Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

Angel Farfan Arcila, 202222183, a.farfana@uniandes.edu.co

Andrés Cáceres, 202214863, a.caceresg@uniandes.edu.co

Estudiante 3, código 3, email 3

Requerimiento <<0,carga de datos>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	archivos csv
Salidas	datastructs con todos los datos necesarios
Implementado (Sí/No)	Sí, Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(M)
Paso 2	O(N)
Paso 3	O(1)
Paso 4	O(1)
Paso 5	O(P)
Paso 6	O(1)
Paso 7	O(Q)
Paso 8	O(C)
TOTAL	***O(N)***

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).

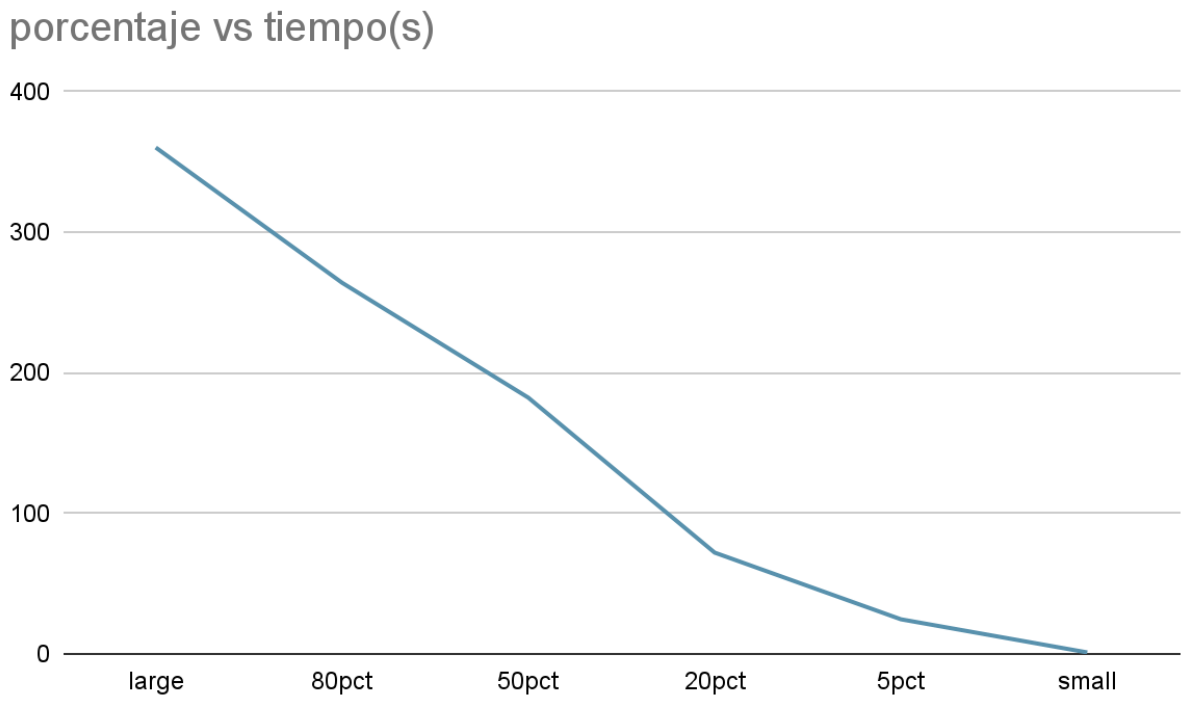
Entrada	**Tiempo (s)**
large	360,27

80pct	264,05
50pct	182,27
20pct	**72,05**
5pct	**24,60**
small	**1,04**

Tablas de datos
Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
large	360,27
80pct	264,05
50pct	182,27
20pct	**72,05**
5pct	**24,60**
small	**1,04**

