ANÁLISIS DEL RETO

Mariana Cediel, 202321548, m.cediel1@uniandes.edu.co

Juan Felipe López, 203220114, jf.lopez234@uniandes.edu.co

Carga de Datos

Descripción

Se llenan todas las estructuras de datos necesarias para la elaboración de todos los requerimientos solicitados.

La carga consta de 3 funciones: add_data_jobs, add_data_skills y add_data_employments_types.

ADD_DATA_JOBS:

```python
def add_data_jobs(data_structs, data):
    #Lleno el mapa de id
    mp.put(data_structs["mapa_id"], data["id"], data)

    #Lleno el mapa de jobs
    data["published_at"] = datetime.strptime(data["published_at"],"%Y-%m-%dT%H:%M:%S.%fZ")
    lista = cargar_mapa(om, data_structs["jobs"], data["published_at"], lt.newList())
    lt.addLast(lista, data)

    #Lleno el mapa del req 1
    data["published_at"] = datetime.strptime(datetime.strftime(data["published_at"],"%Y-%m-%d"), "%Y-%m-%d")
    lista = cargar_mapa(om, data_structs["map_req1"], data["published_at"], lt.newList())
    lt.addLast(lista, data)

    #Lleno mapa req 3
    mapa_paises = data_structs["map_req3"]
    pais = data["country_code"]
    experticia = data["experience_level"]
    mapa_experticia = cargar_mapa(mp, mapa_paises, pais, mp.newMap(numelements=4,
                                                                   maptype="PROBING"))
    arbol_ofertas = cargar_mapa(mp, mapa_experticia, experticia, om.newMap(omaptype="RBT"))
    lista = cargar_mapa(om, arbol_ofertas, data["published_at"], lt.newList())
    lt.addLast(lista, data)

    #Lleno mapa req 7
    mapa_anios = data_structs["map_req7"]
    anio = int(data["published_at"].year)
    valor = {"mapa_paises": mp.newMap(numelements=100),
             "total_ofertas": 0}
    diccionario_anio = cargar_mapa(mp, mapa_anios, anio, valor)
    mapa_paises = diccionario_anio["mapa_paises"]
    diccionario_anio["total_ofertas"] += 1
    diccionario_mapas = cargar_mapa(mp, mapa_paises, data["country_code"], newDict_req7())
    lista_jobs_experticia = cargar_mapa(mp, diccionario_mapas["experticia"], data["experience_level"], lt.newList("ARRAY_LIST"))
    lt.addLast(lista_jobs_experticia, data)
    lista_jobs_ubicacion = cargar_mapa(mp, diccionario_mapas["ubicacion"], data["workplace_type"], lt.newList("ARRAY_LIST"))
    lt.addLast(lista_jobs_ubicacion, data)

    #Lleno mapa req 4
    mapa_ciudades = data_structs["map_req4"]
    mapa_ubicacion =  cargar_mapa(mp,mapa_ciudades,data["city"],mp.newMap(numelements=4))
    lista_jobs = cargar_mapa(mp,mapa_ubicacion,data["workplace_type"],lt.newList(datastructure="ARRAY_LIST"))
    lt.addLast(lista_jobs,data)
```

- Llena el mapa de ids que se usará más adelante para completar la información de las ofertas con su salario mínimo y sus skills en los mapas de los demás requerimientos.
- Se carga el mapa de jobs para mostrar la carga de los datos de todas las ofertas. Es de este mapa que se obtienen las 3 ofertas más recientes y 3 más antiguas.
- Se carga el mapa del requerimiento 1.
- Se carga el mapa del requerimiento 3.
- Se carga el mapa del requerimiento 4.
- Se carga parte del mapa del requerimiento 7.

Entradas	• El data structs, donde se encuentran todos los mapas a llenar.

	• El data (cada una de las líneas del CSV de jobs).
Salidas	Esta función no tiene retorno. Sin embargo, se encarga de llenar los mapas con todos los datos necesarios para resolver todos los requerimientos
Implementado (Sí/No)	Se implementó por Mariana Cediel y Juan Felipe López

ADD_DATA_SKILLS:

```
156
157 def add_data_skills(data_structs, data):
158     #Calculo el salario mínimo de cada una de las ofertas y las agrego a la información de cada una de las ofertas.
159     job = me.getValue(mp.get(data_structs["mapa_id"], data["id"]))
160
161     if "skills" not in job.keys():
162         job["skills"] = lt.newList(datastructure="ARRAY_LIST")
163     else:
164         job["skills"] = job["skills"]
165     lt.addLast(job["skills"], data)
166
167     #Lleno mapa req 7
168     mapa_anios = data_structs["map_req7"]
169     anio = int(job["published_at"].year)
170     diccionario_anio = me.getValue(mp.get(mapa_anios, anio))
171     mapa_paises = diccionario_anio["mapa_paises"]
172     diccionario_mapas = me.getValue(mp.get(mapa_paises, job["country_code"]))
173     lista_jobs_habilidades = cargar_mapa(mp, diccionario_mapas["habilidad"], data["name"], lt.newList("ARRAY_LIST"))
174     lt.addLast(lista_jobs_habilidades, job)
175
```

- Halla las skills de cada oferta.
- Agrega el las skills a la información de todas las ofertas.
- Termina de llenar el mapa del requerimiento 7.

