ANÁLISIS DEL RETO

Miguel Santiago Roa Vallejo, 202322288, ms.roa@uniandes.edu.co

Mattia Riccardi, 202321259, m.riccardi@uniandes.edu.co

Estudiante 3, código 3, email 3

Requerimiento <<n>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como 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.

Graficas
Las gráficas con la representación de las pruebas realizadas.

Análisis
Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis 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

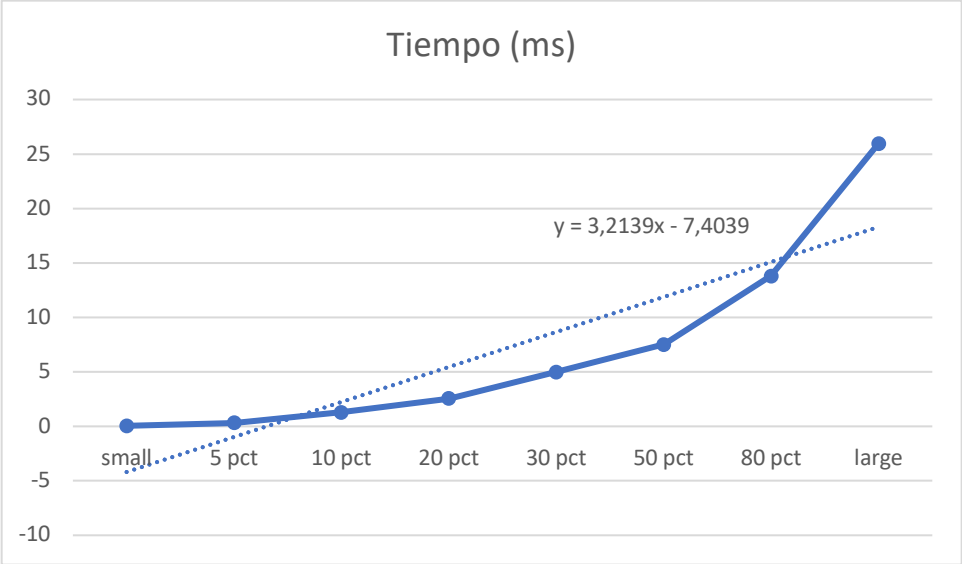
Tablas de datos

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

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 1

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de

```
req_1(data_structs, n_ofertas, codigo_pais, experiencia):
# TODO: Realizar el requerimiento 1
lista = lt.newList()
contador_ofertas = 0
for job in lt.iterator(data_structs["jobs"]):
    if str(job["experience_level"]) == str(experiencia) and str(job["country_code"]) == str(codigo_pais):
        lt.addLast(lista,job)
        contador_ofertas += 1

    if contador_ofertas == n_ofertas:
        break

return lista
```

```
def req_1(control,n_ofertas,codigo_pais,experiencia):
    """
    Retorna el resultado del requerimiento 1
    """
    # TODO: Modificar el requerimiento 1
    tiempo_inicial = get_time()
    lista = model.req_1(control["model"],n_ofertas,codigo_pais,experiencia)
    tiempo_final = get_time()
    tiempo_total = delta_time(tiempo_inicial, tiempo_final)
    return lista, tiempo_total
```

```python
def print_req_1(control):
    print("Listar las últimas N ofertas de trabajo según su país y nivel de experticia")
    n_ofertas = int(input("¿Cuántas ofertas desea consultar: "))
    codigo_pais = str(input("Digite el código del país: "))
    experiencia = str(input("Digite el nivel de experiencia (senior/mid/junior): ")).lower()
    lista, tiempo_total = controller.req_1(control,n_ofertas,codigo_pais,experiencia)
    resultado = []
    contador = 0
    headers = ["Fecha", "Título", "Empresa", "Experiencia", "País", "Ciudad", "Tamaño", "Ubicación de traba
    for job in lt.iterator(lista):
        fila = [
            job["published_at"],
            job["title"],
            job["company_name"],
            job["experience_level"],
            job["country_code"],
            job["city"],
            job["company_size"],
            job["workplace_type"],
            job["open_to_hire_ukrainians"]
        ]
        resultado.append(fila)
        contador += 1
        #print(resultado)
        #print(tabulate([resultado], headers=["Fecha", "Título", "Empresa", "Experiencia", "País", "Ciudad",
    print("")
    print(tabulate(resultado, headers=headers, tablefmt="pretty"))
    print("")
    print("Número total de ofertas de trabajo ofrecidas según la condición: " + str(contador))
    print("Tiempo de carga: " + str(tiempo_total))
```

Entrada	(data_structs, n_ofertas, codigo_pais, experiencia)
Salidas	• El total de ofertas de trabajo ofrecidas según la condición (junior, mid, o senior). • Para cada una de las ofertas de la consulta debe presentar la siguiente información: o Fecha de publicación de la oferta o Título de la oferta o Nombre de la empresa de la oferta o Nivel de experticia de la oferta (es el mismo del filtro) o País de la empresa de la oferta o Ciudad de la empresa de la oferta o Tamaño de la empresa de la oferta o Tipo de ubicación de trabajo (remote, partialy_remote, office) o Disponible a contratar ucranianos (Verdadero o Falso)
Implementado (Sí/No)	Si, lo hizo Mattia Riccardi

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Inicializar una lista	O(1)
Iterar la estructura “jobs” para agregar a la lista	O(n)
Paso ….	O(…)

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 M2**
Memoria RAM	16 GB
Sistema Operativo	macOS 14.0

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	62.163
10 pct	Dato3	41.971
20 pct	Dato4	50.144
30 pct	Dato5	74.300
50 pct	Dato6	98.394
80 pct	Dato7	92.278
large	Dato8	90.678

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

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

Descripción:

El requerimiento se refiere a la implementación de un proceso que filtre ofertas de trabajo según ciertos criterios y presente información específica para cada oferta que cumple con dichos criterios. La entrada del proceso incluye una estructura de datos (data_structs), el número de ofertas a considerar (n_ofertas), el código del país y la

experiencia requerida. Las salidas incluyen el total de ofertas según la experiencia y detalles específicos para cada oferta seleccionada.

