ANÁLISIS DEL RETO

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

Juan Felipe López, 202320114, jf.lopez234@uniandes.edu.co

Requerimiento <<1>>

Descripción

Lista las N ofertas de trabajo más recientes ofrecidas en un país filtrando por el nivel de experticia del puesto

```
def req_1(data_structs, ofertas, codigo_pais, experticia):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    mapa_cod = data_structs["map_req1"]
    map_content = me.getValue(mp.get(mapa_cod, codigo_pais))
    total_ofertas_pais =  lt.size(map_content["junior"]) + lt.size(map_content["mid"]) + lt.size(map_content["senior"])
    rq1 = map_content[experticia]
    num_ofertas = lt.size(map_content[experticia])

    #ordeno la lista de ofertas por fecha, en orden cronológico
    merg.sort(rq1, sort_criteria1)
    if num_ofertas > ofertas:
        rq1 = lt.subList(map_content[experticia], lt.size(map_content[experticia]) - ofertas + 1, ofertas)
    for oferta in lt.iterator(rq1):
        oferta.pop("street")
        oferta.pop("address_text")
        oferta.pop("marker_icon")
        oferta.pop("company_url")
        oferta.pop("remote_interview")
        oferta.pop("id")
        oferta.pop("display_offer")

    return total_ofertas_pais, num_ofertas, rq1
```

Entrada	Los parámetros necesarios para obtener la salida esperada fueron la estructura creada en el modelo para el requerimiento, el número de ofertas a listar, el código del país y el nivel de experticia deseado.
Salidas	La salida de este requerimiento es una tupla de 3 elementos. 1. El total de ofertas de trabajo ofrecidas según el país. 2. El total de ofertas de trabajo ofrecidas según la condición (junior, mid, senior o indiferente). 3. Una lista TAD que contiene las últimas N ofertas de trabajo con el nivel de experticia seleccionado en el país seleccionado ordenada cronológicamente. Cada una de las ofertas es un diccionario que contiene la siguiente información: • Fecha de publicación de la oferta • Título de la oferta

	• Nombre de la empresa de la oferta • Nivel de experticia de la oferta (es el mismo del filtro) • País de la empresa de la oferta • Ciudad de la empresa de la oferta • Tamaño de la empresa de la oferta • Tipo de ubicación de trabajo (remote, partialy, remote, office) • Disponible a contratar ucranianos (Verdadero o Falso)
Implementado (Sí/No)	Se implementó por Mariana Cediel y Juan Felipe López

Análisis de complejidad

Pasos	**Complejidad**
Encontrar el diccionario de experticias del mapa por su código de país.	O(1)
Encontrar la lista de ofertas de la experticia en el país.	O(1)
Ordenar la sub lista de las N ofertas más recientes.	O(n log (n)).
Recorrer la lista ordenada para eliminar la información no pedida.	O(n)
TOTAL	***O(n log(n))***

Este código funciona a partir de un mapa cuyas llaves son códigos de países y sus valores correspondientes diccionarios que clasifican las ofertas (del país de la llave) por la experticia de cada una. Entonces, acceder al diccionario de las ofertas de un país tiene complejidad O(1), y, a su vez, acceder a la lista de ofertas de una experticia dada es también O(1).

Se utiliza merge sort para ordenar la lista de ofertas cronológicamente. Este tipo de sort usa la técnica de “divide y conquistarás” y tiene una complejidad temporal de O(n log(n)).

Finalmente recorremos la lista ordenada y eliminamos de cada oferta la información no solicitada. Es necesario recorrer toda la lista para eliminar esta información de todas las ofertas. Tiene complejidad temporal O(n).

Pruebas Realizadas

Se realizó la prueba del funcionamiento de este requerimiento con:

- Ofertas = 20
- Codigo_pais = US
- Experticia = senior

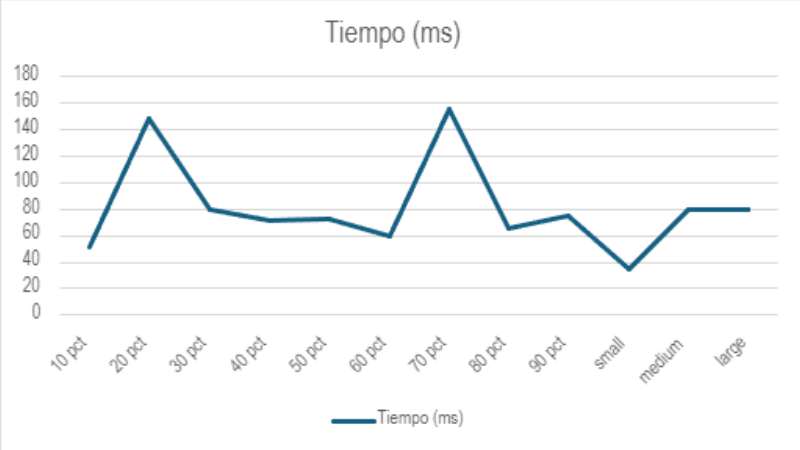
Computador donde se probó

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	51.635
20 pct	148.054
30 pct	79.631
40 pct	71.703
50 pct	72.570
60 pct	60.703
70 pct	156.327
80 pct	66.877
90 pct	75.702
small	35.023
medium	80.258
large	80.269