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

En primer lugar, la carga de datos implica leer dos archivos CSV. Este proceso tiene una complejidad de O(n), donde n es el número de líneas en cada archivo. La lectura de cada línea y la creación de diccionarios se consideran operaciones de tiempo constante.A continuación, hay un bucle for que procesa los datos. Este bucle tiene una complejidad de O(n), ya que itera sobre las líneas del archivo "BA-Grey-Wolf-tracks-utf8-Large.csv". Dentro de este bucle, se realizan operaciones como el formateo de fechas y horas, redondeo y conversión de valores, y asignación de claves y valores a las estructuras de datos. Cada una de estas operaciones también se considera de tiempo constante.Después del bucle principal, hay dos bucles adicionales. El primero itera sobre los elementos de una tabla hash y tiene una complejidad de O(m), donde m es el número de elementos en la tabla. El segundo bucle itera sobre los elementos de una lista y tiene una complejidad de O(p), donde p es el número de elementos en la lista.Dentro de estos bucles, se realizan operaciones como comparaciones, asignaciones, concatenación de cadenas y agregado de elementos a las estructuras de datos. Cada una de estas operaciones también se considera de tiempo constante.Luego, hay otro conjunto de bucles y operaciones para construir una lista y un grafo. Estos bucles tienen una complejidad de O(q), donde q es el número de elementos en una tabla hash y una lista. Las operaciones dentro de estos bucles, cómo agregar elementos a la lista y al grafo, también son de tiempo constante.Finalmente, hay un bucle para construir otra lista y dos bucles para construir listas de valores. Estos bucles tienen complejidades de O(1) en promedio, ya que se iteran sobre listas de longitud fija o realizan operaciones simples como agregar elementos.

Requerimiento <<1>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	identificadores de puntos de encuentro
Salidas	distancia total,total nodos,5 primeros y últimos vértices con sus datos.
Implementado (Sí/No)	Sí, Andrés Cáceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 dfs	O(V+E)
Paso 2 pathto	O(V)
Paso bucle while	O(V)
TOTAL	***O(V+E)***

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).

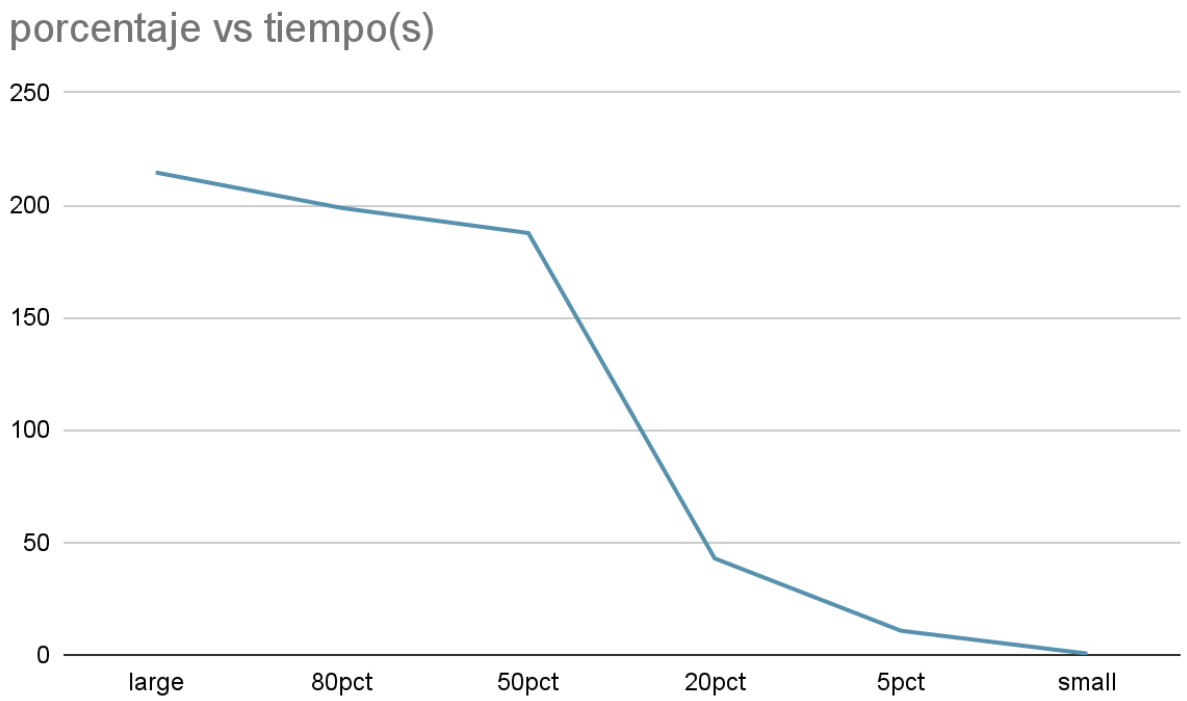
Entrada	**Tiempo (s)**
large	214,53
80pct	198,71
50pct	187,58
20pct	42,91
5pct	10,73
small	0,57

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (s)
large	214,53
80pct	198,71
50pct	187,58
20pct	42,91
5pct	10,73
small	0,57

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

La función realiza una búsqueda en profundidad en el grafo utilizando el algoritmo Depth-First Search (DFS). La complejidad de esta operación depende del tamaño del grafo, que está determinado por la cantidad de vértices y aristas.