Análisis de Complejidad:

Se proporciona un análisis de complejidad que describe la complejidad de cada paso del algoritmo. En este caso, se menciona la inicialización de una lista (O(1)) y la iteración sobre la estructura de trabajos para agregar elementos a la lista (O(n)). La complejidad total se resume como O(n), donde "n" es el número de ofertas de trabajo.

Pruebas Realizadas:

Se han realizado pruebas de tiempo de ejecución y memoria utilizada en diferentes escenarios. Las pruebas se han ejecutado en un procesador Apple M2 con 16 GB de RAM y el sistema operativo macOS 14.0. Se proporcionan datos de tiempo en milisegundos para diversas muestras, desde "small" hasta "large". Se incluyen detalles sobre el procedimiento de prueba, las condiciones y las herramientas utilizadas.

Tablas de Datos:

Se presenta una tabla que muestra la recopilación de datos de las pruebas. Cada fila representa una muestra, con detalles sobre la salida esperada, el tiempo de ejecución en milisegundos y el tamaño de la muestra.

Conclusión:

El análisis del requerimiento, la implementación y las pruebas proporcionan una visión detallada de cómo se abordó este aspecto del sistema. Proporciona información útil para evaluar el rendimiento y la eficiencia del algoritmo implementado.

Requerimiento 7

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripció

App > model.py > req_7

```
908  def req_7(data_structs, n_paises, fecha_inicial, fecha_final):
996      diccionario = {}
997      jobs = data_structs["jobs"]
998      fecha_inicial = convertir_fecha(fecha_inicial)
999      fecha_final = convertir_fecha(fecha_final)
1000     total_ofertas = 0
1001     habilidad_junior = None
1002     habilidad_mid = None
1003     habilidad_senior = None
1004     suma_senior = None
1005     suma_mid = None
1006     suma_junior = None
1007     empresas_junior = None
1008     empresas_mid = None
1009     empresas_senior = None
1010     habilidad_junior_append = None
1011     habilidad_senior_append = None
1012     habilidad_mid_append = None
1013
1014     data_structs["skills"] = sa.sort(data_structs["skills"], compare_id)
1015     #ciudades = {}
1016     for oferta in lt.iterator(jobs):
1017         fecha = convertir_fecha(oferta["published_at"])
1018         if fecha_comparacion(fecha, fecha_inicial) and fecha_comparacion(fecha_final, fecha):
1019             total_ofertas += 1
1020             #if oferta["city"] not in ciudades:
1021                 #ciudades[oferta["city"]] = 1
1022             #else: ciudades[oferta["city"]] += 1
1023             if oferta["experience_level"] == "senior":
1024                 element = buscar_habilidad_oferta(data_structs,oferta)
1025                 #for element in lt.iterator(data_structs["skills"]):
1026                     #print(element)
1027                 if element["id"] == oferta["id"]:
1028                     habilidad_senior = [element["name"]]
1029                     habilidad_senior_append = element["name"]
1030                 suma_senior = int(element["level"])
1031                 #cantidad_senior = 1
1032                 empresas_senior = [oferta["company_name"]]
1033                 empresas_senior_append = oferta["company_name"]
1034             if oferta["experience_level"] == "mid":
1035                 element = buscar_habilidad_oferta(data_structs, oferta)
1036                 #for element in lt.iterator(data_structs["skills"]):
1037                 if element["id"] == oferta["id"]:
1038                     habilidad_mid = [element["name"]]
1039                     habilidad_mid_append = element["name"]
1040                 suma_mid = int(element["level"])
1041                 #cantidad_mid = 1
1042                 empresas_mid = [oferta["company_name"]]
1043                 empresas_mid_append = oferta["company_name"]
1044             if oferta["experience_level"] == "junior":
1045                 element = buscar_habilidad_oferta(data_structs, oferta)
1046                 #for element in lt.iterator(data_structs["skills"]):
1047                 if element["id"] == oferta["id"]:
1048                     habilidad_junior = [element["name"]]
1049                     habilidad_junior_append = element["name"]
1050                 suma_junior = int(element["level"])
1051                 #cantidad_junior = 1
```

n de como abordaron la implementación del requerimiento

Entrada	(data_structs, n_paises, fecha_inicial, fecha_final) El número (N) de países para consulta (ej.: 3, 5, 10 o 20). • La fecha inicial del periodo a consultar (con formato "%Y-%m-%d"). • La fecha final del periodo a consultar (con formato "%Y-%m-%d").
Salidas	El total de ofertas de empleo • Número de ciudades donde se ofertó en los países resultantes de la consulta. • Nombre del país con mayor cantidad de ofertas y su conteo • Nombre de la ciudad con mayor cantidad de ofertas y su conteo • Para el conjunto de las ofertas de trabajo en los países resultantes de la consulta, por cada uno de los tres niveles de experticia (junior, mid y senior) calcule y presente la siguiente información: o Conteo de habilidades diferentes solicitadas en ofertas de trabajo o Nombre de la habilidad más solicitada y su conteo en ofertas de trabajo o Nombre de la habilidad menos solicitada y su conteo en ofertas de trabajo o Nivel mínimo promedio de las habilidades o Conteo de empresas que publicaron una oferta con este nivel o Nombre de la empresa con mayor número de ofertas y su conteo o Nombre de la empresa con menor número de ofertas (al menos una) y su conteo o Número de empresas que publicaron una oferta en este nivel que tienen una o más sedes
Implementado (Sí/No)	Si, Gru