Entrada	• El data structs, donde se encuentran todos los mapas a llenar. • El data (cada una de las líneas del CSV de skills).
Salidas	Esta función no tiene retorno. Sin embargo, se encarga de llenar los mapas con la todos los datos necesarios para resolver los requerimientos
Implementado (Sí/No)	Se implementó por Mariana Cediel

ADD_DATA_EMPLOYMENTS_TYPES:

```
176
177 def add_data_employments_types(data_structs, data):
178     #Calculo las skills de cada una de las ofertas y las agrego a la información de cada una de las ofertas.
179     job = me.getValue(mp.get(data_structs["mapa_id"], data["id"]))
180     salario_min = data["salary_from"]
181     if salario_min != "":
182         salario_min = convertir_salario(float(salario_min), data["currency_salary"])
183         if "salary_from" not in job.keys():
184             job["salary_from"] = float("inf")
185         if salario_min < job["salary_from"]:
186             job["salary_from"] = salario_min
187     else:
188         salario_min = 0
189         job["salary_from"] = salario_min
190
191     #Lleno mapa req 6
192     arbol_fechas6 = data_structs["map_req6"]
193     arbol_salarios_minimos = cargar_mapa(om, arbol_fechas6, job["published_at"], om.newMap(omaptype="RBT"))
194     lista_jobs = cargar_mapa(om, arbol_salarios_minimos, salario_min, lt.newList("ARRAY_LIST"))
195     lt.addLast(lista_jobs, job)
196
```

- Halla el salario mínimo de cada oferta.
- Agrega el salario mínimo a la información de todas las ofertas.
- Llena el mapa del requerimiento 6.

Entradas	• El data structs, donde se encuentran todos los mapas a llenar. • El data (cada una de las líneas del CSV de employments types).
Salidas	Esta función no tiene retorno. Sin embargo, se encarga de llenar los mapas con la todos los datos necesarios para resolver los requerimientos.
Implementado (Sí/No)	Se implementó por Mariana Cediel

Análisis de complejidad

Pasos	**Complejidad**
Recorrer el CSV de jobs para implementar la función add_data_jobs	O(n)
Recorrer el CSV de skillspara implementar la función add_data_skills	O(n)
Recorrer el CSV de employments types para implementar la función add_data_employments_types	O(n).
TOTAL	***O(n)***

Pruebas Realizadas

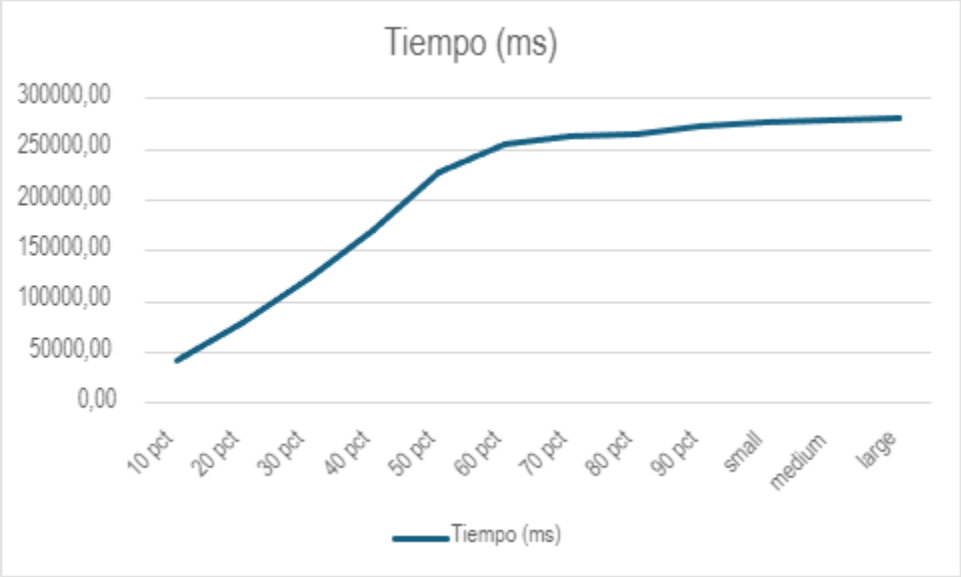
Computador donde se probó

Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

Tablas de datos

Entrada	**Tiempo (ms)**
10 pct	41786,32
20 pct	80400,48
30 pct	122514,17
40 pct	170819,01
50 pct	227401,76
60 pct	254570,97
70 pct	263658,32
80 pct	265821,96
90 pct	272486,33
small	276213,48
medium	278262,06
large	281071,34

Graficas



Análisis

Es la función inicial del programa. Sin esta, no sería posible resolver los demás requerimientos. Podemos observar como el tiempo de ejecución de la misma aumenta a medida que aumenta la cantidad de los datos utilizados.

Requerimiento <<1>>

Descripción

Requerimiento que permite conocer las ofertas laborales publicadas durante un intervalo de fechas específico.

```
252
253  def req_1(data_structs, fecha_inicial, fecha_final):
254      """
255      Función que soluciona el requerimiento 1
256      """
257      # TODO: Realizar el requerimiento 1
258      mapa_fechas = data_structs["map_req1"]
259      fecha_inicial = datetime.strptime(fecha_inicial,"%Y-%m-%d")
260      fecha_final = datetime.strptime(fecha_final,"%Y-%m-%d")
261
262      lista_valores = om.values(mapa_fechas, fecha_final, fecha_inicial)
263      lista = convertir_lista_de_listas(lista_valores)
264
265      total_ofertas = lt.size(lista)
266      if total_ofertas > 10:
267          lista = first_last(lista, 5)
268
269      return total_ofertas, lista
270
```

Entradas	• El data structs, donde se encuentran todos los mapas. • Una fecha inicial en formato %Y-%m-%d. • Una fecha final en formato %Y-%m-%d.
Salidas	Esta función retorna una tupla con dos elementos: 1. El total de ofertas publicadas entre el intervalo de fechas ingresado por el usuario.

	2. Una lista con dichas ofertas. De haber más de 10 ofertas, la lista contiene únicamente las 5 más recientes y antiguas ordenadas de más a menos reciente.
Implementado (Sí/No)	Se implementó por Mariana Cediel.

