**OBSERVACIONES DE LA PRÁCTICA**

Tomas Aponte 202420148

Estudiante 2 Cod XXXX

|  |  |
| --- | --- |
|  | Máquina 1 |
| Procesador | Ryzen 7 5800X |
| Memoria RAM (GB) | 16GB |
| Sistema Operativo | Windows 11 |

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 | 167 | 1.9 |
| 0.5 | 312 | 5.9 |
| 0.7 | 415 | 14.3 |
| 0.9 | 569 | 24.4 |

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

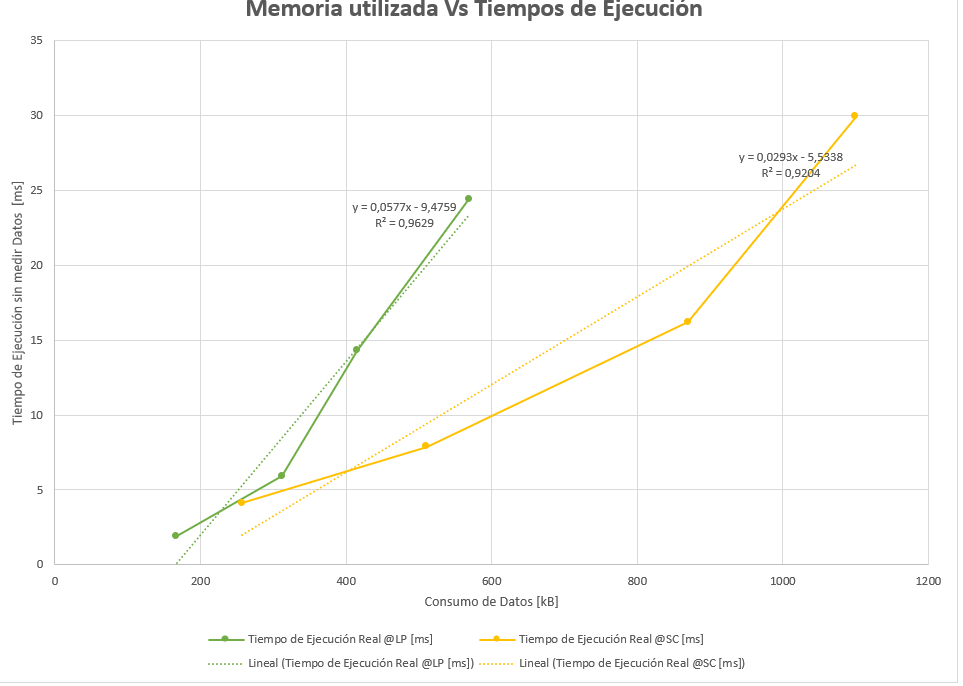
|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo CHAINING*** | | |
| **Factor de Carga (CHAINING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución Real @SC [ms]** |
| 2.00 | 257 | 4.1 |
| 4.00 | 511 | 7.9 |
| 6.00 | 870 | 16.2 |
| 8.00 | 1100 | 29.9 |

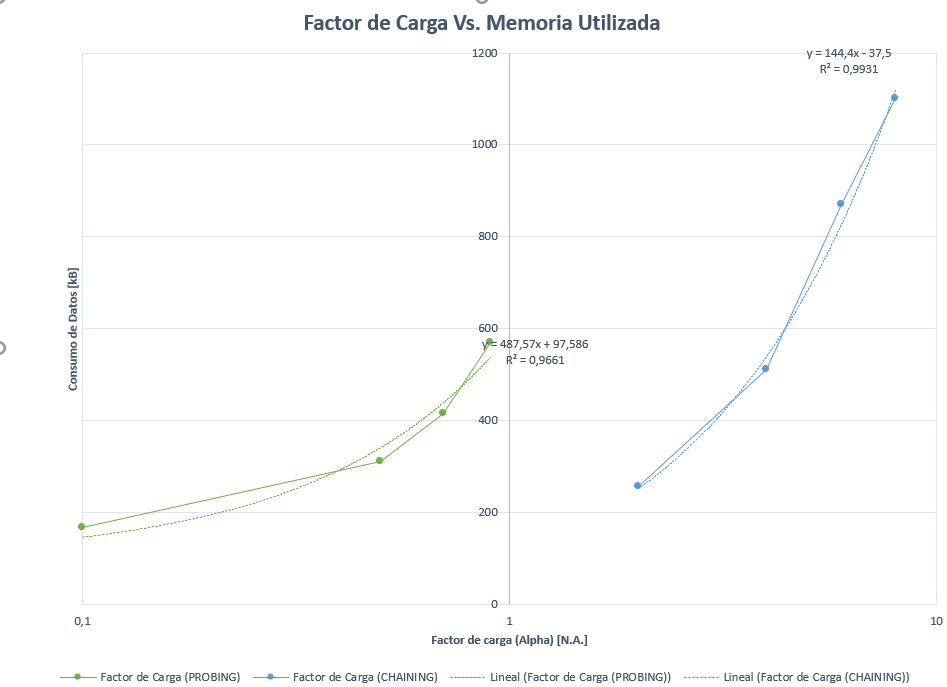
Tabla 3. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING

## **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() y no time.process\_time() porque time.perf\_counter() mide el tiempo de manera que no se ve afectado por los procesos del sistema. En cambio, time.process\_time() mide solo el tiempo que el proceso utiliza y no contabiliza los periodos en los que el proceso puede estar inactivo.

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

Permiten monitorear y comparar el uso de memoria antes y después de realizar una operación, lo que es esencial para optimizar el consumo de memoria.

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

Las mediciones dependen de distintos metodos lo cual puede interferir en el comportamiento del programa y resultar imprecisa.

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?

Si el factor de carga es bajo la tabla de hash se redimensiona más frecuentemente y hay mas operaciones de rehashing. Si el factor de carga es más alto hay menos rehashing pero las búsquedas de elementos en la tabla pueden ser más lentas.

1. ¿Qué cambios percibe en el consumo de memoria al modificar el factor de carga máximo? Respuesta: El factor de carga también influye en el consumo de memoria:

Al aumentar el factor de carga, el consumo de memoria disminuye debido a un mejor aprovechamiento del espacio disponible.

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.

Al aumentar el factor de carga, el consumo de memoria disminuye debido a un mejor aprovechamiento del espacio disponible.

Un factor de carga bajo mantiene más espacio libre en la tabla de hash y reduce el número de colisiones, pero aumenta el consumo de memoria porque hay más espacios vacíos. En cambio, un factor de carga alto utiliza más eficientemente la memoria porque hay menos espacio desperdiciado pero puede haber un aumento de colisiones y tiempo de búsqueda, lo que podria causar más uso de memoria.

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.

El sondeo lineal usa menos memoria porque solo almacena los elementos en la tabla de hash pero sufre de ineficiencia cuando hay muchas colisiones, entonces desperdicia espacio y se afecta el rendimiento. En cambio, el encadenamiento separado maneja mejor las colisiones al utilizar listas vinculadas en cada índice pero esto aumenta el consumo de memoria. En general, sondeo lineal es más eficiente en memoria con baja carga, mientras que encadenamiento separado es más estable en escenarios con alta colisión.