**OBSERVACIONES DE LA PRÁCTICA**

Gabriela Gomez 202420506

Juan Camilo Cancelado

Pedro Archila 202421572

|  |  |
| --- | --- |
|  | Máquina 1 |
| Procesador | Chip M1 |
| Memoria RAM (GB) | 8 GB |
| Sistema Operativo | macOS Sequoia |

Tabla 1. Especificaciones de la máquina para ejecutar las pruebas de rendimiento.

## **Resultados**

|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo PROBING*** | | |
| **Factor de Carga (PROBING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución Real @LP [ms]** |
| 0.1 | 824962,36 | 156703,39 |
| 0.5 | 714707,77 | 45398,06 |
| 0.7 | 706801,51 | 34342,75 |
| 0.9 | 702461,48 | 32113,71 |

Tabla 2. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING

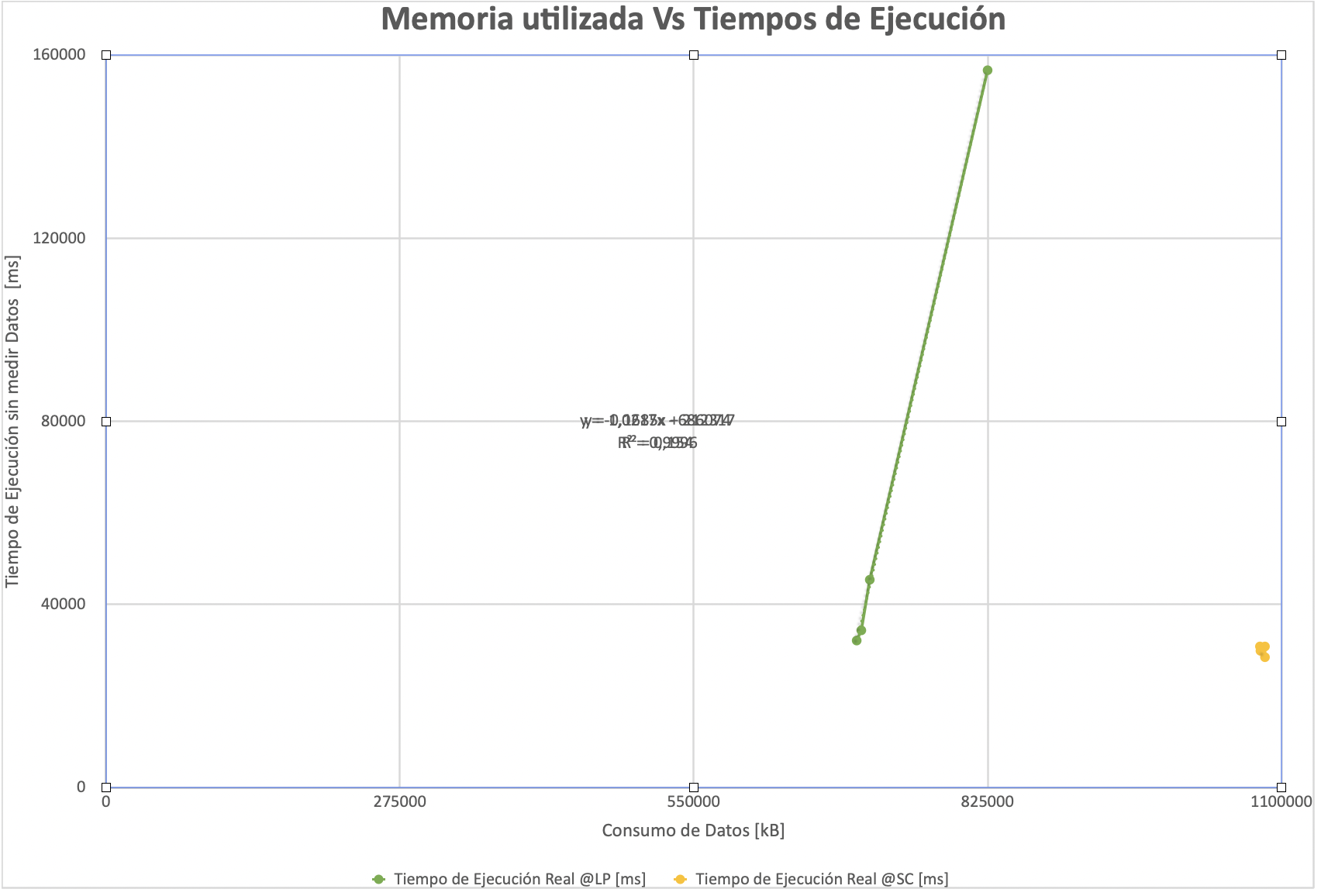
|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo CHAINING*** | | |
| **Factor de Carga (CHAINING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución Real @SC [ms]** |
| 2.00 | 1084500,39 | 30817,26 |
| 4.00 | 1084501,64 | 28475,95 |
| 6.00 | 1080287,30 | 29874,29 |
| 8.00 | 1079795,27 | 30825,08 |

