ANÁLISIS DEL RETO

Isabela Mantilla Mora, <i.mantilla@uniandes.edu.co>, 202215383

Nicolás Sotelo López, <n.sotelol@uniandes.edu.co>, 202113111

Nicolás Sotelo Morales, <n.sotelom@uniandes.edu.co>, 202123752

Requerimiento 1

Descripción

```
def req_1(control,num_ofertas,codigo_pais,nivel_experticia):
    """
    Función que soluciona el requerimiento 1
    """
    cumplen=s.new_list()
    ofertas_pais=0
    for i in range(len(mp.keys(control))):
        key_l=mp.keys(control)[i]
        for j in range(len(key_l)):
            key=key_l[j]
            elemento=mp.get(control,key)
            info=elemento['value']
            pais=info['country_code']
            n_exp=info['experience_level']
            if pais == codigo_pais:
                ofertas_pais +=1
                if n_exp == nivel_experticia:
                    s.add_last(cumplen,info)
    ord_cumplen=merge.merge_sort(cumplen,'published_at', 'title')
    total_ofertas=s.size(ord_cumplen)
    ofertas_r1=[['Fecha publicación','Título oferta',
                 'Nombre empresa',
                 'Nivel experticia',
                 'País','Ciudad',
                 'Tamaño empresa',
                 'Tipo de ubicación',
                 'Contratar ucranianos']]
    for i in range(num_ofertas):
        dato=s.get_element(ord_cumplen,i)
        datos_lista=[dato['published_at'], dato['title'], dato['company_name'],
                     dato['experience_level'], dato['country_code'], dato['city'],
                     dato['company_size'], dato['workplace_type'],
                     dato['open_to_hire_ukrainians']]
        ofertas_r1.append(datos_lista)

    if len(ofertas_r1) > 11:
        primeros=ofertas_r1[0:6]
        ultimos=ofertas_r1[-5:]
        ofertas_r1=primeros+ultimos

    return [total_ofertas,ofertas_r1,ofertas_pais]
```

Este requerimiento se encarga de retornar unos datos y una lista respecto a unas ofertas que cumplen ciertos requisitos. Donde a partir de un número de ofertas, un código de un país y un nivel de experticia ingresado por el usuario, se buscan los datos relacionados en una tabla de hash. Si, no se encuentran el valor de una oferta o varias con estos requerimientos retorna None.

Entrada	Estructuras de datos del modelo, num_ofertas(int), código_pais(str) y nivel_experticia
Salidas	Total ofertas (int) y ofertas_r1(list)
Implementado (Sí/No)	Si. Implementado grupalmente

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
s.newlist()	O(1)
For i in range (len(element)): for i in range(len(elemento)):	O(n)
elemento = mp.get_value(control, i)	O(1)
if pais == codigo_pais and n_exp == nivel_experticia: s.add_last(cumplen,info)	O(1)
ord_cumplen=merge.merge_sort(cumplen,'published_at')	O(n log n)
total_ofertas=s.size(ord_cumplen)	O(1)
for i in range(num_ofertas): dato=s.get_element(ord_cumplen,i)	O(m)
if len(ofertas_r1) > 11:	O(1)
TOTAL	***O(n log n)***

Pruebas Realizadas

Las pruebas realizadas se realizaron en una maquina con estas especificaciones. Los datos ingresados son el número de ofertas, el código del pais y el nivel de experticia. Los datos que se retornan fueron la cantidad de ofertas y el dato de ejemplo que ponemos con cada archivo es la primera oferta.

Procesadores	**AMD Ryzen 5 3450U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	12 GB
Sistema Operativo	Windows 11 Home Single Language

Entrada	**Tiempo (ms)**
small	1362.837
10 pct	1400.766
20 pct	3474.710
30 pct	5214.509
50 pct	8687.894
80 pct	12233.140
large	12093.877

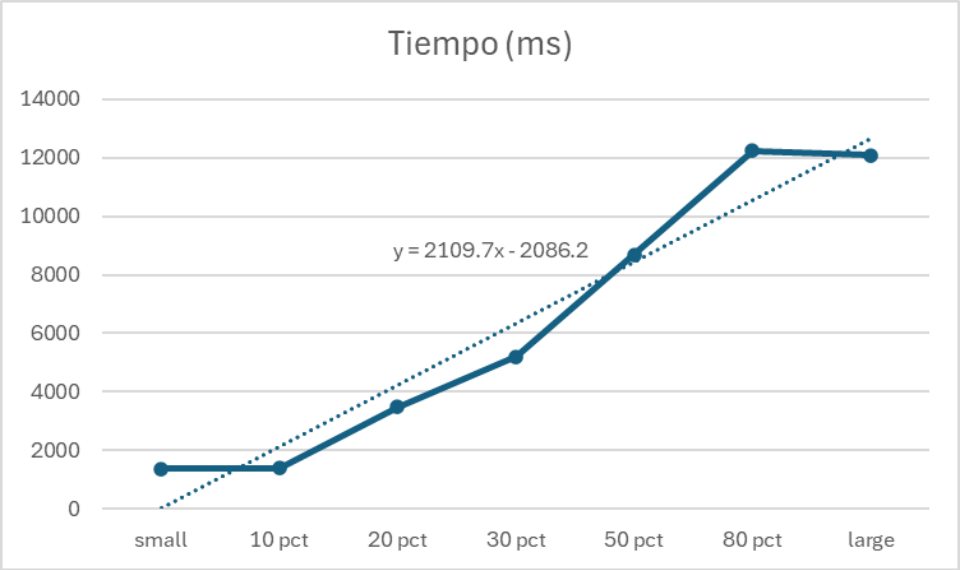
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	IT Business Intelligence Manager	1362.837
10 pct	IT Business Intelligence Manager	1400.766
20 pct	Scrum Master	3474.710
30 pct	Scrum Master	5214.509
50 pct	Scrum Master	8687.894
80 pct	SAP MM S/4HANA Consultant	12233.140
large	QA Engineer	12093.877

Graficas

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 log n). Esto debido al ordenamiento de datos que se utiliza, el merge sort.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos muestran una discrepancia con la línea de tendencia.

Descripción

```
def req_2(control,num_ofertas,nom_empresa,ciudad_oferta):
    """
    Función que soluciona el requerimiento 2
    """
    cumplen=s.new_list()
    for i in range(data_size(control)):
        elemento=mp.get_value(control,i)
        for j in range(len(elemento)):
            info=elemento[j]
            company_name=info['company_name']
            city=info['city']
            if company_name == nom_empresa and city == ciudad_oferta:
                s.add_last(cumplen,info)
    ord_cumplen=merge.merge_sort(cumplen,'published_at')
    total_ofertas=s.size(ord_cumplen)
    ofertas_r2=[['Fecha publicación','País',
                 'Ciudad',
                 'Nombre de la empresa',
                 'Título oferta','Nivel experticia',
                 'Tipo de trabajo']]

    for i in range(num_ofertas):
        dato=s.get_element(ord_cumplen,i)
        datos_lista=[dato['published_at'], dato['country_code'], dato['city'],
                     dato['company_name'], dato['title'], dato['experience_level'],
                     dato['workplace_type']]
        ofertas_r2.append(datos_lista)
    lista_resultado=[]
    if len(ofertas_r2)>11:
        primeros=ofertas_r2[0:6]
        ultimos=ofertas_r2[-5:]
        lista_resultado=primeros+ultimos

    return [total_ofertas, lista_resultado]
```

Este requerimiento se encarga de retornar un entero y una lista relacionada a ofertas con ciertos criterios. Estos criterios son un número n de ofertas, el nombre y la ciudad de una empresa. A partir de estos criterios se busca extraer la información de una tabla de hash. Si al intentar extraer los valores no se encuentran con estos criterios retorna None.

