = get_time()
```

```
    resp_index = lt.new_list()

    surveys = 0

    census = 0

    for i in range(0, lt.size(catalog['registros'])):

        statC = lt.get_element(catalog["registros"], i)["statical_category"]

        if stat_category == statC:

            anio = int(lt.get_element(catalog["registros"], i)["year_collection"])

            if (int(anio) <= int(anioF)) and (int(anio) >= int(anioI)):

                lt.add_last(resp_index, i)

                if lt.get_element(catalog["registros"], i)["source"] ==  "CENSUS":

                    census += 1

                else: surveys +=1

    tamanio_resp = lt.size(resp_index)

    fin = time.time()

    tiempo_ejecucion = (fin - inicio)  # Convertir a milisegundos

    return resp_index, tamanio_resp, census, surveys, tiempo_ejecucion
```

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Categoría estadística para filtrar(ej.: “INVENTORY”, “SALES”, etc.) Año inicial del periodo a consultar (con formato "YYYY ", ej.: “2007”). Año final del periodo a consultar (con formato "YYYY", ej.: “2010”).
Salidas	Tiempo de la ejecución del requerimiento en milisegundos. Número total de registros que cumplieron el filtro. Número total de registros con tipo de fuente/origen “SURVEY” Número total de registros con tipo de fuente/origen “CENSUS” Para el listado de registros resultante, presentar por cada uno de los registros la siguiente información: Tipo de fuente/origen del registro. (ej.: “CENSUS” o “SURVEY”) Año de recopilación del registro. (ej.: “2007”) Fecha de carga del registro. (con formato "%Y-%m-%d".ej.: “2012-05-15”). Frecuencia de la recopilación del registro. (ej.: “ANNUAL”, “WEEKLY”, etc.)

	Nombre del departamento del registro. oUnidad de medición del registro. (ej.: “HEAD”, “$”, etc.) Tipo del producto del registro (ej.: “HOGS”, “SHEEP”, etc.)
Implementado (Sí/No)	Si. Por: Sebastian Lara

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido de catalog (for)	O(n)
Comparación de statical categories(if)	O(1)
Comparación de año (if)	O(1)
Revisión de tipo de source(if)	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Intel i7 14700k**
Memoria RAM	32GB
Sistema Operativo	Windows

Entrada	**Tiempo (s)**
agricultural-20	8.767 ms
agricultural-40	19.974 ms
agricultural-60	23.957 ms
agricultural-80	34.54 ms
agricultural-100	51.283 ms

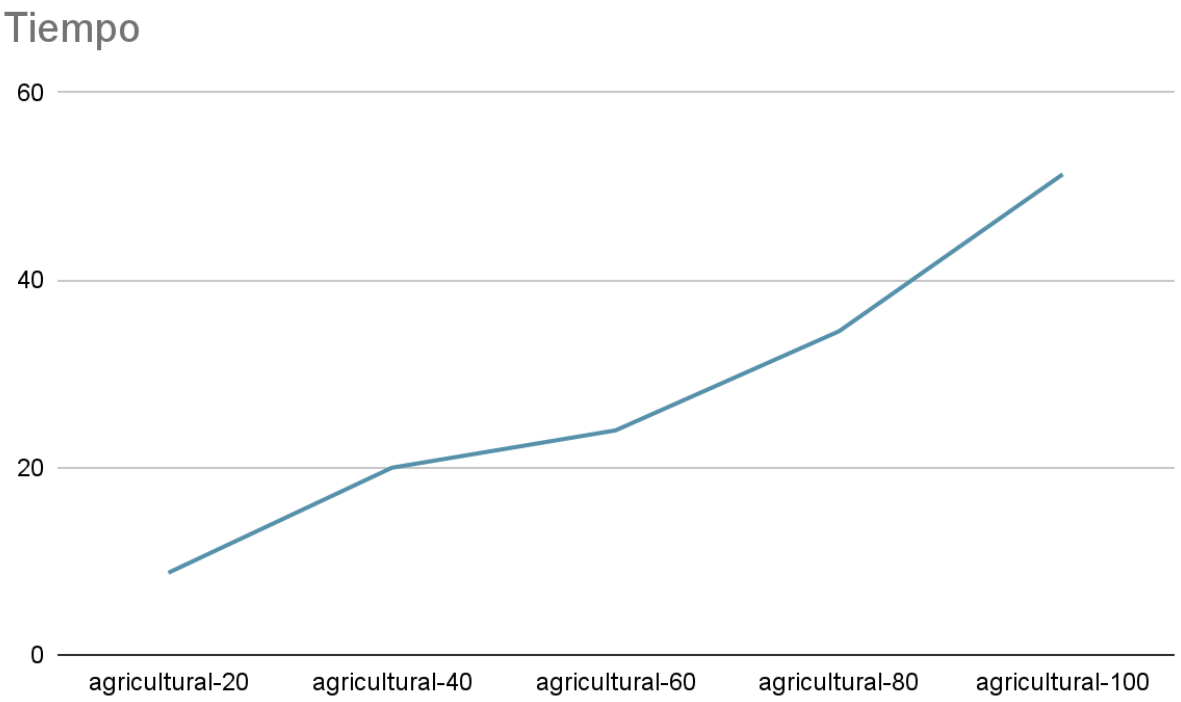
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
agricultural-20	Dato1	8.767 ms

agricultural-40	Dato2	19.974 ms
agricultural-60	Dato3	23.957 ms
agricultural-80	Dato4	34.54 ms
agricultural-100	Dato5	51.283 ms

Gráficas



Análisis

El código filtra los registros por categoría estadística y rango de años de recopilación, después cuenta los registros por tipo de fuente (SURVEY o CENSUS) y finalmente extrae la información que es relevante. Si hay más de 20 registros, se muestran solo los primeros y últimos 5. La eficiencia depende de la estructura de datos utilizada y la cantidad del conjunto de datos.

Requerimiento <<6>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