Graficas



El total de ofertas en US es: 447
El total de ofertas ofrecidas en el pais para la condición senior es 301

title	city	country_code	workplace_type	company_name	company_size	experience_level	published_at	open_to_hire_ukrainians
Cloud Infrastructure Architect	Woodbridge Township	US	remote	Intuitive Cloud	300	senior	2023-05-31T13:26:00.000Z	True
DevSecOps Architect	Woodbridge Township	US	remote	Intuitive.Cloud	300	senior	2023-05-31T13:36:00.000Z	True
Cloud Infrastructure Architect	Woodbridge Township	US	remote	Intuitive Cloud	300	senior	2023-06-01T10:19:00.000Z	True
Fullstack Python Developer (Django + JS)	Clackamas	US	remote	Thinkspace	20	senior	2023-06-06T16:31:59.008Z	False
Senior Full-Stack Developer	San Francisco	US	remote	Awesomic	20	senior	2023-06-08T14:29:48.183Z	True
Backend Engineer	New York	US	remote	All Inspire Health Inc.	30	senior	2023-06-11T16:04:35.072Z	True
Senior DevOps Engineer	Yorktown Heights	US	remote	CouponFollow	Undefined	senior	2023-06-12T20:21:00.000Z	False
Senior Data Scientist	Yorktown Heights	US	remote	CouponFollow	Undefined	senior	2023-06-14T09:47:00.000Z	True
DevOps (relocation)	San Francisco	US	partly_remote	Farel	23	senior	2023-06-20T11:06:00.000Z	True
Senior Backend Engineer	Austin	US	remote	SerpApi, LLC	Undefined	senior	2023-07-17T11:04:00.000Z	True
Creative Director	Austin	US	remote	SerpApi, LLC	Undefined	senior	2023-07-17T11:12:00.000Z	True
Senior UI/UX Designer	Austin	US	remote	SerpApi, LLC	Undefined	senior	2023-07-17T11:19:00.000Z	True
SAP HCM Consultant	Warszawa	US	remote	Simplicity Recruitment	20	senior	2023-08-02T14:54:06.799Z	True
Senior Fullstack Developer	Austin	US	remote	Plenti	20	senior	2023-08-08T10:49:00.000Z	True
Graphic designer	Warszawa	US	remote	Blix Inc.	Undefined	senior	2023-08-30T08:47:00.000Z	False
Staff Engineer – Consumer Tech (f/m/x)	Lisbon	US	remote	InPost	Undefined	senior	2023-08-31T14:23:08.190Z	False
Staff Engineer – Consumer Tech (f/m/x)	Lisbon	US	remote	InPost	Undefined	senior	2023-08-31T14:28:03.569Z	False
Staff Engineer – Consumer Tech (f/m/x)	Lisbon	US	remote	InPost	Undefined	senior	2023-08-31T14:28:08.956Z	False
Senior FullStack Developer	Austin	US	remote	Plenti	50	senior	2023-09-12T07:08:00.000Z	False
Lead DevOps Engineer	Nowy Jork	US	remote	CouponFollow	Undefined	senior	2023-09-15T14:38:09.926Z	True

Análisis

Los tiempos de ejecución cambiaron de forma extraña a medida que aumentó el tamaño de los archivos.

Requerimiento <<3>>

Descripción

```
empresa = data["company_name"]
mapa_emp = data_structs["map_req3"]
if not mp.contains(mapa_emp, empresa):
    empresa_particular = mp.newMap()
    ofertas = lt.newList()
    mp.put(empresa_particular, "ofertas", ofertas)
    lt.addLast(ofertas,data)
    mp.put(empresa_particular,"exp_junior",0)
    mp.put(empresa_particular,"exp_mid",0)
    mp.put(empresa_particular,"exp_senior",0)
    mp.put(mapa_emp, empresa,empresa_particular)
    if experticia == "junior":
        expjunior = me.getValue(mp.get(empresa_particular, "exp_junior"))
        mp.put(empresa_particular, "exp_junior",expjunior+1)
    elif experticia == "mid":
        expmid = me.getValue(mp.get(empresa_particular, "exp_mid"))
        mp.put(empresa_particular, "exp_mid",expmid+1)
    else:
        expsenior = me.getValue(mp.get(empresa_particular, "exp_senior"))
        mp.put(empresa_particular, "exp_senior",expsenior+1)
else:
    empresa_particular = me.getValue(mp.get(mapa_emp, empresa))
    ofertas = me.getValue(mp.get(empresa_particular, "ofertas"))
    lt.addLast(ofertas,data)
    if experticia == "junior":
        expjunior = me.getValue(mp.get(empresa_particular, "exp_junior"))
        mp.put(empresa_particular, "exp_junior",expjunior+1)
    elif experticia == "mid":
        expmid = me.getValue(mp.get(empresa_particular, "exp_mid"))
        mp.put(empresa_particular, "exp_mid",expmid+1)
    else:
        expsenior = me.getValue(mp.get(empresa_particular, "exp_senior"))
        mp.put(empresa_particular, "exp_senior",expsenior+1)
```