A continuación, se calcula un camino desde el nodo de origen hasta el nodo de destino utilizando la función dfs.pathTo(). La complejidad de esta operación depende del tamaño del camino, que está determinado por la longitud del camino calculado mediante la búsqueda en profundidad.

El bucle while itera sobre los elementos del camino calculado. La complejidad de este bucle depende del tamaño del camino.

La función utiliza estructuras de datos como pilas y listas para almacenar y manipular información.

En algunos casos, se realizan operaciones de duplicación de datos, como la copia de elementos de una lista a otra. Esto puede afectar la eficiencia de la implementación.

Algunas operaciones se realizan de forma iterativa, como el cálculo de grados de vértices y la iteración sobre elementos de una lista. Si el tamaño de estas estructuras de datos es grande, podría haber un impacto en la eficiencia del algoritmo.

Requerimiento <<2>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	identificadores de puntos de encuentro de origen y destino
Salidas	camino más corto entre dos puntos de encuentro y sus datos, 5 primeros y 5 últimos.
Implementado (Sí/No)	Sí, Andrés Cáceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 BellmanFord	O(V*E)
Paso 2 pathto	O(V)
Paso 3 cada bucle for	O(V)
TOTAL	***O(V*E)***

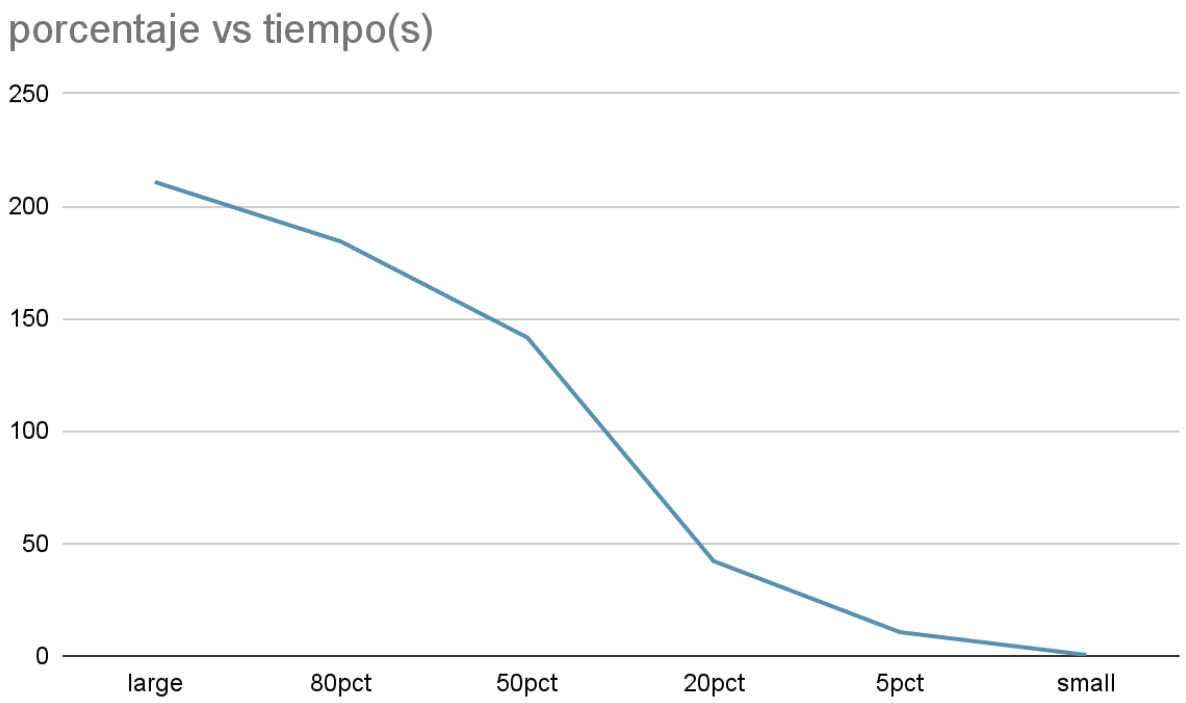
Pruebas Realizadas
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (s)
large	210,72
80pct	184,22
50pct	141,56
20pct	42,14
5pct	10,54
small	0,43