Entrada	Estructuras de datos del modelo, num_ofertas(int),nom_empresa(str) y ciudad_oferta(str).
Salidas	El total de ofertas=”total_ofertas” (int) y lista de ofertas con unas especificaciones=”lista_resultado”.
Implementado (Sí/No)	Si. Implementado grupalmente

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
cumplen=s.new_list()	O(1)
For i in range(data_size(control)): for j in range(len(elemento)): info = elemento[j]	O(n*m)
elemento = mp.get_value(control, i	O(1)
if company_name == nom_empresa and city == ciudad_oferta: s.add_last(cumplen, info)	O(1)
ord_cumplen = merge.merge_sort(cumplen, 'published_at'	O(n log n)
for i in range(num_ofertas): dato = s.get_element(ord_cumplen, i) datos_lista = [...] ofertas_r2.append(datos_lista)	O(1)
if len(ofertas_r2) > 11:	O(1)
TOTAL	***O(n log n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el número de ofertas, el nombre y la ciudad de la empresa.

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

Entrada	Tiempo (ms)
small	80.241
10 pct	83.699
20 pct	241.787
30 pct	199.311
50 pct	323.746
80 pct	490.513
large	587.573

Tablas de datos

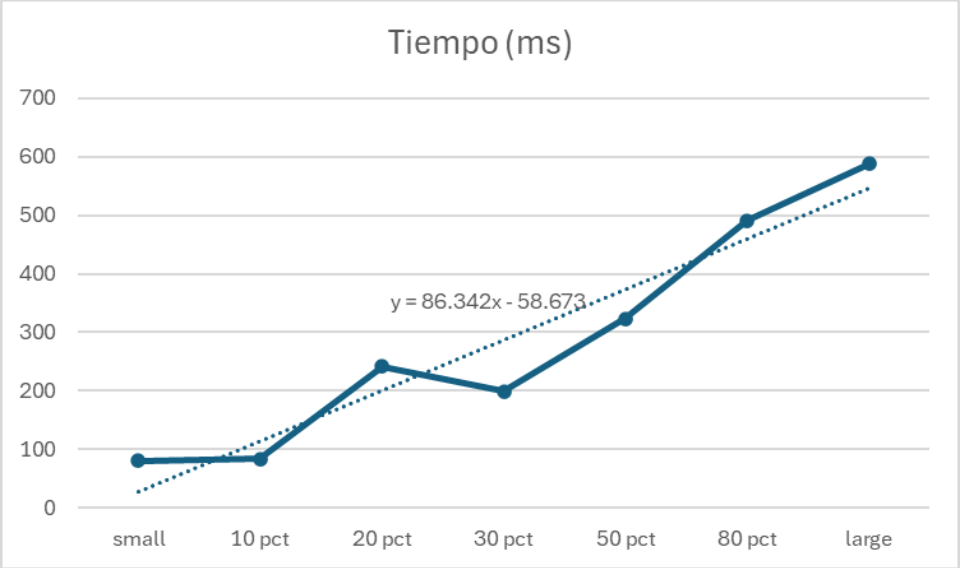
Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	PHP Developer	80.241
10 pct	PHP Developer	83.699
20 pct	PHP Developer	241.787

30 pct	PHP Developer	199.311
50 pct	PHP Developer	323.746
80 pct	PHP Developer	490.513
large	Senior Security Engineer	587.573

Graficas

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 log n). Esto debido al ordenamiento de datos que se utiliza, el merge sort.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos muestran una discrepancia con la línea de tendencia.

Requerimiento 3

Descripción

```python
def req_3(control,nom_empresa,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 3
    """
    fecha_inicial=s.fecha_analisis(fecha_inicial)
    fecha_final=s.fecha_analisis(fecha_final)
    control=control['jobs']
    cumplen=s.new_list()
    total_junior=0
    total_mid=0
    total_senior=0
    for i in range(data_size(control)):
        elemento=mp.get_value(control,i)
        for j in range(len(elemento)):
            info=elemento[j]
            company_name=info['company_name']
            fecha=info['published_at']
            experticia=info['experience_level']
            if company_name == nom_empresa and fecha >= fecha_inicial and fecha <= fecha_final:
                s.add_last(cumplen,info)
                if experticia=='junior':
                    total_junior+=1
                elif experticia=='mid':
                    total_mid+=1
                elif experticia=='senior':
                    total_senior+=1
    cumplen_ord=merge.merge_sort(cumplen, 'published_at')
    total_ofertas=s.size(cumplen_ord)
    ofertas_r2=[['Fecha publicación','Titulo oferta',
                 'Nivel experticia',
                 'Nombre empresa',
                 'Ciudad',
                 'País',
                 'Tamaño de empresa',
                 'Tipo de ubicación',
                 'Contratar ucranianos']]
    for i in range(total_ofertas):
        dato=s.get_element(cumplen_ord,i)
        datos_lista=[dato['published_at'], dato['title'],
                     dato['experience_level'], dato['company_name'], dato['city'],
                     dato['country_code'],dato['company_size'], dato['workplace_type'],
                     dato['open_to_hire_ukrainians']]
        ofertas_r2.append(datos_lista)
    lista_resultado=[]
    if len(ofertas_r2) > 11:
        primeros=ofertas_r2[0:6]
        ultimos=ofertas_r2[-5:]
        lista_resultado=primeros+ultimos

    return [total_ofertas, total_junior,total_mid, total_senior, lista_resultado]
```

Este requerimiento se encarga de retornar resultados y una lista de datos de ofertas publicadas por una empresa en un periodo de tiempo. A partir de los requerimientos se evalúan datos extraídos de una tabla de hash. Si cumplen la comparación son organizadas y contados o puestas en la lista. Si al extraer los datos ninguno tiene los parámetros dados por el usuario se retorna None.

Entrada	Estructuras de datos del modelo, nom_empresa(str), fecha_inicial y fecha_final((con formato "%Y-%m-%d").
Salidas	Varios conteos (int) según oferta y experticia=” total_ofertas que cumplen con este requisito, total de ofertas con nivel de experticia junior que cumplen con los requisitos, total de ofertas con nivel de experticia mid que cumplen con los requisitos y

	total de ofertas con nivel de experticia senior que cumplen con los requisitos". Una lista con las ofertas ordenadas cronológicamente="listado_resultado"
Implementado (Sí/No)	Si. Implementado por Nicolás Sotelo López

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
fecha_inicial = s.fecha_analisis(fecha_inicial) fecha_final = s.fecha_analisis(fecha_final)	O(1). Este paso implica llamar a una función para analizar las fechas, lo cual tiene una complejidad constante. Fecha análisis es O(1)
cumplen=s.new_list()	O(1). Esta función simplemente crea una lista con valores iniciales predeterminados, operación constante
For i in range(len(mp.keys(control)))	O(N). Se itera sobre todos los elementos de la estructura. Además, mp.keys es constante
Key_l=mp.keys(control)[i]	O(1). Es constante ya que devuelve una referencia a la lista interna de llaves la cual ya está en el mapa
For j in range (len(key_l))	O(P). Se itera sobre cada elemento
Elemento1=mp.get(jobs,key1)	O(P). En el peor de los casos, donde P es la cantidad de elementos en la lista de llaves en la posición calculada por el hash
s.add_last (cumplen,info)	O(1). Esta función no necesita recorrer toda la lista ya que tiene una referencia directa al ultimo elemento
if company_name == nom_empresa and fecha >= fecha_inicial and fecha <= fecha_final:	O(1). Python utiliza la técnica de hashing para comparar cadenas y así hacerlo más eficiente
cumplen_ord = merge.merge_sort(cumplen, 'published_at')	O (M log M).(M porque es sobre la lista cumplen) Esto se debe a que merge utiliza dividir y conquistar. Dividir la lista en dos recursivamente (O(log M)) y Conquistar en el caso que fusiona dos sublistas ordenadas de tamaño M/2 lo que tiene una complejidad de O(M)
total_ofertas=s.size(cumplen_ord)	O(1). El tamaño de la lista está en un atributo, por lo que es una operación constante
for i in range(total_ofertas):	O(K). Este paso itera sobre una nueva lista, por lo que debe pasar por cada elemento
dato=s.get_element(cumplen,i)	O(1). La lista almacena sus elementos en una lista adicional ('content') por lo que acceder a un elemento por su índice es constante