Entrada	Los parametros necesarios para obtener la salida esperada fueron el nombre de la empresa, la fecha inicial y la fecha final
Salidas	La respuesta si fue la esperada por el algoritmo, con una baja complejidad espacial. Esto se obtuvo de la salida 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).
Implementado (Sí/No)	Si se implementó, se trabajo individualmente en este requerimiento

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Este codigo fue cargado en la carga de datos, entonces al ejecutar el requerimiento 3, la complejidad de este seria o(1)

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)**
small	0.04
5 pct	0.29
10 pct	0.92
20 pct	2.15
30 pct	4..45
50 pct	7.32
80 pct	13.57
large	20.8

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	0.04
5 pct	Dato2	0.29
10 pct	Dato3	0.92
20 pct	Dato4	2.15

30 pct	Dato5	4.45
50 pct	Dato6	7.32
80 pct	Dato7	13.57
large	Dato8	20.8

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Al ser el requerimiento 3 tener una complejidad temporal O(1), aun asi probando diferentes archivos los tiempos fueron los “esperados” ya que fue un algoritmo con poca memoria y espacio almacenado para el procesamiento de estos mismos.

Requerimiento <<5>>

Descripción

Consultar las ofertas de trabajo publicadas en una ciudad dado un rango de fecha.

```python
def req_5(data_structs, ciudad, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    mapa_ciudades = data_structs["map_req5"]
    ofertas_ciudad_seleccionada = me.getValue(mp.get(mapa_ciudades, ciudad))
    fecha_inicial = datetime.strptime(fecha_inicial,"%Y-%m-%d")
    fecha_final = datetime.strptime(fecha_final,"%Y-%m-%d")

    #Encuentro el total de ofertas publicadas en la ciudad en el periodo de consulta.
    ofertas_ciudad_tiempo = lt.newList(datastructure="SINGLE_LINKED")
    for oferta in lt.iterator(ofertas_ciudad_seleccionada):
        fecha_oferta = datetime.strptime(oferta["published_at"].split("T")[0],"%Y-%m-%d")
        if fecha_inicial <= fecha_oferta and fecha_oferta <= fecha_final:
            lt.addLast(ofertas_ciudad_tiempo, oferta)
    total_ofertas = lt.size(ofertas_ciudad_tiempo)
```

```python
    #Encuentro el total de empresas que publicaron por lo menos una oferta en la ciudad de consulta.
    empresas_ciudad_tiempo = mp.newMap(numelements=203564,
                                       prime=109345121,
                                       maptype="CHAINING",
                                       loadfactor=4)
    for oferta in lt.iterator(ofertas_ciudad_tiempo):
        empresa = oferta["company_name"]
        if not mp.contains(empresas_ciudad_tiempo, empresa):
            ofertas_empresa = lt.newList(datastructure="SINGLE_LINKED")
            lt.addLast(ofertas_empresa, oferta)
            mp.put(empresas_ciudad_tiempo, empresa, ofertas_empresa)
        else:
            ofertas_empresa = me.getValue(mp.get(empresas_ciudad_tiempo, empresa))
            lt.addLast(ofertas_empresa, oferta)
    llaves = mp.keySet(empresas_ciudad_tiempo)
    total_empresas = lt.size(llaves)

    #Empresa con mayor número de ofertas y su conteo. Empresa con menor número de ofertas (al menos una) y su conteo.
    max_ofertas = 0
    min_ofertas = 203564
    for empresa in lt.iterator(llaves):
        num_ofertas_empresa = lt.size(me.getValue(mp.get(empresas_ciudad_tiempo, empresa)))
        if num_ofertas_empresa > max_ofertas:
            max_ofertas = num_ofertas_empresa
            empresa_max = empresa
        if num_ofertas_empresa < min_ofertas:
            min_ofertas = num_ofertas_empresa
            empresa_min = empresa
    rta_empresa_max = (empresa_max, max_ofertas)
    rta_empresa_min = (empresa_min, min_ofertas)
```

```
#El listado de ofertas publicadas ordenadas cronológicamente por fecha y nombre de la empresa.
#Fecha de publicación de la oferta, Título de la oferta, Nombre de la empresa de la oferta,
#Tipo de lugar de trabajo de la oferta, Tamaño de la empresa de la oferta, Tipo de lugar de trabajo de la oferta
merg.sort(ofertas_ciudad_tiempo, sort_criteria5)
for oferta in lt.iterator(ofertas_ciudad_tiempo):
    oferta.pop("street")
    oferta.pop("address_text")
    oferta.pop("marker_icon")
    oferta.pop("company_url")
    oferta.pop("experience_level")
    oferta.pop("remote_interview")
    oferta.pop("open_to_hire_ukrainians")
    oferta.pop("id")
    oferta.pop("display_offer")

return total_ofertas, total_empresas, rta_empresa_max, rta_empresa_min, ofertas_ciudad_tiempo
```