Tabla 3. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING

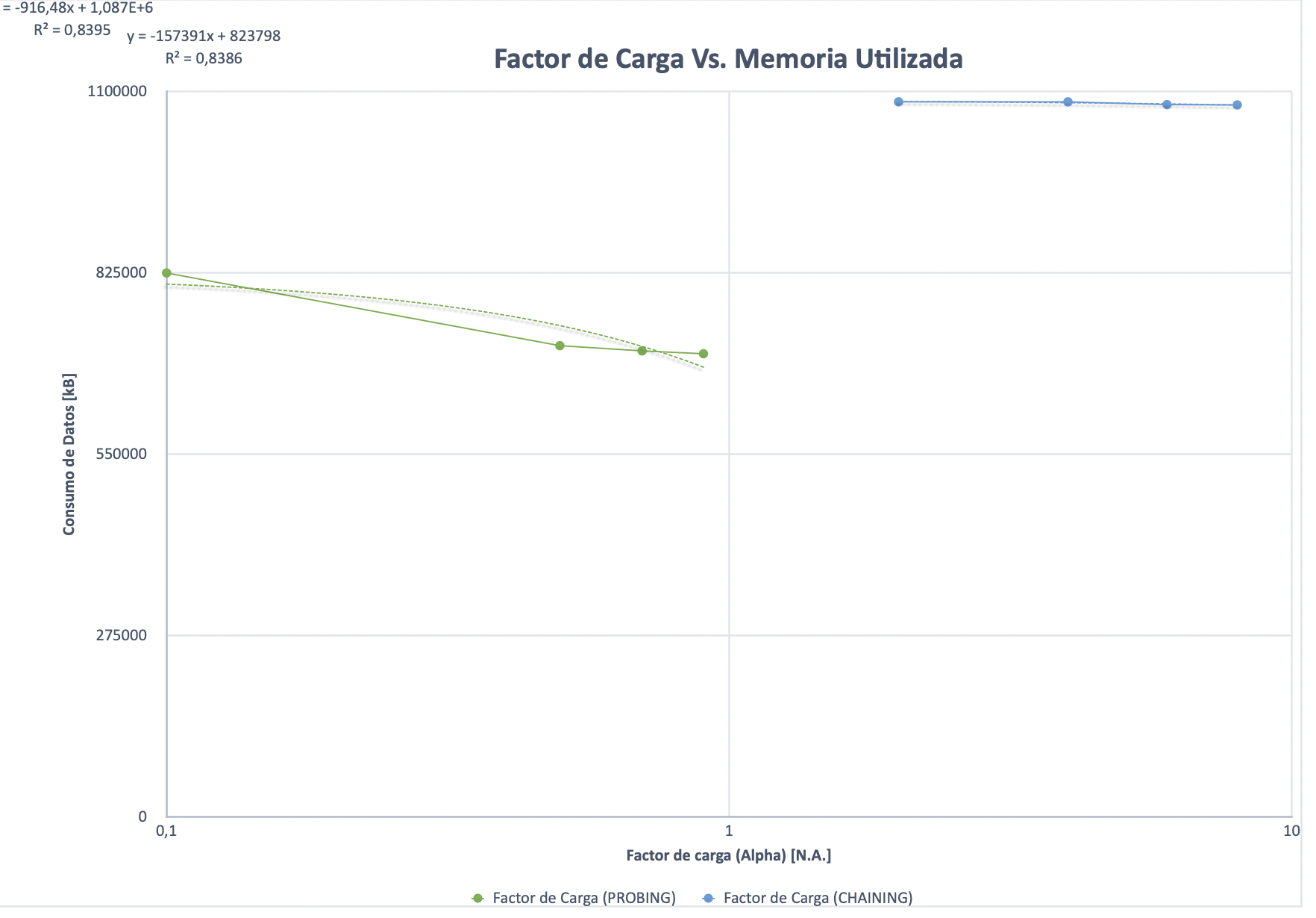
## **Gráficas**

La gráfica generada por los resultados de las pruebas de rendimiento**.**

* Comparación de memoria y tiempo de ejecución para PROBING y CHAINING



* Comparación de factor de carga y memoria para PROBING y CHAINING



# **Preguntas de análisis**

1. ¿Por qué en la función **getTime()** se utiliza **time.perf\_counter()** en vez de otras funciones como **time.process\_time()**?

