**ANÁLISIS DEL RETO**

Pedro Archila,xxxxxx, p.archila@uniandes.edu.co

Gabriela Gomez, 202420506, g.gomezh2@uniandes.edu.co

# **Requerimiento <<1>>**

Plantilla para el documentar y analizar cada uno de los requerimientos.

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | Catalogo con el grafo y vértices, el vértices de origen y el vértice de destino |
| **Salidas** | Una lista con el tiempo que se demora el requerimiento, la cantidad de puntos, los domiciliarios, el camino recorrido y los restaurantes |
| **Implementado (Sí/No)** | Si se implementó - grupal |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Accede al grafo   |  |  | | --- | --- | | grafo = catalog["grafo"] |  | | |  |  | | --- | --- | |  | O(1) | |
| Verificacion de vertices   |  |  | | --- | --- | | gr.contains\_vertex |  | | O(1) |
| |  |  | | --- | --- | | dfs\_result = dfs.dfs(grafo, origen) |  | | |  |  | | --- | --- | |  | O(V + E) | |
| Verificacion de camino   |  |  | | --- | --- | | dfs.has\_path\_to(dfs\_result, destino) |  | | |  | | --- | | O(1) | |
| Se obtiene el camino   |  |  | | --- | --- | | camino = bfs.path\_to(dfs\_result, destino) |  | | |  |  | | --- | --- | |  | O(P) se guarda en una variable el camino encontrado | |
| cuenta la cantidad de vértices (puntos) hay en el camino   |  | | --- | | cantidad\_puntos = st.size(camino) | | |  |  | | --- | --- | |  | O(1) | |
| Inician listas   |  |  | | --- | --- | | domiciliarios = ar.new\_list() y restaurantes = ar.new\_list() |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | for I in range(cantidad\_puntos) |  | | |  |  | | --- | --- | |  | O(P × D × N). | |
| |  |  | | --- | --- | | ar.get\_element(camino, i) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | gr.get\_vertex\_information(grafo, punto) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | mp.value\_set(able\_info) |  | | |  |  | | --- | --- | |  | O(1) | |
| Se recorre   |  |  | | --- | --- | | info\_pedidos["elements"] |  | | |  |  |  | | --- | --- | --- | | O(D) |  |  | |
| |  |  | | --- | --- | | verificar domiciliario esta en la lista  not in domiciliarios["elements"] |  | | |  |  | | --- | --- | |  | O(N) | |
| |  |  | | --- | --- | | ar.add\_last(domiciliarios, domiciliario) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | ar.add\_last(restaurantes, punto) |  | | |  |  | | --- | --- | |  | O(1) | |
| **TOTAL** | **O(V+E+P⋅D⋅N)** |

* V es el número de vértices
* E es el número de aristas.
* P es la cantidad de puntos en el camino (camino)
* D es el número promedio de domicilios por punto.
* N es el número total de domiciliarios únicos procesados.

## **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).

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

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

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Teniendo en cuenta que P, D y N serán números muy pequeños comparados a V y a E se puede llegar a concluir que la complejidad de este requerimiento será un poco mayor a O(V+E), complejidad la cual sale de la implementación de DFS.

# **Requerimiento <<3>>**

Plantilla para el documentar y analizar cada uno de los requerimientos.

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | El catalogo con el grafo y vértices, y el punto el cual se quiere analizar |
| **Salidas** | Una lista que tenga máximo de domiciliarios, máximo de pedidos, tipo de vehículo y el tiempo que se demora el requerimiento |
| **Implementado (Sí/No)** | Si se implementó – Pedro Archila |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Acceder al grafo   |  | | --- | | grafo = catalog[“grafo”] | | |  | | --- | | O(1) | |
| Verificar si el punto existe   |  |  | | --- | --- | | if not gr.contains\_vertex(grafo, punto) |  | | |  | | --- | | O(1) | |
| Se saca la table de hash que tiene la info del vertice   |  |  | | --- | --- | | tabla\_pedidos = gr.get\_vertex\_information(grafo, punto) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Inicialisan diccionario  conteo = {} y vehiculos = {} |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Se sacan los pedidos de la tabla de hash  pedidos = mp.key\_set(tabla\_pedidos) |  | | |  |  | | --- | --- | |  | O(K) | |
| |  | | --- | | **Ciclo** for pedido\_id in pedidos | | |  | | --- | | O(K) | |
| |  |  | | --- | --- | | Se guardan los datos del vertice en una variable  datos = mp.get(tabla\_pedidos, pedido\_id) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Asignan variables a la info sacada del vertice  dom\_id = datos[“domiciliario\_id”], vehículo = datos[“vehículo”] |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Verificar y actualizar  conteo[dom\_id] |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Verificar y actualizar vehículo[dom\_id][vehículo] |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | **Ciclo** for domi in conteo: **(buscar máximo)** |  | | |  |  | | --- | --- | |  | O(U) (U = # domiciliarios) | |
| |  |  | | --- | --- | | **Ciclo** for veh in vehiculos[max\_dom]: |  | | |  |  | | --- | --- | |  | O(T) (T = # tipos vehículo) | |
| **TOTAL** | **O(K+U+T)** |