Entrada	Los parámetros necesarios para obtener la salida esperada fueron la estructura creada en el modelo para el requerimiento, el nombre de la ciudad, la fecha inicial y la fecha final.
Salidas	La salida de este requerimiento es una tupla de 5 elementos. 1. El total de ofertas publicadas en la ciudad en el periodo de consulta. 2. El total de empresas que publicaron por lo menos una oferta en la ciudad de consulta. 3. Empresa con mayor número de ofertas y su conteo 4. Empresa con menor número de ofertas (al menos una) y su conteo. 5. El listado de ofertas publicadas ordenadas cronológicamente por fecha y nombre de la empresa. Cada una de las ofertas presenta la siguiente información: • Fecha de publicación de la oferta • Título de la oferta • Nombre de la empresa de la oferta • Tipo de lugar de trabajo de la oferta • Tamaño de la empresa de la oferta • Tipo de lugar de trabajo de la oferta
Implementado (Sí/No)	Se implementó por Mariana Cediel

Análisis de complejidad

Pasos	**Complejidad**
Encontrar la lista de ofertas de una ciudad en el mapa de ciudades.	O(1)
Llenar la lista de las ofertas en la ciudad en el periodo de tiempo dado.	O(n)
Iterar la lista de las ofertas en la ciudad en el periodo de tiempo dado para llenar el mapa de empresas en la ciudad en el periodo de tiempo dado.	O(n)
Iterar el mapa de empresas en la ciudad en el periodo de tiempo dado para encontrar la ciudad con mayor y menor número de ofertas.	O(n)
Ordenar la lista de las ofertas de la ciudad en el tiempo dado.	O(n log (n))

Recorrer la lista ordenada para eliminar la información no pedida.	O(n)
TOTAL	***O(n log(n))***

Este código funciona a partir de un mapa cuyas llaves son ciudades y sus valores correspondientes listas de las ofertas de una ciudad. Entonces, encontrar la lista de ofertas publicadas en una ciudad tiene complejidad O(1).

Se necesita llenar la lista, inicialmente vacía, de las ofertas en la ciudad en el periodo de tiempo dado. Para esto es necesario recorrer completamente la lista de ofertas en la ciudad para ver si cada oferta se encuentra dentro del rango de fechas. Esta iteración tiene complejidad O(n).

Es necesario ahora llenar un mapa de empresas de la ciudad que se está estudiando. Para esto, es necesario recorrer la lista de ofertas en la ciudad en el periodo de tiempo dado. Esto tiene complejidad, en el peor de los casos, O(n).

Se itera el mapa de las empresas para encontrar aquella con mayor y menor número de ofertas. Es necesario recorrer el mapa completo. Tiene complejidad O(n).

Se quiere ordenar la lista de ofertas de la ciudad en el periodo de tiempo dado. Se utiliza merge sort. Este tipo de merge tiene complejidad temporal de O(n log(n)).

Por último se recorre esta lista ordenada para eliminar la información que no se pide de cada empresa. Hay que recorrer la lista entera. Tiene complejidad O(n).

Pruebas Realizadas

Se realizó la prueba del funcionamiento de este requerimiento con:

- Ciudad = Amsterdam
- Fecha_inicial = 2022-05-31
- Fecha_final = 2022-12-01

Computador donde se probó

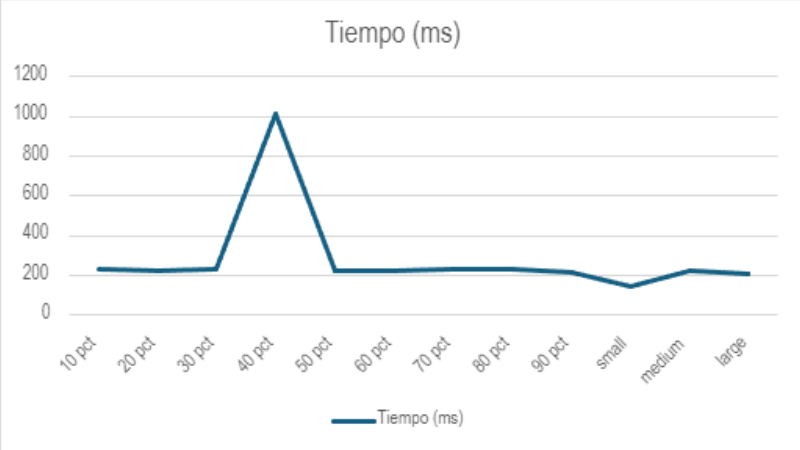
Procesador	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	12 GB
Sistema operativo	Windows 11

Tablas de datos

Entrada	**Tiempo (ms)**
10 pct	232.552
20 pct	222.655
30 pct	233.825
40 pct	1015.383
50 pct	224.456
60 pct	220.209

