ANÁLISIS DEL RETO

Juan Diego Montenegro - 202110276 – jd.montengro@uniandes.edu.co

Nicolas Parra Zorro – 202322257 – n.parraz@uniandes.edu.co

Susana Faciolince Santoyo – 202021066 – s.faciolince@uniandes.edu.co

Equipos <<>>

Tabla con las especificaciones de los equipos con los que se trabajaron.

COMPUTADORES	Juan Montenegro	Nicolas Parra	Susana Faciolince
Procesador	Intel 15 11th	AMD Ryzen 5 5600H	Sala turing
Memoria RAM	24.0GB	8.0GB	Sala turing
S. operativo	Windows 11 -64bits	Windows 11 -64bits	Sala turing

Requerimiento <<Carga>>

Descripción

```python
def load_data(control, memflag=True):
    """
    Carga los datos del reto
    """
    start_time = getTime()

    if memflag is True:
        tracemalloc.start()
        start_memory = getMemory()
    structure = control['model']
    loadjobs(structure)
    loadskills(structure)
    loademployment(structure)
    loadmultilocation(structure)

    stop_time = getTime()
    delta_time = deltaTime(stop_time, start_time)

    if memflag is True:
        stop_memory = getMemory()
        tracemalloc.stop()
        delta_memory = deltaMemory(stop_memory, start_memory)
        return delta_time, delta_memory

    else:
        return delta_time
```

Entrada	Archivos en formato csv.
Salidas	Ninguna (solo carga los datos)
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Abrir csv	O(cte)
Agregar datos a una lista (ltaddlast)	O(cte)
Agregar datos a un mapa (mput)	O(n)
TOTAL	***O(cte)***

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	**Memoria (kB)**	**Tiempo (ms)**
10 – por	56684.43	5433.68
20 - por	109762.369	7272.017
30 - por	169447.882	11754.842
40 - por	225147.702	14702.678
50 - por	274276.987	17225.284
60 - por	306124.094	19946.823
70 - por	316202.084	19856.189
80 - por	323445.524	20431.197
90 - por	328451.858	23303.981
small	187370.118	11921.946
medium	312741.648	20511.615
large	332348.439	21468.300

Análisis

Para la carga de datos necesitamos abrir los archivos usando la librería csv. Continuando para usar las funciones mp.put y lst. Addlast para agregar los datos al diccionario de catalog.

Requerimiento <<Requerimiento 1>>

Descripción

```python
def req_1(catalog, codPais, exp, n):

    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    jobs = catalog['jobs']
    infol, dic = filtro_r1(jobs, codPais, exp)
    x = lst.sublist(infol, 0, n)

    return x , dic

def filtro_r1(mapa,pais,exp):

    lista = lst.new_list()
    dic = {'pais': pais,
           'npais': 0,
           'exp': 0}

    for i in range(len(mapa['keys'])):
        for j in (mapa['keys'][i]):
            key = j[0]
            value = j[1]
            if value['country_code'] == pais:
                dic['npais'] += 1
                if value['experience_level'] == exp:
                    dic['exp'] += 1
                    lst.crit_add_ordered(lista,value,crit1)

    return lista, dic
```

Entrada	Catálogo de los datos, código del país, nivel de experticia y el numero solicitado.
Salidas	Lista con todos los datos filtrados y un diccionario con los contadores solicitados
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
New_list()	O(cte)
For i in range()	O(n)
For j in (mapa[‘keys’][i])	O(n)
Sublist()	O(n)
Crit_add_ordered()	O(n^2)

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

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).

Muestra	**Datos**	**Tiempo (ms)**
10 – por	AE, senior, 7	12.0506
30 - por	AE, senior, 7	21.2829
50 - por	AE, senior, 7	58.0619
70 - por	AE, senior, 7	51.3117
small	AE, senior, 7	27.7749
large	AE, senior, 7	67.9565

Análisis

Para el requerimiento 1 usamos una función auxiliar, la cual se centra en recorrer el mapa de los datos, para cuando cumpla las condiciones de país y experiencia, agregara a una lista nueva ese valor y sumara los valores en un diccionario para facilitar el conteo de los datos. Para finalizar creando la sublista de la cantidad de los datos que solicito el usuario.

