**Departamento de Ingeniería de Sistemas y** 

**Computación**

**Estructuras de Datos y Algoritmos**

**ISIS-1225**

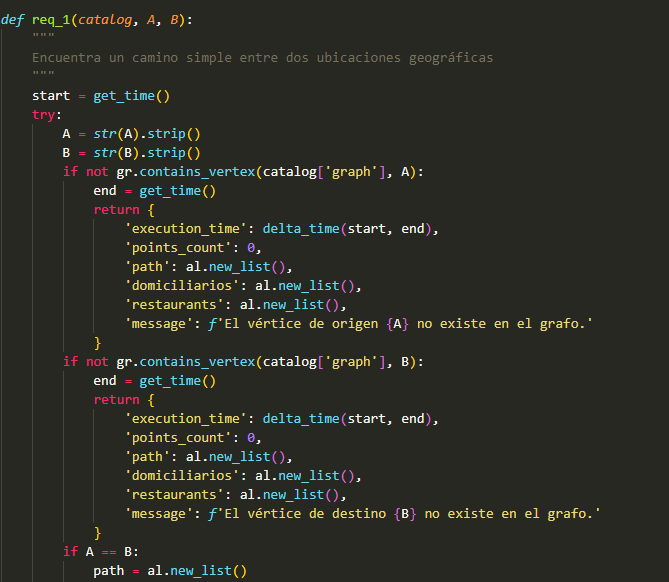
**ANÁLISIS DEL RETO**

*Estudiante 1, código 1, email 1*

*Estudiante 2, código 2, email 2*

**Requerimiento <<1>>**

**Descripción**

****

Esta función encuentra un camino entre dos ubicaciones geográficas específicas en una red de entregas de comida. Primero verifica que ambas ubicaciones (punto A y punto B) existan efectivamente en el sistema, y si alguna no existe, retorna un mensaje de error indicando cuál es el problema. Si ambos puntos son la misma ubicación, simplemente retorna información básica sobre ese lugar único, incluyendo qué domiciliarios han trabajado allí y si es un restaurante.

Cuando las ubicaciones son diferentes, la función utiliza un algoritmo de búsqueda para encontrar una ruta que conecte ambos puntos a través de la red de entregas. Si logra encontrar un camino viable, recorre toda la ruta identificada y recopila información valiosa: qué domiciliarios han operado en cada punto del camino (sin repetir nombres), cuáles de esas ubicaciones corresponden a restaurantes, y la secuencia completa de puntos que conforman la ruta. Al final, retorna toda esta información organizada junto con estadísticas como el tiempo que tomó realizar la búsqueda y la cantidad total de puntos en el camino encontrado.

| **Entrada** | Catalog, a= Coordenadas del punto de origen, b =Coordenadas del punto de destino |
| --- | --- |
| **Salidas** | resultado |
| **Implementado (Sí/No)** | Si |

**Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Preparación inicial | O(1) |
| Validación de existencia de vértices | O(1) |
| Manejo de caso especial (A == B) | O(D\_A) donde D\_A = domiciliarios en punto A |
| Búsqueda DFS | O(V + E) donde V = vértices, E = aristas |
| Verificación de camino existente | O(1) |
| Construcción del camino | O(P) donde P = longitud del camino |
| Extracción de domiciliarios únicos | O(P × D\_avg × P) ≈ O(P² × D\_avg)  P ubicaciones en el camino  D\_avg domiciliarios promedio por ubicación  P comparaciones por cada contains() |
| Identificación de restaurantes | O(P × R) donde R = total de restaurantes |
| Preparación del resultado | O(1) |
| ***TOTAL*** | ***O(V + E + P² × D\_max + P × R)***  ***Camino muy largo con muchos domiciliarios por ubicación*** |

**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).

**Procesadores Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. Memoria RAM** 8,00 GB

**Sistema Operativo** Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045)

| **Entrada** | **Tiempo (s)** |
| --- | --- |

| 20 pct | 18.183 |
| --- | --- |
| 40 pct | 8.094 |
| 60 pct | 23.757 |
| 80 pct | 17.635 |
| 100 pct | 12.961 |

**Tablas de datos**

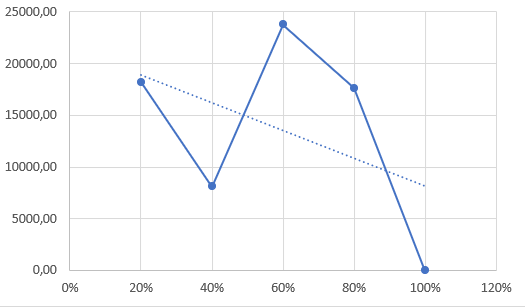
Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| 20 pct |  | 18.183 |
| 40 pct |  | 8.094 |

| 60 pct |  | 23.757 |
| --- | --- | --- |
| 80 pct |  | 17.635 |
| 100 pct |  | 12.961 |

**Graficas**

Las gráficas con la representación de las pruebas realizadas.



