Análisis del reto 2

Estudiante 1: Sergio Iván Rincón, 201914107, si.rincon@uniandes.edu.co
Estudiante 2: Luis Ernesto Tejón, 202113150, l.tejon@uniandes.edu.co

Octubre de 2021

1. Observaciones generales

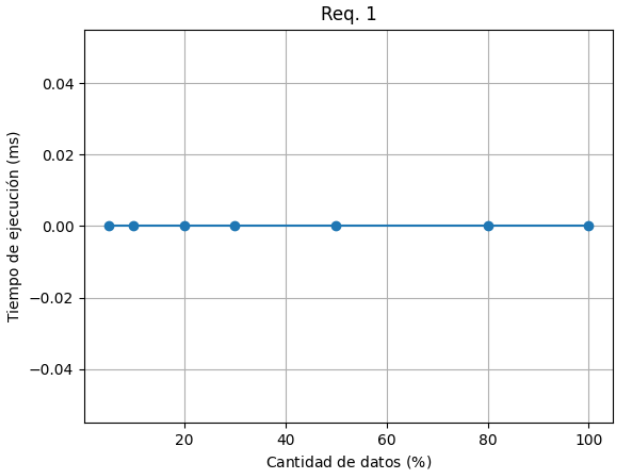
En este caso, y de manera análoga al reto anterior, se decidió priorizar el tiempo de las consultas sobre el tiempo de la carga de datos. De esta manera, en la carga de datos se realizan procesos que hacen que su orden de crecimiento sea alto, pero que luego disminuyen significativamente los tiempos de consulta. Adicionalmente, en los dos primeros requerimientos se decidió mantener el algoritmo empleado para el reto anterior, con una búsqueda binaria modificada que obtiene diferentes rangos. Esto se debe a que los maps no ordenados no permiten mejorar la complejidad logarítmica que se obtiene con esta implementación. Para mejorarla, se deberían usar árboles binarios.

Por último, como aclaración, los tiempos medidos en 2 de los requerimientos es de 0, estos tiempos los medimos con módulo 'time' de Python del mismo modo a como se hizo en alguno de los laboratorios, aunque no estamos seguros de por qué, pensamos que es posible que como las mediciones de tiempo son tan pequeñas que el módulo los aproxima a 0 automáticamente, lo que sí estamos seguros es que todas las consultas son prácticamente igual de rápidas sin importar si se usa el archivo small o large, pues no varían en más que milisegundos.

2. Análisis de complejidad por requerimiento

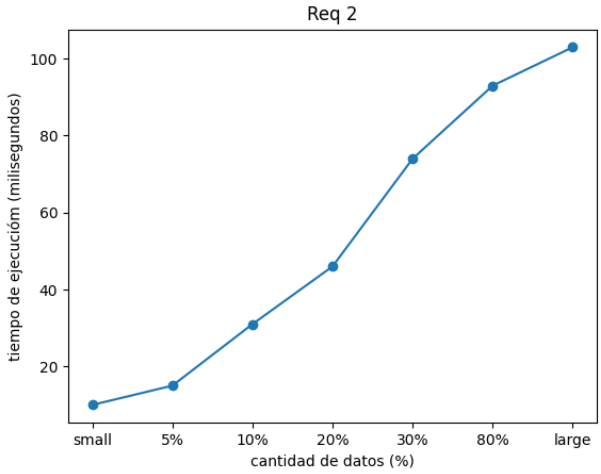
2.1. Requerimiento 1

- Tipo de TAD usado: TAD lista (array list)
- Carga de datos: Para este requerimiento se agregó en la carga inicial de datos un ordenamiento tipo Merge para crear una lista de autores ordenada por fecha de nacimiento.
- Descripción requerimiento: En este requerimiento se empleó un algoritmo de búsqueda binaria modificado que, mediante dos búsquedas binarias con ciertas características, retorna las posiciones inicial y final de un intervalo. Este algoritmo se aplicó para consultar los artistas que nacieron en el año especificado mediante una lista ordenada. Teniendo en cuenta este hecho, la complejidad de la consulta tiende a ser $O(2 \log n)$, ya que la cantidad de obras se determina a partir de las posiciones encontradas. Cabe aclarar que esto es así debido a que sólo es necesario retornar 6 obras, ya que si hubiese que imprimir todas, la complejidad sería $O(n)$.
- Complejidad de la consulta: $O(\log n)$
- Comparación con el reto anterior: En este caso el código se mantuvo, por lo que el rendimiento de la consulta es el mismo.
- Gráfica de pruebas temporales (tiempos reportados en milisegundos y cantidad de datos en porcentaje):