Tablas de datos
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (s)
large	210,72
80pct	184,22
50pct	141,56
20pct	42,14
5pct	10,54
small	0,43

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

La función req_2 utiliza el algoritmo de Bellman-Ford para encontrar el camino más corto desde el nodo de origen hasta el nodo de destino en el grafo. A continuación, se realizan una serie de operaciones en el bucle for para calcular diferentes valores relacionados con el camino y generar una salida.

En cuanto a la complejidad, se mencionó previamente que el algoritmo de Bellman-Ford tiene una complejidad de O(V * E), donde V es el número de vértices y E es el número de aristas en el grafo. Además, se deben considerar las operaciones en el bucle for que dependen del tamaño del camino calculado. Por lo tanto, la complejidad total de la función req_2 estará influenciada principalmente por el tamaño del grafo y el tamaño del camino.

Requerimiento 3

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El grafo y la lista de los lobos (Ambas estan en data_structs)
Salidas	Los SCC mas grandes con sus respectivos lobos.
Implementado (Sí/No)	Si se implementó por Juan Jose Diaz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Funcion Kosaraju	
1. sc = scc.KosarajuSCC(data_structs["graph"])	O(V+E)
2. Lt.iterator(mp.keySet(componentes))	O(N)
3. lt.isPresent(marks, valor)	O(M) M siendo el tamaño de la lista marks
4. Paso 2 y 3 combinados	O(N*M) (Aprox O(N^2)
5. Total Kosaraju	O(N^2)
Funcion Top 5 SCC	Paso ….

1. for key1 in lt.iterator(mp.keySet(sccmap))	O(N)
2. Sort	O(NlogN)
3. For I in range (0,5)	O(5)
4. Total Top 5 SCC	O(N)
Funcion Req (Main)	
5. for key2 in lt.iterator(llaves)	O(5) Porque son las llaves del paso 3
6. for node in lt.iterator(lista):	O(N*5) Se aproxima a O(N)
7. Max, min(Latitudes), max, min(longitudes)	O(N * 5 * 4) Se aproxima a O(N)
8. lt.iterator(data_structs["list_individuals"]	O(N * 5 * 46) Se aproxima a O(N)
9. for t3 in lt.iterator(top3)	O(N * 5 * 46 * 3) Aproxima a O(N)
10. for wolf in lt.iterator(lista_final):	O(N * 5 * X) (2 <= X <= 6) Se aproxima a O(N)
Funcion lista_lobos_fix	
1. for element in lt.iterator(lista_lobos):	O(Y*5) (10 <= Y <= 30) Se aproxima a O(1)
Funcion get_nodes	
1. for node in lt.iterator(top3):	O(3) Aprox O(1)

2. for nodo in lt.iterator(bot3):	O(3) Aprox O(1)
Total cada func:	O(N^2) + O(N) + O(N) + O(1) + O(1)
TOTAL REQ 3	**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).

	Tiempo (s)
Small	0.22s
20	0.98s
50	1.67s
Large	5.2s

Procedimiento

MacBook Pro 2018 intel core i7, librería time python con el uso de decoradores en las funciones.

Graficas

La información obtenida vs la gráfica de O(N^2).

Análisis

Al haber hecho la gráfica hemos obtenido que es bastante parecida, lo cual indica que la aproximación que hemos hecho a O(N^2) está bien, sin embargo hay que tener en cuenta que no es preciso ya que se cancelaron muchas complejidades que podrían haber aumentado el número, por ejemplo todos los O(5*N) que pasaron durante el ciclo en el que se recorre los top 5 nodos es un número muy importante ya que son varios ciclos. De igual forma, si se corre el requerimiento con todas las pestañas cerradas y con el modo normal en vez del debug el tiempo es muy bajo teniendo en cuenta la cantidad de operaciones hechas, con esto podemos afirmar que nuestra solución al requerimiento 3 es una solución rápida y efectiva para la tarea a realizar.