70 pct	231.560
80 pct	235.595
90 pct	218.009
small	141.174
medium	223.077
large	210.392

Graficas




```
El total de ofertas publicadas en  Amsterdam  entre la fecha  2022-05-31  y la fecha  2022-12-01  es  86
El total de empresas que publicaron por lo menos una oferta en  Amsterdam  durante el periodo de consulta es  48
La empresa con mayor número de ofertas en la ciudad durante el periodo de consulta es  ShiftKey  con  8  ofertas.
La empresa con menor número de ofertas en la ciudad durante el periodo de consulta es  Dott  con  1  ofertas.
```

title	city	country_code	workplace_type	company_name	company_size	published_at
Quality Assurance Lead	Amsterdam	NL	remote	Fortum	10000	2022-05-31T07:39:00.000Z
Senior Node.js Developer	Amsterdam	NL	remote	BrainRocket	600	2022-06-13T08:25:00.000Z
Senior Product Designer	Amsterdam	NL	remote	Vosbor	30	2022-06-16T15:51:51.191Z
Manual tester	Amsterdam	NL	remote	Vosbor	30	2022-06-17T07:05:00.000Z
Senior DevOps Engineer (Docker, AWS)	Amsterdam	NL	partly_remote	StuDocu	70	2022-06-20T14:53:47.408Z
Tech Team Lead	Amsterdam	NL	partly_remote	StuDocu	70	2022-06-20T15:08:41.554Z
Lead Product Designer	Amsterdam	NL	remote	MomentRanks	Undefined	2022-06-24T11:23:00.000Z
Senior Database Engineer	Amsterdam	NL	remote	1Password	100	2022-06-28T01:20:55.461Z
Cloud / Infra Architect	Amsterdam	NL	remote	Miro	100	2022-06-28T03:23:59.633Z
Frontend Architect	Amsterdam	NL	remote	Miro	100	2022-06-28T03:30:52.391Z
Senior Staff Software Engineer	Amsterdam	NL	remote	Miro	100	2022-06-28T03:33:07.033Z
Staff Software Engineer (Growth)	Amsterdam	NL	remote	Miro	100	2022-06-28T03:35:30.093Z
SRE	Amsterdam	PL	remote	Nuclia	20	2022-06-28T08:24:00.000Z
Senior Mobile Developer	Amsterdam	NL	remote	RabbitX	Undefined	2022-06-28T14:28:00.000Z
Development Team Lead - EMEA Region	Amsterdam	NL	remote	thoughtbot	200	2022-06-29T07:30:00.000Z
IT-Admin for Purchase and user support	Amsterdam	NL	remote	HanseYachts AG	1000	2022-07-13T08:21:00.000Z
Infrastructure Architect	Amsterdam	NL	remote	StepUp	20	2022-07-13T10:19:00.000Z
Junior .NET Developer	Amsterdam	NL	remote	Oodit	25	2022-07-15T05:39:00.000Z

Análisis

El tiempo de ejecución fue más o menos constante en todos los diferentes tamaños de archivo. Sin embargo, para el archivo 40-pct-jobs.csv se vio un aumento significativo en el tiempo de ejecución.