```
def req_6(catalog, nombre_departamento, initial_date, final_date):

    """
```

```
    Retorna el resultado del requerimiento 6
    """
    start_time = get_time()
    registros_filtrados = []
    total_registros = 0
    total_survey = 0
    total_census = 0



    for i in range(lt.size(catalog["registros"])):
        registro = lt.get_element(catalog["registros"], i)
        departamento = registro["state_name"]


        if departamento == nombre_departamento:
            fecha_carga = registro["load_time"]
            fecha_carga = datetime.strptime(fecha_carga, "%Y-%m-%d
%H:%M:%S").strftime("%Y-%m-%d")


            if initial_date <= fecha_carga <= final_date:
                total_registros +=1


                if registro["source"] == "CENSUS":
                    tipo_fuente = "CENSUS"
                    total_census += 1
                elif registro["source"] == "SURVEY":
                    tipo_fuente = "SURVEY"
                    total_survey += 1
                fecha_carga = registro["load_time"]
```

```
                fecha_carga = datetime.strptime(fecha_carga, "%Y-%m-%d
%H:%M:%S").strftime("%Y-%m-%d")

                year = int(registro["year_collection"])

                registros = {

                    "Fuente": tipo_fuente,

                    "Año_Recopilacion": year,

                    "Fecha_Carga": fecha_carga,

                    "Frecuencia": registro["freq_collection"],

                    "Tipo de Producto": registro["commodity"],

                    "Unidad": registro["unit_measurement"]

                }

                registros_filtrados.append(registros)

    if len(registros_filtrados) > 20:

        registros_filtrados = registros_filtrados[:5] + registros_filtrados[-5:]



    end_time = get_time()

    tiempo_ejecucion = delta_time(start_time, end_time)