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Inicializar variables y estructuras de datos	O(1)
Iterar sobre las ofertas de trabajo	O(n)
Operaciones dentro de la iteración	O(1) (cada operación)
Iterar sobre las habilidades	O(log m), m = tamaño de skills
Crear y actualizar diccionario de países	O(1) (por cada oferta)
Crear y actualizar estructura de datos de países	O(1) (por cada oferta)
Ordenar estructura de datos de países	O(n * log n)
Crear sublistas y estructura de datos final	O(n)
Iterar sobre la sublista de N países	O(n_paises)
Operaciones dentro de la iteración final	O(1) (por cada país)
TOTAL	***O(nlogn) * 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 M2**
Memoria RAM	16 GB
Sistema Operativo	macOS 14.0

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	62.163
10 pct	Dato3	10012.36
20 pct	Dato4	29955.704
30 pct	Dato5	
50 pct	Dato6	141389.557
80 pct	Dato7	234272.700
large	Dato8	265534.982

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

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

El Requerimiento 7 tiene como objetivo realizar consultas sobre ofertas de trabajo en un periodo específico, considerando un número determinado de países. La complejidad del algoritmo se analiza detalladamente, destacando la iteración sobre las ofertas de

trabajo y la ordenación de la estructura de datos de países como los componentes principales. La complejidad total se estima como: O(nlogn) * O(n). Las pruebas realizadas en diferentes muestras, desde "small" hasta "large", revelan un aumento significativo en el tiempo de ejecución con el tamaño del conjunto de datos, siendo la ordenación la operación más costosa. Las condiciones de prueba incluyen procesadores Apple M2, 16 GB de RAM y macOS 14.0 como sistema operativo. En resumen, el algoritmo presenta eficiencia para tamaños pequeños de muestra, pero su rendimiento disminuye para conjuntos de datos más grandes, destacando la importancia de optimizar la ordenación de la estructura de datos.