Se utiliza time.perf\_counter() porque dentro de getTime() es la función más precisa para medir intervalos de tiempo. Además, es una función que sirve para medir el tiempo cuando se usan conjuntos muy grandes de datos.

1. ¿Por qué son importantes las funciones **start()** y **stop()** de la librería **tracemalloc**?

Tracemalloc es una librería que ayuda a medir el consumo de memoria de un código. La función start() empieza el rastreo del consumo de la memoria y guarda la cantidad de memoria que se tiene en un inicio. La función stop() detiene el rastreo de la memoria y guarda la cantidad que se usó de la misma. Con estas funciones se puede realiar un análisis de consumo de memoria por parte de códigos que puedan utilizar datos muy grandes que necesiten de un gran espacio en la computadora.

1. ¿Por qué no se puede medir paralelamente el **uso de memoria** y el **tiempo de ejecución** de las operaciones?

No se pueden medir estos dos aspectos de manera simultanea, pues las funciones y librerías usadas buscan obtener los datos más precisos y óptimos y la medición de una de las dos puede interferir en los resultados de la otra y viceversa.

1. Dado el número de elementos de los archivos (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?

El factor de carga ideal cuando se realiza una tabal de hash usando linear probing es de 0.5, lo cual significa que como máximo estará llena la mitad de la tabla, esto para evitar colisiones. Esto daría como resultado un mapa con capacidad de 20002 espacios, pues el archivo de elementos large tiene un total de 10001 datos. Por otro lado, cuando se realiza un mapa con separate chaining lo ideal es que el factor de carga sea de 4, lo cual resultaría en una tabla de 2501 espacios donde en promedio cada slot tendría 4 elementos.

1. ¿Qué cambios percibe en el tiempo de ejecución al modificar el factor de carga máximo?

Se puede ver que mientras menor sea el factor de carga, más tiempo tardará en realizarse la función pues con un factor de carga pequeño, más veces se tendrá que hacer rehash de la tabla resultando en una tabla con una capacidad muy grande y un tiempo de duración muy grande. Sin embargo, un factor de carga muy grande puede resultar en muchas colisiones lo cual en linear probing es un gran problema para el tiempo de ejecución, pues se demora mucho más en buscar un elemento, no tanto así en separate chaining, pues cada slot puede tener la cantidad de elementos que sea. En separate chaining, un factor de carga grande o pequeño tienen resultados muy similares en cuanto al tiempo de ejecución y se puede ver como el ideal de 4 es donde el separate chainig se demora menos tiempo.

1. ¿Qué cambios percibe en el consumo de memoria al modificar el factor de carga máximo?

Un factor de carga muy pequeño hace que se deba realizar muchas veces un rehash de la tabla terminando en un mapa muy grande que ocupa demasiado espacio, por otro lado, cuando el factor de carga es muy grande la tabla no tendrá que hacer rehash por lo tanto no ocupará tanto espacio. En cuanto al separate chaining, la memoria usada es casi igual usando cualquier factor de carga.

1. ¿Qué cambios percibe en el tiempo de ejecución al modificar el esquema de colisiones? Si los percibe, describa las diferencias y argumente su respuesta.

Según los resultados obtenidos, separate chaining se demora menos en realizarse que linear probing, esto se puede dar debido a la cantidad de rehash que hace y debido a que para poner un elemento debe encontrar la primera posición libre desde el hash del valor, si hay muchas colisiones se demorará mucho más que separate chaining.

1. ¿Qué cambios percibe en el consumo de memoria al modificar el esquema de colisiones? Si los percibe, describa las diferencias y argumente su respuesta.

En cuanto al consumo de memoria se nota una clara diferencia entre linear probing y separate chaining. Si bien las dos están guardando los mismos elementos, la capacidad de la tabla de hash usando linear probing es mucho más grande que la capacidad de la tabla realizada con separate chaining. Esto se da porque el factor de carga en linear probing es mucho menor que el factor de carga con separate chaining y mientras menor es el factor de carga, mayor es el tamaño de la tabla.