ANÁLISIS DEL RETO

Estudiante 1, código 1, email 1

Estudiante 2, código 2, email 2

Nikol Katherin Rodriguez Ortiz, 202317538, nk.rodriguez@uniandes.edu.co

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

en las pruebas realizadas se usan los datos del ejemplo

• Origen: o Latitud: 4.60293518548777, o Longitud: -74.06511801444837 • Destino: o Latitud: 4.693518613347496, o Longitud: -74.13489678235523

Graficas

En este requerimiento no puede tener una grafica ya que solo contamos con un archivo.

Análisis

Requerimiento Ejemplo

Descripción

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.

M	S	T

s	D	0
5	D	0
1	D	1
2	D	2
3	D	4
5	D	7
8	D	1
I	D	2

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>>

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Punto de origen (una localización geográfica con latitud y longitud). Punto de destino (una localización geográfica con latitud y longitud).
Salidas	La distancia total que tomará el camino entre el punto de origen y el de destino. El total de vértices que contiene el camino encontrado. La secuencia de vértices (sus identificadores) que componen el camino encontrado
Implementado (Sí/No)	Si, Santiago Rojas.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Declarar las estructuras de datos.	O(1)
Paso 2 Encontrar los vértices cercanos.	O(N)
Paso 3 Búsqueda Dijkstra	O(V + E)
TOTAL	***O(...)***

Pruebas Realizadas

en las pruebas realizadas se usan los datos del ejemplo

• Origen: o Latitud: 4.60293518548777, o Longitud: -74.06511801444837 • Destino: o Latitud: 4.693518613347496, o Longitud: -74.13489678235523

Graficas

En este requerimiento no puede tener una grafica ya que solo contamos con un archivo.

Análisis

Paso 1:

```
def req_1(model, latO, longO, latD, longD):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1

    totalV = 0
    totalD = 0
    hashmap = model["info"]
    listahash = mp.valueSet(hashmap)
```

se inicializan las variables que serán usadas. O(1)

Paso 2:

```
sVertex = verticeCercano(listahash, latO, longO)
eVertex = verticeCercano(listahash, latD, longD)
```

Para encontrar el vértice más cercano, hay que recorrer cada nodo del grafo, por tanto la complejidad es O(N).

Paso 3:

```
djResult = djk.Dijkstra(model["bogota"], sVertex)
totalV = 0


if djk.hasPathTo(djResult, eVertex):

        path = djk.pathTo(djResult, eVertex)


        path_list = list(lt.iterator(path))

        camino = path_list


totalV = len(path_list)

for arco in camino:
    totalD += float(arco["weight"])
nCamino = []
for arco in camino:
    if not(arco["vertexA"] in nCamino):

        nCamino.append(arco["vertexA"])
    if not(arco["vertexB"] in nCamino):

        nCamino.append(arco["vertexB"])
#totalD = calcularDistanciaEnCamino(hashmap, camino)

#print(camino)

return nCamino, totalD, totalV
```

El algoritmo de Dijkstra tiene una complejidad de O(V+E) donde representan los vértices y arcos del grafo.

Requerimiento <<2>>

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Punto de origen (una localización geográfica con latitud y longitud). Punto de destino (una localización geográfica con latitud y longitud).
Salidas	La distancia total que tomará el camino entre el punto de origen y el de destino. El total de vértices que contiene el camino encontrado. La secuencia de vértices (sus identificadores) que componen el camino encontrado
Implementado (Sí/No)	Si, Santiago Rojas.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Declarar las estructuras de datos.	O(1)
Paso 2 Encontrar los vértices cercanos.	O(N)
Paso 3 Busqueda BFS	O(V + E)
TOTAL	***O(...)***

Pruebas Realizadas

en las pruebas realizadas se usan los datos del ejemplo

• Origen: o Latitud: 4.60293518548777, o Longitud: -74.06511801444837 • Destino: o Latitud: 4.693518613347496, o Longitud: -74.13489678235523

Graficas

En este requerimiento no puede tener una grafica ya que solo contamos con un archivo.