model.py M | Git Graph | view.py M | controller.py M

App > model.py > req_7

```python
def req_7(data_structs, n_paises, fecha_inicial, fecha_final):
    for paises in diccionario.keys():
        diccionario_add = diccionario[paises]
        lt.addLast(estructura_de_datos_paises, diccionario_add)
    quk.sort(estructura_de_datos_paises, cmp_function_ordenar_paises_con_mas_ofertas_nombres)

    #print(estructura_de_datos_paises)

    if n_paises > lt.size(estructura_de_datos_paises):
        n_paises = lt.size(estructura_de_datos_paises)
    sublista_de_las_N_paises = lt.subList(estructura_de_datos_paises, 1, n_paises)

    lista_final = lt.newList("ARRAY_LIST")
    for x in lt.iterator(sublista_de_las_N_paises):
        ciudad_con_mas_ofertas = max(x["ciudades"], key=x["ciudades"].count)
        conteo_ciudad_mas_ofertas = x["ciudades"].count(ciudad_con_mas_ofertas)
        numero_ciudades = len(x["ciudades"])
        #junior
        habilidad_mas_solicitada_junior = max(x["junior"]["habilidades_junior"], key=x["junior"]["habilidades_junior"].count)
        conteo_habilidad_mas_solicitada_junior = x["junior"]["habilidades_junior"].count(habilidad_mas_solicitada_junior)
        habilidad_menos_solicitada_junior = min(x["junior"]["habilidades_junior"], key=x["junior"]["habilidades_junior"].count)
        conteo_habilidad_menos_solicitada_junior = x["junior"]["habilidades_junior"].count(habilidad_menos_solicitada_junior)
        promedio_junior = int(x["junior"]["suma_junior"]) / int(x["junior"]["contador_junior"])
        empresas_junior_con_mas_ofertas = max(x["junior"]["empresas_junior"], key=x["junior"]["empresas_junior"].count)
        conteo_empresas_junior_con_mas_ofertas = x["junior"]["empresas_junior"].count(empresas_junior_con_mas_ofertas)

        #mid
        habilidad_mas_solicitada_mid = max(x["mid"]["habilidades_mid"], key=x["mid"]["habilidades_mid"].count)
        conteo_habilidad_mas_solicitada_mid = x["mid"]["habilidades_mid"].count(habilidad_mas_solicitada_mid)
        habilidad_menos_solicitada_mid = min(x["mid"]["habilidades_mid"], key=x["mid"]["habilidades_mid"].count)
        conteo_habilidad_menos_solicitada_mid = x["mid"]["habilidades_mid"].count(habilidad_menos_solicitada_mid)
        promedio_mid = int(x["mid"]["suma_mid"]) / int(x["mid"]["contador_mid"])
        empresas_mid_con_mas_ofertas = max(x["mid"]["empresas_mid"], key=x["mid"]["empresas_mid"].count)
        conteo_empresas_mid_con_mas_ofertas = x["mid"]["empresas_mid"].count(empresas_mid_con_mas_ofertas)

        #senior
        habilidad_mas_solicitada_senior = max(x["senior"]["habilidades_senior"], key=x["senior"]["habilidades_senior"].count)
        conteo_habilidad_mas_solicitada_senior = x["senior"]["habilidades_senior"].count(habilidad_mas_solicitada_senior)
        habilidad_menos_solicitada_senior = min(x["senior"]["habilidades_senior"], key=x["senior"]["habilidades_senior"].count)
        conteo_habilidad_menos_solicitada_senior = x["senior"]["habilidades_senior"].count(habilidad_menos_solicitada_senior)
        promedio_senior = int(x["senior"]["suma_senior"]) / int(x["senior"]["contador_senior"])
        empresas_senior_con_mas_ofertas = max(x["senior"]["empresas_senior"], key=x["senior"]["empresas_senior"].count)
        conteo_empresas_senior_con_mas_ofertas = x["senior"]["empresas_senior"].count(empresas_senior_con_mas_ofertas)

        diccionario_final = {"Nombre Pais" : x["nombre pais"],
                             "Ciudad mas ofertas" : ciudad_con_mas_ofertas,
                              "Conteo" : conteo_ciudad_mas_ofertas,
                              "Numero ciudades" : numero_ciudades,
                              "Junior" : {"Habilidad mas solicitada" : habilidad_mas_solicitada_junior,
                                          "Conteo" : conteo_habilidad_mas_solicitada_junior,
                                          "Habilidad menos solicitada" : habilidad_menos_solicitada_junior,
                                          "Conteo" : conteo_habilidad_menos_solicitada_junior,
                                          "Promedio" : round(promedio_junior,3),
                                          "Empresa mas ofertas" : empresas_junior_con_mas_ofertas,
                                          "Conteo" : conteo_empresas_junior_con_mas_ofertas},
                              "Mid" : {"Habilidad mas solicitada" : habilidad_mas_solicitada_mid,
                                          "Conteo" : conteo_habilidad_mas_solicitada_mid,
                                          "Habilidad menos solicitada" : habilidad_menos_solicitada_mid,
                                          "Conteo" : conteo_habilidad_menos_solicitada_mid,
                                          "Promedio" : round(promedio_mid,3),
                                          "Empresa mas ofertas" : empresas_mid_con_mas_ofertas,
                                          "Conteo" : conteo_empresas_mid_con_mas_ofertas},
                              "Senior" : {"Habilidad mas solicitada" : habilidad_mas_solicitada_senior,
                                          "Conteo" : conteo_habilidad_mas_solicitada_senior,
                                          "Habilidad menos solicitada" : habilidad_menos_solicitada_senior,
                                          "Conteo" : conteo_habilidad_menos_solicitada_senior,
                                          "Promedio" : round(promedio_senior,3),
                                          "Empresa mas ofertas" : empresas_senior_con_mas_ofertas,
                                          "Conteo" : conteo_empresas_senior_con_mas_ofertas}
                              }
```

model.py M | Git Graph | view.py M | controller.py M

App > model.py > req_7

```python
def req_7(data_structs, n_paises, fecha_inicial, fecha_final):
            if oferta["country_code"] not in diccionario:
                pais = oferta["country_code"]
                cantidad_ofertas_pais = 1
                ciudades = [oferta["city"]]

                diccionario[pais] = {"nombre pais" : pais,
                                     "cantidad ofertas" : cantidad_ofertas_pais,
                                     "ciudades" : ciudades,
                                     "junior" : {"habilidades_junior" : habilidad_junior,
                                                 "suma_junior" : suma_junior,
                                                 "contador_junior" : 1,
                                                 "empresas_junior" : empresas_junior},
                                     "mid" : {"habilidades_mid" : habilidad_mid,
                                              "suma_mid" : suma_mid,
                                                 "contador_mid" : 1,
                                                 "empresas_mid" : empresas_mid},
                                     "senior" : {"habilidades_senior" : habilidad_senior,
                                                 "suma_senior" : suma_senior,
                                                 "contador_senior" : 1,
                                                 "empresas_senior" : empresas_senior}
                                     }
            else:
                diccionario[pais]["cantidad ofertas"] += 1
                diccionario[pais]["ciudades"].append(oferta["city"])
                #JUNIOR
                if (habilidad_junior_append != None) and (diccionario[pais]["junior"]["habilidades_junior"] != None):
                    diccionario[pais]["junior"]["habilidades_junior"].append(habilidad_junior_append)
                if (suma_junior != None) and (diccionario[pais]["junior"]["suma_junior"] != None):
                    diccionario[pais]["junior"]["suma_junior"] +=suma_junior
                diccionario[pais]["junior"]["contador_junior"] += 1
                if (empresas_junior != None) and (diccionario[pais]["junior"]["empresas_junior"] != None):
                    diccionario[pais]["junior"]["empresas_junior"].append(empresas_junior_append)

                #MID

                #print("antes")
                #print(habilidad_mid)
                if (habilidad_mid_append !=  None) and (diccionario[pais]["mid"]["habilidades_mid"] != None):
                    #print("despues")
                    #print(habilidad_mid)
                    diccionario[pais]["mid"]["habilidades_mid"].append(habilidad_mid_append)
                if (suma_mid != None) and (diccionario[pais]["mid"]["suma_mid"] != None):
                    diccionario[pais]["mid"]["suma_mid"] +=suma_mid
                diccionario[pais]["mid"]["contador_mid"] += 1
                if (empresas_mid != None) and (diccionario[pais]["mid"]["empresas_mid"] != None):
                    diccionario[pais]["mid"]["empresas_mid"].append(empresas_mid_append)

                #SENIOR
                if (habilidad_senior_append != None) and (diccionario[pais]["senior"]["habilidades_senior"] != None):
                    diccionario[pais]["senior"]["habilidades_senior"].append(habilidad_senior_append)
                if (suma_senior != None) and (diccionario[pais]["senior"]["suma_senior"] != None):
                    diccionario[pais]["senior"]["suma_senior"] +=suma_senior
                diccionario[pais]["senior"]["contador_senior"] += 1
                if (empresas_senior != None) and (diccionario[pais]["senior"]["empresas_senior"] != None):
                    diccionario[pais]["senior"]["empresas_senior"].append(empresas_senior_append)
    #numero_ciudades = len(ciudades)
```