    return {

        "Tiempo de ejecución": tiempo_ejecucion,

        "Total registros filtrados": total_registros,

        "Total registros (SURVEY)": total_survey,

        "Total registros (CENSUS)": total_census,

        "Registros": registros_filtrados

    }
```

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Catalog: Diccionario que contiene la lista de datos

	nombre_departamento: Nombre del departamento a buscar en la base de datos initial_date: Fecha inicial del rango de consulta final_date: Fecha final del rango de consulta
Salidas	Tiempo de ejecución del requerimiento Número total de registros encontrados Número total de registros con fuente tipo “SURVEY” Número total de registros con fuente tipo “CENSUS” Lista con los registros encontrados, si hay mas de 20, retornar los primero 5 y los ultimos 5
Implementado (Sí/No)	Si se implementó

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido de catalog (for)	O(n)
Comparación del departamento (if)	O(1)
Comparación del rango de fechas (if)	O(1)
Conteo de registros por fuente (if)	O(1)
Filtracion de registros si hay mas de 20	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Apple M3**
Memoria RAM	8 GB
Sistema Operativo	macOS

CALIFORNIA, 2000-01-01, 2020-01-01

Entrada	**Tiempo (s)**
agricultural-20	56.695
agricultural-40	90.001
agricultural-60	127.069
agricultural-80	209.274
agricultural-100	177.483

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
agricultural-20	Dato1	56.695
agricultural-40	Dato2	90.001
agricultural-60	Dato3	127.069
agricultural-80	Dato4	209.274
agricultural-100	Dato5	177.483

Gráficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Se puede observar en la grafica un crecimiento en los tiempos de ejecución conforme aumenta el tamaño de los datos, aunque con una ligera irregularidad en *agricultural-100*, donde el tiempo disminuye en comparación con *agricultural-80*. La linea de crecimiento no es completamente lineal, lo que indica que pueden existir optimizaciones o problemas en el procesamiento de ciertos datos.

Requerimiento <<7>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

```
def req_7(catalog, departamento, anio_inicio, anio_fin):

    """

    Retorna el resultado del requerimiento 7

    """
```

```
    # TODO: Modificar el requerimiento 7


    start_time = get_time()


    registros = catalog["registros"]


    registros_filtrados = [
        lt.get_element(registros, i) for i in range(lt.size(registros))
        if lt.get_element(registros, i)["state_name"] == departamento and
           anio_inicio <= int(lt.get_element(registros, i)["year_collection"]) <=
anio_fin
    ]


    if not registros_filtrados:
        return "No records found for the given filters."


    ingresos_por_anio = {}
    survey_count = 0
    census_count = 0
    invalid_unit_count = 0
    registros_count = len(registros_filtrados)


    for registro in registros_filtrados:
        anio = int(registro["year_collection"])
        valores = registro["value"].replace(",", "").strip()
        tipo = registro["source"]


        if "$" in valores or not valores.isdigit():
```

```
            invalid_unit_count += 1
            continue

        ingreso = float(valores)

        if tipo == "SURVEY":
            survey_count += 1
        elif tipo == "CENSUS":
            census_count += 1

        if anio not in ingresos_por_anio:
            ingresos_por_anio[anio] = 0

        ingresos_por_anio[anio] += ingreso

    if not ingresos_por_anio:
        return "No valid income data found."

    max_anio = max(ingresos_por_anio, key=ingresos_por_anio.get)
    min_anio = min(ingresos_por_anio, key=ingresos_por_anio.get)

    end_time = get_time()
    execution_time = delta_time(start_time, end_time)

    result = {
        "execution_time_ms": execution_time,
        "total_records": registros_count,
```

```
    "max_income_period": {
        "year": max_anio,
        "status": "MAYOR",
        "income": ingresos_por_anio[max_anio]
    },
    "min_income_period": {
        "year": min_anio,
        "status": "MENOR",
        "income": ingresos_por_anio[min_anio]
    },
    "valid_records_count": registros_count - invalid_unit_count,
    "invalid_unit_count": invalid_unit_count,
    "survey_count": survey_count,
    "census_count": census_count,
}