* K proviene del recorrido de pedidos.
* U del ciclo que busca el domiciliario con más pedidos.
* T del ciclo que busca el vehículo más usado por ese domiciliario.

## **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).

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

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

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

La complejidad de este requerimiento es muy buena, pues es lineal. Se usan varios ciclos separados para analizar todos los datos necesarios y a la vez no se tiene una complejidad demasiado alta.

# **Requerimiento <<4>>**

Plantilla para el documentar y analizar cada uno de los requerimientos.

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | Catalogo con el grafo y vértices, el punto a de análisis y el punto b de análisis |
| **Salidas** | Una lista con el camino que resulto, los domiciliarios en común y el tiempo que se demora el requerimiento |
| **Implementado (Sí/No)** | Si se implementó – Gabriela Gomez |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Se accede al grafo   |  |  | | --- | --- | | grafo = catalog[“grafo”] |  | | |  |  | | --- | --- | |  | O(1) | |
| Verificación de existencia de vertices   |  |  | | --- | --- | | (contains\_vertex) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | BFS desde punto\_a (bfs.bfs) |  | | |  |  | | --- | --- | |  | O(V + E) | |
| |  |  | | --- | --- | | Verificación de camino a punto\_b (bfs.has\_path\_to) |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Obtener camino (bfs.path\_to) |  | | |  | | --- | | O(L) | |
| |  |  | | --- | --- | | Recorrer camino (while i <= tam\_camino) |  | | |  | | --- | | O(L) | |
| |  |  | | --- | --- | | Obtener pedidos por punto (gr.get\_vertex\_information) |  | | |  | | --- | | O(1) | |
| |  |  | | --- | --- | | Obtener info de pedidos (mp.value\_set) |  | | |  |  | | --- | --- | |  | O(P\_i) por punto | |
| |  |  | | --- | --- | | Recorrer pedidos y registrar domiciliarios únicos (while j <= ar.size(info\_ped)) |  | | |  |  | | --- | --- | |  | O(P\_i) por punto → O(∑P\_i) | |
| |  |  | | --- | --- | | Agregar lista de domiciliarios por punto a listas\_domiciliarios |  | | |  |  | | --- | --- | |  | O(1) por punto | |
| |  |  | | --- | --- | | Obtener primera lista de domiciliarios |  | | |  |  | | --- | --- | |  | O(1) | |
| |  |  | | --- | --- | | Recorrer domiciliarios de la primera lista (while i <= tam\_primera) |  | | |  |  | | --- | --- | |  | O(D₁) | |
| |  |  | | --- | --- | | Verificar si está en todas las listas (while j <= ar.size(...)) |  | | |  |  | | --- | --- | |  | O(L × D) | |
| |  |  | | --- | --- | | Agregar comunes si no están ya (ar.is\_present(comunes, dom)) |  | | |  |  | | --- | --- | |  | O(C) | |
| |  |  | | --- | --- | | Construir camino legible  (while i <= ar.size(camino)) |  | | |  |  | | --- | --- | |  | O(L) | |
| **TOTAL** | **O(V+E+L⋅P+D1​⋅L⋅D)** |

* V: número de vértices en el grafo.
* E: número de aristas en el grafo.
* L: largo del camino desde punto\_a hasta punto\_b.
* P: número de pedidos promedio por punto.
* D\_1: domiciliarios en el primer punto del camino.
* D: domiciliarios promedio por punto en el camin

## **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).

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

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

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Para este requerimiento es necesario utilizar bfs pues se nos pide encontrar el camino más corto entre dos vértices. Además, para encontrar el domiciliario que más aparecía en este camino era necesario realizar algunos ciclos. Sin embargo, estos ciclos tendrán una complejidad relativamente baja, pues L, P, D1 y D serán siempre mucho menores a V y a E, pues no suponen demasiados datos. Es por esto que se pude considerar esta complejidad como algo un poco mayor a O(V+E).