Requerimiento <<4>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Origen latitud y longitud, Destino latitud y longitud
Salidas	lon_lat_1_nearest[1], lon_lat_2_nearest[1], total_weight, len(hiper_nodes_route), number_nodes_individuals,total_segments,list_3_first_last['elements']
Implementado (Sí/No)	Ángel Farfán Arcila

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de listas lon_lat_1_list y lon_lat_2_list	O(N)
Ordenamiento de listas lon_lat_1_list y lon_lat_2_list:	O(N log N)
Realización de búsqueda de ruta utilizando el algoritmo de Dijkstra:	O(E + V log V)
list_vertices_path	O(P)
Procesamiento de vértices y construcción de listas adicionales:	O(S)
list_3_first_last	O(M)
Distho	O(N^2)
Paso Último	O(V + E)
TOTAL	O(N^2)

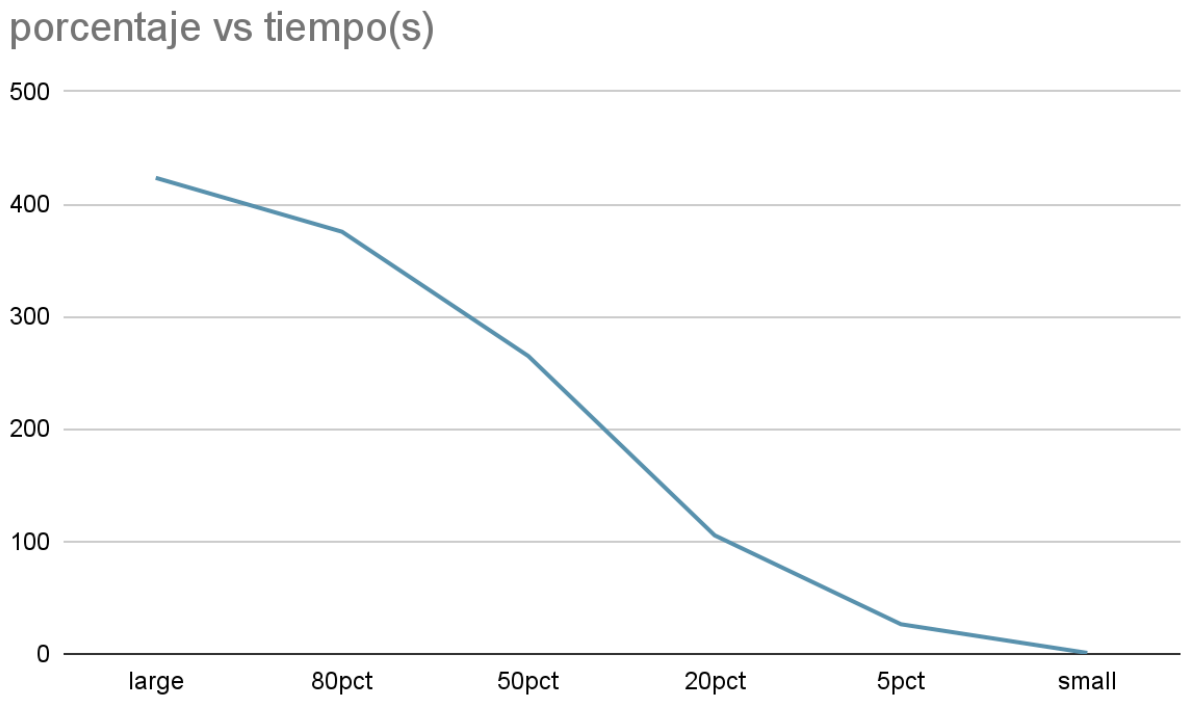
Pruebas Realizadas
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (s)
large	423,35
80pct	375,41
50pct	264,84
20pct	105,42
5pct	26,36
small	0,86

Tablas de datos
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (s)
large	423,35
80pct	375,41
50pct	264,84
20pct	105,42
5pct	26,36
small	0,86

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis
El análisis de la complejidad en términos de Big O notation para el código proporcionado es el siguiente:

La creación de las listas `lon_lat_1_list` y `lon_lat_2_list` es la primera parte del código. En esta sección, se recorre `data_structs['list_hiper_nodes']` y se agregan elementos a las listas `lon_lat_1_list` y `lon_lat_2_list`. La complejidad de esta operación depende del tamaño de `data_structs['list_hiper_nodes']`, que podemos llamar 'n'. Por lo tanto, la complejidad es O(n). Después de crear las listas `lon_lat_1_list` y `lon_lat_2_list`, se realiza un ordenamiento utilizando la función `quk.sort`. El tiempo de ordenamiento depende del tamaño de las listas, que en el peor caso es 'n'. Por lo tanto, la complejidad es O(n log n) en promedio. Luego, se utiliza la función `lt.firstElement` para obtener el primer elemento de las listas ordenadas `lon_lat_1_list` y `lon_lat_2_list`. Esta operación es constante y no depende del tamaño de las listas. Por lo tanto, la complejidad es O(1). En la siguiente sección, se realizan operaciones para convertir los valores de `lon_lat_1_nearest` y `lon_lat_2_nearest` a cadenas y se reemplazan algunos caracteres. Estas operaciones son proporcionales a la longitud de las cadenas resultantes, pero se consideran operaciones de tiempo constante en la notación Big O. Por lo tanto, la complejidad es O(1). Después de eso, se realiza una búsqueda del camino más corto utilizando el algoritmo de Dijkstra en el grafo `data_structs['graph']`. La complejidad de este algoritmo depende del número de nodos y aristas en el grafo. Si llamamos 'V' al número de nodos y 'E' al número de aristas, entonces la complejidad en el peor caso es O((V + E) log V). A continuación, se recorre el

camino más corto obtenido y se realizan operaciones en función de los nodos visitados. La complejidad de este recorrido depende del tamaño del camino más corto, que podemos llamar 'm'. Por lo tanto, la complejidad es O(m). Finalmente, se crea la lista `list_3_first_last` y se agregan elementos a ella. En esta parte, se recorren los elementos de `hiper_nodes_route` y se realizan operaciones en función de cada elemento. La complejidad de esta operación depende del tamaño de `hiper_nodes_route`, que en el peor caso es 'n'.

Requerimiento <<5>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	identificadores de 2 puntos de encuentro, máxima distancia a recorrer, mínima cantidad de nodos.
Salidas	Camino más largo entre 2 puntos de encuentro y todos sus datos.
Implementado (Sí/No)	En parte sí. Andrés Cáceres

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

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)**

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Gráficas
Las gráficas con la representación de las pruebas realizadas.

Análisis
Análisis de resultados de la implementación, tener en cuenta las pruebas realizadas y el análisis de complejidad.

El req_5 se empezó y se dejó bastante avanzado, pero por cuestiones de tiempo y disponibilidad no se pudo terminar.

Requerimiento <<6>>
Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción
Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Fecha inicial del análisis (con formato "%Y-%m-%d"). Fecha final del análisis (con formato "%Y-%m-%d") El sexo registrado del animal (animal-sex).
Salidas	list_individual_short_char['elements'][0],shortest_path[1],len(hiper_nodes_route_shortest),path_shortest_size,list_3_first_last_shortest,list_individual_large_char['elements'][0],larger_path[1],len(hiper_nodes_route_larger),path_larger_size,list_3_first_last_larger
Implementado (Sí/No)	Ángel Farfán Arcila

Análisis de complejidad
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(N)
Paso 2	O(N^2)
Paso 3	O(1)
Paso 4	O((V + E) log V)
Paso 5	O(1)
	O(V + E)
TOTAL	***O(***N^2)