Análisis de complejidad

Pasos	**Complejidad**
Hallar la lista de listas que conforman todas las ofertas publicadas en el intervalo de fechas ingresado por el usuario.	O(log(n))
Convertir la lista de listas en una única lista.	O(n*m)
En caso de que la lista obtenida tenga más de 10 elementos, se restringe a una lista con las 5 ofertas más recientes y las 5 más antiguas. En este caso, no es necesario hacer un sort de ningún tipo, pues el árbol inicial fue ordenado por fechas de mayor a menor. Este proceso lo hicimos mediante una función llamada first_last.	O(10)
TOTAL	***O(n*m)***

Pruebas Realizadas

Computador donde se probó

Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

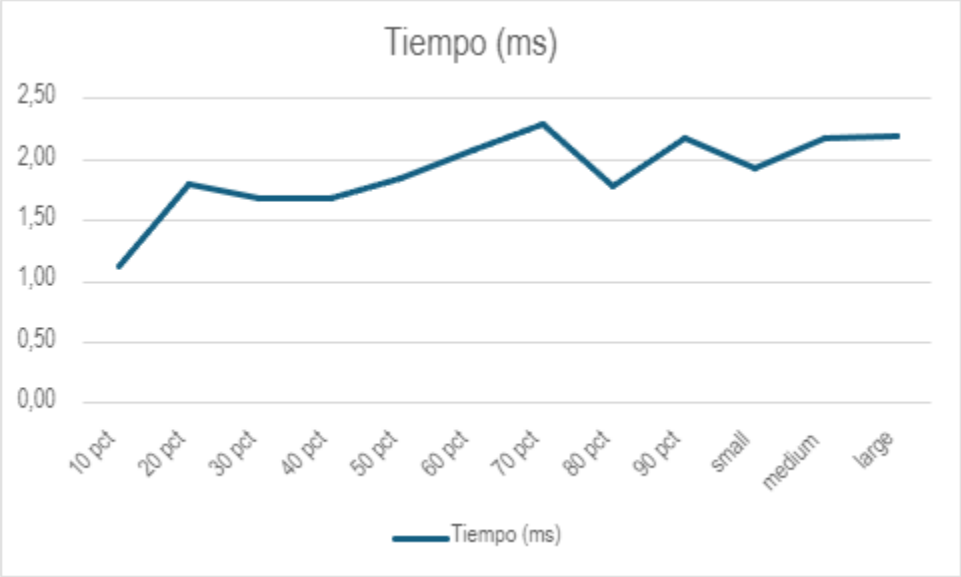
Tablas de datos

Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Fecha inicial: 2022-04-13
- Fecha final: 2022-04-14

Entrada	**Tiempo (ms)**
10 pct	1,13
20 pct	1,80
30 pct	1,68
40 pct	1,68
50 pct	1,85
60 pct	2,09
70 pct	2,30
80 pct	1,79
90 pct	2,18
small	1,93
medium	2,18
large	2,20

Graficas



Análisis

Aunque el tiempo de ejecución no pareció ser de ningún modo proporcional a la cantidad de datos, podemos ver como la complejidad temporal es casi constante. No varía de gran manera.

Requerimiento <<3>>

Descripción

Requerimiento que permite consultar las N ofertas laborales más recientes para un país y que requieran un nivel de experiencia específico.

```
def req_3(data_structs, numero_ofertas, codigo_pais, experticia):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    mapa_paises = data_structs["map_req3"]

    mapa_pais_exp = me.getValue(mp.get(mapa_paises, codigo_pais))
    arbol_ofertas_pais_exp = me.getValue(mp.get(mapa_pais_exp, experticia))

    lista_valores = om.valueSet(arbol_ofertas_pais_exp)
    lista_final = convertir_lista_de_listas(lista_valores)
    total_ofertas = lt.size(lista_final)

    merg.sort(lista_final, sort_date_salario)
    if total_ofertas > numero_ofertas:
        lista_final = first(lista_final, numero_ofertas)

    return total_ofertas, lista_final
```

Entradas	• El data structs, donde se encuentran todos los mapas. • El número de ofertas a consultar. • El código del país a consultar. • El nivel de experticia a consultar.
Salidas	Esta función retorna una tupla con dos elementos:

	1. El total de ofertas publicadas con el código del país y el nivel de experticia ingresados por el usuario. 2. Una lista con dichas ofertas. La lista contiene el número de ofertas que ingresó el usuario. Estas son las ofertas publicadas más recientes.
Implementado (Sí/No)	Se implementó por Mariana Cediel.

Análisis de complejidad

Pasos	**Complejidad**
Hallar el mapa de las ofertas filtradas por experticia del pais por el código del país ingresado por el usuario.	O(1)
Hallar el árbol de las ofertas del país ingresado, que requieren el nivel de experticia ingresado por parámetro ordenadas por fecha de publicación.	O(1)
Hallar la lista de listas que conforman todas las ofertas publicadas en el intervalo de fechas ingresado por el usuario.	O(log(n))
Convertir la lista de listas en una única lista.	O(n*m)
Se ordena la lista obtenida por fecha y salario de mayor a menor.	O(n log(n))
En caso de que el número de elementos de la lista encontrada sea mayor a el número de ofertas ingresado por el usuario, se restringe a una lista que tenga únicamente el número de ofertas ingresado por parámetro.	O(x), siendo x el número ingresado por el usuario por parámetro.
TOTAL	***O(n log (n))***

Pruebas Realizadas

Computador donde se probó

Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

Tablas de datos

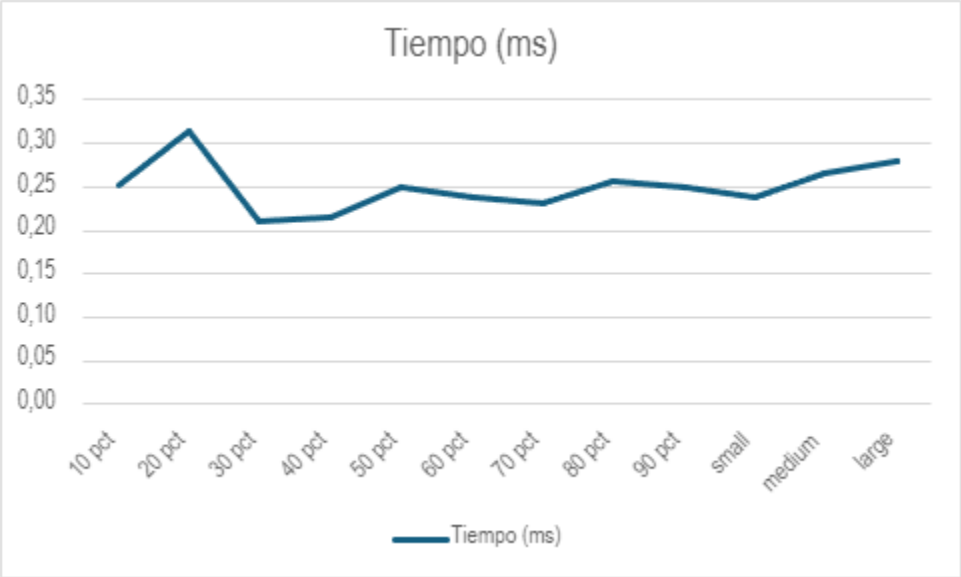
Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Numero de ofertas: 10
- Codigo del pais: AE
- Nivel de experticia: senior

