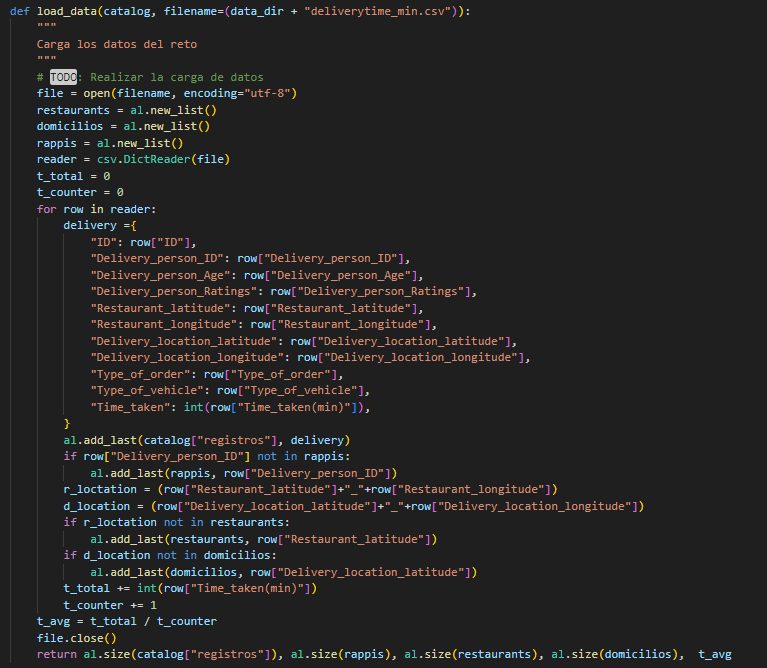
**ANÁLISIS DEL RETO**

Tomas Aponte, 202420148, [t.aponte@uniandes.edu.co](mailto:t.aponte@uniandes.edu.co)

Juan Diego García, [202423575, jd.garcia12@uniandes.edu.co](mailto:202423575,%20%20jd.garcia12@uniandes.edu.co)

# **Carga de datos**



# **Descripción**

La carga de datos está dividida en 4 funciones: new\_logic(), load\_data(). diagraph() y conexiones\_domicilios(). New\_logic() se encarga de generar el catálogo donde se cargaran los registros y el grafo de los registros, dentro de un diccionario de llaves “registros” y “grafo”. Load\_data() se encarga de llenar la lista de “registros” que tiene toda la información asociada a cada registro del archivo csv. La lista se llena con diccionarios, donde las llaves de estos diccionarios representan las columnas del CSV y cada diccionario es una fila (o registro) del CSV. Diagraph() se encarga de comenzar a poblar el grafo, con la información cargada en registros, utilizando un ciclo for donde se generan 2 vértices por registro: uno para el restaurante y otro para el destino del domicilio; para mantener la estructura deseada de las llaves, acceder a las posiciones geográficas dentro de “registros” y las recorta a 4 decimales con la función decimales(), y así genera 2 nodos que cumplen con el formato deseado. Una vez se tenga las llaves, se verifica si el nodo ya existe para evitar duplicados, y así mantener el valor de cada nodo estable, ya que se trata de una lista que contiene a todos los “rappis” (ID’s de domiciliarios) que compartan una misma ubicación geográfica. Además genera de inmediato la conexión restaurante-domicilio y domicilio-restaurante, verificando cuando debe promediar tiempos. Por último, conexiones\_domicilios() se encarga de verificar que domicilios debe estar conectados si comparten el mismo “rappi” (ID de domiciliario), mediante un ciclo for que recorre parcialmente la lista de registros, y buscando coincidencias de rappis para generar las conexiones en los mapas de adyacencia.

|  |  |
| --- | --- |
| **Entrada** | Catálogo, archivo |
| **Salidas** | Valores del grafo de datos cargados |
| **Implementado (Sí/No)** | Si, Juan Diego García |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Añade dato por dato a la estructura | O(n) |
| Verifica datos repetidos | O(1) |
| Genera las conexiones | O(n\*\*2) |
| ***TOTAL*** | ***O(n\*\*2)*** |