Análisis

Paso 1:

```
def req_1(model, latO, longO, latD, longD):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1

    totalV = 0
    totalD = 0
    hashmap = model["info"]
    listahash = mp.valueSet(hashmap)
```

se inicializan las variables que serán usadas. O(1)

Paso 2:

```
sVertex = verticeCercano(listahash, latO, longO)
eVertex = verticeCercano(listahash, latD, longD)
```

Para encontrar el vértice más cercano, hay que recorrer cada nodo del grafo, por tanto la complejidad es O(N).

Paso 3:

```
bfsResult = bfs.BreathFirstSearch(model["bogota"], sVertex)
totalV = 0


if bfs.hasPathTo(bfsResult, eVertex):

        path = bfs.pathTo(bfsResult, eVertex)


        path_list = list(lt.iterator(path))

        camino = path_list

totalV = len(path_list)
totalD = calcularDistanciaEnCamino(hashmap, camino)


return camino, totalD, totalV
```

El algoritmo de BFS tiene una complejidad de O(V+E) donde representan los vértices y arcos del grafo.

Requerimiento <<3>>

Descripción

Este requerimiento encuentra el camino de menor costo de fibra optica para pasar por los m puntos con mayor numero de comparendos

Entrada	La cantidad de cámaras de video que se desean instalar (M). La localidad donde se desean instalar.
Salidas	El total de vértices de la red.

	Los vértices incluidos (identificadores). Los arcos incluidos (Id vértice inicial e Id vértice final). La cantidad de kilómetros de fibra óptica extendida. El costo (monetario) total.
Implementado (Sí/No)	Si se implementó: Juan Felipe Ruiz Sosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Declarar las estructuras de datos y recorrer los nodos en busca de los n con mas comparendos	O(N)
Paso 2 Usar algoritmo de Dijsktra	O(ElogV)
Paso 3 Buscar la distancia entre los caminos para saber el costo	O(V + E)
TOTAL	***O(***ElogV***)***

Complejidad

la complejidad total es O(ElogV)

.

Pruebas Realizadas

en las pruebas realizadas se usan los datos del ejemplo

• 20 cámaras en la localidad de Chapinero.

Gráficas

En este requerimiento no puede tener una gráfica ya que solo contamos con un archivo.

Análisis

El requerimiento 3 busca el camino de menos costo que una los n vertices que tengan mayor numero de comparendos en una localidad dada. Como toca recorrer todo el grafo principal para saber el camino mas corto y usamos el algoritmo de Dijsktra la complejidad es O(ElogV)

Requerimiento <<6>>

Descripción

Este requerimiento busca obtener los caminos más cortos para que los policías atiendan los M comparendos más graves

Entrada	La cantidad de comparendos que se desea responder (M) • La estación de policía más cercana al comparendo más grave.
Salidas	El total de vértices del camino. Los vértices incluidos (identificadores). Los arcos incluidos (Id vértice inicial e Id vértice final). La cantidad de kilómetros del camino.
Implementado (Sí/No)	Si se implementó: Juan Felipe Ruiz Sosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Declarar las estructuras de datos y recorrer los nodos en busca de los n comparendos mas graves	O(N)

Paso 2 Usar algoritmo de Dijsktra	O(ElogV)
Paso 3 Buscar la distancia entre los caminos	O(V + E)
TOTAL	***O(***ElogV***)***

Complejidad

la complejidad total es O(ElogV)

.

Pruebas Realizadas

en las pruebas realizadas se usan los datos del ejemplo

• 20 comparendos más graves de la ciudad

Gráficas

En este requerimiento no puede tener una gráfica ya que solo contamos con un archivo.

Análisis

El requerimiento 6 busca el camino de menor costo que una los n vértices que tengan los comparendos mas graves partiendo desde una estación de policia. Como toca recorrer todo el grafo principal para saber el camino más corto y usamos el algoritmo de Dijsktra la complejidad es O(ElogV)

Requerimiento <<7>>

Descripción

Este requerimiento encuentra el camino “más corto” en términos del número de menor cantidad de comparendos entre dos puntos geográficos localizados en los límites de la ciudad de Bogotá.