return result
```

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Nombre del departamento a filtrar (ej.: “NEW MEXICO”, “CALIFORNIA” o “COLORADO”). Año inicial del periodo a consultar (con formato "YYYY ", ej.: “2007”). Año final del periodo a consultar (con formato "YYYY", ej.: “2010”).
Salidas	Tiempo de la ejecución del requerimiento en milisegundos. Número total de registros que cumplieron el filtro. Para los períodos de años con mayor y menor ingresos, presentar para cada uno la siguiente información: Año de recopilación del registro. (ej.: “2007”)oIndicación si es el periodo con mayor y/o menor ingresos. (ej.: “MAYOR”, “MENOR”)

	Valor de ingresos para el periodo.oNúmero de registros que cumplen el periododel filtro. Número de registros con unidad de medida $ que su valor no es válido. Número total de registros con tipo de fuente/origen “SURVEY” Número total de registros con tipo de fuente/origen “CENSUS
Implementado (Sí/No)	Si se implementó.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido de catalog (for)	O(n)
Comparación del departamento (if)	O(1)
Revisión de si existen registros filtrados(if)	O(1)
Recorido de registros filtrados(for)	O(n)
Revision de valores validos en los filtrados(if)	O(1)
Clasificacion de census o survey(if - elif)	O(1)
Comparacion de año con ingresos por año(if)	O(1)
Revision de si hay ingresos por año(if)	O(1)
TOTAL	***O(2n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Intel i7 14700k**
Memoria RAM	32GB
Sistema Operativo	Windows

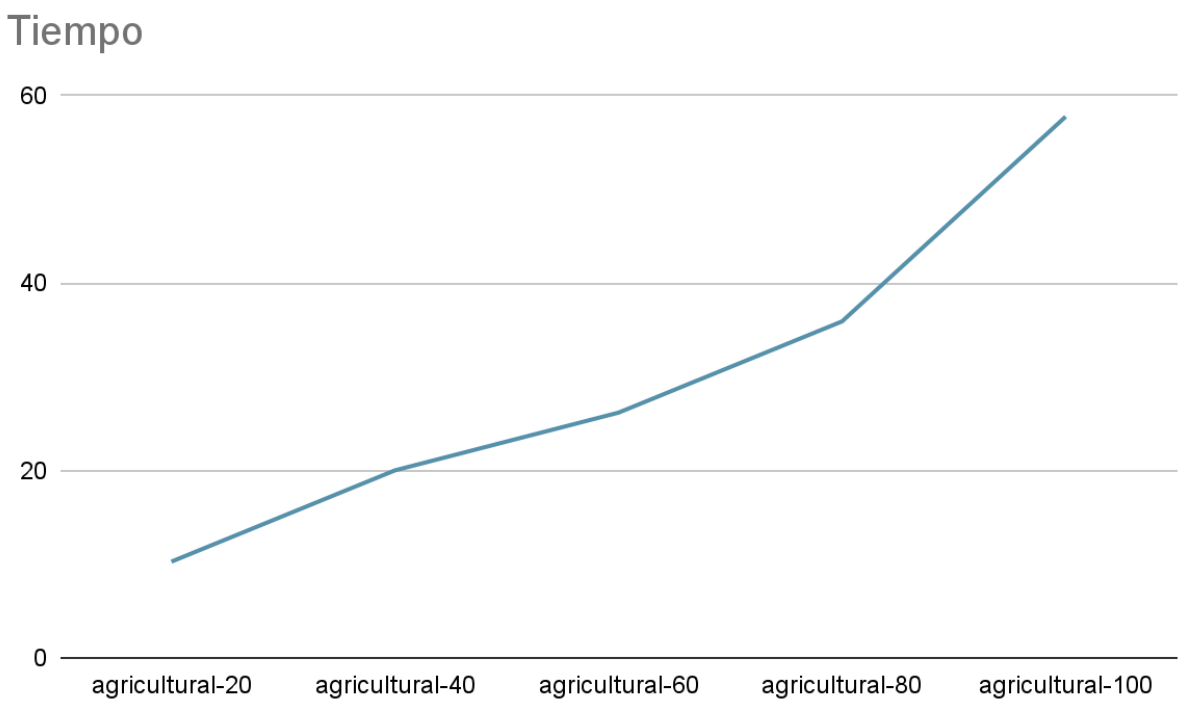
Entrada	**Tiempo (s)**
agricultural-20	10.271 ms
agricultural-40	20.012 ms
agricultural-60	26.173 ms
agricultural-80	35.909 ms
agricultural-100	57.759 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
agricultural-20	Dato1	10.271 ms
agricultural-40	Dato2	20.012 ms
agricultural-60	Dato3	26.173 ms
agricultural-80	Dato4	35.909 ms
agricultural-100	Dato5	57.759 ms

Gráficas



Análisis

El código filtra los registros por departamento y rango de años, después identifica el año con mayor y menor ingreso basado en los registros con unidad de medida $. Se cuentan registros válidos e inválidos,

diferenciando entre fuentes SURVEY y CENSUS. La eficiencia depende del método de búsqueda y el filtrado que se aplique.

Requerimiento <<8>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(...)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(...)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Gráficas

Las gráficas con la representación de las pruebas realizadas.

Análisis
Análisis de resultados de la implementación, tener en cuenta las pruebas realizadas y el análisis de complejidad.

Requerimiento Ejemplo

Descripción