2.2. Requerimiento 2

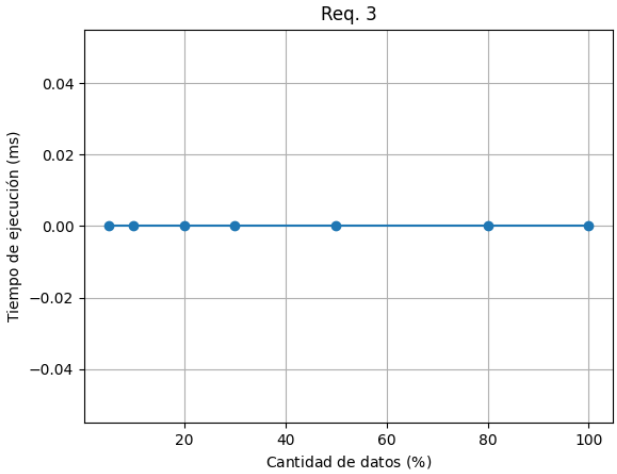
- **Tipo de TAD usado:** TAD lista (array list)
- **Carga de datos:** Para este requerimiento se agregó en la carga inicial de datos un ordenamiento de tipo Merge para crear una sublista de las obras, pero ordenada por la fecha de adquisición.
- **Descripción requerimiento:** En el requerimiento al ya estar organizada la lista, hizo falta únicamente el uso de 2 búsquedas binarias para hallar las posiciones de la menor obra con la fecha inicial (o la menor obra con una fecha inmediatamente mayor) y la posición de la mayor obra con la fecha final (o la inmediatamente menor) con lo que, al ser de tipo array se pudo "cortar" la lista a partir de esas posiciones en O (1). Al final en la lista resultante se hizo una iteración para ver cuántas obras son adquiridas por compra la cual tiene una complejidad de n.
- **Complejidad de la consulta:** tilda($2logn + m$) con n el número de obras totales en el catálogo y m el número de obras en el rango de fechas dado. En notación Big O sería $O(logn + m)$ puesto que no podemos despreciar m, ya que dependiendo de la consulta puede ser del mismo tamaño que n, sin embargo en el caso promedio sería despreciable por lo que en el caso promedio la complejidad sería $O(logn)$
- **Comparación con el reto anterior:** En este requerimiento decidimos hacer uso de la misma solución del Reto 1, esto debido a que para usar mapas habría que haber creado un mapa con llaves las fechas de adquisición y valores las obras de cada fecha, para luego al momento de consultar, ir fecha por fecha viendo si está en el rango y luego organizarlas, lo que hubiera tenido una complejidad de n+mlogm con n el número de fechas distintas y m el número de obras en el rango. Esta solución hubiera tenido una complejidad menor en la carga de datos pero mayor en la consulta, por lo que se optó por continuar con la solución del reto anterior.
- **Gráfica de pruebas temporales (en milisegundos):**



2.3. Requerimiento 3 (implementado por Sergio Rincón)

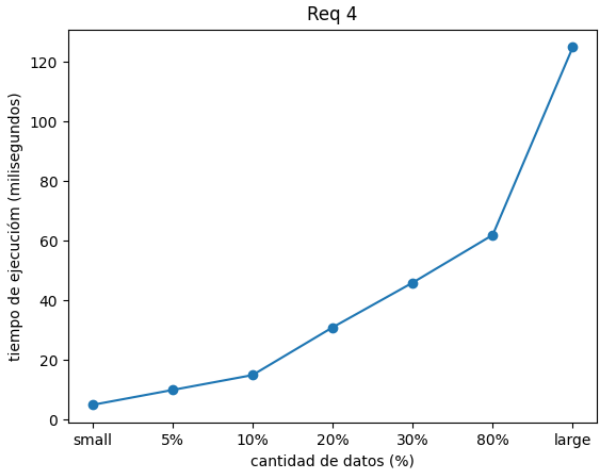
- Tipo de TAD usado: TAD lista (array list) y TAD map (linear probing)
- Carga de datos: Para cargar estos datos se generó un map de tipo linear probing usando como llave el ID de los artistas y como valor un diccionario pequeño que contenía una lista de obras organizada por medio, y varios datos sobre el artista y los medios empleados. Además, y para encontrar el ID por nombre, se creo una lista organizada por nombre de diccionarios de dos elementos que contenían el nombre y el ID de los diferentes artistas.
- Descripción requerimiento: Contando con los datos cargados la consulta primero hace una búsqueda binaria en la lista de artistas ordenada por nombre, encontrando así el ID del artista. Con este valor se dirige al map y encuentra el diccionario con la lista de obras organizada y los datos asociados. Dentro de estos datos se encuentra el medio más empleado y, para obtener las obras en las que se empleó, se usa el algoritmo de búsqueda binaria modificada sobre la lista de obras organizada.