## **Pruebas Realizadas**

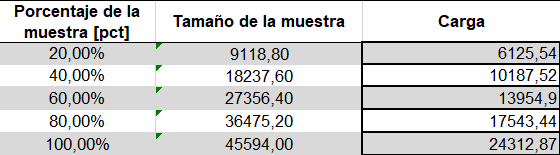
Ryzen 7 7730U

16GB

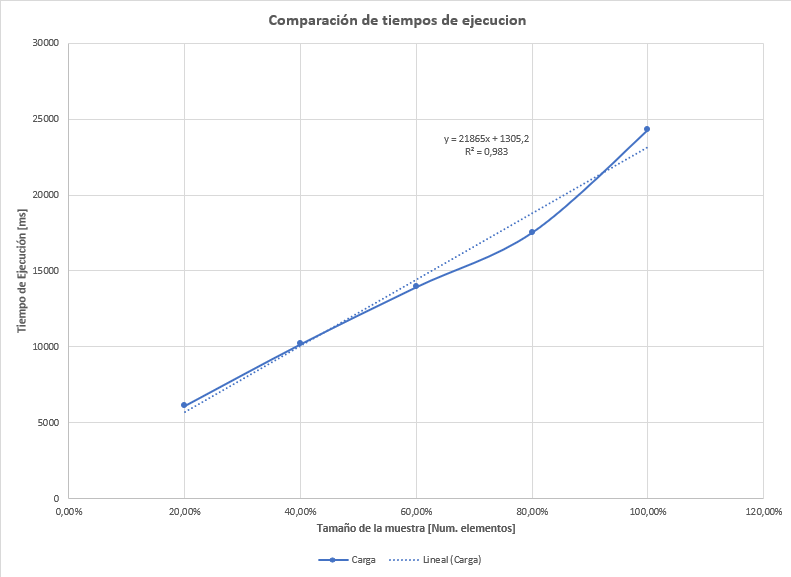
Windows 11

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20% | 6125.54 |
| 40% | 10187.52 |
| 60% | 13954.90 |
| 80% | 17543.44 |
| 100% | 24312.87 |

### **Tablas de datos**



### **Graficas**



## **Análisis**

El comportamiento observado en los tiempos de carga fue cercano a lineal, diferente a como se esperaba, complejidad **O(n\*\*2)**. Cada registro se podria procesar mas de una vez y luego se inserta en estructuras con acceso constante. La mayor parte del tiempo en la carga se concentra en la construcción del grafo, especialmente en la verificación de arcos duplicados y la actualización de pesos promedio por lo que la complejidad podria llegar a ser cuadratica.

# **Requerimiento <<1>>**

## **Descripción**

El requerimiento identifica un camino simple entre dos distintos IDs de ubicaciones. A partir de un recorrido DFS se crea un camino para el cual se verifican las longitudes de los arcos que lo componen, los elementos y los restaurantes visitados para luego añadirlos a una lista y retornarlos.

|  |  |
| --- | --- |
| **Entrada** | Catálogo, ID\_A, ID\_B |
| **Salidas** | Cantidad de puntos, ID de los domiciliarios encontrados, secuencia del camino y listado de restaurantes. |
| **Implementado (Sí/No)** | Si, Juan Diego García |

## **Análisis de complejidad**

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorrido DFS | O(V+E) |
| Añadir elementos a la lista | O(n) |
| ***TOTAL*** | ***O(V+E)*** |