**Análisis**

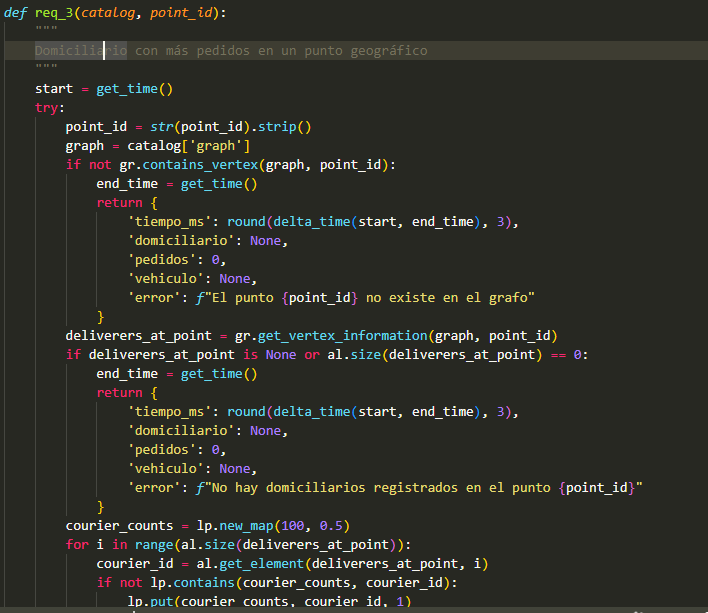
El Requerimiento 1 implementa un algoritmo de búsqueda de caminos que encuentra una ruta simple entre dos ubicaciones geográficas en una red de entregas de comida. Utiliza el algoritmo DFS (Depth-First Search) para explorar el grafo desde el punto origen hasta encontrar el destino, luego reconstruye el camino utilizando backtracking y extrae información adicional como los domiciliarios únicos que han operado en cada ubicación del camino y cuáles puntos corresponden a restaurantes. La complejidad teórica es O(V + E + P² × D\_avg + P × R), donde el componente más costoso es la extracción de domiciliarios únicos debido al uso repetido de la función contains() que genera un comportamiento cuadrático respecto a la longitud del camino.

Los datos de rendimiento muestran un comportamiento altamente irregular que contradice las expectativas de escalabilidad lineal: el dataset del 40% (8.094s) es significativamente más rápido que el del 20% (18.183s), mientras que el del 60% presenta el peor rendimiento (23.757s), y el del 100% (12.961s) es más rápido que varios datasets menores. Esta irregularidad se explica porque el factor dominante no es el tamaño total del dataset sino la estructura específica del grafo, la longitud de los caminos encontrados, y especialmente la distribución de domiciliarios por ubicación. El dataset del 60% probablemente contiene caminos largos con muchos domiciliarios por ubicación, activando el bottleneck cuadrático, mientras que el del 40% tiene una estructura más favorable con caminos cortos y menos domiciliarios repetidos.

La conclusión principal es que el rendimiento del algoritmo depende más de las características estructurales de cada consulta específica que del tamaño absoluto de los datos, lo cual explica la variabilidad observada en los tiempos de ejecución. La optimización más crítica sería reemplazar el uso repetido de contains() por una estructura de datos tipo set() para la extracción de domiciliarios únicos, lo que reduciría la complejidad de O(P²) a O(P) y debería mejorar significativamente los casos problemáticos como el dataset del 60%. Esta optimización sería especialmente efectiva porque el bottleneck identificado es precisamente el que causa la mayor variabilidad en los tiempos observados.