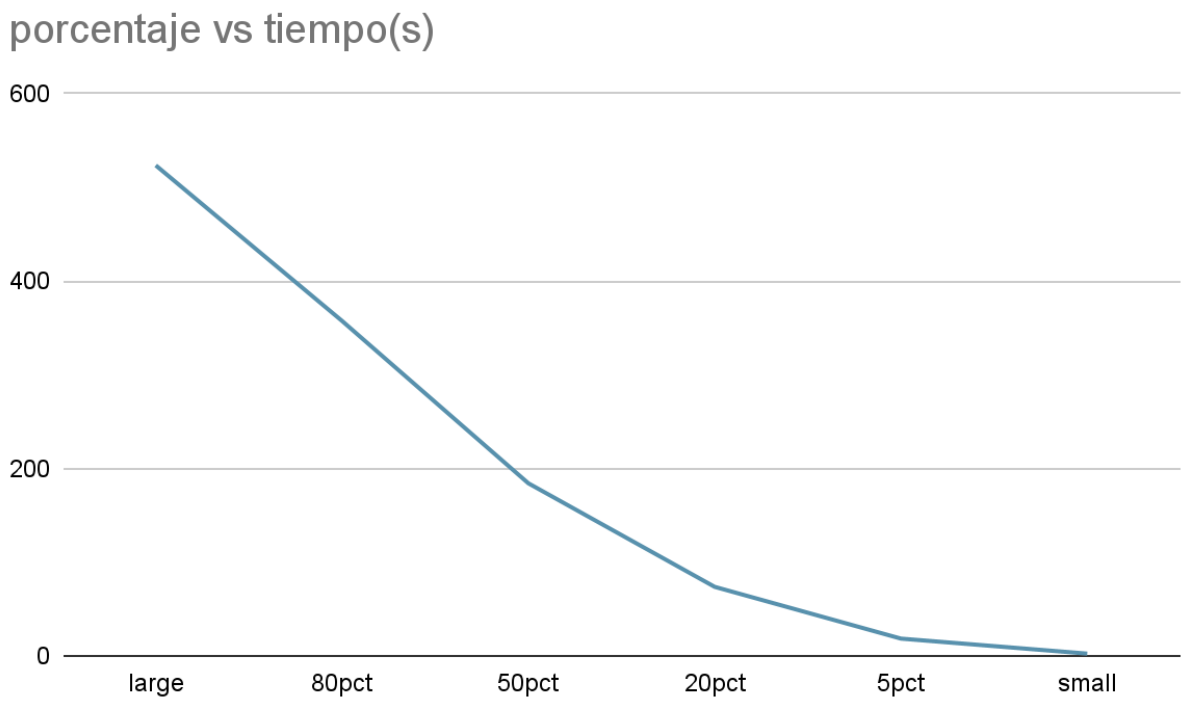
Pruebas Realizadas
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (s)
large	523,47
80pct	357,46
50pct	184,19
20pct	73,68
5pct	18,43
small	2,41

Tablas de datos
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (s)
large	523,47
80pct	357,46
50pct	184,19
20pct	73,68
5pct	18,43
small	2,41

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

El bucle inicial tiene una complejidad de O(N), donde N es el tamaño de la lista de individuos. Dentro del bucle, se realizan operaciones de inserción en listas, que generalmente tienen una complejidad de O(1) en promedio. Además, el bucle de filtrado y el bucle de búsqueda de caminos más cortos iteran sobre tablas hash, y su complejidad promedio se considera O(1) por cada elemento. El algoritmo de Dijkstra se utiliza para encontrar los caminos más cortos en un grafo. Su complejidad depende del número de vértices y aristas en el grafo. En el código proporcionado, no se especifica la implementación utilizada para el grafo, por lo que no se puede determinar la complejidad exacta. Sin embargo, en general, el algoritmo de Dijkstra tiene una complejidad de O((V + E) log V), donde V es el número de vértices y E es el número de aristas. En conclusión, el código proporcionado tiene principalmente complejidad lineal (O(n)) en el bucle inicial y complejidad constante (O(1)) en las operaciones de inserción y acceso a listas y mapas. Las complejidades del algoritmo de Dijkstra y las operaciones en las tablas hash dependen de las implementaciones específicas utilizadas, pero en general, pueden considerarse como O(1) en promedio si se manejan de manera eficiente.

Requerimiento <<7>>

Plantilla para documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	• Fecha inicial del análisis (con formato "%Y-%m-%d”). • Fecha final del análisis (con formato "%Y-%m-%d”). • Temperatura ambiente mínima (en grados centígrados). • Temperatura ambiente máxima (en grados centígrados).
Salidas	scc.connectedComponents(scc_graph),rows['elements'],part_2['elements']
Implementado (Sí/No)	Ángel Farfán Arcila

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(N)
Paso 2	O(M)
Paso 3	O(N LOG N)
Paso 4	O(Q)
Paso 5	O(R)
Paso 6	O(S)
Paso 7	O(N^2)
Paso 8	O(V)
Paso 9	O((V + E) log V)
Paso 10	O(V + E)
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).