Requerimiento <<Requerimiento 2>>

Descripción

```python
def req_2(catalog, ciudad,emp, n):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    jobs = catalog['jobs']
    infol = filtro_r2(jobs, ciudad,emp)
    size = lst.size(infol)
    x = lst.sublist(infol, 0, n)
    return x, size

def filtro_r2(mapa,ciudad,emp):
    """
    """

    lista_nueva=lst.new_list()

    for i in range(len(mapa['keys'])):
        for j in (mapa['keys'][i]):
            value = j[1]
            if value['city'] == ciudad:
                if value['company_name'] == emp:
                    lst.crit_add_ordered(lista_nueva,value, crit2)

    return (lista_nueva)
```

Entrada	Catálogo de los datos, ciudad, nombre de la empresa y el numero solicitado.
Salidas	Lista con todos los datos filtrados y el tamaño de la lista completa (antes de cortar los datos por el valor n que ingresa el usuario)
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
New_list()	O(cte)
For i in range()	O(n)
For j in (mapa[‘keys’][i])	O(n)
Crit_add_ordered()	O(n^2)
Sublist()	O(n)
TOTAL	***O(n^2)***

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).

Muestra	**Datos**	**Tiempo (ms)**
10 – por	Abu Dhabi, Unikie, 3	11.2879
30 - por	Abu Dhabi, Unikie, 6	36.3149
50 - por	Abu Dhabi, Unikie, 6	34.3753
70 - por	Abu Dhabi, Unikie, 6	49.5646
small	Abu Dhabi, Unikie, 6	60.8818
large	Abu Dhabi, Unikie, 6	96.8218

Análisis

Para el requerimiento 2 (asi como en el requerimiento 1) usamos una función auxiliar, la cual se centra en recorrer el mapa de los datos, para cuando cumpla las condiciones de ciudad y nombre de empresa, agregara a una lista nueva ese valor y sumara los valores en un diccionario para facilitar el conteo de los datos. Para finalizar creando la sublista de la cantidad de los datos que solicito el usuario.

Requerimiento <<Requerimiento 3>>

Descripción

```python
def req_3(catalog, empresa, fi, ff):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    jobs = catalog['jobs']
    infol, dic = filtro_r3(jobs, empresa, fi, ff)
    return infol, dic

def filtro_r3(mapa,empresa,fi, ff):
    """
    retorna un mapa y una lista de las ofertas en un pais segun nivel de experticia.
    """

    lista_nueva=lst.new_list()
    dic = {'mid': 0,
           'senior': 0,
           'junior': 0}

    for i in range(len(mapa['keys'])):
        for j in (mapa['keys'][i]):

            key = j[0]
            value = j[1]

            if value['company_name']==empresa:
                if fi <= value['published_at'] <= ff:
                    lista_nueva = lst.crit_add_ordered(lista_nueva,value, crit3)
                    if value['experience_level'] == 'mid':
                        dic['mid'] += 1
                    if value['experience_level'] == 'senior':
                        dic['junior'] += 1
                    if value['experience_level'] == 'junior':
                        dic['senior'] += 1

    return lista_nueva, dic
```

Entrada	Catálogo de los datos, nombre de empresa, fecha inicial y fecha final.
Salidas	Lista con todos los datos filtrados y el diccionario con la cantidad de ofertas según experiencia
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
New_list()	O(cte)
For i in range()	O(n)
For j in (mapa['keys'][i])	O(n)
Crit_add_ordered()	O(n^2)
Sublist()	O(n)
TOTAL	***O(n^2)***

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).

Muestra	**Datos**	**Tiempo (ms)**
10 – por	1Password, 2022-04-15, 2022-08-01	12.1625
30 - por	1Password, 2022-04-15, 2022-08-01	26.4733
50 - por	1Password, 2022-04-15, 2022-08-01	35.8647
70 - por	1Password, 2022-04-15, 2022-08-01	57.0969
small	1Password, 2022-04-15, 2022-08-01	27.0
large	1Password, 2022-04-15, 2022-08-01	79.3885

Análisis

Para el requerimiento 3 (asi como en el requerimiento 2) usamos una función auxiliar, la cual se centra en recorrer el mapa de los datos, para cuando cumpla las condiciones de nombre de empresa y rango de fechas. Agregara a una lista nueva ese valor y revisara que tipo de experiencia es para sumarlo en los datos del diccionario. Para finalizar creando la sublista de la cantidad de los datos que solicito el usuario.