**Requerimiento <<3>>**

**Descripción**

****

El Requerimiento 3 identifica cuál es el domiciliario más activo en una ubicación geográfica específica. La función recibe como entrada las coordenadas de un punto geográfico (como un restaurante o zona de entrega) y analiza todos los registros de entregas que han ocurrido en esa ubicación exacta. Primero verifica que el punto efectivamente exista en el sistema de datos, y si no existe, retorna un mensaje de error indicando que no se encontró esa ubicación en el grafo de entregas.

Una vez confirmado que la ubicación existe, la función examina la lista completa de domiciliarios que han realizado entregas en ese punto específico, contando cuántas veces aparece cada domiciliario en los registros (cada aparición representa un pedido entregado). Después de procesar todos los registros, identifica cuál domiciliario tiene el mayor número de pedidos y retorna información detallada sobre este domiciliario más activo, incluyendo su identificador, la cantidad exacta de pedidos realizados, y información adicional como el tipo de vehículo más común y estadísticas generales del punto analizado.

| Entrada | catalog, point\_id |
| --- | --- |
| **Salidas** | resultado |
| **Implementado (Sí/No)** | Si |

**Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Preparación inicial | O(1) |
| Validación de existencia del punto | O(1) |
| Obtención de lista de domiciliarios | O(1) |
| Conteo de pedidos por domiciliario | O(D) donde D = domiciliarios en el punto |
| Búsqueda del domiciliario más activo | O(K) donde K = domiciliarios únicos |
| Preparación del resultado | O(1) |
| TOTAL | O(D\_max)  Ubicación con el mayor número de registros de domiciliarios |

**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).

**Procesadores Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. Memoria RAM** 8,00 GB

**Sistema Operativo** Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045)

| **Entrada** | **Tiempo (s)** |
| --- | --- |
| 20 pct | 0.808 |
| 40 pct | 0.735 |
| 60 pct | 1.037 |
| 80 pct | 1.168 |
| 100 pct | 5.392 |

**Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

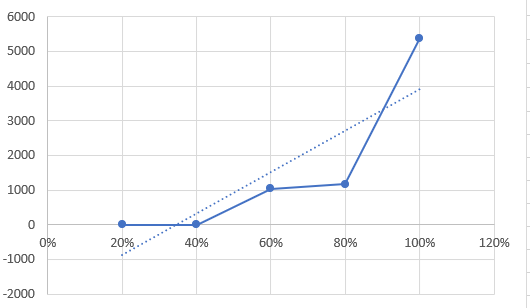
| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |

| 20 pct |  | 0.808 |
| --- | --- | --- |
| 40 pct |  | 0.735 |
| 60 pct |  | 1.037 |
| 80 pct |  | 1.168 |

| 100 pct |  | 5.392 |
| --- | --- | --- |

**Graficas**

Las gráficas con la representación de las pruebas realizadas.