  A partir de lo anterior, el orden de crecimiento de este algoritmo es logarítmico, debido a la realización de las busquedas binarias. Por su parte, la consulta en el map es cercana a $O(1)$.
- Complejidad de la consulta: $O(\log n)$ en promedio, pero si sólo hubiese un autor con obras de una sola técnica sería $O(n)$
- Comparación con el reto anterior: En el reto anterior la implementación fue similar, pero usando diccionarios en todos los casos. Tendiendo en cuenta este hecho, la complejidad es similar, aunque se observó que la carga de datos desmejora un poco con la inclusión del map. Adicionalmente, se podría implementar más maps y hacer más de dos indexamientos en vez de usar la búsqueda binaria para consultar las obras del medio más usado, pero esto podría agregarle una mayor complejidad a la carga de datos, algo que se quiso evitar, ya que la carga de datos ya tardaba un tiempo considerable.
- Gráfica de pruebas temporales (en milisegundos):



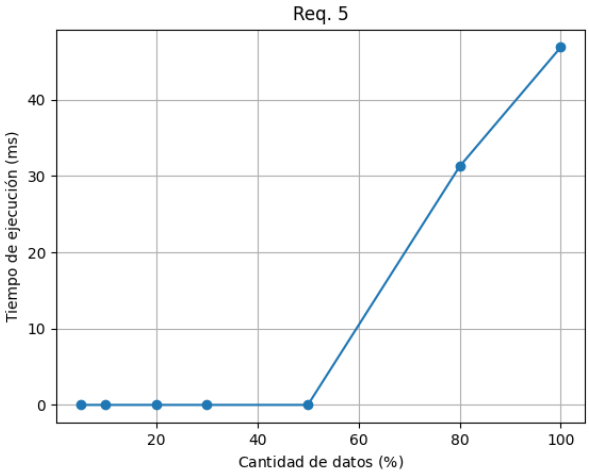
2.4. Requerimiento 4

- **Tipo de TAD usado:** TAD lista (array list), TAD map (Chaining) y TAD map (linear probing)
- **Carga de datos:** Para este requerimiento al cargar los datos se creo una copia de la lista de artistas y se organizó con un merge por su 'ConstituentID', luego se crearon dos mapas tipo 'Probing' con keys las nacionalidades y en unos los values eran listas vacías tipo 'Array' mientras que en el segundo mapa los values eran mapas vacíos tipo 'Chaining', luego se iteró sobre la lista de obras para que, con los ConstituentID asignados a las obras y con el uso de una búsqueda binaria, encontrar sus autores y se hacía put y addlast (en cada mapa respectivamente) de esa obra en los key de las nacionalidades de sus creadores. Por último se iteró sobre los mapas de nacionalidades para hacer dos listas de listas que luego se organizó según el size de cada sublista. nacionalidad.
- **Descripción requerimiento:** Básicamente en este requerimiento solo se consulta la información cargada en los datos, pues de una de las listas finales creadas en la carga de datos, se obtiene su primer elemento el cual es la lista de obras del país con más obras únicas, y del mismo modo se obtiene los primero 10 elemento de la otra lista para así tener las listas de obras de los 10 países que tiene más obras
- **Complejidad de la consulta:** O(1), solo se lee la información guardada en el catálogo y se imprime.
- **Comparación con el reto anterior:** En el reto anterior este requerimiento tuvo una complejidad mucho mayor en la carga de datos, ya que al no hacer uso de mapas fue necesario para crear una de estas listas fue necesario hace uso de la función IsPresent() en algunos casos, lo cual tiene una complejidad de n.
- **Gráfica de pruebas temporales (en milisegundos):**