model.py M
Git Graph
view.py M
controller.py M
App > model.py > req_7
def req_7(data_structs, n_paises, fecha_inicial, fecha_final):
habilidad_mas_solicitada_senior = max(x["senior"]["habilidades_senior"], key=x["senior"]["habilidades_senior"].count)
conteo_habilidad_mas_solicitada_senior = x["senior"]["habilidades_senior"].count(habilidad_mas_solicitada_senior)
habilidad_menos_solicitada_senior = min(x["senior"]["habilidades_senior"], key=x["senior"]["habilidades_senior"].count)
conteo_habilidad_menos_solicitada_senior = x["senior"]["habilidades_senior"].count(habilidad_menos_solicitada_senior)
promedio_senior = int(x["senior"]["suma_senior"]) / int(x["senior"]["contador_senior"])
empresas_senior_con_mas_ofertas = max(x["senior"]["empresas_senior"], key=x["senior"]["empresas_senior"].count)
conteo_empresas_senior_con_mas_ofertas = x["senior"]["empresas_senior"].count(empresas_senior_con_mas_ofertas)
diccionario_final = {"Nombre Pais" : x["nombre pais"],
"Ciudad mas ofertas" : ciudad_con_mas_ofertas,
"Conteo" : conteo_ciudad_mas_ofertas,
"Numero ciudades" : numero_ciudades,
"Junior" : {"Habilidad mas solicitada" : habilidad_mas_solicitada_junior,
"Conteo" : conteo_habilidad_mas_solicitada_junior,
"Habilidad menos solicitada" : habilidad_menos_solicitada_junior,
"Conteo" : conteo_habilidad_menos_solicitada_junior,
"Promedio" : round(promedio_junior,3),
"Empresa mas ofertas" : empresas_junior_con_mas_ofertas,
"Conteo" : conteo_empresas_junior_con_mas_ofertas},
"Mid" : {"Habilidad mas solicitada" : habilidad_mas_solicitada_mid,
"Conteo" : conteo_habilidad_mas_solicitada_mid,
"Habilidad menos solicitada" : habilidad_menos_solicitada_mid,
"Conteo" : conteo_habilidad_menos_solicitada_mid,
"Promedio" : round(promedio_mid,3),
"Empresa mas ofertas" : empresas_mid_con_mas_ofertas,
"Conteo" : conteo_empresas_mid_con_mas_ofertas},
"Senior" : {"Habilidad mas solicitada" : habilidad_mas_solicitada_senior,
"Conteo" : conteo_habilidad_mas_solicitada_senior,
"Habilidad menos solicitada" : habilidad_menos_solicitada_senior,
"Conteo" : conteo_habilidad_menos_solicitada_senior,
"Promedio" : round(promedio_senior,3),
"Empresa mas ofertas" : empresas_senior_con_mas_ofertas,
"Conteo" : conteo_empresas_senior_con_mas_ofertas}
}
lt.addLast(lista_final, diccionario_final)
#print(sublista_de_las_N_paises)
pais_con_mas_ofertas = lt.firstElement(sublista_de_las_N_paises)["nombre pais"]
conteo_pais_con_mas_ofertas = lt.firstElement(sublista_de_las_N_paises)["cantidad ofertas"]
return total_ofertas, pais_con_mas_ofertas, conteo_pais_con_mas_ofertas, lista_final
def buscar_habilidad_oferta(data_structs, jobs):
lista_skills = data_structs["skills"]
id_jobs = jobs["id"]
#voy a usar búsqueda binaria
top = lt.size(lista_skills)
valor_ref = 0
if(top != None):
indice_inicial = 1
while indice_inicial <= top:
valor_ref = (indice_inicial + top) / 2
valor_ref = int(valor_ref)
elemento_buscado= lt.getElement(lista_skills, valor_ref)
if(top != None):
if elemento_buscado["id"] < id_jobs:
indice_inicial = valor_ref + 1
elif elemento_buscado["id"] > id_jobs:
top = valor_ref - 1
else:
if(elemento_buscado != None):
return elemento_buscado
def cmp_function_ordenar_paises_con_mas_ofertas_nombres(dato1,dato2):
cantidad_1 = dato1["cantidad ofertas"]
cantidad_2 = dato2["cantidad ofertas"]