**Análisis**

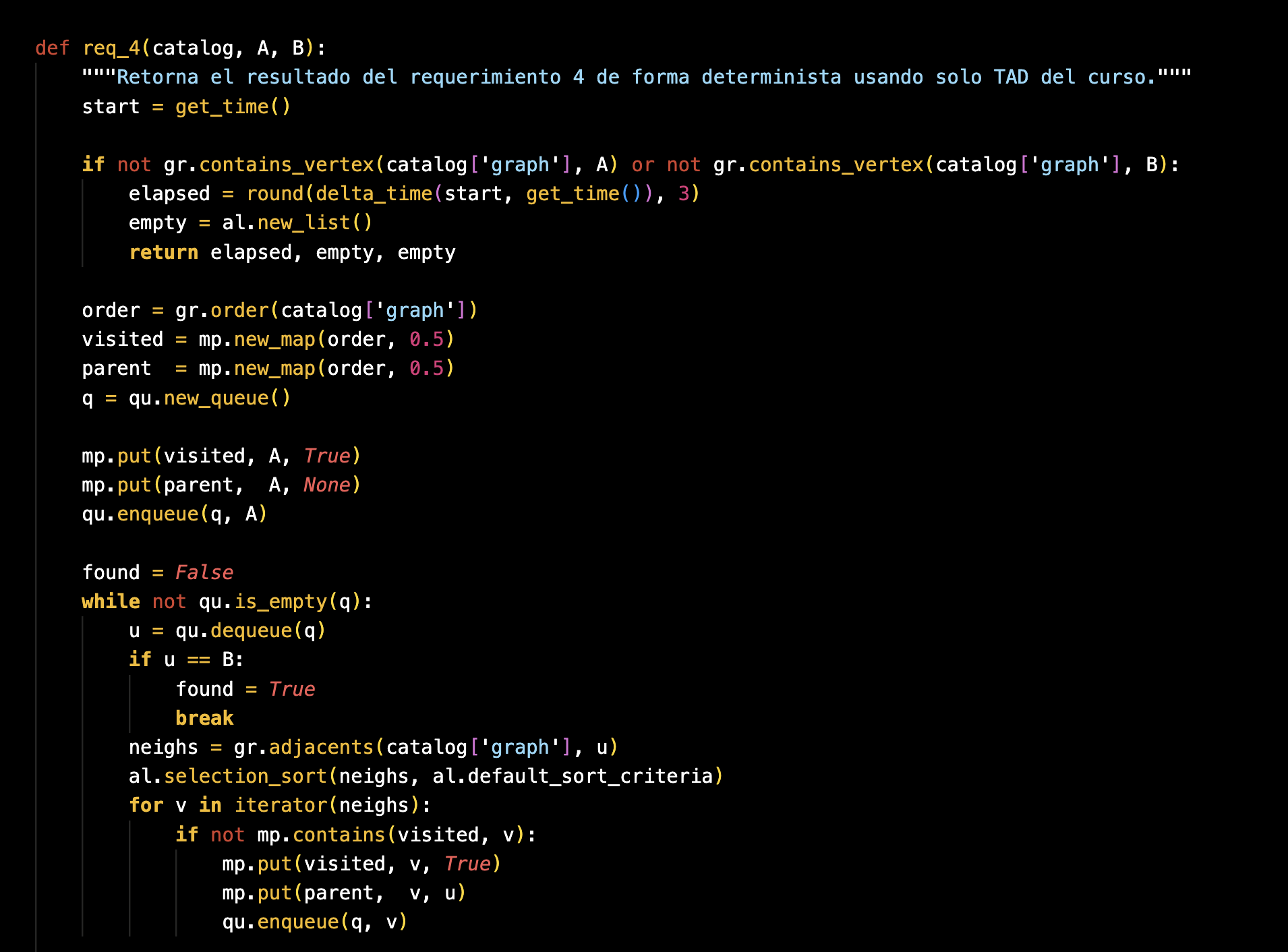
4El Requerimiento 3 implementa un algoritmo de análisis puntual que identifica el domiciliario más activo en una ubicación geográfica específica mediante un proceso de conteo lineal. Su diseño es intrínsecamente eficiente con complejidad O(D), donde D representa únicamente el número de registros de domiciliarios en la ubicación consultada, no el tamaño total del dataset. El algoritmo simplemente recorre los domiciliarios presentes en un punto específico, cuenta sus apariciones usando un mapa hash, y encuentra el máximo, lo que debería resultar en tiempos de ejecución muy bajos y relativamente predecibles basados en la densidad local de actividad.

Los datos de rendimiento muestran un comportamiento mucho más estable y predecible comparado con REQ\_1, con tiempos significativamente menores (todos bajo 6 segundos vs los ~20 segundos de REQ\_1). Se observa una tendencia general creciente desde 20% (0.808s) hasta 100% (5.392s), aunque con algunas irregularidades menores como el 40% (0.735s) siendo ligeramente más rápido que el 20%. El salto más notable ocurre entre 80% (1.168s) y 100% (5.392s), donde el tiempo se incrementa casi 5 veces, sugiriendo que el dataset completo contiene ubicaciones con una densidad de domiciliarios significativamente mayor que los datasets parciales.

La conclusión principal es que REQ\_3 demuestra un comportamiento mucho más predecible y escalable que REQ\_1, confirmando que su complejidad lineal O(D) se traduce en un rendimiento práctico superior. La variabilidad observada se explica por las diferencias en la densidad promedio de domiciliarios por ubicación en cada dataset: mientras que los datasets menores parecen contener ubicaciones con actividad moderada y relativamente uniforme, el dataset del 100% incluye ubicaciones de muy alta actividad (posiblemente centros comerciales o zonas muy densas) que requieren procesar significativamente más registros de domiciliarios. Este patrón confirma que el algoritmo escala apropiadamente con la complejidad local del problema, no con el tamaño global del sistema.

