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

Johan Camilo Suarez Sinisterra, 202214091, jc.suarezs12@uniandes.edu.co

Juan David Camargo Parra, 202220493, jd.camargop1@uniandes.edu.co

Requerimiento <<1>>

Descripción

```
def organizar_actividades_por_saldo_a_pagar(data):

    año, elements = listas_aspecto(data, "Año")
    a = model.data_size(año)
    x = []
    for i in range(1, a+1):
        year = model.get_data_by_pos(elements["data"], i)
        data = model.actividades_por_saldo(year)
        x.append(data)

    return x
```

```python
def actividades_por_saldo(lista):
    """
    Función que soluciona el requerimiento 1
    """

    datos = lista["data"]
    respuesta = quk.sort(datos, cmp_saldo)
    element = get_data_by_pos(respuesta, 1)

    a = {
        "Año": element["Año"],
        "Código actividad económica": element["Código actividad económica"],
        "Nombre actividad económica": element["Nombre actividad económica"],
        "Código sector económico": element["Código sector económico"],
        "Nombre sector económico": element["Nombre sector económico"],
        "Código subsector económico": element["Código subsector económico"],
        "Nombre subsector económico": element["Nombre subsector económico"],
        "Total ingresos netos": element["Total ingresos netos"],
        "Total costos y gastos": element["Total costos y gastos"],
        "Total saldo a pagar": element["Total saldo a pagar"],
        "Total saldo a favor": element["Total saldo a favor"]
    }

    return a
```

```python
def elementos_unicos(data_structs, aspecto):
    size = lt.size(data_structs)
    respuesta = new_data_structs(2)
    for i in range(1, size+1):
        x = get_data_by_pos(data_structs, i)
        present = lt.isPresent(respuesta["data"], x[aspecto])
        if present == 0:
            respuesta["data"] = add_data(respuesta["data"], x[aspecto]

    return respuesta

def elementos_segun_aspecto(data_structs, aspecto, valor):
    size = lt.size(data_structs)
    respuesta = new_data_structs(2)
    for i in range(1, size+1):
        x = get_data_by_pos(data_structs, i)
        if x[aspecto] == valor:
            respuesta["data"] = add_data(respuesta["data"], x)

    return respuesta

def lista_by_aspecto(datastructs, aspecto):
    elementos = elementos_unicos(datastructs, aspecto)
    respuesta = new_data_structs(2)
    size = data_size(elementos)

    for i in range(1, size+1):
        valor = get_data_by_pos(elementos["data"], i)
        x = elementos_segun_aspecto(datastructs, aspecto, valor)
        respuesta["data"] = add_data(respuesta["data"], x)

    return elementos, respuesta
```

Mediante una función llamada lista_by_aspecto separamos la lista en varias sublistas con elementos por año, entonces por cada sublista aplicamos un quick sort y agarramos el primer elemento para finalmente juntar cada uno de estos en una sola lista e imprimir

Entrada	La lista con todos los registros
Salidas	Una lista con la actividad económica de mayor saldo a pagar por año
Implementado (Sí/No)	Si se implementa de manera grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Ejecutar en view	O(1)
Conseguir elementos únicos	O(N)
Formar las sublistas por cada elemento unico	X*O(N) (X representa la cantidad de elementos únicos)
Hacer quicksort en cada sublista	X*O(N/X^3/2) (N/X representa a la cantidad promedio de elementos por lista)
Agarrar el firstElement de cada sublista	X*O(1)
Añadir todos los elementos a una lista a la vez que se cambian los campos necesarios en el diccionario que tiene la informacion de la fila	X*O(1)
Se imprimen los resultados	O(1)
TOTAL	***O(N) + X(O(N)+O((N/X)^3/2)) Aproximadamente***

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 core i5 11400H
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	30.6145999
5 pct	41.15620
10 pct	63.085299
20 pct	110.02439
30 pct	167.92540
50 pct	256.890800
80 pct	418.507400
large	533.707

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, teniendo en cuenta las pruebas realizadas y el análisis de complejidad.