Requerimiento <<6>>

Descripción

Clasifica las N ciudades con mayor cantidad de ofertas de trabajo dado el año de publicación y nivel de experticia de la oferta.

```
def req_6(data_structs, numero_ciudades, experticia, anio):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    dic_anios = data_structs["dic_req_6"]
    #Encuentro la lista de ofertas del año dado con la experticia dada
    ofertas_anio_exp = dic_anios[anio][experticia]
    total_ofertas = lt.size(ofertas_anio_exp)

    #Encuentro el total de empresas que tienen ofertas publicadas en el año dado con la experticia dada
    #Encuentro el total de ciudades que tienen ofertas publicadas en el año dado con la experiencia dada.
    empresas = lt.newList(datastructure="SINGLE_LINKED")
    ciudades = mp.newMap(numelements=17,
                         prime=109345121,
                         maptype="CHAINING",
                         loadfactor=4)

    for oferta in lt.iterator(ofertas_anio_exp):
        empresa = oferta["company_name"]
        ciudad = oferta["city"]
        if not lt.isPresent(empresas, empresa):
            lt.addLast(empresas, empresa)
        if not mp.contains(ciudades, ciudad):
            info_ciudad = {
                "nombre": ciudad,
                "pais": oferta["country_code"],
                "ofertas": lt.newList(datastructure="SINGLE_LINKED"),
                "num_ofertas": 0,
                "empresas_ciudad": mp.newMap(numelements=17,
                                             prime=109345121,
                                             maptype="CHAINING",
                                             loadfactor=4),
                "num_empresas": 0,
                "mayor_empresa": ()
                }
            lt.addLast(info_ciudad["ofertas"], oferta)
            info_ciudad["num_ofertas"] = lt.size(info_ciudad["ofertas"])
```

```
            if not mp.contains(info_ciudad["empresas_ciudad"], empresa):
                info_empresa = {
                    "nombre": empresa,
                    "ofertas_empresa": lt.newList(datastructure="SINGLE_LINKED"),
                    "num_ofertas": 0
                }
                lt.addLast(info_empresa["ofertas_empresa"], oferta)
                info_empresa["num_ofertas"] = lt.size(info_empresa["ofertas_empresa"])
                mp.put(info_ciudad["empresas_ciudad"], empresa, info_empresa)
            else:
                info_empresa = me.getValue(mp.get(info_ciudad["empresas_ciudad"], empresa))
                lt.addLast(info_empresa["ofertas_empresa"], oferta)
                info_empresa["num_ofertas"] = lt.size(info_empresa["ofertas_empresa"])
            mp.put(ciudades, ciudad, info_ciudad)
        else:
            info_ciudad = me.getValue(mp.get(ciudades, ciudad))
            lt.addLast(info_ciudad["ofertas"], oferta)
            info_ciudad["num_ofertas"] = lt.size(info_ciudad["ofertas"])
            if not mp.contains(info_ciudad["empresas_ciudad"], empresa):
                info_empresa = {
                    "nombre": empresa,
                    "ofertas_empresa": lt.newList(datastructure="SINGLE_LINKED"),
                    "num_ofertas": 0
                }
                lt.addLast(info_empresa["ofertas_empresa"], oferta)
                info_empresa["num_ofertas"] = lt.size(info_empresa["ofertas_empresa"])
                mp.put(info_ciudad["empresas_ciudad"], empresa, info_empresa)
            else:
                info_empresa = me.getValue(mp.get(info_ciudad["empresas_ciudad"], empresa))
                lt.addLast(info_empresa["ofertas_empresa"], oferta)
                info_empresa["num_ofertas"] = lt.size(info_empresa["ofertas_empresa"])

    total_empresas = lt.size(empresas)
    lista_ciudades = mp.keySet(ciudades)
    total_ciudades = lt.size(lista_ciudades)
    if total_ciudades > numero_ciudades:
        total_ciudades = numero_ciudades
```

```python
    #Encuentro la ciudad con mayor cantidad de ofertas y su conteo y la ciudad con menor cantidad de ofertas y su conteo
    max_ofertas_ciu = 0
    min_ofertas_ciu = 203564
    lista_dic_ciudades = lt.newList(datastructure="SINGLE_LINKED") #La lista final
    for ciudad in lt.iterator(lista_ciudades):
        info_ciudad = me.getValue(mp.get(ciudades, ciudad))
        num_ofertas_ciudad = info_ciudad["num_ofertas"]
        if num_ofertas_ciudad > max_ofertas_ciu:
            max_ofertas_ciu = num_ofertas_ciudad
            ciudad_max = ciudad
        if num_ofertas_ciudad < min_ofertas_ciu:
            min_ofertas_ciu = num_ofertas_ciudad
            ciudad_min = ciudad
        lista_empresas_ciudad = mp.keySet(info_ciudad["empresas_ciudad"])
        num_empresas_ciudad = lt.size(lista_empresas_ciudad)
        info_ciudad["num_empresas"] = num_empresas_ciudad
        max_ofertas_emp = 0
        for empresa in lt.iterator(lista_empresas_ciudad):
            info_empresa = me.getValue(mp.get(info_ciudad["empresas_ciudad"], empresa))
            num_ofertas_empresa = info_empresa["num_ofertas"]
            if num_ofertas_empresa > max_ofertas_emp:
                max_ofertas_emp = num_ofertas_empresa
                empresa_max = empresa
        mayor_empresa = (empresa_max, max_ofertas_emp)
        info_ciudad["mayor_empresa"] = mayor_empresa
        lt.addLast(lista_dic_ciudades, info_ciudad)
    rta_ciudad_max = (ciudad_max, max_ofertas_ciu)
    rta_ciudad_min = (ciudad_min, min_ofertas_ciu)

    #Ordeno la lista de ciudades por numero de ofertas.
    merg.sort(lista_dic_ciudades, sort_criteria6)
    if lt.size(lista_dic_ciudades) > numero_ciudades:
        lista_dic_ciudades = lt.subList(lista_dic_ciudades, 1, numero_ciudades)
    for ciudad in lt.iterator(lista_dic_ciudades):
        ciudad.pop("ofertas")
        ciudad.pop("empresas_ciudad")

    return numero_ciudades, total_empresas, total_ofertas , rta_ciudad_max,  rta_ciudad_min, lista_dic_ciudades
```

Entrada	Los parámetros necesarios para obtener la salida esperada fueron la estructura creada en el modelo para el requerimiento, el número de ciudades a listar, el nivel de experticia y el año.
Salidas	La salida de este requerimiento es una tupla de 6 elementos. 1. El total de ciudades que cumplen con las condiciones de la consulta (valor menor o igual a N) 2. El total de empresas que cumplen con las condiciones de la consulta 3. El total de ofertas publicadas que cumplen con las condiciones de la consulta 4. Nombre de la ciudad con mayor cantidad de ofertas de empleos y su conteo 5. Nombre de la ciudad con menor cantidad de ofertas de empleos y su conteo 6. El listado de las ciudades ordenadas por el número de ofertas publicadas y nombre de la ciudad. Cada una de las ciudades resultantes contiene la siguiente información: • Nombre de la ciudad. • País de la ciudad.