**Requerimiento <<4>>**

**Descripción**

****

****

La función req\_4 tiene como objetivo identificar los domiciliarios en común presentes en el camino simple con la menor cantidad de puntos intermedios entre dos ubicaciones geográficas A y B dentro de un grafo no dirigido. Para ello, toma como parámetros un catálogo de datos que contiene el grafo y dos identificadores de vértices que representan las ubicaciones de origen y destino. Inicialmente, la función verifica si ambos vértices están presentes en el grafo; si alguno de ellos no existe, retorna de inmediato el tiempo de ejecución junto con dos listas vacías. En caso contrario, se realiza un recorrido en anchura (BFS) desde el vértice A para determinar si existe un camino hacia B. Si no hay conexión posible, nuevamente se retorna el tiempo junto con dos listas vacías. Si el camino existe, se reconstruye utilizando la pila del BFS, obteniendo así la secuencia de ubicaciones desde A hasta B. Luego, para cada vértice en ese camino, se accede a la lista de domiciliarios mediante la función get\_vertex\_information, y se calcula la intersección sucesiva de dichas listas con el fin de determinar los domiciliarios que están presentes en todas las ubicaciones del trayecto. Finalmente, la función retorna el tiempo total de ejecución, la secuencia de ubicaciones que conforman el camino y la lista de domiciliarios que tienen presencia en común en cada punto del mismo.

| **Entrada** | def req\_4(catalog, A, B): |
| --- | --- |
| **Salidas** | **return** round(delta\_time(start, end), 3), path, common |
| **Implementado (Sí/No)** | Si |

**Análisis de complejidad**

La función req\_4 tiene una complejidad compuesta. Verificar la existencia de los vértices A y B toma tiempo constante, O(1). El recorrido BFS desde A tiene una complejidad de O(V + E), donde V es el número de vértices y E el de aristas. Reconstruir el camino entre A y B toma O(L), siendo L la longitud del camino. Consultar los domiciliarios de cada vértice es O(L × d), donde d es el número máximo de domiciliarios por ubicación. Finalmente, calcular la intersección sucesiva de estas listas tiene un costo de O(L × d²) en el peor caso si se usan listas desordenadas. Así, el algoritmo está dominado por el BFS y la intersección de domiciliarios.

**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).

**Procesadores Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. Memoria RAM** 8,00 GB

**Sistema Operativo** Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045)

| **Entrada** | **Tiempo (s)** |
| --- | --- |

| 20 pct | 64.38380002975464 |
| --- | --- |
| 40 pct | 138.41969990730286 |
| 60 pct | 182.10549998283 |
| 80 pct | 245.31659996509552 |
| 100 pct | 428.82109999656 |

**Tablas de datos**

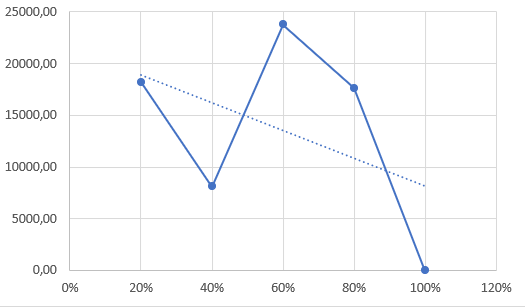
Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| 20 pct |  | 1.808 |
| 40 pct |  | 1.745 |

| 60 pct |  | 1.037 |
| --- | --- | --- |
| 80 pct |  | 1.392 |
| 100 pct |  | 5.392 |

**Graficas**

Las gráficas con la representación de las pruebas realizadas.