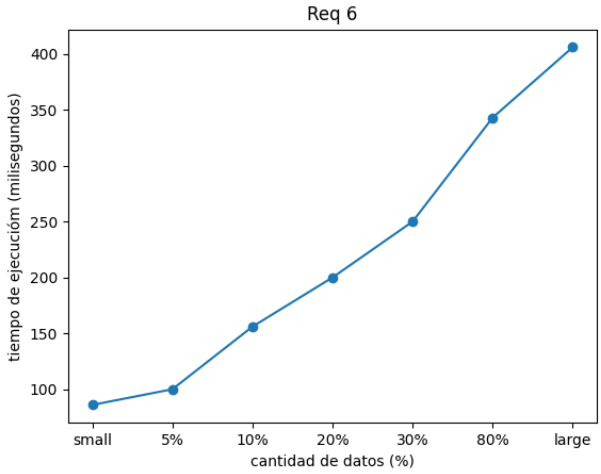
2.5. Requerimiento 5

- Tipo de TAD usado: TAD lista (array list) y TAD map (linear probing)
- Carga de datos: En este caso se crea un map por departamento que contiene un diccionario con una lista de obras organizada por fecha y otros datos asociados al departamento como peso, obras mas costosas, etc. Esto permite que en la carga de datos se guarden los datos de respuesta del requerimiento.
- Descripción requerimiento: En este caso se busca en el map el nombre del departamento y se encuentra su diccionario asociado. Luego, con esta información se extraen del diccionario los datos asociados al peso, costo de transporte y de la lista de obras organizada por antiguedad se obtienen las obras más antiguas. Además, y con los IDs de los autores, se hace una búsqueda binaria en la lista de autores para obtener los nombres. En este caso se decidió realizar una búsqueda binaria en esta última parte para no agregar más maps a la carga de datos, ya que el tiempo de es bastante alto y los tiempos con la búsqueda siguen siendo mínimos al emplear el archivo large.
- Complejidad de la consulta: $O(\log n)$
- Comparación con el reto anterior: En este caso la complejidad de la consulta no podía mejorar porque en el reto anterior se implementó de manera análoga pero con diccionarios (de hecho, en el reto anterior la complejidad está mal resportada, debido a que no se tuvo en cuenta la búsqueda binaria de artistas). Además, se observó que la complejidad de la carga de datos aumentó en este caso, algo asociado a la implementación del map.
- Gráfica de pruebas temporales (en milisegundos):



2.6. Requerimiento 6 (bono)

- **Tipo de TAD usado:** TAD map (Linear probing)
- **Carga de datos:** En este requerimiento en la carga de datos se creó un mapa tipo Linear probing con keys el Id de cada artista y values una lista con las obras de ese artistas. Aunque naturalmente esto tendría una complejidad alta, el aporte a la carga de datos fue bastante pequeño, pues se aprovechó de la misma iteración que se hizo sobre las obras en el requerimiento 4, con lo que se logró hacer que el aporte fuera mínimo, pues no se hizo ninguna iteración nueva solo se subió en 1 el número de asignaciones por iteración.
- **Descripción requerimiento:** Gracias a que en el requerimiento 1 ya se había creado una función para dar una lista de artistas en rango de fechas dado en tan solo $2logn$ se reutilizó esa misma función para encontrar los artistas en el rango, luego, utilizando el mapa creado en la carga de datos, se iba artista por artista en el rango de años, viendo sus obras, y guardando, cuántas obras hizo, cuánto medios uso, y cuál fue el medio que más uso y se iba guardando la información de cada artista en una lista, esto último tendría un complejidad promedio de m, con m el número de obras totales de los artistas en el rango de años dado. Por último se organizaba esa lista de artistas del rango usando un merge sort que a su vez usaba una función de comparación que seguía los criterios planteados en el enunciado.
- **Complejidad de la consulta:** tilda($2logn + m + klogk$) (con n es el número de artistas totales, m el número de obras totales de los artistas en el rango de años y k el número de artistas en el rango) y en notación Big O, puesto que claramente m es el término que más afecta a la complejidad, sería $O(m)$, con m el número de obras en el rango de años, que en el peor de los casos serían todas, aunque en un caso promedio sería solo pequeño porcentaje.
- **Gráfica de pruebas temporales (en milisegundos):**



3. Datos adicionales

3.1. Ambientes de prueba

Los requerimiento 1, 3 y 5 se les midió el tiempo en la maquina 1, mientras que la carga de datos y los requerimiento 2, 4 y 6, se les midió el tiempo en la maquina 2.

	Máquina 1	**Máquina 2**
Procesadores	AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2.61GHz
Memoria RAM (GB)	16GB	12GB
Sistema Operativo	Windows 10 home 64-bits	Windows 10 home 64-bits

3.2. Tiempos de la carga de datos (en segundos):