Requerimiento <<Requerimiento 4>>

Descripción

```python
def req_4(catalog, code, fi, ff):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    jobs = catalog['jobs']
    dic = filtro_r4(jobs, code, fi, ff)
    return dic

def filtro_r4(mapa, code, fi, ff):
    lista_ofertas = lst.new_list()
    lista_empresas = lst.new_list()
    lista_ciudades = lst.new_list()
    for i in range(len(mapa['keys'])):
        for j in(mapa['keys'][i]):
            key = j[0]
            value = j[1]
            if value['country_code'] == code:
                if fi <= value['published_at'] <= ff:
                    lista_ofertas = lst.addlast(lista_ofertas, value)

            if value['country_code'] == code:
                if fi <= value['published_at'] <= ff:
                    if lst.is_present(lista_empresas, value['company_name']) == False:
                        lista_empresas = lst.addlast(lista_empresas, value)

            if value['country_code'] == code:
                if fi <= value['published_at'] <= ff:
                    if lst.is_present(lista_ciudades, value['city']) == False:
                        lista_ciudades = lst.addlast(lista_ciudades, value)

    total_ofertas = lst.size(lista_ofertas)
    total_empresas = lst.size(lista_empresas)
    total_ciudades = lst.size(lista_ciudades)

    response = {
        "TOTAL OFERTAS": total_ofertas,
        "TOTAL EMPRESAS": total_empresas,
        "TOTAL CIUDADES": total_ciudades
    }
```

Entrada	Catálogo de los datos, código del país, fecha inicial y fecha final.
Salidas	Un diccionario con la cantidad de ofertas, empresas y ciudades según las condiciones especificadas.
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

New_list()	O(cte)
For i in range()	O(n)
For j in (mapa[‘keys’][i])	O(n)
addlast()	O(cte)
Size()	O(cte)
TOTAL	***O(n)***

Análisis

Para este requerimiento, se recorre la información y se almacena en 3 listas de diferente estilo cada una filtrando por ofertas, empresas y ciudades. Esto se realiza con la intención de que una vez filtradas todas las ofertas en base a una característica específica, entonces sea mucho más rápido y sencillo devolver la cantidad requerida, ejecutando la función .size() en las listas creadas.