Entrada	**Tiempo (ms)**
10 pct	0,25
20 pct	0,32
30 pct	0,21
40 pct	0,22
50 pct	0,25
60 pct	0,24

70 pct	0,23
80 pct	0,26
90 pct	0,25
small	0,24
medium	0,27
large	0,28

Graficas



Análisis

Esta fue la función con menos tiempo de ejecución. Su tiempo nunca fue mayor a los 0,35 milisegundos. La prueba con el 20 porciento se demoró mucho más que las anteriores. Sin embargo, siguió siendo una función muy eficiente.

Requerimiento <<4>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	numero_ofertas, ciudad, ubicacion
Salidas	Las ofertas dependiendo de la ciudad y la ubicacion que el usuario ingrese
Implementado (Sí/No)	Si se implementó y quien lo hizo. Juan Felipe López

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1. mapa_ciudades = data_structs["map_req4"]	O(1)
Paso 2 mapa_ubicacion = me.getValue(mp.get(mapa_ciudades, ciudad))	O(1)
Paso 3 lista_jobs = me.getValue(mp.get(mapa_ubicacion, ubicacion))}	O(1)
Paso 4 merg.sort(lista_jobs,sort_date_salario)	***O(n Log N)***
Paso 5 total_ofertas = lt.size(lista_jobs)	***O(1)***
if total_ofertas>numero_ofertas:	***O(1)***
lista_jobs= lt.subList(lista_jobs,1,numero_ofertas)	***O(1)***
return total_ofertas, lista_jobs	***O(1)***
Total	***O(LogN)***

Pruebas Realizadas

AMD Ryzen 7 6800HS with Radeon Graphics

Memoria Ram 16 GB

Windows 11

Entrada	**Tiempo (ms)**
small	0.02
5 pct	0.1
10 pct	1.10
20 pct	2.22
30 pct	4.32
50 pct	6.21
80 pct	11.98
large	24.6

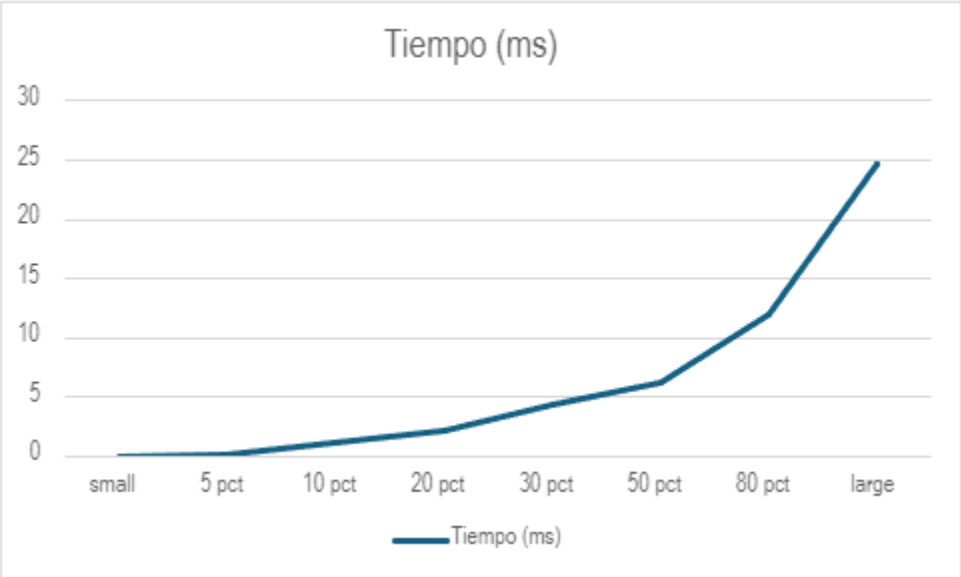
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**

small	Dato1	0.02
5 pct	Dato2	0.1
10 pct	Dato3	1.10
20 pct	Dato4	2.22
30 pct	Dato5	4.32
50 pct	Dato6	6.21
80 pct	Dato7	11.98
large	Dato8	24.6

Graficas



Análisis

La función req_4 recupera una lista de ofertas basadas en un código anónimo, las ordena cronológicamente y las filtra según las habilidades requeridas, finalmente devuelve las top n ofertas