## **Pruebas Realizadas**

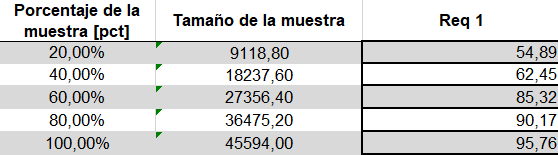
Ryzen 7 7730U

16GB

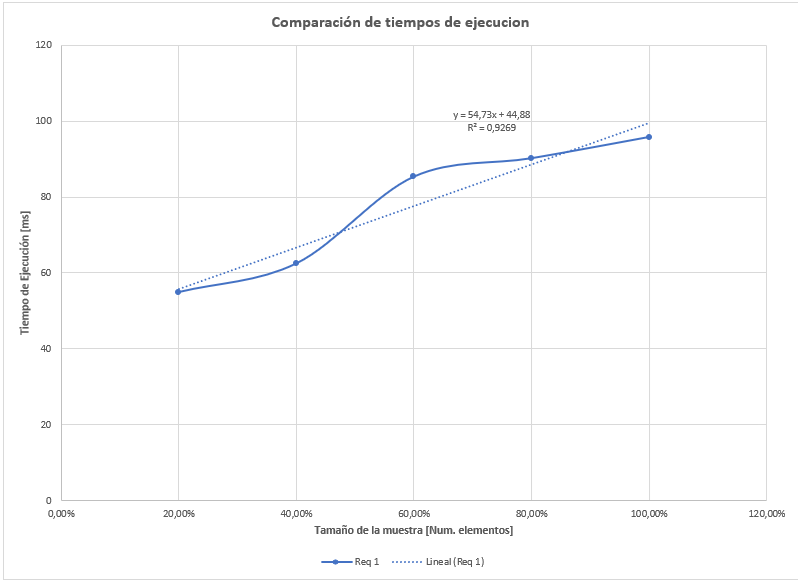
Windows 11

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20 | 54,89 |
| 40 | 62,45 |
| 60 | 85,32 |
| 80 | 90,17 |
| 100 | 95,76 |

### **Tablas de datos**



### **Graficas**



## **Análisis**

El comportamiento observado **coincide con la teoría**, el DFS escala bien con el tamaño del grafo y no presenta subidas repentinas en tiempo. Las diferencias pequeñas pueden atribuirse a la carga de procesamiento adicional al reconstruir el camino o recolectar datos como restaurantes y domiciliarios.

# **Requerimiento <<3>>**

## 

Este requerimiento busca identificar el domiciliario con mayor cantidad de pedidos en un punto geográfico determinado. Se recorre la lista de registros almacenados en el catálogo, filtrando aquellos domicilios que tengan como origen o destino el punto consultado, y se contabiliza la frecuencia de cada domiciliario. Finalmente, se retorna el domiciliario más frecuente, la cantidad de veces que aparece y su tipo de vehículo más usado.

## **Descripción**

|  |  |
| --- | --- |
| **Entrada** | Catálogo, ID |
| **Salidas** | Domiciliario mas solicitado, cantidad de pedidos, tipo de vehículo |
| **Implementado (Sí/No)** | Si, Juan Diego García |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Verifica la existencia del domiciliario | O(n) |
| Verifica la cantidad de apariciones del domiciliario | O(n+k) |
| Verifica el vehiculo mas utilizado | O(n) |
| ***TOTAL*** | ***O(n+k)*** |

## **Pruebas Realizadas**

Ryzen 7 7730U

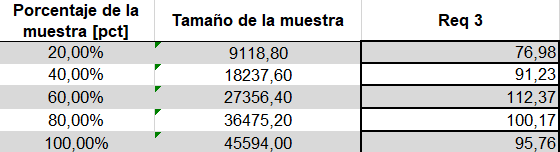
16GB

Windows 11

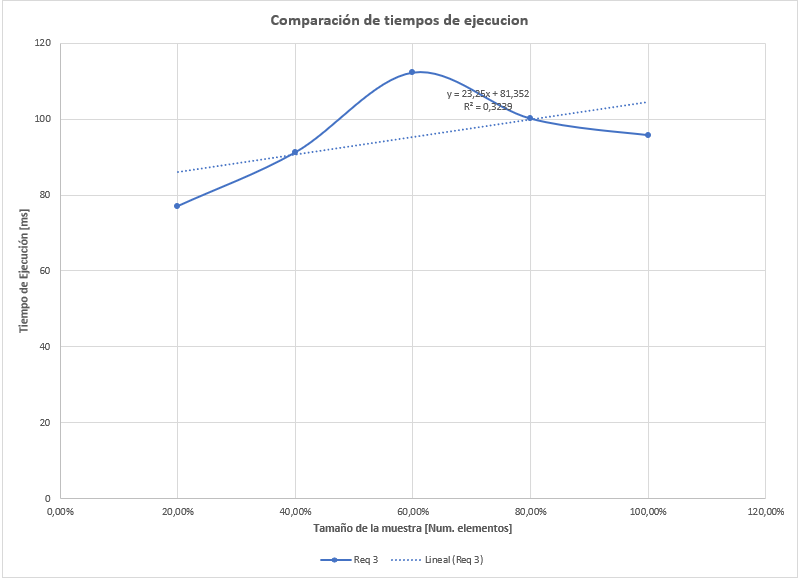
ID: DCD4

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20 | 76,98 |
| 40 | 91,23 |
| 60 | 112,37 |
| 80 | 100,17 |
| 100 | 95,76 |