Requerimiento <<3>>

Descripción

```python
def req_3(control):
    """
    Retorna el resultado del requerimiento 3
    """
    # TODO: Modificar el requerimiento 3
    año, elementos = listas_aspecto(control, "Año")
    size = model.data_size(año)
    respuesta1 = new_controller(2)
    respuesta2 = new_controller(2)
    for i in range(1, size+1):
        data = model.get_data_by_pos(elementos["data"], i)
        x, y = model.subsector_menor_retenciones(data)
        respuesta1["model"]["data"] = model.add_data(respuesta1["model"]["data"], x)
        respuesta2["model"]["data"] = model.add_data(respuesta2["model"]["data"], y)

    return respuesta1["model"]["data"]["elements"], respuesta2["model"]["data"]["elements"]
```

```
def subsector_menor_retenciones(lista):
    comparar = new_data_structs(2)
    subsector, data = lista_by_aspecto(lista["data"], "Código subsector económico")
    size = data_size(subsector)
    for k in range(1, size+1):
        subsector_elements = lt.getElement(data["data"], k)
        names = lt.firstElement(subsector_elements["data"])
        size2 = data_size(subsector_elements)
        a = {
            "Año":names["Año"],
            "Código sector económico": names["Código sector económico"],
            "Nombre sector económico": names["Nombre sector económico"],
            "Código subsector económico": names["Código subsector económico"],
            "Nombre subsector económico": names["Nombre subsector económico"],
            "Total retenciones": int(names["Total retenciones"]),
            "Total ingresos netos": int(names["Total ingresos netos"]),
            "Total costos y gastos": int(names["Total costos y gastos"]),
            "Total saldo a pagar": int(names["Total saldo a pagar"]),
            "Total saldo a favor": int(names["Total saldo a favor"])
        }
```

```
        for j in range(2, size2+1):
            adition = lt.getElement(subsector_elements["data"], j)
            a["Total retenciones"] += int(adition["Total retenciones"])
            a["Total ingresos netos"] += int(adition["Total ingresos netos"])
            a["Total costos y gastos"] += int(adition["Total costos y gastos"])
            a["Total saldo a pagar"] += int(adition["Total saldo a pagar"])
            a["Total saldo a favor"] += int(adition["Total saldo a favor"])

        comparar["data"] = add_data(comparar["data"], a)

    comparar["data"] = se.sort(comparar["data"], cmp_sub_menor_retencion)
    subsector_data = lt.firstElement(comparar["data"])
    subsector_index = lt.isPresent(subsector["data"], subsector_data["Código sector económico"])
    activities = lt.getElement(data["data"], subsector_index)
    activities_size = data_size(activities)
    activities = quk.sort(activities["data"], cmp_sub_menor_retencion)


    if activities_size < 6:
        return subsector_data, activities
    else:
        a = primeros_ultimos(activities, 3, "adelante")
        b = primeros_ultimos(activities, 3, "atras")
        bsize = data_size(b)
        for i in range(1, bsize):
            x = get_data_by_pos(activities, i)
            a = add_data(a, x)

        return subsector_data, a
```

Entrada	La lista con todos los registros
Salidas	Una tupla con [0] = una lista con la información de cada subsector, y [1] = Las 3 actividades económicas que más aportaron, y las 3 que menos aportaron
Implementado (Sí/No)	Se implementa de manera individual: Johan Camilo Suarez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ejecutar en view	O(1)
Conseguir elementos únicos	O(N)
Formar las sublistas por cada elemento unico	X*O(N) (X representa la cantidad de elementos únicos)
últimos dos pasos pero por cada sublista	Xb*(X*O(N/X))(Xb representa a la cantidad de subsectores)
Recorrer cada sublista para sumar las contribuciones	Xb*(O(N/X)
Shellsort de la lista de cada subsector	O(N/Xb^2)*Xb
Se imprimen los resultados	
TOTAL	

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 core i5 11400H**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	30.6145999
5 pct	41.15620
10 pct	63.085299
20 pct	110.02439
30 pct	167.92540
50 pct	256.890800
80 pct	418.507400
large	533.707