Requerimiento <<6>>

Descripción

Requerimiento que permite consultar las N ciudades que presentan la mayor cantidad de ofertas laborales publicadas entre un par de fechas y que estén en un rango de salario ofertado.

```python
def req_6(data_structs, numero_ciudades, fecha_inicial, fecha_final, salario_min_inicial, salario_min_final):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    arbol_fechas = data_structs["map_req6"]
    fecha_inicial = datetime.strptime(fecha_inicial,"%Y-%m-%d")
    fecha_final = datetime.strptime(fecha_final,"%Y-%m-%d")

    lista_fecha_salario = lt.newList(datastructure="ARRAY_LIST")
    mapa_ciudades = mp.newMap(numelements=1000,
                              prime = 109345121,
                              maptype="CHAINING",
                              loadfactor=1)
    lista_arboles_salarios = om.values(arbol_fechas, fecha_inicial, fecha_final)
    ciudad_mayor = None
    max_ofertas = 0
    for arbol_salarios in lt.iterator(lista_arboles_salarios):
        lista_valores = om.values(arbol_salarios, salario_min_inicial, salario_min_final)
        lista_jobs = convertir_lista_de_listas(lista_valores)
        for job in lt.iterator(lista_jobs):
            lt.addLast(lista_fecha_salario, job)
            lista_ofertas_ciudad = cargar_mapa(mp, mapa_ciudades, job["city"], lt.newList(datastructure="ARRAY_LIST"))
            lt.addLast(lista_ofertas_ciudad, job)
            if lt.size(lista_ofertas_ciudad) > max_ofertas:
                max_ofertas = lt.size(lista_ofertas_ciudad)
                ciudad_mayor = job["city"]
```

```python
    total_ofertas = lt.size(lista_fecha_salario)
    lista_ciudades = mp.keySet(mapa_ciudades)
    total_ciudades = lt.size(lista_ciudades)
    merg.sort(lista_ciudades, sort_city)
    if lt.size(lista_ciudades) > numero_ciudades:
        lista_ciudades = first(lista_ciudades, numero_ciudades)

    lista_jobs_mayor = me.getValue(mp.get(mapa_ciudades, ciudad_mayor))
    merg.sort(lista_jobs_mayor, sort_date_salario)
    if lt.size(lista_jobs_mayor) > 10:
        lista_jobs_mayor = first_last(lista_jobs_mayor, 5)

    return total_ofertas, total_ciudades, lista_ciudades, lista_jobs_mayor
```

Entradas	• El data structs, donde se encuentran todos los mapas. • Una fecha inicial en formato %Y-%m-%d. • Una fecha final en formato %Y-%m-%d. • Rango inferior de salario mínimo. • Rango superior de salario mínimo.
Salidas	Esta función retorna una tupla con 4 elementos: 1. El total de ofertas publicadas en el intervalo de fechas elegico, cuyo salario mínimo se encuentra entre el rango ingresado por el usuario. 2. El total de ciudades donde se publicaron ofertas que cumplan estas condiciones. 3. La lista de las ciudades en que se publicaron ofertas ordenadas alfabéticamente. 4. La lista de ofertas publicadas restringida a las 5 más recientes y las 5 más antiguas en caso de ser inicialmnete una lista dde más de 10 elementos.
Implementado (Sí/No)	Se implementó por Mariana Cediel.

Análisis de complejidad

Pasos	**Complejidad**
Hallar la lista de árboles ordenados por salarios de las ofertas que se encuentran en el rango de fechas ingresado por el usuario.	O(log(n))
Recorrer la lista de árboles para encontrar las ofertas que cumplen ambas condiciones.	O(n*m)
Hallar las ofertas de cada árbol de salarios que se encuentren en el rango de salarios ingresado por parámetro.	O(log(n))

Convertir la lista de listas en una única lista.	O(n*m)
Llenar la lista de todas las ofertas que cumplen con las condiciones de fecha y salario mínimo y el mapa de todas las ciudades que tienen ofertas que cumplen.	O(n)
Ordenar la lista de las ciudades a través de un merge sort.	O(n log(n))
Si hay más ciudades que el número de ciudades ingresado por parámetro, se restringe la lista a una con tan solo la cantidad de elementos que pidió el usuario.	O(x), siendo x el número ingresado por parámetro.
Ordenar la lista de todas las ofertas por fecha y salario usando un merge sort.	O(n log(n))
En caso de que la lista obtenida tenga más de 10 elementos, se restringe a una lista con las 5 ofertas más recientes y las 5 más antiguas. En este caso, no es necesario hacer un sort de ningún tipo, pues el árbol inicial fue ordenado por fechas de mayor a menor. Este proceso lo hicimos mediante una función llamada first_last.	O(10)
TOTAL	***O(n*m)***

Pruebas Realizadas

Computador donde se probó

Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

Tablas de datos

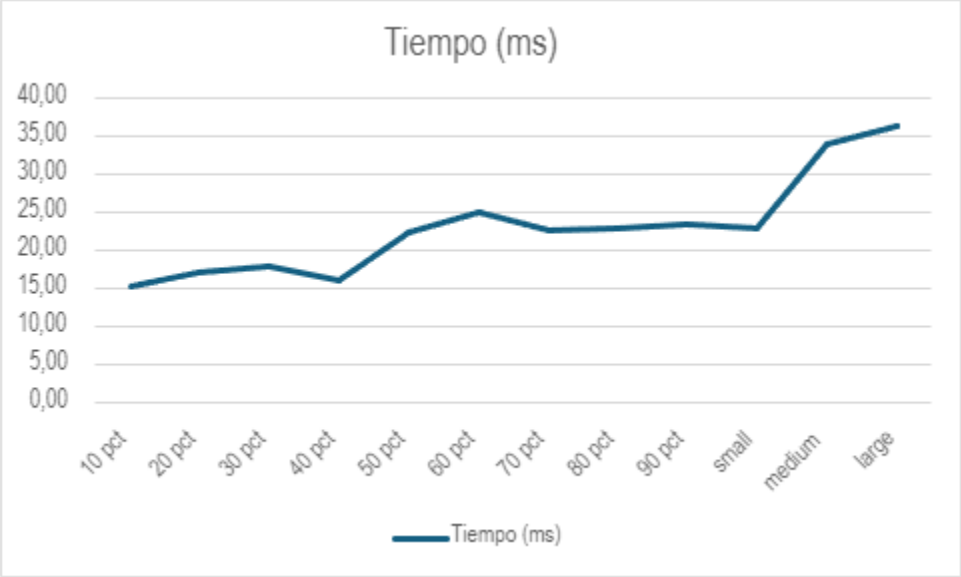
Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Numero de ciudades: 10
- Fecha inicial: 2022-04-13
- Fecha final: 2022-04-14
- Límite inferior del salario mínimo en dólares: 2000
- Límite superior del salario mínimo en dólares: 5000

Entrada	**Tiempo (ms)**
10 pct	15,38
20 pct	17,32
30 pct	18,03
40 pct	16,10
50 pct	22,51
60 pct	25,04
70 pct	22,85
80 pct	23,01
90 pct	23,50

small	22,94
medium	34,17
large	36,32

Graficas



Análisis

En este requerimiento podemos observar como el tiempo de ejecución aumenta a medida que aumenta la cantidad de datos.