If len(ofertas_r3)>11	O(1). Python utiliza la técnica de hashing para comparar integers y así hacerlo más eficiente
TOTAL	***O(M log M) o O(NxP) Depende de los valores especificos de M,N y P***

Pruebas Realizadas

Las pruebas realizadas se realizaron en una maquina con estas 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	389.575
10 pct	140.709
20 pct	267.739
30 pct	451.488
50 pct	437.236
80 pct	1231.331
large	754.430

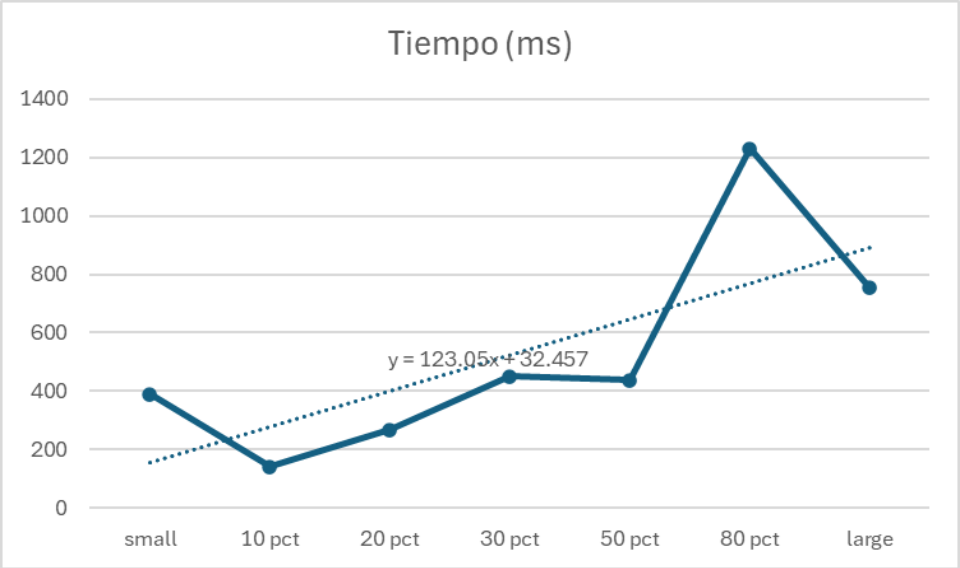
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	PHP Developer	389.575
10 pct	Mid Frontend Developer	140.709
20 pct	Senior Technical Support Engineer	267.739
30 pct	Senior Frontend Developer	451.488
50 pct	Senior Frontend Developer	437.236
80 pct	Senior DevOps Engineer	1231.331
large	Senior DevOps Engineer	754.430

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Aunque la complejidad del algoritmo es relativamente alta por O(MlogM) o (NxP), el tiempo de ejecución en milisegundos parece razonable para el conjunto de datos proporcionados. Las pruebas realizadas muestran que el algoritmo es escalable, ya que el tiempo de ejecución aumenta a medida que aumenta el tamaño de datos. Sin embargo, siempre hay un margen para mejoras y optimizaciones adicionales en las técnicas de conteo datos y acceder a la información

Requerimiento 4

Descripción

```python
def req_4(control,codigo_pais,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    fecha_inicial=s.fecha_analisis(fecha_inicial)
    fecha_final=s.fecha_analisis(fecha_final)
    jobs=control['jobs']
    cumplen=s.new_list()
    pais=s.new_list()
    for i in range(data_size(jobs)):
        elemento=mp.get_value(jobs,i)
        for i in range(len(elemento)):
            info=elemento[i]
            paiss=info['country_code']
            fecha=info['published_at']
            if paiss == codigo_pais:
                s.add_last(pais,info)
                if fecha >= fecha_inicial and fecha <= fecha_final:
                    s.add_last(cumplen,info)

    cumplen_ord=merge.merge_sort(cumplen,'published_at')
    total_ofertas=s.size(cumplen_ord)
    empresas=s.conteo(pais,'company_name')
    total_empresas=s.size(empresas)
    ciudades=s.conteo(pais,'city')
    total_ciudades=s.size(ciudades)
    ciudad_mas=s.first_element(ciudades)
    ciudad_menos=s.last_element(ciudades)

    ofertas_r1=[['Fecha publicación','Título oferta',
                 'Nivel experticia',
                 'Nombre empresa',
                 'Ciudad',
                 'Tipo de ubicación',
                 'Tipo Trabajo'
                 'Contratar ucranianos']]
    for i in range(total_ofertas):
        dato=s.get_element(cumplen_ord,i)
        datos_lista=[dato['published_at'], dato['title'],
                     dato['experience_level'], dato['company_name'], dato['city'],
                     dato['workplace_type'], dato['remote_interview'],
                     dato['open_to_hire_ukrainians']]
        ofertas_r1.append(datos_lista)

    if len(ofertas_r1) > 11:
        primeros=ofertas_r1[0:6]
        ultimos=ofertas_r1[-5:]
        ofertas_r1=primeros+ultimos

    return [total_ofertas,total_empresas,total_ciudades,ciudad_mas,ciudad_menos,ofertas_r1]
```

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, codigo_pais(str), fecha_inicial y fecha_final((con formato "%Y-%m-%d").
Salidas	Conteos respecto a las ofertas, empresas y ciudades=”total_ofertas,total_empresas,total_ciudades,ciudad_mas, ciudad_menos”. Lista de ofertas=”ofertas_r1”
Implementado (Sí/No)	Si. Implementado por Nicolás Sotelo Morales

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
fecha_inicial = s.fecha_analisis(fecha_inicial) fecha_final = s.fecha_analisis(fecha_final)	O(1)
jobs = control['jobs'] cumplen = s.new_list() pais = s.new_list()	O(1)

for i in range(len(mp.keys(jobs))):
key_l=mp.keys(jobs)[i]
for j in range(len(key_l)):
	O(n^2)

elemento = mp.get_value(jobs, i)	O(1)

if paiss == codigo_pais:
s.add_last(pais, info)
if fecha >= fecha_inicial and fecha <= fecha_final:
s.add_last(cumplen, info)
	O(1)

cumplen_ord = merge.merge_sort(cumplen, 'published_at')	O(n log n)
empresas = s.conteo(pais, 'company_name') ciudades = s.conteo(pais, 'city') ciudad_menos = s.last_element(ciudades)	O(m^2)
total_empresas = s.size(empresas) total_ciudades = s.size(ciudades) ciudad_mas = s.first_element(ciudades)	O(1)
for i in range(total_ofertas):	O(k)
if len(ofertas_r1) > 11	O(1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el código del pais y un periodo con fecha inicial/ fecha final.

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

Entrada	Tiempo (ms)
small	72951.981
10 pct	27298.332
20 pct	55819.355
30 pct	94643.080
50 pct	202443.442
80 pct	228172.991
large	374055.964

