ANÁLISIS DEL RETO

Estudiante 1, Pablo Castrillon,202122150, p.castrillon@uniandes.edu.co

Estudiante 2, David Felipe Monroy, 202211146, d.monroyh@uniandes.edu.co

Estudiante 3, Sergio David Laverde, ,

Requerimiento 1

```
def req_1(catalog,Id_1,Id_2):
    start_time=get_time()
    recorrido=dfo.depth_first_order(catalog,Id_1)
    camino=dfs.path_to(recorrido,Id_2)
    info_usuarios=[]
    for i in camino:
        info_usuarios.append(gr.get_vertex_information(catalog),i)
    end_time=get_time()
    delta_time=end_time-start_time
    return info_usuarios,delta_time
```

Descripción

Utilizamos el método DFO que nos encuentra el camino en menores

Entrada	El catálogo con el grafo del problema, una Id inicial y una Id final, con las que se hace la búsqueda

Salidas	Lista de cuentas en las que se hace el camino para las 2 personas
Implementado (Sí/No)	Si, grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Algoritmo DFS	O(V+E)
Impresión del resultado	O(1)
TOTAL	***O(V+E)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron: 2016-06-06 12:12:12 y 2016-06-06 12:12:12

Procesador	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
Memoria RAM	8GB
Sistema operativo	Windows 10 - 64 bits

Entrada	**Tiempo (s)**
small	0.09
Médium(25k)	0.138
large	0.102

Análisis

Requerimiento 2

```
def req_2(catalog,Id_1,Id_2):
    start_time=get_time()
    if gr.get_vertex_information(catalog,Id_1)['USER_TYPE']=='basic' and gr.get_vertex_information(catalog
        recorrido=bfs.breath_first_search(catalog,Id_1)
        camino=bfs.path_to(recorrido,Id_2)
        info_usuarios=[]
        for i in camino:
            info_usuarios.append(gr.get_vertex_information(catalog,i))
    else:
        info_usuarios='Los usuarios no son de tipo básico'
    end_time=get_time()
    delta_time=end_time-start_time
    return info_usuarios,len(info_usuarios),delta_time
```

Descripción

Utilizamos el algoritmo bfs y buscamos el camino entre la raiz y el nodo objetivo

Entrada	El catálogo, el índice 1 y el índice 2
Salidas	Lista con los usuarios que pasan por el camino
Implementado (Sí/No)	Si, grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
BFS	O(V+E)
Path to	O(N)
TOTAL	***O(V+E)***

Análisis

Requerimiento 3

```
def req_3(catalog,Id):
    start_time=get_time()
    max=0
    max_amigo=()
    for i in map.get(catalog['vertices'],Id):
        if is_friend(catalog,Id,i):
            if al.size(map.get(catalog['vertices'],i))>=max:
                max=(al.size(map.get(catalog['vertices'],i)),i)
                max_amigo=i
    end_time=get_time()
    delta_time=end_time-start_time
    return max,max_amigo,delta_time

def is_friend(catalog,A_id, B_id):
    if al.is_present(map.get(catalog['vertices'],A_id),B_id):
        if al.is_present(map.get(catalog['vertices'],B_id),A_id):
            return True
    else:
        return False
```

Descripción

Pasa por la lista de adyacencia del usuario 1, revisa si cada seguido es seguidor también, y va buscando el que tenga máximos seguidores

Entrada	El grafo, el Id de la persona a analizar
Salidas	Amigo con más seguidores, y el número de seguidores que tiene
Implementado (Sí/No)	Si, Pablo Castrillon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Consigue la lista de adyacencia de Id	O(1)
Revisa quienes son amigos	O(N)
Compara el número de seguidos	O(1)
Total	O(N)

-

Requerimiento 4

```
def req_4(catalog,Id_A,Id_B):
    start_time=get_time()
    amigos_comun=[]
    for i in map.get(catalog['vertices'],Id_A)['elements']:
        if al.is_present(map.get(catalog['vertices'],Id_B),i):
            amigos_comun.append(i)
    end_time=get_time()
    delta_time=end_time-start_time
    return amigos_comun,delta_time
```

Descripción

Revisa los seguidos de A, y revisa si están en los seguidos de B. Si sí es así, los anota en la lista de amigos en común

Entrada	El catálogo, ambos Id

Salidas	Lista de amigos en común
Implementado (Sí/No)	Si, Sergio Laverde

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Pasar por la lista de seguidos de A	O(N)
Buscar en la lista de B	O(N)
Insertar	O(1)
TOTAL	***O(N)***

- N es el numero de elementos.

Requerimiento 5

```
def req_5(catalog, Id, N):
    """
    Retorna el resultado del requerimiento 5
    """
    adlist = map.get(catalog['vertices'],Id)
    adlist2 = al.new_list()
    for i in adlist['elements']:
        if is_friend(catalog,Id,i):
            al.add_last(adlist2,i)

    al.quick_sort(adlist2,sortcrit5)
    list = al.sub_list(adlist,0,N)
    return list

def sortcrit5(catalog,Id1,Id2):
    is_sorted = False
    if gr.out_degree(catalog,Id1) > gr.out_degree(catalog,Id2):
        is_sorted = True
    return is_sorted
```

Descripción

Se recorre el índice de fechas y se asigna la franja horaria, con la cual se añade a las diferentes listas

Entrada	Catalog(Grafo) Id, y el número de amigos
Salidas	Listas con los usuarios con más seguidos

Implementado (Sí/No)	Si, David Monroy

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Consigue la lista de adyacencia de Id	O(1)
Revisa si son amigos	O(N)
Organiza los amigos por número de seguidores	O(n log(n))
TOTAL	***O(N)***

.