Requerimiento <<7>>

Descripción

Requerimiento que permite contabilizar las ofertas laborales publicadas para un país y un año específico según alguna propiedad de interés como lo son el nivel de experticia requerido, el tipo de ubicación del trabajo, o habilidad específica.

```python
def req_7(data_structs, anio, codigo_pais, propiedad):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    mapa_anios = data_structs["map_req7"]
    diccionario_anio = me.getValue(mp.get(mapa_anios, anio))
    mapa_paises = diccionario_anio["mapa_paises"]
    diccionario_mapas = me.getValue(mp.get(mapa_paises, codigo_pais))
    mapa_propiedad = diccionario_mapas[propiedad]

    total_ofertas_anio = diccionario_anio["total_ofertas"]
    total_ofertas_propiedad = 0
    x = []
    y = []
    valor_min = float("inf")
    valor_max = 0
    lista_final = lt.newList("ARRAY_LIST")
    for llave in lt.iterator(mp.keySet(mapa_propiedad)):
        lista_jobs_propiedad = me.getValue(mp.get(mapa_propiedad, llave))
        total_ofertas_propiedad += lt.size(lista_jobs_propiedad)
        x.append(llave)
        y.append(lt.size(lista_jobs_propiedad))
        if lt.size(lista_jobs_propiedad) > valor_max:
            valor_max = lt.size(lista_jobs_propiedad)
        if lt.size(lista_jobs_propiedad) < valor_min:
            valor_min = lt.size(lista_jobs_propiedad)
        lt.addLast(lista_final, lista_jobs_propiedad)
    fig,ax = plt.subplots()
    ax.barh(x, y)
    plt.show()
    lista_final = convertir_lista_de_listas(lista_final)

    if lt.size(lista_final) > 10:
        lista_final = first_last(lista_final, 5)

    return total_ofertas_anio, total_ofertas_propiedad, (valor_min, valor_max), lista_final
```

Entradas	• El data structs, donde se encuentran todos los mapas. • El año. • El código del país. • La propiedad por analizar
Salidas	Esta función retorna una tupla con cuatro elementos: 1. El total de ofertas publicadas en el año escogido por el usuario. 2. El total de ofertas utilizadas para crear el gráfico de la propiedad que se escogió. 3. Una tupla con dos elementos: a. El valor mínimo representado en el gráfico. b. El valor máximo representado en el gráfico. 4. La lista final de ofertas a imprimir.
Implementado (Sí/No)	Se implementó por Mariana Cediel.

Análisis de complejidad

Pasos	**Complejidad**
Hallar el diccionario de años en el mapa de años.	O(1)
Hallar el mapa de los países dentro del diccionario de años.	O(1)
Hallar el diccionario de las 3 propiedades dentro del mapa del país ingresado por parámetro.	O(1)
Hallar el mapa de la propiedad en el diccionario utilizando la propiedad ingresada por parámetro por el usuario.	O(1)
Encontrar el total de ofertas en el año escogido por el usuario.	O(1)
Llenar la lista final de ofertas y los ejes de la gráfica.	O(n)
Convertir la lista de listas final en una única lista.	O(n*m)
En caso de que la lista obtenida tenga más de 10 elementos, se restringe a una lista con las 5 ofertas más recientes y las 5 más antiguas. En este caso, no es necesario hacer un sort de ningún tipo, pues el árbol inicial	O(10)

fue ordenado por fechas de mayor a menor. Este proceso lo hicimos mediante una función llamada first_last.	
TOTAL	***O(n*m)***

Pruebas Realizadas

Computador donde se probó

Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

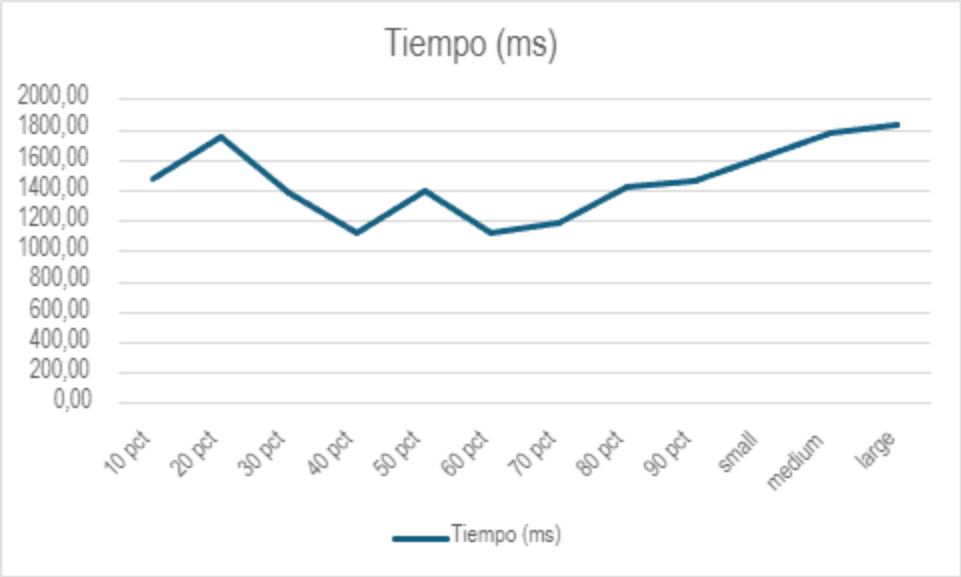
Tablas de datos

Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Año: 2023
- Código de país: US
- Propiedad: experticia

Entrada	**Tiempo (ms)**
10 pct	1481,81
20 pct	1756,78
30 pct	1388,91
40 pct	1125,69
50 pct	1398,65
60 pct	1121,00
70 pct	1184,07
80 pct	1430,54
90 pct	1472,52
small	1627,68
medium	1781,00
large	1834,00

Graficas



Análisis

Este requerimiento se comportó de manera peculiar a la hora de realizar las pruebas. Desde el 20 porciento hasta el 60 porciento se puede decir que se observó una disminución en el tiempo de ejecucuión. Sin embargo, de ahí para adelante, este solo aumentó.

