**OBSERVACIONES DE LA PRÁCTICA**

Estudiante 1 Cod 202420972

Estudiante 2 Cod 202420978

Estudiante 3 Cod 202420380

|  |  |
| --- | --- |
|  | Máquina 1 |
| Procesador | Chip 3 |
| Memoria RAM (GB) | 128 GB |
| Sistema Operativo | 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 | |  | | --- | | 14165365.43 | | |  | | --- | | 276134.01 | |
| 0.5 | 3179718.32 | 46992.85 |
| 0.7 | 2400392.20 | 37497,00 |
| 0.9 | 1971988.06 | 32179.43 |

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

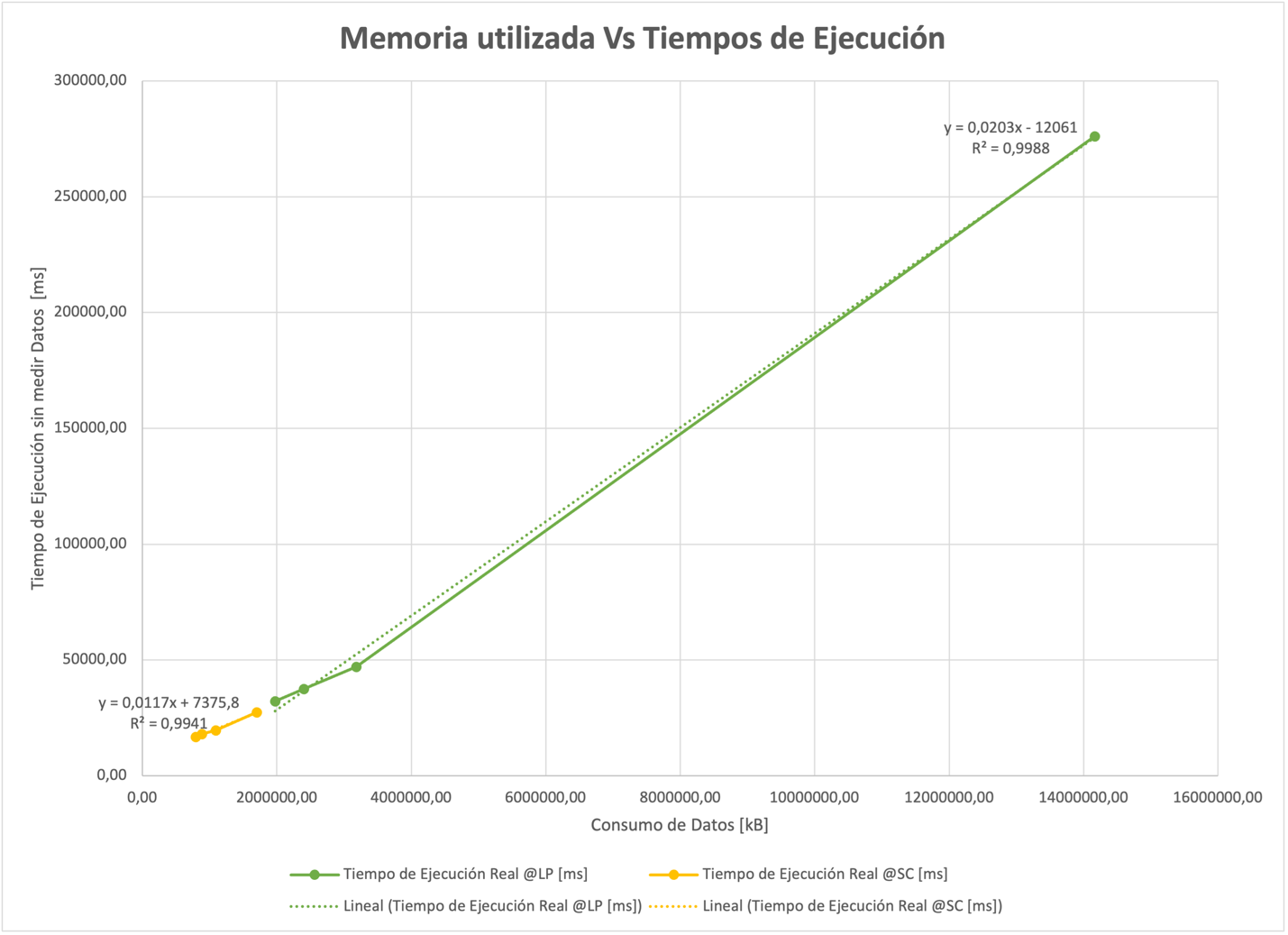
|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo CHAINING*** | | |
| **Factor de Carga (CHAINING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución Real @SC [ms]** |
| 2.00 | |  | | --- | | 1698809.11 | | |  | | --- | | 27466.22 | |
| 4.00 | 1088613.95 | 19591.56 |
| 6.00 | 884653.55 | 17956.07 |
| 8.00 | 787569.83 | 16805.40 |