	• El total de ofertas hechas en la ciudad. • Número de empresas que publicaron por lo menos una oferta en la ciudad • Nombre de la empresa con mayor número de ofertas y su conteo
Implementado (Sí/No)	Se implementó por Mariana Cediel

Análisis de complejidad

Pasos	**Complejidad**
Encontrar la lista de ofertas de una experticia en un año.	O(1)
Llenar la lista de empresas y el mapa de ciudades a partir de la lista de ofertas.	O(n)
Iterar el mapa de ciudades para encontrar la ciudad con mayor y menor número de ofertas con la experticia dada en el año dado.	O(n^2)
Iterar el mapa de empresas en una ciudad para encontrar la empresa con más ofertas y llenar la lista de las ciudades	O(n)
Ordenar la lista de las ciudades.	O(n log (n))
Iterar la lista ordenada de las ciudades para eliminar la información no solicitada.	O(n)
TOTAL	***O(n^2)***

Este código funciona a partir de un diccionario cuyas llaves son los años 2022 y 2023 y el valor de cada una es un diccionario que tiene de llaves “junior”, “mid” y “senior” y como valores listas de las ofertas que cumplan la condición del año y la condición de la experticia. Entonces, acceder a la lista que se necesita analizar tiene complejidad O(1).

Es necesario llenar una lista con los nombres de todas las empresas que publicaron ofertas en el año dado, con el nivel de experticia dada y un mapa de las todas las ciudades en que se publicaron ofertas con las mismas condiciones. Para esto, es necesario iterar la lista de todas las ofertas que cumplen estas condiciones. Este proceso tiene complejidad O(n).

Se itera el mapa de ciudades para encontrar la ciudad con el mayor y el menor número de ofertas. Este paso tiene complejidad O(n^2). Esto puesto que dentro de esta iteración se hace otra, volviendo entonces, esta iteración exponencial.

Se itera el mapa de empresas de una ciudad para encontrar la empresa con más ofertas publicadas. Tiene complejidad temporal de O(n).

Se quiere ordenar la lista de ofertas de la ciudad en el periodo de tiempo dado. Se utiliza merge sort. Este tipo de merge tiene complejidad temporal de O(n log(n)).

Por último, se recorre esta lista ordenada para eliminar la información que no se pide de cada empresa. Hay que recorrer la lista entera. Tiene complejidad O(n).

Pruebas Realizadas

Se realizó la prueba del funcionamiento de este requerimiento con:

- Numero_ciudades = 12
- Experticia = junior
- Anio = 2022

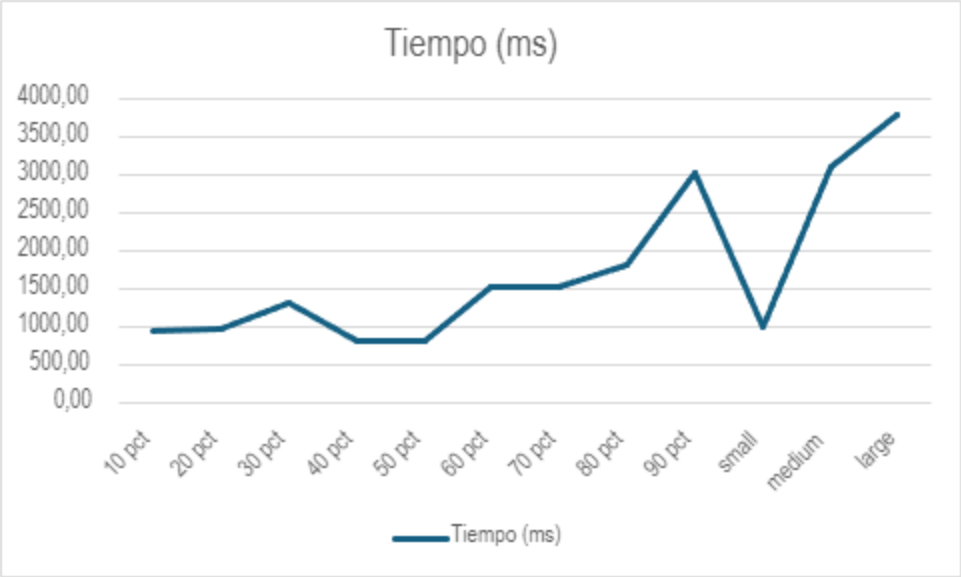
Computador donde se probó

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	953,79
20 pct	972,22
30 pct	1321,67
40 pct	821,38
50 pct	833,98
60 pct	1527,04
70 pct	1538,94
80 pct	1833,60
90 pct	3031,02
small	1014,21
medium	3127,52
large	3793,67