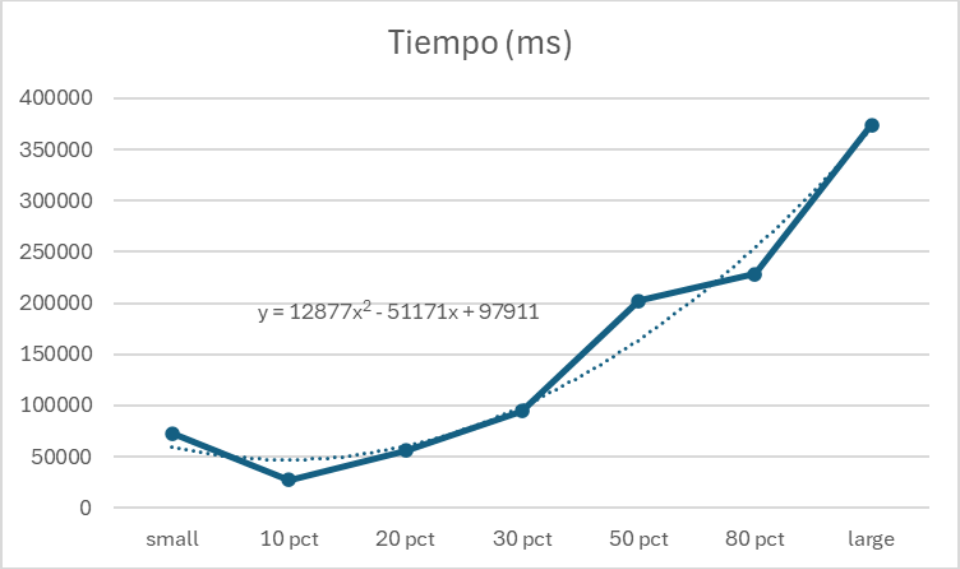
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Programista Laravel lub Symfony	72951.981
10 pct	Mentor / Trener kursu Python	27298.332
20 pct	Mentor / Trener kursu Python	55819.355
30 pct	Mentor / Trener kursu Python	94643.080
50 pct	Mentor / Trener kursu Python	202443.442
80 pct	Mentor / Trener kursu Python	228172.991
large	Salesforce Developer	374055.964

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A Como se observa en el gráfico, esta función tiene complejidad cuadrática, esto, ya que tiene que recorrer todos los datos contenidos, primero para extraer las llaves de los mapas, y luego para con estas, extraer los valores de dichos mapar. A esta gráfica se le realizó una regresión cuadrática donde la línea de tendencia cuadra de forma muy acertadamente con los datos obtenidos en las pruebas.

Requerimiento 5

```
def req_5(control,ciudad_oferta,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 5
    """
    fecha_inicial=s.fecha_analisis(fecha_inicial)
    fecha_final=s.fecha_analisis(fecha_final)
    jobs=control['jobs']
    cumplen=s.new_list()
    ciudad=s.new_list()
    for i in range(len(mp.keys(jobs))):
        key_l=mp.keys(jobs)[i]
        for j in range(len(key_l)):
            key=key_l[j]
            elemento=mp.get(jobs,key)
            info=elemento['value']
            ciudad_o=info['city']
            fecha=info['published_at']
            if ciudad_o == ciudad_oferta:
                s.add_last(ciudad,info)
                if fecha >= fecha_inicial and fecha <= fecha_final:
                    s.add_last(cumplen,info)

    cumplen_ord=merge.merge_sort(cumplen,'published_at', 'company_name')
    total_ofertas=s.size(cumplen_ord)
    empresas=s.conteo(ciudad,'company_name')
    total_empresas=s.size(empresas)
    empresa_mas=s.first_element(empresas)
    empresa_menos=s.last_element(empresas)

    ofertas_ciudad=[['Fecha publicación','Título oferta',
                     'Nombre empresa',
                     'Tipo de ubicación',
                     'Tamaño de la empresa',
                     'Tipo de trabajo']]

    for i in range(total_ofertas):
        dato=s.get_element(cumplen_ord,i)
        datos_lista=[dato['published_at'], dato['title'],
                     dato['company_name'],dato['workplace_type'],dato['company_size'],
                     dato['remote_interview']]
        ofertas_ciudad.append(datos_lista)

    if len(ofertas_ciudad) > 11:
        primeros=ofertas_ciudad[0:6]
        ultimos=ofertas_ciudad[-5:]
        ofertas_ciudad=primeros+ultimos
    return [total_ofertas,total_empresas,empresa_mas,empresa_menos,ofertas_ciudad]
```

Descripción

Este requerimiento se encarga de retornar datos y una lista relacionada a ofertas publicadas en una ciudad durante un periodo de tiempo. Por lo que por medio de una ciudad y un periodo de tiempo se comparan los datos de una tabla de hash, si cumplen los requisitos forman parte de la respuesta. En el caso que no se encuentren datos con los requerimientos retorna None.

Entrada	Estructuras de datos del modelo, ciudad_oferta(str), fecha_inicial y fecha_final((con formato "%Y-%m-%d").
Salidas	Conteos y totales relacionados a las ofertas y a las empresas (“total_ofertas, total_empresas, empresa_mas, empresa_menos”) junto a una lista de las ofertas con ciertas especificaciones=” ofertas_ciudad”.
Implementado (Sí/No)	Si. Implementado por Isabela Mantilla Mora

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
fecha_inicial = s.fecha_analisis(fecha_inicial) fecha_final = s.fecha_analisis(fecha_final)	O(1)
jobs = control['jobs'] cumplen = s.new_list() ciudad = s.new_list()	O(1)

for i in range(data_size(jobs)):
for i in range(len(elemento)):
  info = elemento[i]
	O(n*m)

elemento = mp.get_value(jobs, i)	O(1)

if ciudad_o == ciudad_oferta:
  s.add_last(ciudad, info)
  if fecha >= fecha_inicial and fecha <= fecha_final:
  s.add_last(cumplen, info)
	O(1)

cumplen_ord = merge.merge_sort(cumplen, 'published_at')	O(n log n)
empresas = s.conteo(pais, 'company_name') empresa_menos = s.last_element(empresa)	O(n)
total_empresas = s.size(empresas) total_ofertas = s.size(cumple_ord) empresa_mas = s.first_element(ciudades)	O(1)
for i in range(total_ofertas):	O(k)
if len(ofertas_ciudad) > 11	O(1)
TOTAL	***O(n log n) o O(n*m) depende del valor de m***

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	367.814
10 pct	172.115
20 pct	211.487
30 pct	326.346
50 pct	486.216

80 pct	584.654
large	630.241

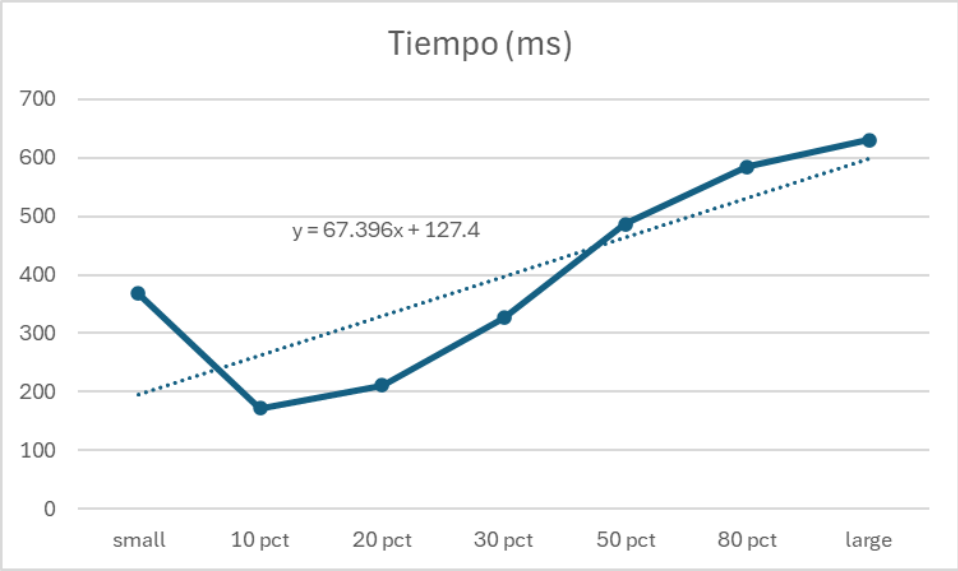
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Data governance specialist	367.814
10 pct	Data governance specialist	172.115
20 pct	Data governance specialist	211.487
30 pct	Data governance specialist	326.346
50 pct	Data governance specialist	486.216
80 pct	Data governance specialist	584.654
large	Data governance specialist	630.241

Graficas

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.