Requerimiento <<4>>

Descripción

```python
def req_4(data_structs, codigo_pais, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    lista_ofertas = data_structs["model"]["jobs"]
    ofertas_de_trabajo_rango_fechas = lt.newList("ARRAY_LIST")
    empresas = [] #hace referencia a las empresas que hicieron una publicación en el país indicado
    ciudades_sin_repetir = {} #hace referencia a las ciudades que pertenecen al país de búsqueda

    #fecha_inicial = datetime.strptime(fecha_inicial, "%Y-%m-%dT%H:%M:%S.%fZ")
    fecha_inicial = convertir_fecha(fecha_inicial)

    fecha_final = convertir_fecha(fecha_final)

    #fecha_final= datetime.strptime(fecha_final, "%Y-%m-%dT%H:%M:%S.%fZ")
    conteo = 0
    for ofertas in lt.iterator(lista_ofertas):
        fecha1 = convertir_fecha(ofertas["published_at"])


        if((fecha_comparacion(fecha1, fecha_inicial)) and (fecha_comparacion(fecha_final, fecha1)) and (ofertas["country_code"] == codigo_pais)):

            lt.addLast(ofertas_de_trabajo_rango_fechas, ofertas)
            if(ofertas["company_name"] not in empresas):
                empresas.append(ofertas["company_name"])
            if(ofertas["city"] not in ciudades_sin_repetir):
                ciudades_sin_repetir[ofertas["city"]] = 1
            else:
                ciudades_sin_repetir[ofertas["city"]] += 1


    nombre_ciudad_mas_ofertas = ""
    cantidad_mayor = -1
    nombre_ciudad_menos_ofertas = ""
    cantidad_menor = -1
    iteracion = 1
    for ciudad in ciudades_sin_repetir:
        if(iteracion == 1):
            nombre_ciudad_menos_ofertas = ciudad
            cantidad_menor = ciudades_sin_repetir[ciudad]
            iteracion += 1
        if(ciudades_sin_repetir[ciudad] > cantidad_mayor):
            cantidad_mayor = ciudades_sin_repetir[ciudad]
            nombre_ciudad_mas_ofertas = ciudad
```

```python
            if(ciudades_sin_repetir[ciudad] < cantidad_menor):
                cantidad_menor = ciudades_sin_repetir[ciudad]
                nombre_ciudad_menos_ofertas = ciudad

    total_ofertas = lt.size(ofertas_de_trabajo_rango_fechas)
    total_empresas = len(empresas)
    total_ciudades = len(ciudades_sin_repetir)

    ofertas_de_trabajo_rango_fechas_ordenadas = sa.sort(ofertas_de_trabajo_rango_fechas, cmp_ofertas_by_empresa_y_fecha)

    if(total_ofertas == 0):
        return False

    return total_ofertas, total_empresas, total_ciudades, nombre_ciudad_mas_ofertas, cantidad_mayor, nombre_ciudad_menos_ofertas, cantidad_menor, ofertas_de_trabajo_rango_fechas_ordenadas
```

Este requerimiento se encarga de retornar el número total de ofertas que cumplen con los parámetros indicados por el usuario. Lo primero que hace es crear un ARRAY_LIST donde se van a guardar todas las ofertas que cumplan con las características buscadas por el usuario, luego se crea una lista para ir guardando las empresas (y así saber el número de empresas en total), y por último un diccionario donde va a ir almacenando las ciudades y el número de ofertas que tiene cada una. Retorna el número total de ofertas, el total de empresas que cumplen con los parámetros solicitados, los nombres de las ciudades con más y menos ofertas, con sus respectivas cantidades de ofertas de trabajo, y un ARRAY_LIST con todas las ofertas ordenadas. En caso en que no haya ninguna oferta que cumpla con lo solicitado, retorna None.

Entrada	Data_stucts (estructura de datos de los Jobs), codigo_pais, fecha_inicial, fecha_final
Salidas	La cantidad de ofertas, empresas, ciudades que cumplen con dichos filtros del usuario. También retorna la ciudad con más y menos oferta, y finalmente el ARRAY_LIST con toda la información filtrada y ordenada. Si no hay ninguna oferta retorna None.
Implementado (Sí/No)	Si. Miguel Santiago Roa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Hacer una iteración en la estructura de datos para iniciar a filtrarla (iterator)	O(n)
Añadir una oferta a la lista en caso en que esta cumpla con los requisitos (addLast)	O(1)
Agregar a la lista las empresas que no se repiten (append)	O(1)
Añadir información al diccionario según si la ciudad ya fue agregada o no.	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron hechas en una máquina con las siguientes especificaciones:

Procesadores	**AMD Ryzen 3 3250U with Radeon Graphics 2.60 GHz**
Memoria RAM	8 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador basado en x64 Windows 11

Las pruebas fueron realizadas usando como filtro los siguientes datos:

Nombre del país: PL Año inicial: 1900-00-00 Año final: 2024-00-00

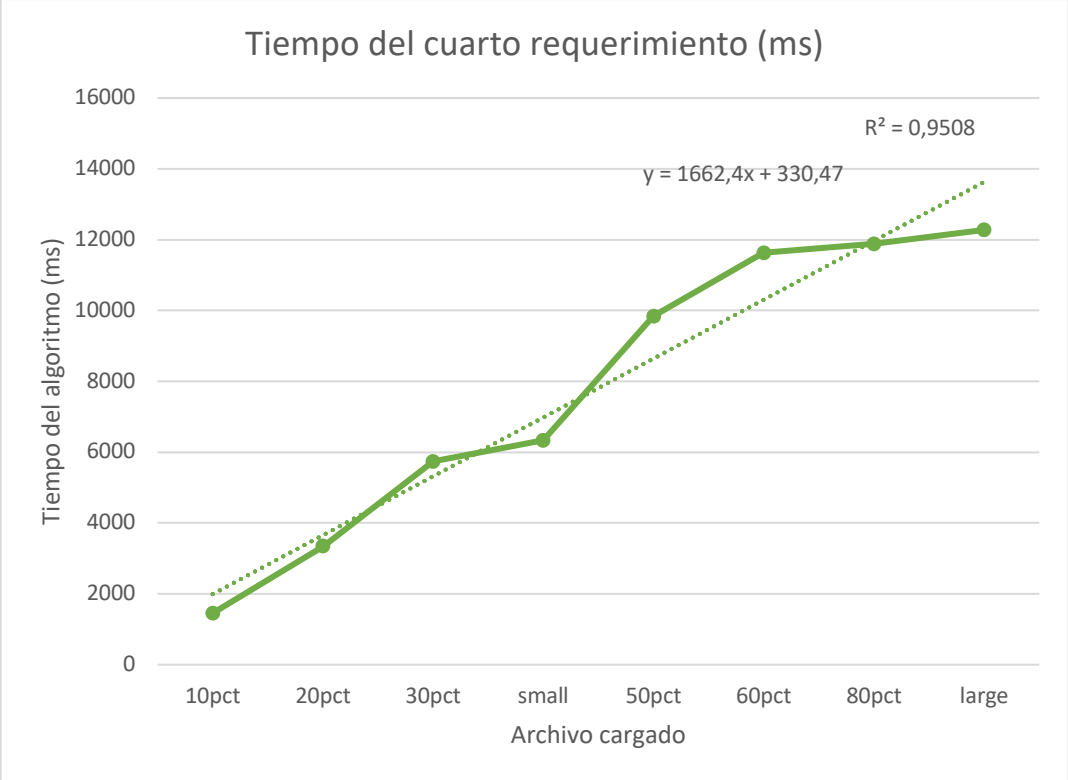
Entrada	**Tiempo (s)**
10pct	1453,3090999992564
20pct	3341,5531000001356
30pct	5735,3149999994785
small	6334,318400000222
50pct	9838,474899999797
60pct	11629,527799999341
80pct	11881,972399999388
large	12274,84499999974

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10pct	Dato1	1453.3090999992564
20pct	Dato2	3341.5531000001356
30pct	Dato3	5735.3149999994785
Small	Dato4	6334.318400000222
50pct	Dato5	9838.474899999797
60pct	Dato6	11629.527799999341
80pct	Dato7	11881.972399999388
large	Dato8	12274.84499999974

Graficas



Análisis

Nótese que el algoritmo efectivamente tiene un comportamiento muy parecido al de una función lineal O(n), lo cual se evidencia ya que entre más aumenta la cantidad de datos, el tiempo crece con una pendiente de forma lineal. Al hacer la regresión, se obtiene un R cuadrado muy cercano a 1, lo cual indica adicionalmente la gran similitud de este algoritmo con una función lineal. Esto es de esperar, ya que el algoritmo de mayor complejidad de este requerimiento es únicamente un for para recorrer todas las ofertas de trabajo (O(n)). Nótese que precisamente, entre más aumenta la cantidad de ofertas,

también aumenta el tiempo, pues justamente el rendimiento del requerimiento depende de cuantos datos tiene que iterar el for. De igual forma, los algoritmos de agregar un elemento a la lista o diccionario (no obstante sean O(1), se van a repetir n veces, ya que se encuentran adentro del for. Sin embargo, se evidencia que esto no afecta casi nada el tiempo de ejecución, puesto que sin importar que estos algoritmos estén (append, agregar al diccionario), siempre se van a repetir n veces al estar adentro del for.

Requerimiento <<3>>

```
def req_3(data_structs, empresa, fecha_inicial, fecha_final):
• La fecha inicial del periodo a consultar (con formato "%Y-%m-%d").
• La fecha final del periodo a consultar (con formato "%Y-%m-%d").
La respuesta esperada debe contener:
• Número total de ofertas.
• Número total de ofertas con experticia junior.
• Número total de ofertas con experticia mid.
• Número total de ofertas con experticia senior.
• El listado de ofertas de la empresa ordenados cronológicamente por fecha y país (v.gr. Para dos
ofertas con la misma fecha, el orden lo decide el país de forma alfabética). Donde para cada uno de
los elementos resultantes contendrá la siguiente información:
o Fecha de la oferta.
o Título de la oferta.
o Nivel de experticia requerido
o Ciudad de la empresa de la oferta
o País de la empresa de la oferta
o Tamaño de la empresa de la oferta
o Tipo de lugar de trabajo de la oferta.
o Disponible a contratar ucranianos (Verdadero o Falso).
Recomendaciones:
• Antes de empezar el desarrollo del requerimiento analice los archivos e identifique posibles valores
para datos como código de país, nivel de experticia, nombre de la empresa y fechas de consulta.
    """
    # TODO: Realizar el requerimiento 3
    jobs = data_structs["jobs"]
    ofertas_en_rango = lt.newList()

    fecha_inicial = convertir_fecha(fecha_inicial)
    fecha_final = convertir_fecha(fecha_final)
    n_ofertas_experticia = {"senior": 0, "mid": 0, "junior": 0}
    for job in lt.iterator(jobs):

        fecha1 = convertir_fecha(job["published_at"])

        if ((fecha_comparacion(fecha1, fecha_inicial)) and (fecha_comparacion(fecha_final, fecha1)) and (job["company_name"] == empresa)):
            lt.addLast(ofertas_en_rango, job)
            if job["experience_level"] == "senior":
                n_ofertas_experticia["senior"] += 1
            elif job["experience_level"] == "mid":
                n_ofertas_experticia["mid"] += 1
            elif job["experience_level"] == "junior":
                n_ofertas_experticia["junior"] += 1

    ofertas_en_rango_ordenadas = sa.sort(ofertas_en_rango, cmp_ofertas_by_pais_y_fecha)
    total_ofertas = lt.size(ofertas_en_rango)

    return total_ofertas, n_ofertas_experticia, ofertas_en_rango_ordenadas
```