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

## **Gráficas**

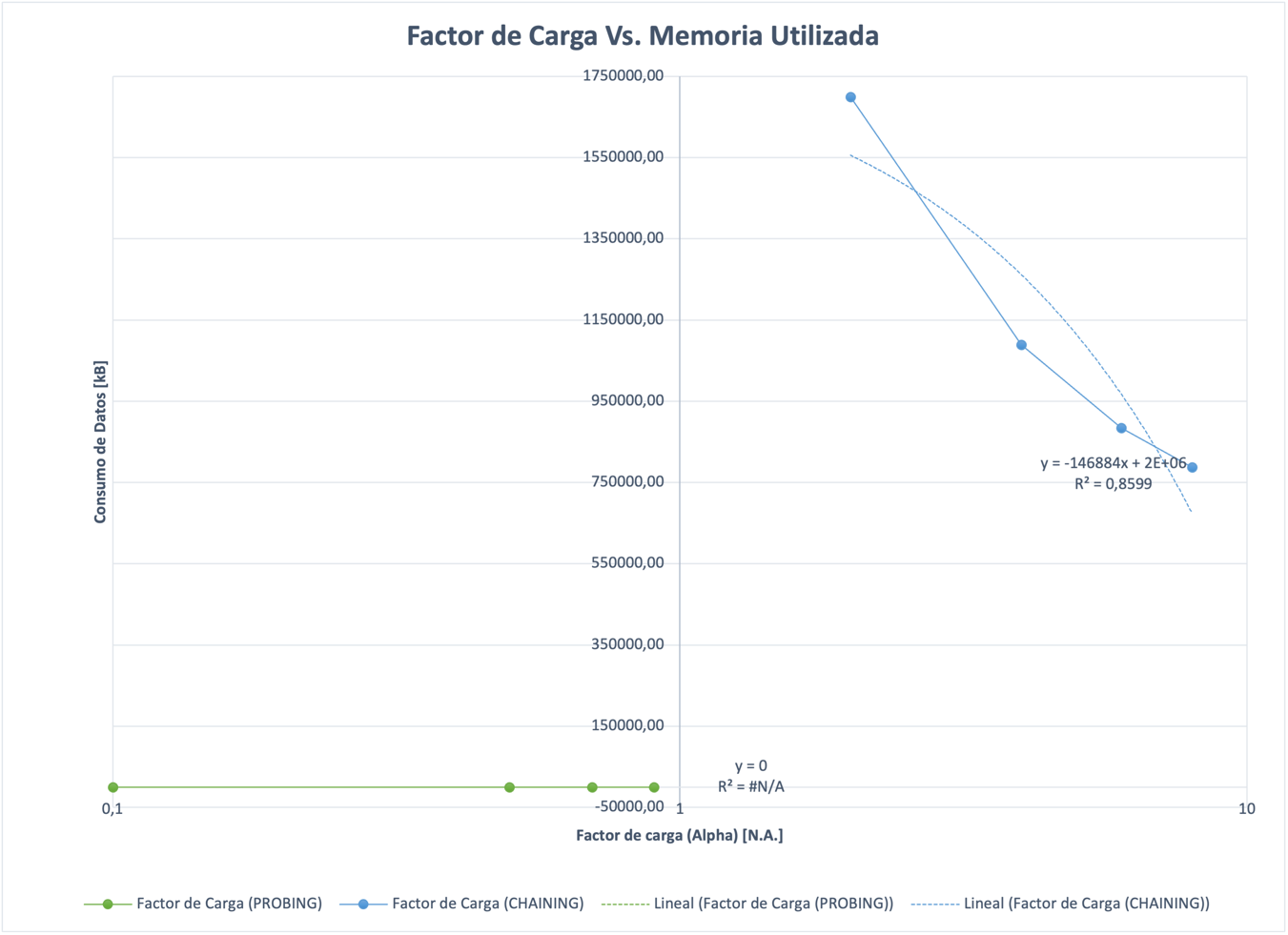
La gráfica generada por los resultados de las pruebas de rendimiento**.**

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



**Gráfica 1: Memoria vs Tiempos de Ejecución para Linear Probing y Separate Chaining.**

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



**Gráfica 2: Factor de Carga vs Memoria Utilizada para Linear Probing y Separate 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()**?