Requerimiento <<Requerimiento 5>

Descripción

Para este requerimiento, creé una función que recorre todas las ofertas y las filtra al tiempo en un mapa y en una lista que se construye ordenada de acuerdo con lo que pide el reto. Luego, se recorre y crea una lista con las empresas que dieron ofertas y un nuevo mapa que le asigna el número de ofertas que hizo; después, se recorre la lista, consultando el número de ofertas en el mapa para determinar la empresa con mayor y menor número de ofertas, y la cantidad de ofertas que dieron.

```
def req_5(jobs,ciudad,fecha1,fecha2):
    """
    Función que soluciona el requerimiento 5
    """
    (mapa_ofertas,lista_ofertas)=filtro_r5(jobs['jobs'],ciudad,fecha1,fecha2)
    num_ofertas=mapa_ofertas['load']

    (mapa_empresas,lista_empresas)=empresas_r5(mapa_ofertas)
    (mejor_empresa,peor_empresa,num_ofertas_max,num_ofertas_min)=empresas_extremales_r5(mapa_empresas,lista_empresas)
    num_empresas=lista_empresas['size']

    dic={
        'num_ofertas':num_ofertas,
        'num_empresas':num_empresas,
        'mejor_empresa':mejor_empresa,
        'max_ofertas':num_ofertas_max,
        'peor_empresa':peor_empresa,
        'min_ofertas':num_ofertas_min
        }

    return (dic,lista_ofertas)
```

```python
def filtro_r5(mapa,ciudad,fecha1,fecha2):
    """
    Da un mapa y una lista de las ofertas en la ciudad entre ambas fechas.
    """

    mapa_nuevo=mp.new_map(int(mapa['capacity']//500)+1)
    lista_nueva=lst.new_list()

    for i in range(len(mapa['keys'])):
        for j in (mapa['keys'][i]):
            key = j[0]
            value = j[1]

            if value['city']==ciudad:
                if (fecha1 <= value['published_at']) and (value['published_at'] <= fecha2):
                    mapa_nuevo=mp.put(mapa_nuevo,value,key)
                    lst.crit_add_ordered(lista_nueva,value,crit_r5)

    return (mapa_nuevo,lista_nueva)
```

```python
def empresas_r5(mapa):
    """
    A partir del mapa filtrado, crea una lista con las empresas relevantes y un mapa que a cada empresa asigna el n
    """
    mapa_emp=mp.new_map(int(mapa['capacity']//5)+1) #TODO: Cambiar 5 por el numero promedio de ofertas en algun eje
    lista_emp=lst.new_list()

    for i in range(len(mapa['keys'])):
        for j in range(len(mapa['keys'][i])):
            key=mapa['keys'][i][j][0]
            value=mapa['keys'][i][j][1]
            empresa=value['company_name']

            if not mp.contains(mapa_emp,empresa):
                num_ofertas=1
                mapa_emp=mp.put(mapa_emp,num_ofertas,empresa)
                lista_emp=lst.addlast(lista_emp,empresa)
            else:
                for k in range(len(mapa_emp['keys'][mp.hash_fun(mapa_emp, empresa)])):
                    value = mapa_emp['keys'][mp.hash_fun(mapa_emp, empresa)][k]
                    llave = value[0]
                    num = value[1]
                    if llave == empresa:
                        mapa_emp['keys'][mp.hash_fun(mapa_emp, empresa)][k] = (llave, num+1)

    return (mapa_emp,lista_emp)
```

```python
def empresas_extremales_r5(mapa,lista):
    '''
    A partir del mapa y la lista anterior, retorna la empresa con más ofertas laborales y la empresa con menos ofertas laborales,
    junto con el número de ofertas que cada una ofrece.
    Si hay dos empresas con la cantidad maximal, devuelve la primera en salir.
    Si hay dos empresas con la cantidad minimal, devuelve la primera en salir.
    '''
    it = lst.iterator(lista)
    empresa=it['value']

    mejor_empresa=empresa
    peor_empresa=empresa
    num = mp.getvalue(mapa, empresa)
    num_ofertas_max=num
    num_ofertas_min=num

    while lst.hasNext(it):
        it=lst.Next(it)
        empresa=it['value']
        num_ofertas=mp.getvalue(mapa, empresa)

        if num_ofertas>num_ofertas_max:
            num_ofertas_max=num_ofertas
            mejor_empresa=empresa

        if num_ofertas<num_ofertas_min:
            num_ofertas_min=num_ofertas
            peor_empresa=empresa

    return (mejor_empresa,peor_empresa,num_ofertas_max,num_ofertas_min)
```

```python
def crit_r5(oferta1,oferta2):
    '''
    Es el criterio de comparación para dos ofertas. Retorna True si la primera d
    '''
    fecha1=oferta1['published_at']
    fecha2=oferta2['published_at']
    empresa1=oferta1['company_name'].lower()
    empresa2=oferta2['company_name'].lower()

    if fecha1<fecha2 or (fecha1==fecha2 and empresa1<=empresa2):
        return True
    else:
        return False
```

Entrada	Ciudad, fecha1, fecha2
Salidas	Una lista ordenada con todas las ofertas filtradas en orden y un diccionario que contiene: el número de ofertas y de empresas que dieron ofertas, la empresa con mayor y menos ofertas, junto con la cantidad de ofertas que hicieron.
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
filtro_r5()	O(N2)
empresas_r5()	O(N)
empresas_extremales_r5()	O(N)
crit_r5()	O(1)
req_5()	O(N2)
TOTAL	***O(N2)***

Análisis

Lo más complejo a nivel temporal del requerimiento es armar la lista de forma ordenada desde el inicio, que dio complejidad cuadrática: la función filtro_r5() sería lineal si simplemente agregaramos las nuevas ofertas al final y, si después usaramos un algoritmo de ordenamiento adecuado, reduciriamos la complejidad temporal a linearítmica. Por problemas de tiempo, no logramos esta optimización.

Por otra parte, creamos dos mapas nuevos con tamaños mucho mayores a lo necesario (dividimos el tamaño del mapa que almacena todos los datos por número para minimizar esto, pero faltó tiempo para hacer más purebas y encontrar un número menor), lo que fue más costoso espacialmente de lo que pudo haber sido. Sin embargo, esto garantiza que nunca fue necesario hacer rehash, lo que hizo que la creación de ambos mapas sea solo de orden lineal.