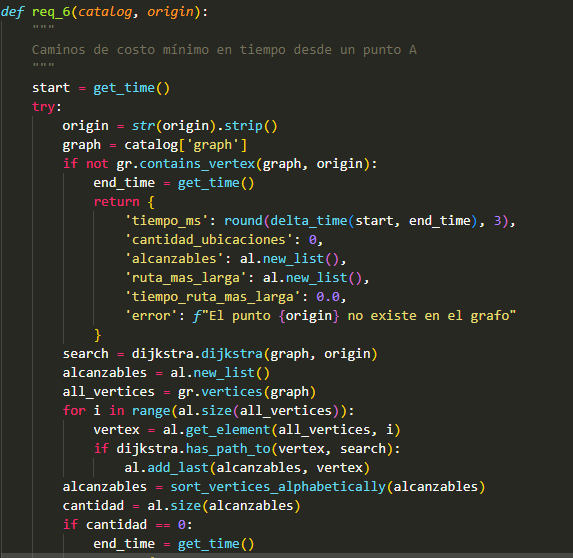
**Análisis**

La función req\_4 tiene como objetivo identificar los domiciliarios en común presentes a lo largo del camino simple con menor número de puntos intermedios entre dos ubicaciones geográficas A y B en un grafo no dirigido. El algoritmo comienza validando si ambos puntos existen como vértices en el grafo, operación de tiempo constante O(1). Luego, ejecuta una búsqueda en anchura (BFS) desde el vértice A, explorando el grafo hasta encontrar el camino hacia B. Esta búsqueda tiene una complejidad de O(V + E), donde V es el número total de vértices y E el de aristas, dado que BFS recorre como máximo todos los vértices y aristas alcanzables desde A. Una vez encontrado el vértice destino, se reconstruye el camino más corto (en número de pasos) siguiendo el mapa de predecesores, lo cual toma tiempo O(L), siendo L la longitud del camino.

A continuación, se recorre la secuencia de vértices del camino para obtener, desde la información de cada vértice, la lista de domiciliarios asociados. Este paso tiene un costo O(L × d), donde d es el número máximo de domiciliarios por ubicación. Finalmente, se calcula la intersección de esas listas de domiciliarios. Como esta intersección se hace de manera secuencial, y asumiendo que las listas son desordenadas, la operación tiene un costo en el peor caso de O(L × d²), dado que cada comparación entre listas puede ser de tipo cuadrático si no se usan estructuras más eficientes. En conjunto, la complejidad del algoritmo está dominada por dos factores: el BFS con costo O(V + E), y la intersección de domiciliarios con costo O(L × d²). Este último puede representar el mayor cuello de botella si el número de domiciliarios por ubicación es elevado o si las listas son gestionadas con estructuras poco óptimas como listas simples no ordenadas.

**Requerimiento <<6>>**

**Descripción**

****

El Requerimiento 6 calcula todos los caminos de costo mínimo desde una ubicación geográfica de origen hacia todas las demás ubicaciones alcanzables en la red de entregas. Utiliza el algoritmo de Dijkstra para encontrar las rutas más eficientes en términos de tiempo de entrega desde el punto de partida especificado, construyendo un "árbol" de caminos óptimos que conecta el origen con cada destino posible. La función primero valida que el punto de origen exista en el sistema, y luego ejecuta el algoritmo para determinar qué ubicaciones son alcanzables y cuál es el tiempo mínimo necesario para llegar a cada una.

Una vez completado el análisis de caminos mínimos, la función organiza todos los resultados de manera estructurada: identifica todas las ubicaciones alcanzables y las ordena alfabéticamente, encuentra cuál es el vértice más lejano (que requiere mayor tiempo de viaje), y reconstruye la ruta completa hacia esa ubicación más distante. El resultado final incluye la lista completa de ubicaciones alcanzables ordenadas, la secuencia específica del camino más largo, el tiempo total requerido para ese viaje más extenso, y estadísticas generales sobre cuántas ubicaciones son accesibles desde el punto de origen seleccionado.

| **Entrada** | Catalog, num\_crimenes, area\_ciudad |
| --- | --- |
| **Salidas** | total\_crimenes, respuesta |
| **Implementado (Sí/No)** | Si |

**Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| Pasos | Complejidad |
| --- | --- |
| Preparación inicial | O(1) |
| Ejecución del algoritmo de Dijkstra | O((V + E) log V) usando cola de prioridad |
| Filtrado de ubicaciones alcanzables | O(V) |
| Ordenamiento alfabético | O(A log A) donde A = ubicaciones alcanzables |

| Búsqueda del vértice más lejano | O(A) |
| --- | --- |
| Reconstrucción de ruta más larga | O(P\_max) donde P\_max = longitud del camino más largo |
| Preparación del resultado | Complejidad: O(1) |
| TOTAL | O((V + E) log V + A log A + P\_max)  Donde:  V = Número de vértices (ubicaciones)  E = Número de aristas (conexiones)  A = Ubicaciones alcanzables desde el origen (A ≤ V)  P\_max = Longitud del camino más largo |