Requerimiento <<8>>

Descripción

Requerimiento que realiza mapas interactivos de los demás requerimientos.

```python
def req_8(data_structs, requerimiento, fecha_inicial, fecha_final, numero_ofertas, codigo_pais, experticia, numero_ciudades, salario_min_inicial, salario_min_final, anio, propied
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    if requerimiento == 1:
        tupla = req_1(data_structs, fecha_inicial, fecha_final)
        lista_final = tupla[1]
    elif requerimiento == 3:
        tupla = req_3(data_structs, numero_ofertas, codigo_pais, experticia)
        lista_final = tupla[1]
    elif requerimiento == 6:
        tupla = req_6(data_structs, numero_ciudades, fecha_inicial, fecha_final, salario_min_inicial, salario_min_final)
        lista_final = tupla[3]
    elif requerimiento == 7:
        tupla = req_7(data_structs, anio, codigo_pais, propiedad)
        lista_final = tupla[3]

    m = folium.Map([23, 25], zoom_start=2)

    for oferta in lt.iterator(lista_final):
        info = "published at: " + datetime.strftime(oferta["published_at"],"%Y-%m-%d") + " - title: " + oferta["title"] + " - company name: " + oferta["company_name"] + " - coun
        popup = folium.Popup(info, min_width=150, max_width=150)
        folium.Marker(location=[oferta["latitude"], oferta["longitude"]],
                      tooltip="Click me!",
                      popup= popup,
                      icon=folium.Icon(color="red")).add_to(m)
    m.save("map.html")
    return m
```

Entradas	• El data structs, donde se encuentran todos los mapas. • Una fecha inicial en formato %Y-%m-%d. • Una fecha final en formato %Y-%m-%d. • Un número de ofertas. • Un código de país. • Un nivel de experticia.

	• Un número de ciudades. • Un límite inferior de salario mínimo. • Un límite superior de salario mínimo. • Un año. • Una propiedad.
Salidas	Esta función retorna un mapa.
Implementado (Sí/No)	Se implementó por Mariana Cediel.

Análisis de complejidad

Pasos	**Complejidad**
En caso de escogerse el requerimiento 1.	O(n*m)
En caso de escogerse el requerimiento 3.	O(n log(n))
En caso de escogerse el requerimiento 6.	O(n*m)
En caso de escogerse el requerimiento 7.	O(n*m)
Se ponen los marcadores en el mapa.	O(n)
TOTAL	***O(n*m) o O(n log(n))***

Pruebas Realizadas

Computador donde se probó

Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

Tablas de datos

PARA REQ 1

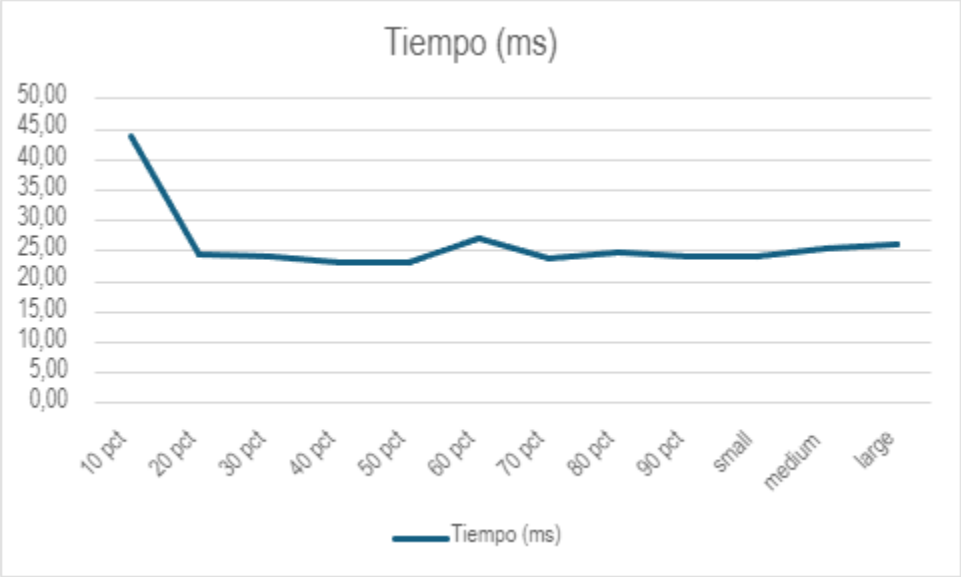
Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Fecha inicial: 2022-04-13
- Fecha final: 2022-04-14

Entrada	**Tiempo (ms)**
10 pct	43,85
20 pct	24,60
30 pct	24,30
40 pct	23,11
50 pct	23,21
60 pct	27,26
70 pct	23,76
80 pct	24,87
90 pct	24,00
small	24,26
medium	25,34