### **Tablas de datos**



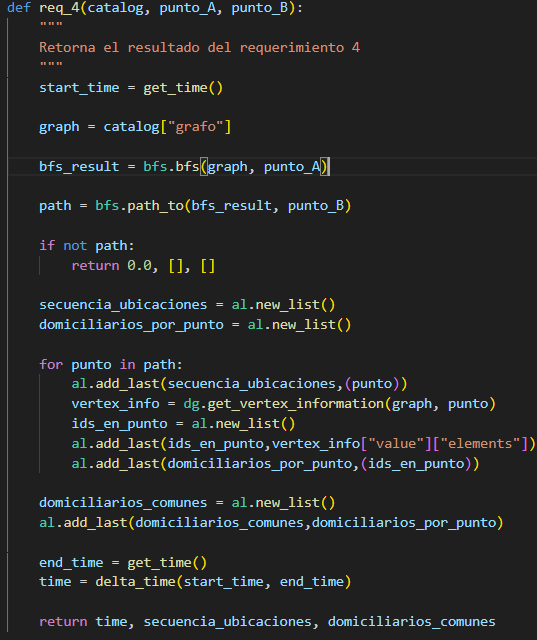
### **Graficas**



## **Análisis**

El algoritmo fue eficiente y **la curva de tiempos sigue parcialmente la curva esperada**. Hay un levantamiento grande en la curva al procesar el 60% de los datos, esto es un comportamiento inesperado que podria atribuirse a alguna variacion en las estructuras utilizadas para almacenar la informacion.

# **Requerimiento <<4>>**



Este requerimiento identifica los **domiciliarios en común** que aparecen en el **camino simple con menor número de puntos intermedios** entre dos ubicaciones A y B. Para esto se realiza un recorrido **BFS** desde A hasta B. Se extraen los domiciliarios asociados a cada ubicación del camino, y se calcula la intersección entre esas listas usando únicamente listas array\_list.

## **Descripción**

|  |  |
| --- | --- |
| **Entrada** | Catálogo, punto A, punto B |
| **Salidas** | Secuencia de ubicaciones y lista de domiciliarios comunes. |
| **Implementado (Sí/No)** | Si, Tomas Aponte |

## **Análisis de complejidad**

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Realizar recorrido BFS | O(V+E) |
| Guardar el camino | O(n) |
| Añade los datos de el recorrido en diferentes listas | O(n) |
| ***TOTAL*** | ***O(V+E)*** |