Entrada	Punto de origen (una localización geográfica con latitud y longitud). Punto de destino (una localización geográfica con latitud y longitud)
Salidas	El total de vértices del camino. Los vértices incluidos (identificadores). Los arcos incluidos (Id vértice inicial e Id vértice final). La cantidad de comparendos del camino. o La cantidad de kilómetros del camino.
Implementado (Sí/No)	Si se implementó: Nikol Rodriguez

Análisis de complejidad

parte 1

```
def req_7(model, lato, longo, latd, longd):

    """

    Función que soluciona el requerimiento 7

    """

    # TODO: Realizar el requerimiento 7

    grafo = model["grafo_comparendos"]

    hashmap = model["info"]

    listahash = mp.valueSet(hashmap)

    distancia_origen = math.inf

    info_cercano_origen = None

    camino = []
```

La complejidad es O(1) sin tener en cuenta la asignación `mp.getvalueset`, ya que se debe realizar una búsqueda lineal, lo cual es O(N).

Parte dos.

```
for submapa in lt.iterator(listahash):

        coordenadas = me.getValue(mp.get(submapa, 'coordenadas'))

        lat = coordenadas[1]

        long = coordenadas[0]

        distancy1 = calculate_distancy(lat, long, lato, longo)

        if distancy1 < distancia_origen:

            distancia_origen = distancy1

            info_cercano_origen = submapa

    distancia_destino = math.inf

    info_cercano_destino = None

    for submapa in lt.iterator(listahash):

        coordenadas = me.getValue(mp.get(submapa, 'coordenadas'))

        lat = coordenadas[1]

        long = coordenadas[0]

        distancy2 = calculate_distancy(lat, long, latd, longd)

        if distancy2 < distancia_destino:

            distancia_destino = distancy2

            info_cercano_destino = submapa
```

La complejidad de esta parte es O(V) donde V son los vértices que se recorren en la lista hash para conocer el vértice más cercano.

Parte 3

```
origen = me.getValue(mp.get(info_cercano_origen, "id"))

    destino = me.getValue(mp.get(info_cercano_destino, "id"))
```

```
    print('Origen:', origen, '. Destino:', destino)

    search = djk.Dijkstra(grafo, origen)

    haspath = djk.hasPathTo(search, destino)

    print(haspath)

    if haspath:

        pathTo = djk.pathTo(search, destino)

        for minipath in lt.iterator(pathTo):

            if camino == []:

                camino.append(minipath['vertexA'])

            camino.append(minipath['vertexB'])



    return camino
```

La complejidad de esta parte es O(ElogV) ya que se usa el algoritmo de camino más corto Dijkstra

Complejidad

la complejidad total es O(ElogV)

.

Pruebas Realizadas

en las pruebas realizadas se usan los datos del ejemplo

• Origen: o Latitud: 4.60293518548777, o Longitud: -74.06511801444837 • Destino: o Latitud: 4.693518613347496, o Longitud: -74.13489678235523

Gráficas

En este requerimiento no puede tener una gráfica ya que solo contamos con un archivo.

Análisis

El requerimiento 7 aborda la búsqueda del camino más corto en términos de cantidad de comparendos entre dos ubicaciones geográficas en Bogotá.La implementación se basa en estructuras de datos como grafos y utiliza el algoritmo de Dijkstra para encontrar rutas eficientes. El análisis de complejidad proporciona una visión clara del rendimiento del algoritmo en diferentes etapas que en total es O(ElogV). Se han realizado pruebas satisfactorias con datos de ejemplo.

Requerimiento Ejemplo

Descripción

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.

M	S	T

s	D	0
5	D	0
1	D	1
2	D	2
3	D	4
5	D	7
8	D	1
I	D	2

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.