Requerimiento <<5>>

Descripción

```
def req_5(control):
    """
    Retorna el resultado del requerimiento 5
    """
    # TODO: Modificar el requerimiento 5
    año, elementos = listas_aspecto(control, "Año")
    size = model.data_size(año)
    respuesta1 = new_controller(2)
    respuesta2 = new_controller(2)
    for i in range(1, size+1):
        data = model.get_data_by_pos(elementos["data"], i)
        x, y = model.calculos_subsector_req_5(data)
        respuesta1["model"]["data"] = model.add_data(respuesta1["model"]["data"], x)
        respuesta2["model"]["data"] = model.add_data(respuesta2["model"]["data"], y)

    return respuesta1["model"]["data"]["elements"], respuesta2["model"]["data"]["elements"]
```

```
def calculos_subsector_req_5(datastructs):
    """
    Función que soluciona el requerimiento 5
    """

    comparar = new_data_structs(2)
    subsector, data = lista_by_aspecto(datastructs["data"], "Código subsector económico")
    size = data_size(subsector)
    for k in range(1, size+1):
        subsector_elements = lt.getElement(data["data"], k)
        names = lt.firstElement(subsector_elements["data"])
        size2 = data_size(subsector_elements)
        a = {
            "Año":names["Año"],
            "Código sector económico": names["Código sector económico"],
            "Nombre sector económico": names["Nombre sector económico"],
            "Código subsector económico": names["Código subsector económico"],
            "Nombre subsector económico": names["Nombre subsector económico"],
            "Descuentos tributarios": int(names["Descuentos tributarios"]),
            "Total ingresos netos": int(names["Total ingresos netos"]),
            "Total costos y gastos": int(names["Total costos y gastos"]),
            "Total saldo a pagar": int(names["Total saldo a pagar"]),
            "Total saldo a favor": int(names["Total saldo a favor"])
        }

        for j in range(2, size2+1):
            adition = lt.getElement(subsector_elements["data"], j)
            a["Descuentos tributarios"] += int(adition["Descuentos tributarios"])
            a["Total ingresos netos"] += int(adition["Total ingresos netos"])
            a["Total costos y gastos"] += int(adition["Total costos y gastos"])
            a["Total saldo a pagar"] += int(adition["Total saldo a pagar"])
            a["Total saldo a favor"] += int(adition["Total saldo a favor"])
```

```
    for j in range(2, size2+1):
        adition = lt.getElement(subsector_elements["data"], j)
        a["Descuentos tributarios"] += int(adition["Descuentos tributarios"])
        a["Total ingresos netos"] += int(adition["Total ingresos netos"])
        a["Total costos y gastos"] += int(adition["Total costos y gastos"])
        a["Total saldo a pagar"] += int(adition["Total saldo a pagar"])
        a["Total saldo a favor"] += int(adition["Total saldo a favor"])

    comparar["data"] = add_data(comparar["data"], a)

comparar["data"] = se.sort(comparar["data"], cmp_descuentos_tributarios)
subsector_data = lt.firstElement(comparar["data"])
subsector_index = lt.isPresent(subsector["data"], subsector_data["Código sector económico"])
activities = lt.getElement(data["data"], subsector_index)
activities_size = data_size(activities)
activities["data"] = quk.sort(activities["data"], cmp_descuentos_tributarios)

if activities_size < 6:
    return subsector_data, activities
else:
    a = primeros_ultimos(activities, 3, "adelante")
    b = primeros_ultimos(activities, 3, "atras")
    bsize = data_size(b)
    for i in range(1, bsize):
        x = get_data_by_pos(activities["data"], i)
        a["data"] = add_data(a["data"], x)

    return subsector_data, a
```

Llama dos veces a listas by aspecto para separar los datos por años, luego para separar cada una de estas mini listas por subsectores, entonces recorre las listas de sectores para sumar los datos financieros, luego hacer un shellsort y toma el elemento mayor, que es el subsector con mayor descuento tributario. Con este dato agarra la lista del subsector correspondiente y le aplica un quick sort para encontrar los primeros 3 y ultimos 3 que aporten para esta cifra

Entrada	La lista con todos los registros
Salidas	Dos listas, una con los datos de los subsectores y otra con las 3 mejores y 3 menores actividades economicas segun su aporte a esta cifra
Implementado (Sí/No)	Si se implementa, por Juan David Camargo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ejecutar en view	O(1)
Conseguir elementos únicos	O(N)
Formar las sublistas por cada elemento unico	X*O(N) (X representa la cantidad de elementos únicos)

últimos dos pasos pero por cada sublista	Xb*(X*O(N/X))(Xb representa a la cantidad de subsectores)
Recorrer cada sublista para sumar las contribuciones	Xb*(O(N/X)
Shellsort de la lista de cada subsector	O(N/Xb^2)*Xb
Se imprimen los resultados	
TOTAL	

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 core i5 11400H**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	
5 pct	
10 pct	
20 pct	
30 pct	
50 pct	
80 pct	
large	

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, teniendo en cuenta las pruebas realizadas y el análisis de complejidad.