Requerimiento 6

Descripción

```
def req_6(control,num_ciudades,nivel_experticia,ano_consulta):
    """
    Función que soluciona el requerimiento 6
    """
    inicio=datetime(ano_consulta,1,1)
    fin=datetime(ano_consulta,12,31,23,59,59)
    jobs=control['jobs']
    emply=control['employments']
    cumplen=s.new_list()
    for i in range(len(mp.keys(jobs))):
        key_l=mp.keys(jobs)[i]
        for j in range(len(key_l)):
            key=key_l[j]
            elemento=mp.get(jobs,key)
            info=elemento['value']
            nivel=info['experience_level']
            fecha=info['published_at']
            if fecha >= inicio and fecha <= fin:
                if nivel_experticia == 'indiferente':
                    s.add_last(cumplen,info)
                else:
                    if nivel == nivel_experticia:
                        s.add_last(cumplen,info)
    ciudades=s.conteo_info(cumplen,'city')
    total_ciudades=s.size(ciudades)
    if total_ciudades < num_ciudades:
        resultado_total_ciudades=total_ciudades
    else:
        resultado_total_ciudades=num_ciudades
    empresas=s.conteo(cumplen,'company_name')
    resultado_total_empresas=s.size(empresas)
    resultado_total_ofertas=s.size(cumplen)
    ciudad_mas=s.first_element(ciudades)
    ciudad_menos=s.last_element(ciudades)

    ofertas_r1=[['Ciudad', 'País', 'Ofertas hechas',
                 'Promedio salario', 'Número empresas',
                 'Empresa más ofertas', 'ID Mejor oferta',
                 'ID Peor oferta']]

    for i in range(total_ciudades):
        dato = s.get_element(ciudades,i)
        ciudad=dato[0]
        empresas_ciudad=s.new_list()
        salario = 0
        sal_max = 0
        sal_min = 9999999999
        sal_max_info = None
        sal_min_info = None
        for j in range(s.size(cumplen)):
            elem = s.get_element(cumplen,j)
            city = elem['city']
            if city == ciudad:
                s.add_last(empresas_ciudad,elem)
                aidi = elem['id']
                info_sal = mp.get(emply,aidi)['value']
                if info_sal['salary_from'] != '':
                    sal_info = (int(info_sal['salary_to'])+int(info_sal['salary_from']))/2
                    sal_info2=int(info_sal['salary_to'])
                    salario += sal_info
                    if sal_info2 > sal_max:
                        sal_max_info = elem['id']
                        sal_max=sal_info2
                    if sal_info2 < sal_min:
                        sal_min_info = elem['id']
                        sal_min=sal_info2
        empresas_conteo = s.conteo(empresas_ciudad,'company_name')
        salario=salario/s.size(empresas_ciudad)

        datos_lista=[ciudad, dato[1]['country_code'], dato[2],
                     round(salario,2), s.size(empresas_conteo),
                     s.first_element(empresas_conteo), sal_max_info, sal_min_info]
        ofertas_r1.append(datos_lista)

    ofertas_r1=ofertas_r1[0:num_ciudades+1]
    if len(ofertas_r1) > 11:
        primeros=ofertas_r1[0:6]
        ultimos=ofertas_r1[-5:]
        ofertas_r1=primeros+ultimos

    return [resultado_total_ciudades, resultado_total_empresas,
            resultado_total_ofertas, ciudad_mas, ciudad_menos, ofertas_r1]
```

Este requerimiento se encarga de clasificar las N ciudad con mayor cantidad de ofertas de trabajo dado unos requerimientos. A partir de un año de trabajo dado y un nivel de experiencia la función compara esta información con la de una tabla de hash para extraerla. Al extraer la información se retorna unos conteos, datos y una lista relacionados a las ciudades. Si no se encuentran ciudades con estos requerimientos se retorna None.

Entrada	Estructuras de datos del modelo, num_ciudades(int), nivel_experticia y ano_consulta (con formato "%Y").
Salidas	Conteos relacionados a las ciudades y a sus ofertas ("resultado_total_ciudades, resultado_total_empresas, resultado_total_ofertas, ciudad_mas, ciudad_menos") y el listado de las ciudades con información, si no existen se retorna None
Implementado (Sí/No)	Si. Implementado grupalmente

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
inicio=datetime(ano_consulta,1,1) y los otros datos de inicialización o de acceso al archivo en control	O(1)

for i in range(len(mp.keys(jobs))):
key_l=mp.keys(jobs)[i]
for j in range(len(key_l)):
	O(n^2)

ciudades=s.conteo_info(cumplen,'city')	O(n log n)

total_ciudades=s.size(ciudades)
if total_ciudades < num_ciudades:
resultado_total_ciudades=total_ciudades
else:
resultado_total_ciudades=num_ciudades
	O (1)

empresas=s.conteo(cumplen,'company_name'	O(m^2)
resultado_total_empresas=s.size(empresas) resultado_total_ofertas=s.size(cumplen) ciudad_mas=s.first_element(ciudades) ciudad_menos=s.last_element(ciudades)	O (1)

for i in range(total_ciudades):
dato = s.get_element(ciudades,i)
ciudad=dato[0]
empresas_ciudad=s.new_list()
salario = 0
sal_max = 0
sal_min = 9999999999
sal_max_info = None
sal_min_info = None
for j in range(s.size(cumplen)):
	O (m^2)

TOTAL	***O(n^2)***

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	1085.182
10 pct	268.253
20 pct	571.273
30 pct	958.296
50 pct	3801.710
80 pct	4089.167
large	3206.087

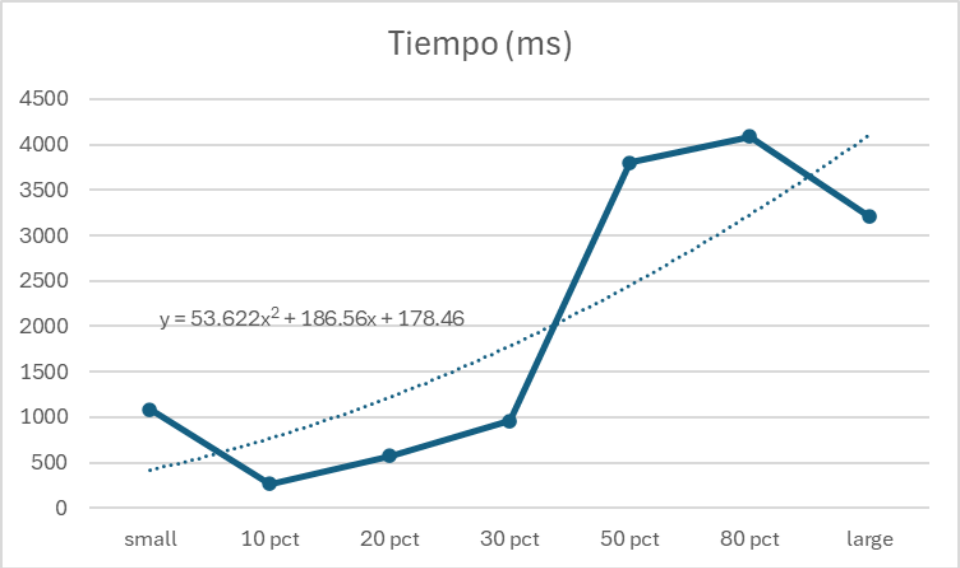
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Warszawa	1085.182
10 pct	Warszawa	268.253
20 pct	Warszawa	571.273
30 pct	Warszawa	958.296
50 pct	Warszawa	3801.710
80 pct	Warszawa	4089.167
large	Warszawa	3206.087

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A Como se observa en el gráfico, esta función tiene complejidad cuadrática, esto, ya que tiene que recorrer todos los datos contenidos, primero para extraer las llaves de los mapas, y luego para con estas, extraer los valores de dichos mapa. A esta gráfica se le realizó una regresión cuadrática donde la línea de tendencia cuadra bastante en la forma de la curva.