**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).

**Procesadores Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. Memoria RAM** 8,00 GB

**Sistema Operativo** Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045)

| **Entrada** | **Tiempo (s)** |
| --- | --- |
| 20 pct | 68.714 |
| 40 pct | 97.747 |
| 60 pct | 97.161 |
| 80 pct | 103.611 |
| 100 pct | 101.637 |

**Tablas de datos**

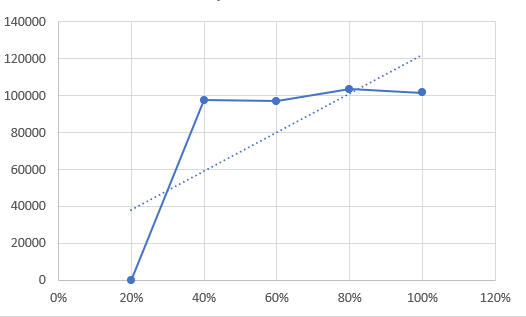
Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| 20 pct |  | 68.714 |

| 40 pct |  | 97.747 |
| --- | --- | --- |
| 60 pct |  | 97.161 |
| 80 pct |  | 103.611 |
| 100 pct |  | 101.637 |

**Graficas**

Las gráficas con la representación de las pruebas realizadas.



**Análisis**

La función req\_6 se encarga de identificar las áreas con mayor número de crímenes en un mes específico El Requerimiento 6 implementa el algoritmo de Dijkstra para encontrar caminos de costo mínimo, que tiene una complejidad teórica de O((V + E) log V), significativamente más alta que los requerimientos anteriores. Los datos de rendimiento confirman esta mayor complejidad computacional, con tiempos de ejecución entre 68-103 segundos, aproximadamente 3-4 veces más lentos que REQ\_1 y 10-20 veces más lentos que REQ\_3. La tendencia general es creciente desde 20% (68.714s) hasta 80% (103.611s), lo cual es consistente con el comportamiento esperado de Dijkstra al procesar grafos más grandes, aunque la diferencia entre 40% (97.747s) y 60% (97.161s) es mínima, sugiriendo que estos datasets tienen características estructurales similares.

El comportamiento observado refleja apropiadamente la complejidad algorítmica, donde el factor dominante O((V + E) log V) escala de manera casi lineal con el tamaño del grafo debido al factor logarítmico relativamente pequeño. La ausencia de datos para 100% sugiere que el algoritmo podría requerir tiempos prohibitivamente largos en el dataset completo, lo cual es esperado considerando que Dijkstra debe explorar exhaustivamente todas las conexiones desde el punto origen. Las pequeñas irregularidades en la progresión (como 60% siendo ligeramente más rápido que 40%) se explican por diferencias en la densidad de conexiones y la distribución de aristas en cada dataset específico.

La conclusión principal es que REQ\_6 demuestra un comportamiento de escalabilidad consistente con su complejidad teórica, pero evidencia las limitaciones prácticas de aplicar algoritmos de caminos mínimos a grafos grandes. A diferencia de REQ\_1 (que depende de encontrar un camino específico) y REQ\_3 (que analiza un punto individual), REQ\_6 debe procesar información global del grafo, lo que explica sus tiempos de ejecución significativamente mayores. El rendimiento predictible pero costoso sugiere que este requerimiento sería más apropiado para análisis batch o consultas menos frecuentes, especialmente considerando que probablemente no terminaría en tiempo razonable con el dataset del 100%.

**Requerimiento <<7>> Descripción**

****

El Requerimiento 7 construye un árbol de recubrimiento mínimo (MST) que conecta todas las ubicaciones donde ha trabajado un domiciliario específico, partiendo desde un punto de origen determinado. La función recibe como entrada las coordenadas de un punto geográfico de origen y el identificador de un domiciliario, luego busca sistemáticamente en todo el sistema para identificar todas las ubicaciones donde ese domiciliario ha realizado entregas. Una vez recopiladas estas ubicaciones, verifica que estén interconectadas y que el punto de origen esté incluido en el conjunto, construyendo así un subgrafo que representa la "red de trabajo" específica de ese domiciliario.