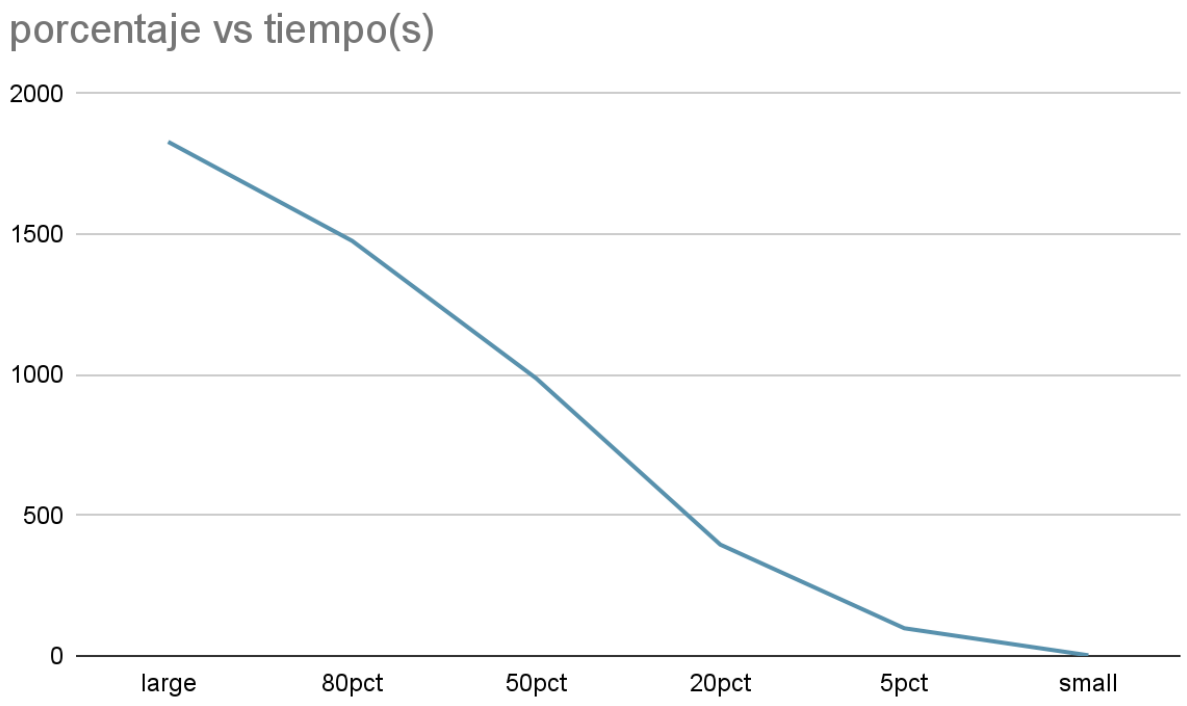
Entrada	Tiempo (s)
large	+1828,68
80pct	+1476,23
50pct	+987,23
20pct	395,85
5pct	98,27

small	2,06

Tablas de datos
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (s)
large	+1828,68
80pct	+1476,23
50pct	+987,23
20pct	395,85
5pct	98,27
small	2,06

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

La creación de estructuras de datos vacías (grafos, listas y mapas) generalmente se considera O(1) en términos de complejidad, ya que implica la inicialización de una estructura vacía. Sin embargo, las operaciones de formateo y análisis de fechas pueden tener una complejidad lineal, O(n), donde n es la longitud de la cadena de fecha. Por otro lado, los bucles que recorren estructuras de datos como listas y tablas de hash tienen una complejidad dependiente del tamaño de los datos en esas estructuras, que denotaremos como m. Por lo tanto, la complejidad sería O(m). Además, algunas operaciones de inserción y búsqueda en las estructuras de datos también tienen una complejidad dependiente del tamaño de los datos, nuevamente denotado como m, lo que implica una complejidad O(m). En cuanto al algoritmo de búsqueda de componentes fuertemente conectados (SCC), su complejidad es de O(V + E), donde V es el número de vértices y E es el número de aristas en el grafo. Esto significa que la complejidad del algoritmo está relacionada con el tamaño del grafo y la cantidad de conexiones entre los vértices. Por otro lado, las operaciones de ordenamiento pueden tener una complejidad de O(n log n) en el peor de los casos, donde n es el tamaño de la lista que se desea ordenar. Esto implica que la complejidad del ordenamiento aumenta a medida que crece el tamaño de la lista.