Requerimiento 7

Descripción

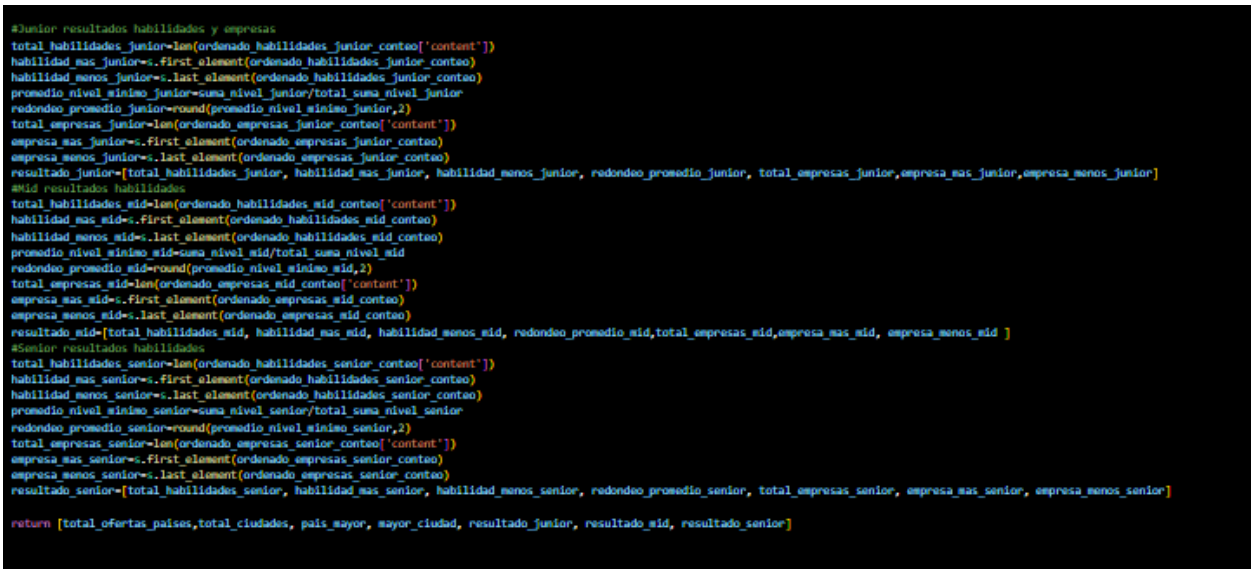
```python
def req_7(control,num_paises,ano_consulta,mes_consulta):
    """
    Función que soluciona el requerimiento 7
    """
    inicio=datetime(ano_consulta,mes_consulta,1)
    fin=datetime(ano_consulta,mes_consulta,31,23,59,59)
    jobs=control['jobs']
    skills=control['skills']
    cumplen=s.new_list()
    for i in range(len(mp.keys(jobs))):
        key_li=mp.keys(jobs)[i]
        for j in range(len(key_li)):
            key1=key_li[j]
            elemento1=mp.get(jobs,key1)
            info1=elemento1['value']
            fecha1=info1['published_at']
            if fecha1 >= inicio and fecha1 <= fin:
                s.add_last(cumplen,info1)
                idi=info1['id']
                experiencia=info1['experience_level']

    resultado2=s.conteo_info_req7(cumplen,'country_code')
    ordenado=merge.merge_sort(resultado2, 1, 0 )
    #N paises
    total_content=len(ordenado['content'])
    if num_paises<=total_content:
        ordenado=ordenado['content'][:num_paises]
    else:
        ordenado=ordenado['content'][:total_content]
    total_ofertas_paises=0
    for sublista in ordenado:
        total_ofertas_paises+=sublista[1]
    #mayor pais
    pais_mayor_oferta=ordenado[0][0]
    conteo_pais_mayor_oferta=ordenado[0][1]
    pais_mayor=[pais_mayor_oferta,conteo_pais_mayor_oferta]
    #mayor ciudad
    ciudad_max_numero = None
    max_numero = 0
    #Mapa para cada nivel de experticia donde está el id como llave y la empresa como valor
    mapa_junior=mp.new_map(50000)
    mapa_mid=mp.new_map(50000)
    mapa_senior=mp.new_map(50000)
    for pais, numero, ciudades, experticia in ordenado:
        for ciudad, num in ciudades:
            if num > max_numero:
                max_numero = num
                ciudad_max_numero = ciudad
        for nivel, diccionario_id in experticia.items():
            if nivel == 'junior':
                for llave, valor in diccionario_id.items():
                    mp.put(mapa_junior, llave, valor)
            elif nivel == 'mid':
                for llave, valor in diccionario_id.items():
                    mp.put(mapa_mid, llave, valor)
            elif nivel == 'senior':
                for llave, valor in diccionario_id.items():
                    mp.put(mapa_senior, llave, valor)
    mayor_ciudad=[ciudad_max_numero, max_numero]
    #Total ciudades
    total_ciudades = 0
    for elemento in ordenado:
        total_ciudades += len(elemento[2])
```

```
#Habilidades
#lista de id en cada mapa
id_junior=mp.keys(mapa_junior)
id_junior=[sublista for sublista in id_junior if len(sublista) > 0]
id_junior = [dato for sublista in id_junior for dato in sublista]
id_mid=mp.keys(mapa_mid)
id_mid=[sublista for sublista in id_mid if len(sublista) > 0]
id_mid = [dato for sublista in id_mid for dato in sublista]
id_senior=mp.keys(mapa_senior)
id_senior=[sublista for sublista in id_senior if len(sublista) > 0]
id_senior = [dato for sublista in id_senior for dato in sublista]
#Recorrido archivo skills y comparar id
habilidades_junior=s.new_list()
habilidades_mid=s.new_list()
habilidades_senior=s.new_list()
suma_nivel_junior=0
suma_nivel_mid=0
suma_nivel_senior=0
total_suma_nivel_junior=0
total_suma_nivel_mid=0
total_suma_nivel_senior=0
for i in range(len(mp.keys(skills))):
    key_li=mp.keys(skills)[i]
    for j in range(len(key_li)):
        key1=key_li[j]
        elemento1=mp.get(skills,key1)
        info1=elemento1['value']
        id=info1['id']
        nombre_skill=info1['name']
        nivel_skill=int(info1['level'])
        if id in id_junior:
            s.add_last(habilidades_junior,info1)
            suma_nivel_junior+=nivel_skill
            total_suma_nivel_junior += 1
        elif id in id_mid:
            s.add_last(habilidades_mid,info1)
            suma_nivel_mid+=nivel_skill
            total_suma_nivel_mid += 1
        elif id in id_senior:
            s.add_last(habilidades_senior,info1)
            suma_nivel_senior+=nivel_skill
            total_suma_nivel_senior += 1


habilidades_junior_conteo=s.conteo(habilidades_junior,'name')
ordenado_habilidades_junior_conteo=merge.merge_sort(habilidades_junior_conteo, 1, 0 )
habilidades_mid_conteo=s.conteo(habilidades_mid,'name')
ordenado_habilidades_mid_conteo=merge.merge_sort(habilidades_mid_conteo, 1, 0 )
habilidades_senior_conteo=s.conteo(habilidades_senior,'name')
ordenado_habilidades_senior_conteo=merge.merge_sort(habilidades_senior_conteo, 1, 0 )
```

```
#empresas
#Valores empresas junior
empresas_junior=s.new_list()
for i in range(len(mp.keys(mapa_junior))):
    key_li=mp.keys(mapa_junior)[i]
    for j in range(len(key_li)):
        key1=key_li[j]
        elemento1=mp.get(mapa_junior,key1)
        if elemento1 is not None and elemento1['value'] is not None:
            info1=elemento1['value']
            s.add_last(empresas_junior,[info1])
empresas_junior_conteo=s.conteo(empresas_junior,0)
ordenado_empresas_junior_conteo=merge.merge_sort(empresas_junior_conteo, 1, 0 )
#Valores empresas mid
empresas_mid=s.new_list()
for i in range(len(mp.keys(mapa_mid))):
    key_li=mp.keys(mapa_mid)[i]
    for j in range(len(key_li)):
        key1=key_li[j]
        elemento1=mp.get(mapa_mid,key1)
        if elemento1 is not None and elemento1['value'] is not None:
            info1=elemento1['value']
            s.add_last(empresas_mid,[info1])
empresas_mid_conteo=s.conteo(empresas_mid,0)
ordenado_empresas_mid_conteo=merge.merge_sort(empresas_mid_conteo, 1, 0 )
#Valores empresas senior
empresas_senior=s.new_list()
for i in range(len(mp.keys(mapa_senior))):
    key_li=mp.keys(mapa_senior)[i]
    for j in range(len(key_li)):
        key1=key_li[j]
        elemento1=mp.get(mapa_senior,key1)
        if elemento1 is not None and elemento1['value'] is not None:
            info1=elemento1['value']
            s.add_last(empresas_senior,[info1])
empresas_senior_conteo=s.conteo(empresas_senior,0)
ordenado_empresas_senior_conteo=merge.merge_sort(empresas_senior_conteo, 1, 0 )
```



```
#Junior resultados habilidades y empresas
total_habilidades_junior=len(ordenado_habilidades_junior_conteo['content'])
habilidad_mas_junior=s.first_element(ordenado_habilidades_junior_conteo)
habilidad_menos_junior=s.last_element(ordenado_habilidades_junior_conteo)
promedio_nivel_minimo_junior=suma_nivel_junior/total_suma_nivel_junior
redondeo_promedio_junior=round(promedio_nivel_minimo_junior,2)
total_empresas_junior=len(ordenado_empresas_junior_conteo['content'])
empresa_mas_junior=s.first_element(ordenado_empresas_junior_conteo)
empresa_menos_junior=s.last_element(ordenado_empresas_junior_conteo)
resultado_junior=[total_habilidades_junior, habilidad_mas_junior, habilidad_menos_junior, redondeo_promedio_junior, total_empresas_junior,empresa_mas_junior,empresa_menos_junior]
#Mid resultados habilidades
total_habilidades_mid=len(ordenado_habilidades_mid_conteo['content'])
habilidad_mas_mid=s.first_element(ordenado_habilidades_mid_conteo)
habilidad_menos_mid=s.last_element(ordenado_habilidades_mid_conteo)
promedio_nivel_minimo_mid=suma_nivel_mid/total_suma_nivel_mid
redondeo_promedio_mid=round(promedio_nivel_minimo_mid,2)
total_empresas_mid=len(ordenado_empresas_mid_conteo['content'])
empresa_mas_mid=s.first_element(ordenado_empresas_mid_conteo)
empresa_menos_mid=s.last_element(ordenado_empresas_mid_conteo)
resultado_mid=[total_habilidades_mid, habilidad_mas_mid, habilidad_menos_mid, redondeo_promedio_mid,total_empresas_mid,empresa_mas_mid, empresa_menos_mid ]
#Senior resultados habilidades
total_habilidades_senior=len(ordenado_habilidades_senior_conteo['content'])
habilidad_mas_senior=s.first_element(ordenado_habilidades_senior_conteo)
habilidad_menos_senior=s.last_element(ordenado_habilidades_senior_conteo)
promedio_nivel_minimo_senior=suma_nivel_senior/total_suma_nivel_senior
redondeo_promedio_senior=round(promedio_nivel_minimo_senior,2)
total_empresas_senior=len(ordenado_empresas_senior_conteo['content'])
empresa_mas_senior=s.first_element(ordenado_empresas_senior_conteo)
empresa_menos_senior=s.last_element(ordenado_empresas_senior_conteo)
resultado_senior=[total_habilidades_senior, habilidad_mas_senior, habilidad_menos_senior, redondeo_promedio_senior, total_empresas_senior, empresa_mas_senior, empresa_menos_senior]

return [total_ofertas_paises,total_ciudades, pais_mayor, mayor_ciudad, resultado_junior, resultado_mid, resultado_senior]
```