Graficas




```
Ingrese el numero de ciudades a solicitar: 12
Ingrese el nivel de experticia solicitada: junior
Ingrese el año a solicitar: 2022
El total de ciudades que cumplen con las condiciones de la consulta es  12
El total de empresas que cumplen con las condiciones de la consulta es  1758
El total de ofertas publicadas que cumplen con las condiciones de la consulta es  8960
El nombre de la ciudad con mayor cantidad de ofertas de empleos es  Warszawa  con  2035  ofertas.
El nombre de la ciudad con menor cantidad de ofertas de empleos es  Mountain View  con  1  ofertas.
```

nombre	pais	num_ofertas	num_empresas	mayor_empresa
Warszawa	PL	2035	791	('Nexio Management', 42)
Wroclaw	PL	1039	490	('SoftServe', 25)
Krakow	PL	1023	512	('Comarch', 26)
Gdansk	PL	612	299	('Lufthansa Systems', 22)
Poznan	PL	597	320	('Dataedo', 12)
Katowice	PL	356	179	('Reply', 53)
Lodz	PL	354	196	('Commerzbank', 22)
Lublin	PL	195	116	('Asseco Business Solutions', 14)
Szczecin	PL	187	108	('GlobalLogic', 17)
Bydgoszcz	PL	181	104	('Atos', 9)
Gdynia	PL	161	85	('Soldevelo', 27)
Rzeszow	PL	160	97	('Ideo Sp. z o.o.', 7)

Análisis

Los tiempos de ejecución cambiaron de forma extraña a medida que aumentó el tamaño de los archivos.

Requerimiento <<7>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

```
mapa_anio = data_structs["map_req7"]
anio = data["published_at"].split("-")[0]
mes = data["published_at"].split("-")[1]
oferta = mp.newMap()
rta = {"total_ofertas":lt.size(ofertas),
       "pais_mayor_ofertas":pais_top,
       "ciudad_mayor_ofertas":ciudad_top,
       "ofertas":oferta
       }
if not mp.contains(mapa_anio, anio):
    mapa_mes = mp.newMap()
    ofertas = lt.newList
    mp.put(mapa_anio, anio, mapa_mes)
    mp.put(mapa_mes, mes, rta)
    lt.addLast(ofertas, data)
    paises = Counter(ofertas["country_code"])
    pais_top = paises.most_common(1)
    ciudad = Counter(ofertas["city"])
    ciudad_top = ciudad.most_common(1)
    lista_final = {"empresas": empresass,
                   "empresa_max": empresas_max,
                   "empresas_min": empresas_min}
    if experticia == "junior":
        expjunior = me.getValue(mp.get(oferta, "exp_junior"))
        empresas = Counter(ofertas["company_name"])
        empresass = empresas.most_common(1)
        dic_inverso = {}
        for dato, repeticiones in ofertas.items():
            if repeticiones not in dic_inverso:
                dic_inverso[repeticiones] = []
            dic_inverso[repeticiones].append(dato)
        mp.put(oferta, "exp_junior",lista_final)
        empresas_max = max(dic_inverso)
        empresas_min = min(dic_inverso)
    elif experticia == "mid":
        expmid = me.getValue(mp.get(oferta, "exp_mid"))
        dic_inverso = {}
        for dato, repeticiones in ofertas.items():
            if repeticiones not in dic_inverso:
                dic_inverso[repeticiones] = []
            dic_inverso[repeticiones].append(dato)
        empresas_max = max(dic_inverso)
        empresas_min = min(dic_inverso)
        mp.put(oferta, "exp_mid",lista_final)
    else:
        expsenior = me.getValue(mp.get(oferta, "exp_senior"))
        dic_inverso = {}
        for dato, repeticiones in ofertas.items():
            if repeticiones not in dic_inverso:
                dic_inverso[repeticiones] = []
            dic_inverso[repeticiones].append(dato)
        empresas_max = max(dic_inverso)
        empresas_min = min(dic_inverso)
        mp.put(oferta, "exp_senior",lista_final)
```

Entrada	Los parametros necesarios para obtener la salida esperada fueron los numero de paises a consultar, el año y el mes
Salidas	La respuesta si fue la esperada por el algoritmo, La salida fue 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 de experticia 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 de experticia que tienen una o más sedes
Implementado (Sí/No)	Si se implementó, se trabajo en grupo en este requerimiento

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Este codigo fue cargado en la carga de datos, entonces al ejecutar el requerimiento 1, la complejidad de este seria o(1)

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)**
small	0
5 pct	0
10 pct	0
20 pct	0
30 pct	0
50 pct	0

80 pct	0
large	0

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0
5 pct	Dato2	0
10 pct	Dato3	0
20 pct	Dato4	0
30 pct	Dato5	0
50 pct	Dato6	0
80 pct	Dato7	0
large	Dato8	0

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Al ser el requerimiento 1 tener una complejidad temporal O(1), aun asi probando diferentes archivos los tiempos fueron los “esperados” ya que fue un algoritmo con poca memoria y espacio almacenado para el procesamiento de estos mismos.