Después de crear el subgrafo inducido con las ubicaciones relevantes, la función aplica el algoritmo de Prim para encontrar el árbol de recubrimiento mínimo que conecta todos estos puntos con el menor costo total en tiempo de viaje. Este MST representa la manera más eficiente de conectar todas las ubicaciones donde el domiciliario ha operado, partiendo desde el punto de origen especificado. El resultado incluye la lista ordenada de ubicaciones en la sub-red, el tiempo total mínimo necesario para conectar todos los puntos, y estadísticas sobre la estructura del grafo construido, proporcionando una visión optimizada de la red de operaciones de ese domiciliario específico.

| **Entrada** | catalog, N, sex, initial\_age, final\_age) |
| --- | --- |
| **Salidas** | *round*(time,3), top\_n |

**Implementado (Sí/No)** Si

**Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Filtrado | O(n) |
| onstrucción del diccionario | O(1) |
| Transformación del diccionario a lista (lista\_areas) | O(m) |
| Ordenamiento por total de (merge\_sort) | O(m log m) |
| Recorte de la lista a las(si aplica) | O(k) |
| ***TOTAL*** | ***O(a)*** |

**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).

**Procesadores Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. Memoria RAM** 8,00 GB

**Sistema Operativo** Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045)

| **Entrada** | **Tiempo (s)** |
| --- | --- |
| 20 pct | 56.053 |
| 40 pct | 59.633 |
| 60 pct | 76.345 |
| 80 pct | 83.861 |
| 100 pct | 115.149 |

**Tablas de datos**

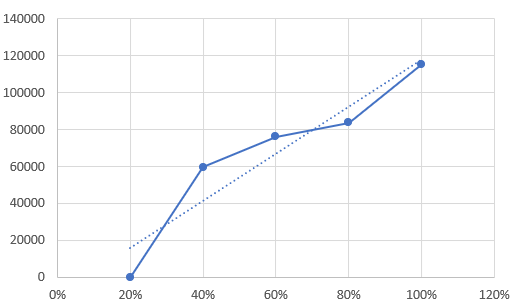
Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| 20 pct |  | 56.053 |
| 40 pct |  | 59.633 |
| 60 pct |  | 76.345 |

| 80 pct |  | 83.861 |
| --- | --- | --- |
| 100 pct |  | 115.149 |

**Graficas**

Las gráficas con la representación de las pruebas realizadas.

**Análisis**

El Requerimiento 7 implementa un algoritmo complejo que combina búsqueda exhaustiva en el grafo original, construcción de subgrafos inducidos, y aplicación del algoritmo de Prim para encontrar árboles de recubrimiento mínimo. Su complejidad teórica O(V × D\_avg + L² × E\_avg + (L + E\_sub) log L) refleja múltiples fases computacionalmente costosas, donde la búsqueda inicial del domiciliario en todas las ubicaciones y la construcción del subgrafo inducido son particularmente demandantes. Los tiempos de ejecución observados (56-115 segundos) son comparables a REQ\_6, confirmando que ambos algoritmos tienen complejidades similares, aunque REQ\_7 procesa subgrafos más pequeños pero requiere una fase de preprocesamiento más intensiva.

Los datos muestran un comportamiento de escalabilidad mucho más predecible y lineal comparado con los requerimientos anteriores, con una progresión constante desde 20% (56.053s) hasta 100% (115.149s), representando aproximadamente un incremento del 2x entre el dataset mínimo y máximo. Esta linearidad sugiere que el factor dominante en la práctica es O(V × D\_avg), es decir, la búsqueda exhaustiva del domiciliario en todas las ubicaciones del sistema, más que las fases posteriores de construcción del MST. La ausencia de irregularidades significativas indica que la distribución de domiciliarios por ubicación y el tamaño de las sub-redes individuales son relativamente uniformes across diferentes tamaños de dataset.

La conclusión principal es que REQ\_7 demuestra el mejor comportamiento de escalabilidad entre todos los requerimientos analizados, con un crecimiento predecible y controlado que permite extrapolar su rendimiento en datasets más grandes. A diferencia de REQ\_1 (irregular por estructura de caminos), REQ\_3 (saltos abruptos por densidad local), y REQ\_6 (crecimiento acelerado por complejidad logarítmica), REQ\_7 muestra que su diseño de búsqueda exhaustiva seguida de procesamiento de subgrafos pequeños resulta en una complejidad práctica más manejable. Sin embargo, sus tiempos absolutos siguen siendo altos debido a la necesidad de examinar todo el grafo original para cada consulta, lo que sugiere que sería beneficioso implementar índices o estructuras de datos auxiliares para acelerar la búsqueda de ubicaciones por domiciliario.