# **Requerimiento <<6>>**

Plantilla para el documentar y analizar cada uno de los requerimientos.

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | Catalogo de los datos organizados en un grafo y vértice de origen |
| **Salidas** | Una lista con el tiempo de ejecución, cantidad de ubicaciones visitadas, una lista de los vértices visitados en orden, el camino que se sigue desde el origen al vértice que implica más tiempo y el tiempo que se demora el domiciliario en hacer ese camino |
| **Implementado (Sí/No)** | Si se implementó - grupal |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 | O(1) Saca el grafo |
| Paso 2 | O(VlogE) Dijkstra |
| Paso 3 | O(1) saca el número de ubicaciones que se visitan en el dijkstra |
| Paso 4 | O(1) Revisa que la cantidad no sea 0 |
| Paso 5 | O(V) Se calcula la distancia desde el origen a cada elemento visitado en el dijkstra |
| Paso 6 | O(V) revisa si ese tiempo es mayor que el que está guardado en la variable de tiempo maximo |
| Paso 7 | O(V) añade cada vertice a una lista de vertices alcanzables |
| Paso 8 | O(VlogV) Hace merge sort de la lista de vertices alcanzables |
| Paso 9 | O(V) Construye el camino que se sigue entre el origen y el vertice de tiempo maximo |
| ***TOTAL*** | ***O(VlogE + 4V + VlogV)*** |

## **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.

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Es un requerimiento que tiene una complejidad relativamente baja teniendo en cuenta la cantidad de datos que se están utilizando. La implementación de Dijkstra ayuda mucho a disminuir la complejidad de este requerimiento

# **Requerimiento <<7>>**

Plantilla para el documentar y analizar cada uno de los requerimientos.

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | Catalogo con el grafo y vértices, el vertice de origen y le domiciliario a analizar |
| **Salidas** | Una lista con el tiempo que se demora, el numero de ubicaciones, el orden de los vértices y el tiempo del recorrido del mst |
| **Implementado (Sí/No)** | Si se implementó - grupal |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 | O(1) saca el grafo del catalogo |
| Paso 2 | O(VlogE) Hace eager prim desde el vertice origen |
| Paso 3 | O(entre V y V^2) Se saca cada vértice que ha sido visitado en el prim, cada vértice se revisa que no sea vacio, si no es vacio se hace un key\_set de todos los domicilios en ese vertice. Luego se revisa cada uno de los domicilios y si no aparece en ninguno el domiciliario buscado entonces se sale del while. |
| Paso 4 | O(1) Revisa si el mst es válido o no, si no es válido vuelve a hacer un nuevo mst eliminando el vértice que no tiene al domiciliario y se vuelve a hacer toda la función (recursión) |
| Paso 5 | O(1) Se saca la cantidad de ubicaciones visitadas en el mst. |
| Paso 6 | O(V) se hace una lista de todos los vertices visitados usando un for |
| Paso 7 | O(VlogV) se ordena esa lista usando merge sort |
| Paso 8 | O(V) se calcula el pesototal del mst usando un for |
| ***TOTAL*** | ***O(VlogE + V^2 + 2V + VlogV)*** |

## **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.

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Este requerimiento tiene un crecimiento de complejidad muy grande gracias a ese V^2 que aparece en la complejidad. Eso se da por la recursión usando dos whiles uno dentro de otro. Lo bueno es que como los vertices son siempre mucho menos que los arcos la complejidad podría ser peor.

# **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.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| small | Dato1 | 0.05 |
| 5 pct | Dato2 | 0.33 |
| 10 pct | Dato3 | 1.28 |
| 20 pct | Dato4 | 2.54 |
| 30 pct | Dato5 | 4.98 |
| 50 pct | Dato6 | 7.51 |
| 80 pct | Dato7 | 13.81 |
| large | Dato8 | 25.97 |

### **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.