Este requerimiento se encarga de clasificar n países con mayor número de ofertas e identificar las habilidades solicitadas de las ofertas. A partir de un número de países, el año y mes de la consulta se comparan y se extraen datos de una tabla de hash. Después de extraer los datos que cumplen los parámetros se organizan unos conteos y una lista de ofertas. Si ningún país cumple con los requerimientos se retorna None.

Entrada	Estructuras de datos del modelo, num_paises (int), ano_consulta, a (con formato "%Y") y mes_consulta (con formato "%m").
Salidas	Los conteos (int) o nombre de un dato (str) y el listado de ofertas (list), si no existe se retorna None
Implementado (Sí/No)	Si. Implementado grupalmente

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
inicio=datetime(ano_consulta, mes_consulta,1) Fin=daterime(ano_consulta, mes_consulta, 31,23,59,59)	O(1). Esto se debe a que la creación de un objeto datetime implica simplemente asignar valores a sus atributos, en un tiempo fijo y predecible
cumplen=s.new_list()	O(1). Esta función simplemente crea una lista con valores iniciales predeterminados, operación constante
For i in range(len(mp.keys(jobs)))	O(N). Se itera sobre todos los elementos de la estructura. Además, mp.keys es constante
Key_l=mp.keys(jobs)[i]	O(1). Es constante ya que devuelve una referencia a la lista interna de llaves la cual ya está en el mapa
For j in range (len(key_l))	O(P). Itera sobre cada elemento

Elemento1=mp.get(jobs,key1)	O(P). En el peor de los casos, donde P es la cantidad de elementos en la lista de llaves en la posición calculada por el hash
s.add_last (cumplen,info)	O(1). Esta función no necesita recorrer toda la lista ya que tiene una referencia directa al ultimo elemento
if fecha1 >=inicio and fecha1<=fin:	O(1). Python utiliza la técnica de hashing para comparar cadenas y así hacerlo más eficiente
Resultado 2=s.conteo_info_req7(cumple, 'country_code')	O(M^2). Hay dos bucles en la función
ordenado= merge.merge_sort(cumplen, 'country_code')	O (M log M).(M porque es sobre la lista cumplen) Esto se debe a que merge utiliza dividir y conquistar. Dividir la lista en dos recursivamente (O(log M)) y Conquistar en el caso que fusiona dos sublistas ordenadas de tamaño M/2 lo que tiene una complejidad de O(M)
Mapa_junior=mp.new_map(50000) Mapa_mid=mp.new_map(50000) Mapa_senior=mp.new_map(50000)	O(capacity). Capacity es el tamaño inicial de la tabla

for pais, numero, ciudades, experticia in ordenado:
for ciudad, num in ciudades:
if num > max_numero:
max_numero = num
ciudad_max_numero = ciudad
for nivel, diccionario_id in experticia.items():
if nivel == 'junior':
for llave, valor in diccionario_id.items():
mp.put(mapa_junior, llave, valor)
elif nivel == 'mid':
for llave, valor in diccionario_id.items():
mp.put(mapa_mid, llave, valor)
elif nivel == 'senior':
for llave, valor in diccionario_id.items():
mp.put(mapa_senior, llave, valor)
	O(MxAxB): Debe realizar tres bucles con tres listas distintas, ordenado, ciudades, experticia

mp.put(mapa_junior, llave, valor)	O(C) en el peor de los casos, donde C es la cantidad de elementos en la lista de llaves en la posicion caculada por el hash
For elemento in ordendado	O(M)
mp.keys (mapa_junior)	O(1). Es constante ya que devuelve una referencia a la lista interna de llaves la cual ya está en el mapa