Esto se debe a un tema de eficacia, por como están implementadas ambas funciones. Porque **time.perf\_counter()** mide el tiempo real (o wall-clock time) con alta precisión, incluyendo pausas del sistema y procesos externos. En cambio, **time.process\_time()** solo mide el tiempo de CPU, ignorando tiempos de espera o ejecucción paralela lo cual no refleja el tiempo total real que tarda el algoritmo. Por lo tanto, si queremos datos reales que muestren fielmente los datos, es mucho más recomendable utilizar la función **time.perf\_counter().**

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

Las funciones start() y stop() de la librería tracemalloc son importantes porque permiten activar y desactivar el rastreo del uso de la memoria para medir exactamente cuánta memoria usa una función específica, sin interferencias de otros procesos del programa. Mientras que start() comienza a registrar las asignaciones de memoria, stop() detiene ese rastro. Esto es útil en el laboratorio porque permite comparar con claridad cuánto consume cada estructura (por ejemplo, probing vs chaining) sin que se mezclen otras operaciones, y así tener una medición precisa y aislada del comportamiento de memoria.

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

Porque tracemalloc y time.perf\_counter() afectan el rendimiento si se ejecutan al mismo tiempo. Al medir ambos simultáneamente, se pierde precisión porque cada medición interfiere con la otra. Por eso, para obtener resultados confiables, se recomienda medir primero el tiempo y luego la memoria (o viceversa), y así asegurar que los valores registrados correspondan únicamente a la operación analizada y no a efectos secundarios del monitoreo en paralelo.

1. Dado el número de elementos de los archivos (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?
2. ¿Qué cambios percibe en el tiempo de ejecución al modificar el factor de carga máximo?

Según los datos del laboratorio, cuando el factor de carga aumenta, el tiempo también aumenta. Esto es evidente con la gráfica 1 que en ambos tipos de manejo de colisiones existe un aumento en el tiempo de ejecucción al modificar el factor de carga. Sin embargo, esta es más notoria en la linear probing que en el chaining. Como muestra, la pendiente del manejo de linear probing muestra una pendiente de 0,02, mostrando una clara relación entre el aumento del factor de carga con el tiempo de ejeccución. Al contrario, la pendiente de separate chaining es de apenas 0,01, la mitad de la anterio.

Esta tendencia se explica por que el aumento del factor de carga implica también un aumento significativo de las colisiones. Por tanto, entre más colisiones más se demorará el tiempo de ejecución pues se deben corregir estos casos excepcionales.

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

Según los datos del laboratorio y lo que se observa en la gráfica 2, al aumentar el factor de carga, el consumo de memoria disminuye. Esto se debe a que una tabla más llena usa menos espacio libre, lo que reduce la cantidad total de memoria reservada. En el caso de separate chaining, esta relación es clara: la memoria cae de forma pronunciada a medida que sube el factor de carga. La pendiente de la regresión lineal es de aproximadamente –146884, lo que indica una fuerte relación negativa entre ambas variables, con un coeficiente de determinación de R² = 0.8599, que respalda esta correlación.

Por otro lado, en el caso de linear probing, la línea es completamente plana con pendiente cero. Esto implica que el consumo de memoria no cambia al variar el factor de carga, lo cual tiene sentido, ya que en probing se usa una tabla de tamaño fijo y no hay estructuras adicionales como listas. En conlcusión, chaining consume más memoria, pero es sensible al factor de carga, mientras que probing mantiene un consumo constante independientemente del mismo.

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.

Sí se perciben cambios importantes al modificar el esquema de colisiones. De acuerdo con los datos del laboratorio y lo mostrado en la gráfica de tiempo, el esquema de linear probing es mucho más sensible al aumento del factor de carga, lo que se refleja en un aumento más pronunciado en el tiempo de ejecución. Esto se debe a que cuando hay colisiones, linear probing tiene que recorrer secuencialmente la tabla hasta encontrar una posición vacía, lo que hace más costosa cada operación de inserción o búsqueda.

En contraste, separate chaining tiene un comportamiento más estable, ya que al manejar colisiones mediante listas enlazadas por índice, evita la necesidad de moverse dentro del arreglo principal. Por eso, incluso cuando el factor de carga es alto, el tiempo de ejecución no crece tan abruptamente. Esto se refleja en que la pendiente del gráfico de tiempo para chaining es más baja que la de probing, mostrando que su rendimiento es más predecible y escalable. Por tanto, aunque ambos esquemas muestran aumentos de tiempo cuando hay más colisiones, el efecto es mucho mayor en probing.

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.

Evidentemente, chaining consume más memoria porque necesita estructuras adicionales (listas por índice). Esto se ve claramente en los datos del laboratorio: el esquema de chaining tiene un consumo de memoria mucho más alto que probing en todos los casos. En probing, como no se crean listas sino que todo se guarda directamente en la tabla, el uso de memoria se mantiene constante. Por eso, en la gráfica, la línea de probing es completamente plana mientras que la de chaining baja con el factor de carga, pero siempre está por encima. Esto pasa porque en chaining cada vez que hay colisión se tiene que guardar un nuevo nodo en la lista, lo cual ocupa espacio adicional en memoria dinámica. En conclusión, aunque chaining mejora el tiempo, lo hace a costa de gastar mucha más memoria.