```python
def get_data(data_structs, id):
    """
    Retorna un dato a partir de su ID
    """
    pos_data = lt.isPresent(data_structs["data"], id)
    if pos_data > 0:
        data = lt.getElement(data_structs["data"], pos_data)
        return data
    return None
```

Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

Entrada	Estructuras de datos del modelo, ID.
Salidas	El elemento con el ID dado, si no existe se retorna None
Implementado (Sí/No)	Si. Implementado por Juan Andrés Ariza

Análisis de complejidad
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Obtener el elemento (getElement)	O(1)
TOTAL	***O(n)***

Pruebas Realizadas
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Entrada	Tiempo (ms)
small	0.05
5 pct	0.33
10 pct	1.28
20 pct	2.54
30 pct	4.98
50 pct	7.51
80 pct	13.81
large	25.97

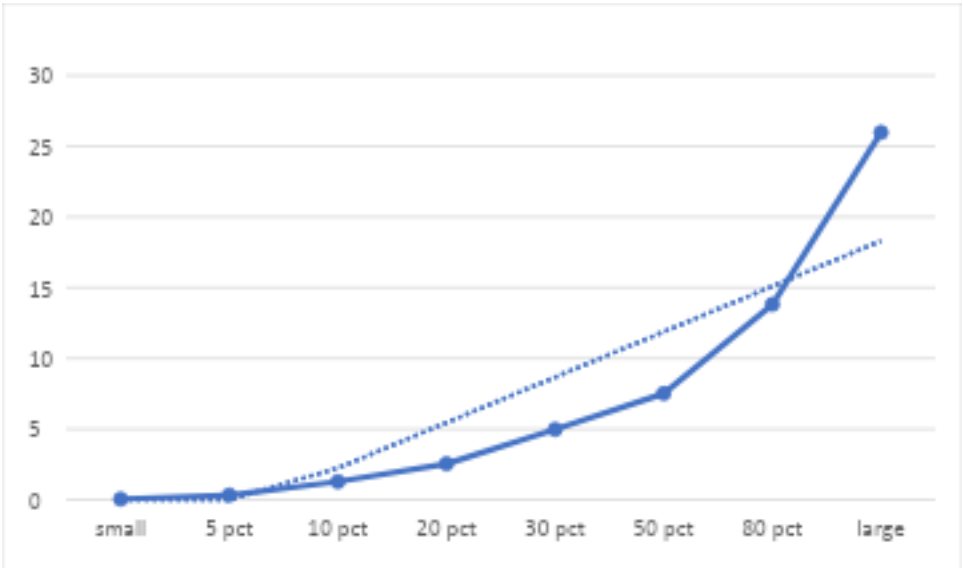
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0.05
5 pct	Dato2	0.33
10 pct	Dato3	1.28
20 pct	Dato4	2.54
30 pct	Dato5	4.98
50 pct	Dato6	7.51
80 pct	Dato7	13.81
large	Dato8	25.97

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.