## **Pruebas Realizadas**

Ryzen 7 7730U

16GB

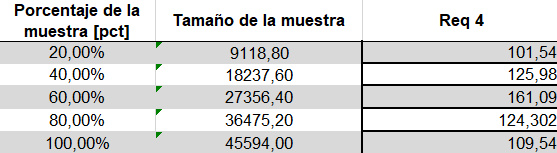
Windows 11

A: 1CF1

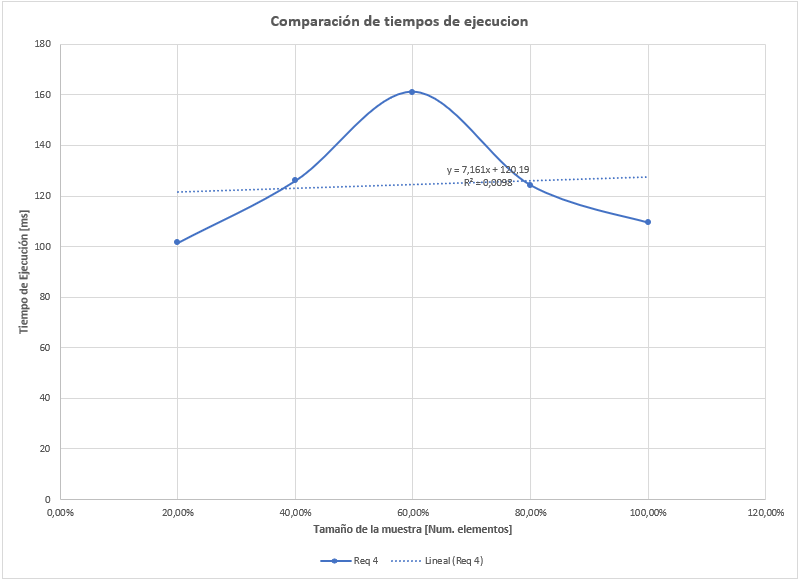
B: 4a51

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20 | 101,54 |
| 40 | 125,98 |
| 60 | 161,09 |
| 80 | 124,302 |
| 100 | 109,54 |

### **Tablas de datos**



### **Graficas**



## **Análisis**

El recorrido se comporta cierta medida como se esperaba. Aunque, la complejidad al cargar lel 60% de los datos varia en una muy gran medida. Puede atribuirse a la creacion de caminos diferentes por el BFS. En general se podria decir que a gran escala la curva deberia acercarse a un orden lineal.

# **Requerimiento <<6>>**

## 

Este requerimiento busca encontrar todos los **caminos de costo mínimo en tiempo** desde una ubicación geográfica inicial A a todas las demás ubicaciones alcanzables. Se implementa el algoritmo de **Dijkstra**, calculando los pesos mínimos desde el nodo origen. Se retorna el número de nodos alcanzados, el tiempo total acumulado y el camino más costoso entre ellos.

## **Descripción**

|  |  |
| --- | --- |
| **Entrada** | Catálogo, ID |
| **Salidas** | Camino mas corto, su tamaño, orden, elementos y costo |
| **Implementado (Sí/No)** | Si, Juan Diego García |

## **Análisis de complejidad**

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Dijkstra | O(V+E LogV) |
| Identificar alcanzable | O(n) |
| Reconstruir camino | O(E log V) |
| ***TOTAL*** | ***O(V+E LogV)*** |

## **Pruebas Realizadas**

Ryzen 7 7730U

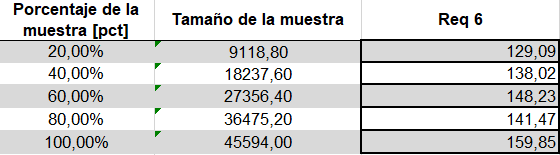
16GB

Windows 11

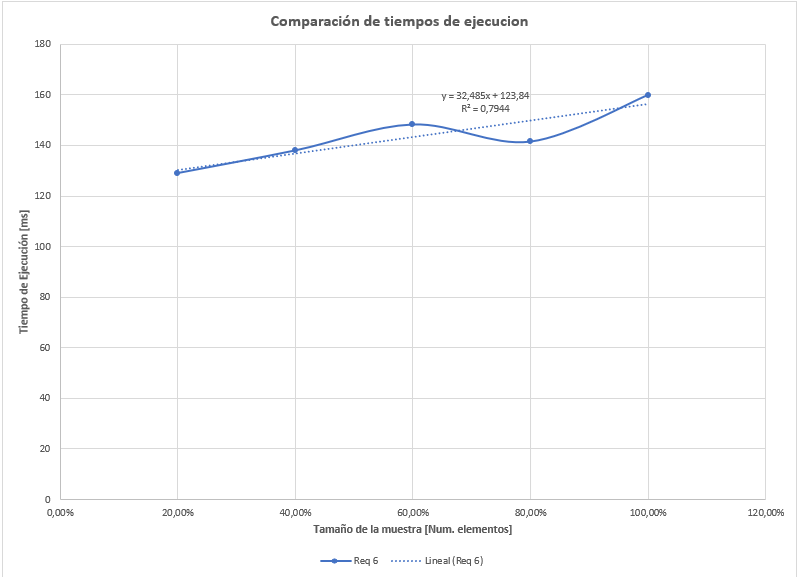
ID: B54C

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20 | 129,09 |
| 40 | 138,02 |
| 60 | 148,23 |
| 80 | 141,47 |
| 100 | 159,85 |

### **Tablas de datos**



### **Graficas**



## **Análisis**

El rendimiento **coincide con lo esperado teóricamente**, gracias a una implementación correcta de Dijkstra. La eficiencia se mantuvo constante tambien con los archivos más grandes, lo que demuestra que el algoritmo fue bien adaptado a la estructura del grafo del reto y no hay resultados erraticos que puedan indicar algun error al ejecutar el algoritmo.