large	26,00

Graficas



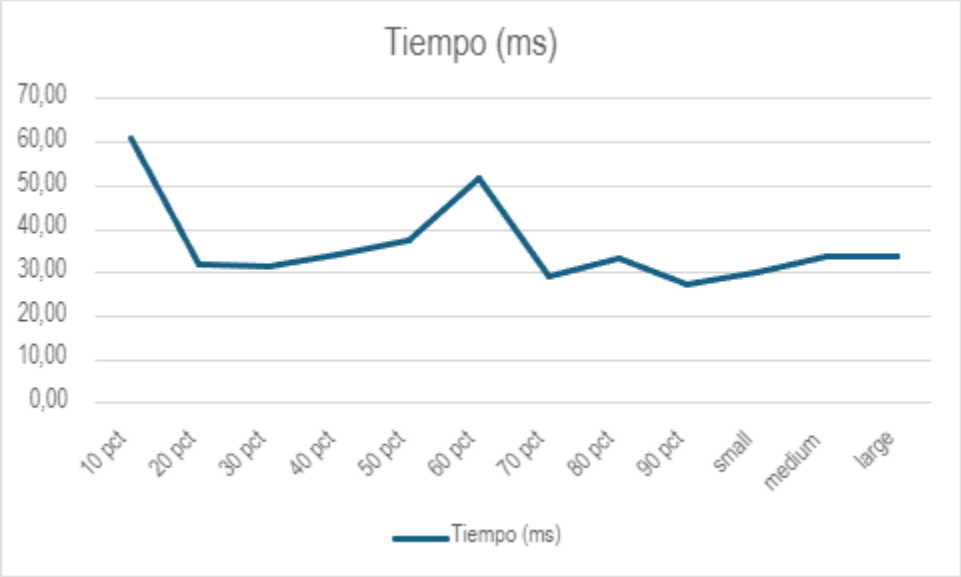
PARA REQ 3

Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Numero de ofertas: 15
- Código de país: US
- Experticia: senior

Entrada	**Tiempo (ms)**
10 pct	61,04
20 pct	32,15
30 pct	31,36
40 pct	34,36
50 pct	37,40
60 pct	51,84
70 pct	29,26
80 pct	33,22
90 pct	27,57
small	30,34
medium	33,80
large	34,00

Graficas



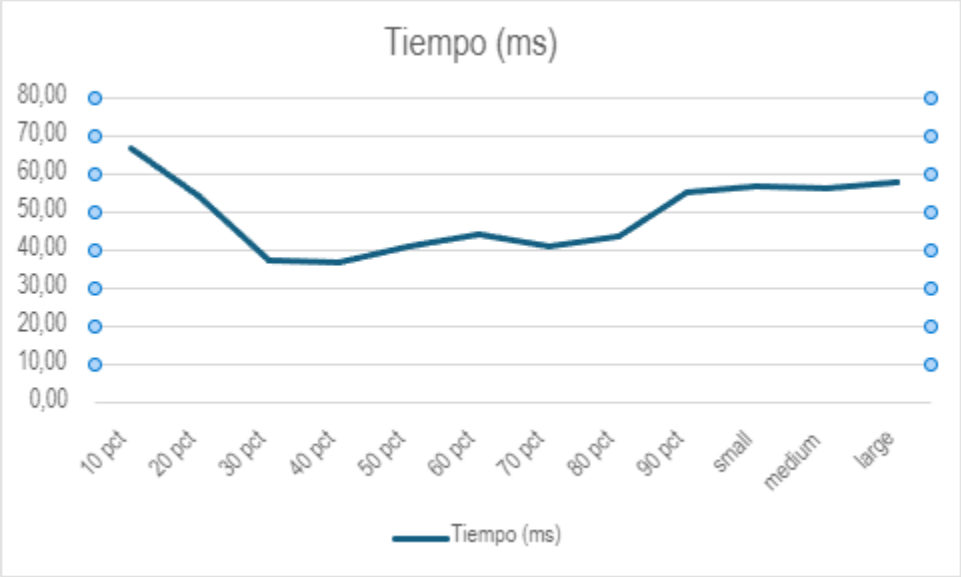
PARA REQ 6

Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Numero de ciudades: 10
- Fecha inicial: 2022-04-13
- Fecha final: 2022-04-14
- Límite inferior salario mínimo: 2000
- Límite superior salario mínimo: 5000

Entrada	**Tiempo (ms)**
10 pct	66,91
20 pct	54,34
30 pct	37,79
40 pct	37,23
50 pct	41,18
60 pct	44,38
70 pct	41,41
80 pct	44,05
90 pct	55,63
small	57,23
medium	56,33
large	58,21

Graficas



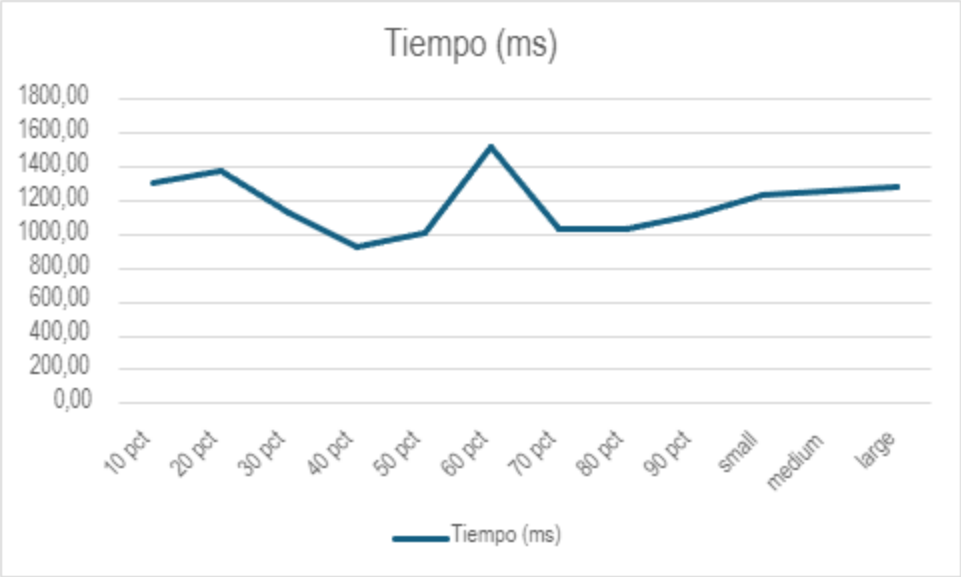
PARA REQ 7

Las pruebas se hicieron ingresando los siguientes datos por parámetro:

- Año: 2023
- Código del país: US
- Propiedad: experticia

Entrada	**Tiempo (ms)**
10 pct	1310,58
20 pct	1382,28
30 pct	1125,71
40 pct	926,37
50 pct	1008,91
60 pct	1524,76
70 pct	1029,84
80 pct	1033,90
90 pct	1118,07
small	1234,09
medium	1266,00
large	1279,23

Graficas



Análisis

Este es un requerimiento que depende estrictamente del requerimiento que esté representando.