habilidades_junior=s.new_list()	O(1). Esta función simplemente crea una lista con valores iniciales predeterminados, operación constante
For i in range(len(mp.keys(skills)))	O(E). Se itera sobre todos los elementos de la estructura. Además, mp.keys es constante
Key_l=mp.keys(skills)[i]	O(1). Es constante ya que devuelve una referencia a la lista interna de llaves la cual ya está en el mapa
For j in range (len(key_l))	O(F). Itera sobre cada elemento
Elemento1=mp.get(skills,key1)	O(F). En el peor de los casos, donde P es la cantidad de elementos en la lista de llaves en la posición calculada por el hash
s.add_last (habilidades,info)	O(1). Esta función no necesita recorrer toda la lista ya que tiene una referencia directa al ultimo elemento
Habilidades_junior_conteo=s.conteo(habilidades_junior, 'name') Habilidades_mid_conteo=s.conteo(habilidades_mid, 'name') Habilidades_senior_conteo=s.conteo(habilidades_senior, 'name')	O(G^2). Hay dos bucles en la función
ordenado= merge.merge_sort(Habilidades_junior_conteo, 1,0) ordenado= merge.merge_sort(Habilidades_mid_conteo, 1,0) ordenado= merge.merge_sort(Habilidades_senior_conteo, 1,0)	O (M log M).(M porque es sobre la lista cumplen) Esto se debe a que merge utiliza dividir y conquistar. Dividir la lista en dos recursivamente (O(log M)) y Conquistar en el caso que fusiona dos sublistas ordenadas de tamaño M/2 lo que tiene una complejidad de O(M)
Empresas_junior=s.new_list Empresas_midr=s.new_list Empresas_senior=s.new_list	O(1). Esta función simplemente crea una lista con valores iniciales predeterminados, operación constante
For i in range(len(mp.keys(mapa_junior)))	O(C). Se itera sobre todos los elementos de la estructura. Además, mp.keys es constante
Key_l=mp.keys(mapa_junio)[i]	O(1). Es constante ya que devuelve una referencia a la lista interna de llaves la cual ya está en el mapa
For j in range (len(key_l))	O(H). Itera sobre cada elemento
Elemento1=mp.get(mapa_junior,key1)	O(I). En el peor de los casos, donde P es la cantidad de elementos en la lista de llaves en la posición calculada por el hash
s.add_last (empressas_junior,info)	O(1). Esta función no necesita recorrer toda la lista ya que tiene una referencia directa al ultimo elemento

empresas_junior_conteo=s.conteo(empresas_junior, 'name') empresas_mid_conteo=s.conteo(empresas_mid, 'name') empresas_senior_conteo=s.conteo(empresas_senior, 'name')	O(G^2). Hay dos bucles en la función
ordenado= merge.merge_sort(empresass_junior_conteo, 1,0) ordenado= merge.merge_sort(empresas_mid_conteo, 1,0) ordenado= merge.merge_sort(empresas_senior_conteo, 1,0)	O (M log M).(M porque es sobre la lista cumplen) Esto se debe a que merge utiliza dividir y conquistar. Dividir la lista en dos recursivamente (O(log M)) y Conquistar en el caso que fusiona dos sublistas ordenadas de tamaño M/2 lo que tiene una complejidad de O(M)
habilidad_mas_junior=s.first_element(ordenado_habilidades_junior_conteo) habilidad_menos_junior=s.last_element(ordenado_habilidades_junior_conteo)	O(1). Tiene una referencia al primer y ultimo elemento
TOTAL	***O(M log M) o O(MxAxB) Depende de los valores especificos de M,A y B***

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	34384.284
10 pct	4500.725
20 pct	11740.515
30 pct	38810.987
50 pct	76972.959
80 pct	127800.296
large	91116.241

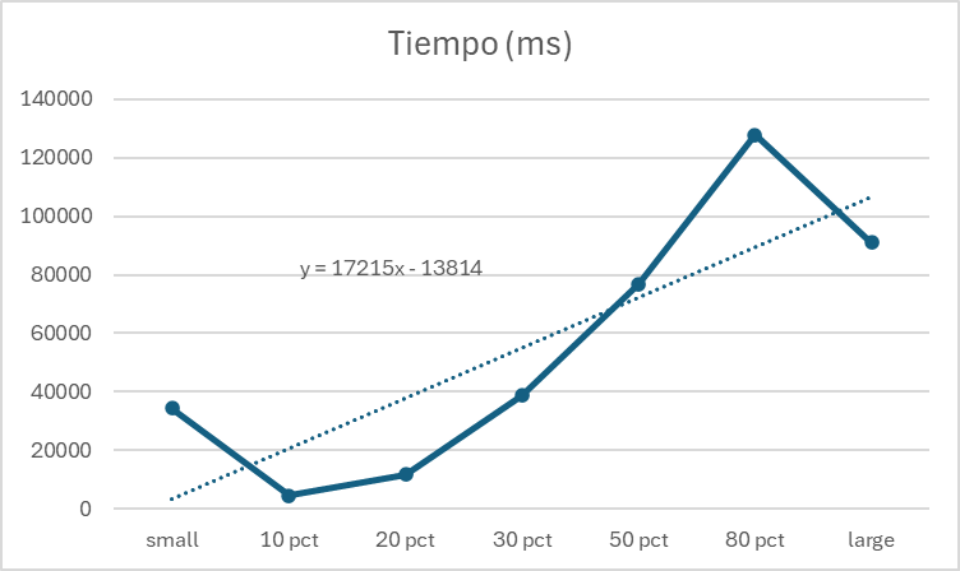
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	6281	34384.284
10 pct	1847	4500.725
20 pct	3771	11740.515
30 pct	5672	38810.987
50 pct	9769	76972.959
80 pct	11749	127800.296
large	11830	91116.241

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Por la cantidad de recorridos que debe hacer el requerimiento para acceder a la información en varios archivos, se debe demorar mucho tiempo comparado con los demás. Es contundente la variación del tiempo de ejecución por cada porcentaje de archivo, por lo que se puede inferir que la forma en que se guarda la información y se maneja se debe controlar y simplificar mucho más para que reduzca dichos tiempos

Requerimiento 8

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 identificar los países con mayor y menor ofertas según un rango de fechas. A partir del análisis de datos de una tabla de hash ,que adicional a los anteriores requisitos tengan un rango salarial, y se extrae para obtener una respuesta de dos partes. La primera es las estadísticas y la segunda parte los países. Si no identifica los datos retorna None.

Entrada	Estructuras de datos del modelo, nivel_experiencia(str), divisa(str), fecha_inicial y fecha_final((con formato "%Y-%m-%d").
Salidas	Una primera parte con estadísticas y la segunda en el menor/mayor pais.
Implementado (Sí/No)	Si. Implementado grupalmente

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

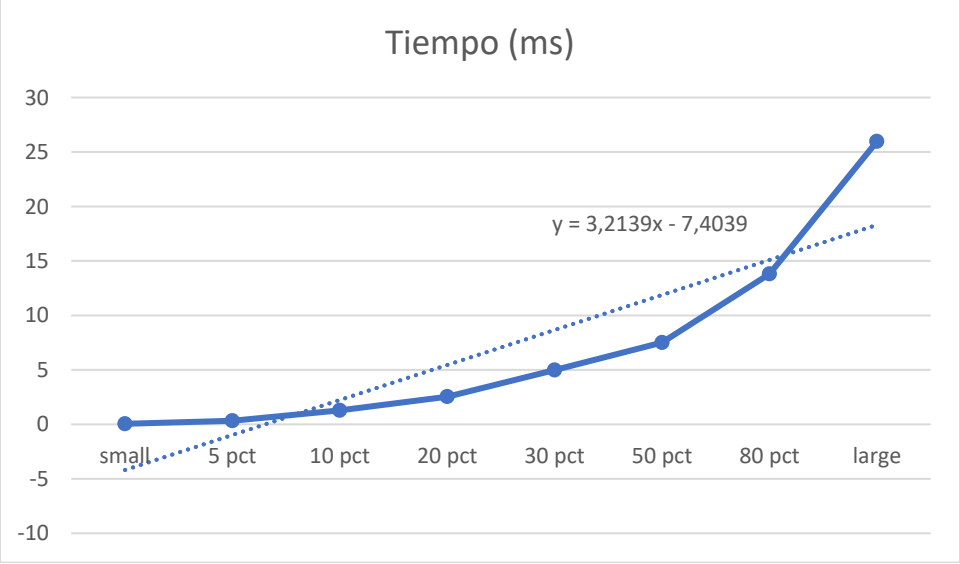
Tablas de datos

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

Graficas

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.