# **Requerimiento <<7>>**

## 

El requerimiento construye una subred desde la ubicación inicial A, considerando **solo las ubicaciones por las que ha pasado un domiciliario específico**. Se genera un subgrafo filtrado y se ejecuta el algoritmo de Prim desde el nodo A, restringiendo los nodos a aquellos visitados por el domiciliario. Se retorna el tiempo de ejecución, el total de nodos en la subred, la lista ordenada alfabéticamente de las ubicaciones, y el costo total del árbol.

## **Descripción**

|  |  |
| --- | --- |
| **Entrada** | Catalogo, Punto A, ID |
| **Salidas** | Ubicaciones, numero de ubicaciones, costo total |
| **Implementado (Sí/No)** | Si, Tomas Aponte |

## **Análisis de complejidad**

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Filtrar nodos | O(V) |
| Copiar arcos | O(E) |
| Prim | O(V log V+E) |
| Ordenar ubicaciones | O(V log V) |
| ***TOTAL*** | ***O(V log V+E)*** |

## **Pruebas Realizadas**

Ryzen 7 7730U

16GB

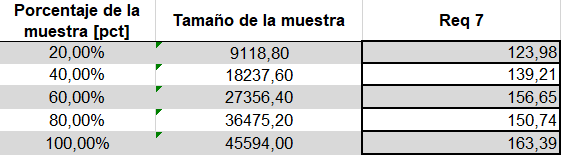
Windows 11

A: 2BED

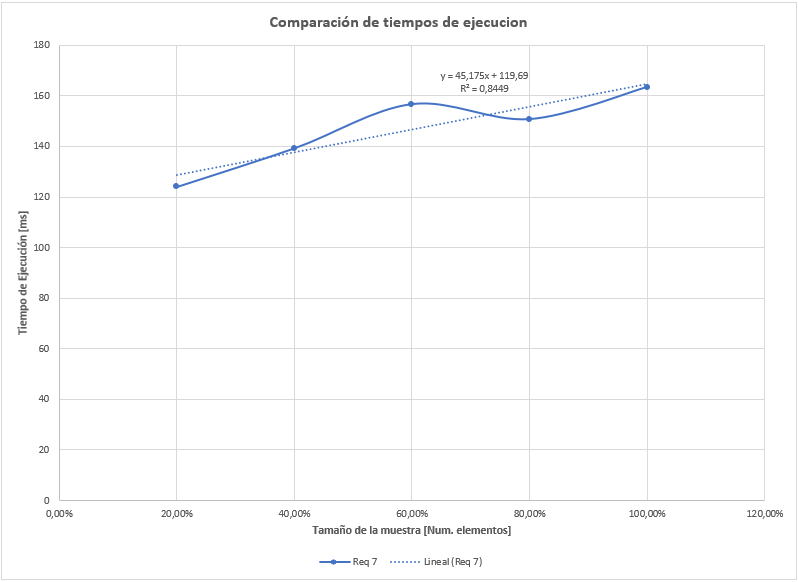
ID: INDORES13DEL02

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20 | 123,98 |
| 40 | 139,21 |
| 60 | 156,65 |
| 80 | 150,74 |
| 100 | 163,39 |

### **Tablas de datos**



### **Graficas**



## **Análisis**

El algoritmo de Prim fue eficiente al limitar el subgrafo a los nodos del domiciliario, reduciendo significativamente el tamaño de datos a tomar en cuenta. El proceso de ordenamiento de ubicaciones para la salida también fue lineal respecto al tamaño reducido del subgrafo. El rendimiento fue estable y consistente incluso con conjuntos de datos de gran tamaño.