Descripción

Este requerimiento va a hacer un recorrido sobre las ofertas de trabajo, para luego hacer un filtro adicional, ir añadiendo la información pertinente en un ARRAY nuevo, y con base en la información agregada, iniciar a calcular la cantidad de ofertas con nivel de experticia senior, mid y junior.

Entrada	La estructura de datos, la empresa específica con la que se va a hacer el filtro, y el rango de fechas.

Salidas	Total de ofertas que cumplen ese filtro, la cantidad de ofertas de cada experticia, y las ofertas ordenadas en dicho rango.
Implementado (Sí/No)	Si. Mattia Riccardi.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear un ARRAY vacío	Temporal: O(1)
Recorrer la estructura de datos de los trabajos	O(n)
Añadir al último en un ARRAY	O(1)
Hacer un ordenamiento de un ARRAY con shellSort	O(nlogn)
TOTAL	***O(nlogn)***

Pruebas Realizadas

Las pruebas realizadas fueron hechas en una máquina con las siguientes especificaciones:

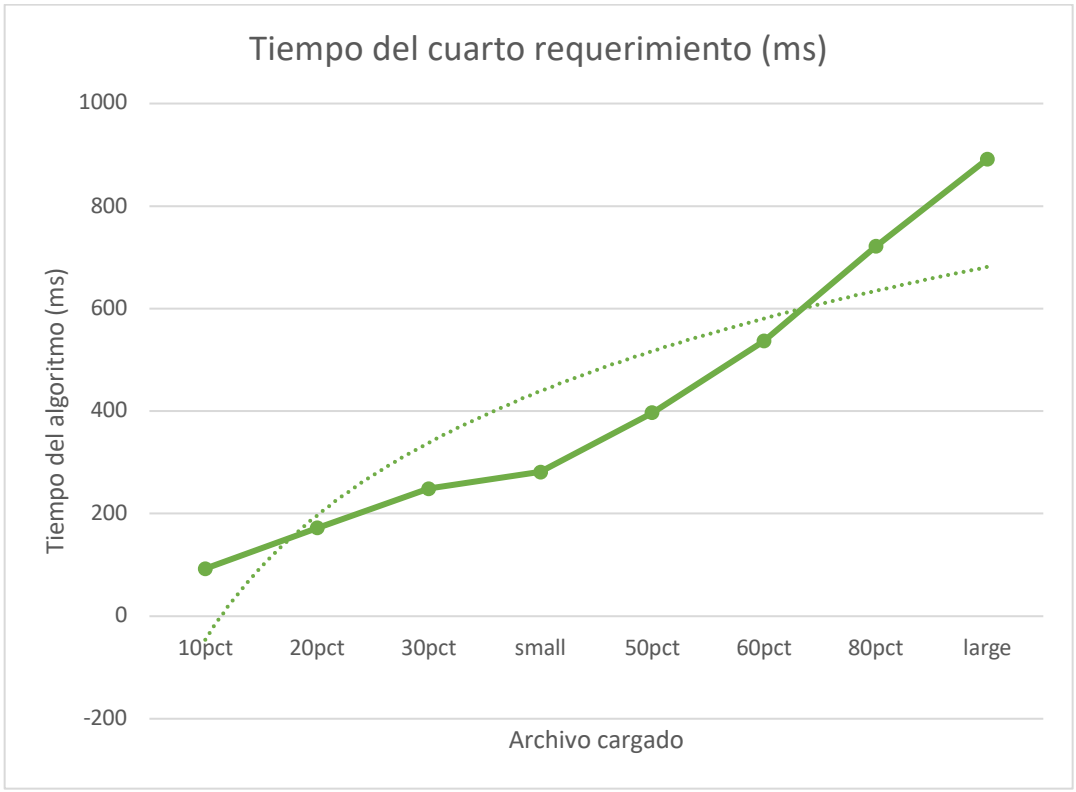
Procesadores	**AMD Ryzen 3 3250U with Radeon Graphics 2.60 GHz**
Memoria RAM	8 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador basado en x64 Windows 11

Las pruebas fueron realizadas usando como filtro los siguientes datos:

Nombre de la empresa: Softax Año inicial: 1900-00-00 Año final: 2024-00-00

Entrada	**Tiempo (s)**
10pct	92,8221
20pct	172,1165
30pct	249,0611
small	281,0687
50pct	396,5789
60pct	536,6073
80pct	721,4561
large	891,7756

Análisis


Tiempo del cuarto requerimiento (ms)
1000
800
600
400
200
0
-200
Tiempo del algoritmo (ms)
10pct
20pct
30pct
small
50pct
60pct
80pct
large
Archivo cargado