**OBSERVACIONES DE LA PRÁCTICA**

Juan David Echeverry 1 Cod

Nicolás Castaño Cod 202420324

Daniel Rincon Cod 202420898

|  |  |
| --- | --- |
|  | Máquina 1 (Unica) |
| Procesador | M4 standard |
| Memoria RAM (GB) | 16 GB |
| Sistema Operativo | MacOS |

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 | 6250.6 | 3237.44 |
| 0.5 | 1250.3 | 870.06 |
| 0.7 | 892.875 | 627.61 |
| 0.9 | 694.5 | 621.82 |

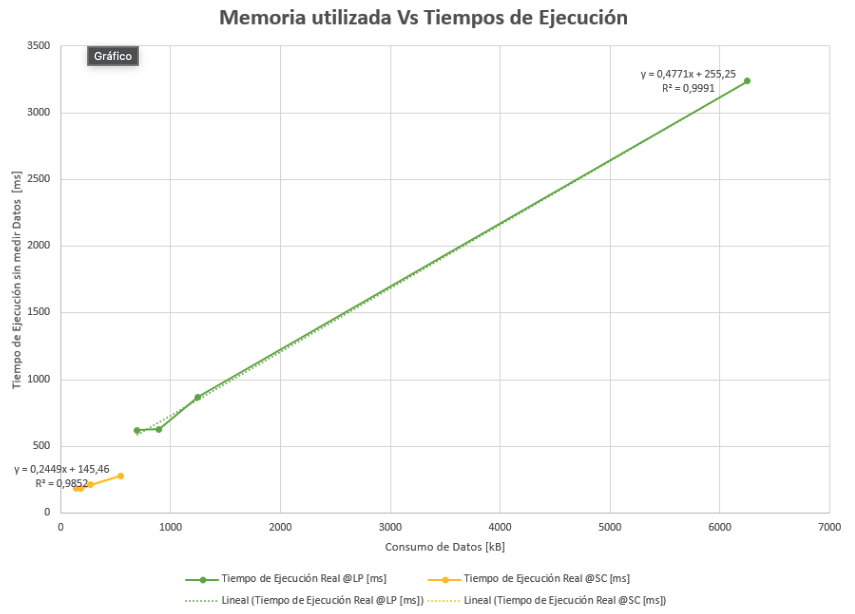
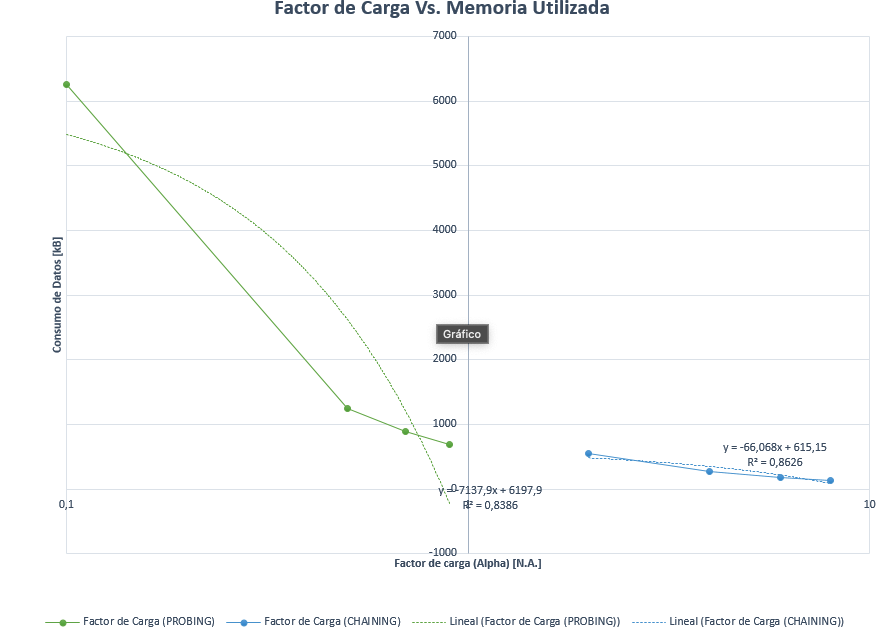
Tabla 2. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING

|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo CHAINING*** | | |
| **Factor de Carga (CHAINING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución Real @SC [ms]** |
| 2.00 | 546.8 | 280.14 |
| 4.00 | 273.4 | 212.45 |
| 6.00 | 182.32 | 182.99 |
| 8.00 | 136.71 | 185.24 |

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()**?

Time.perf\_counter() se utiliza para medir en tiempo real (wall-clock time).

Esta función es utilizada para examinar y registrar la duración de procesos, tomando en cuenta su espera, como por ejemplo el uso de CPU.

A diferencia de time.process\_process() , time.perf\_counter sirve más como medidor de rendimiento de funciones completas.

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

Dado que al usar funciones como start (empezar el rastreo de memoria) y stop (detiene las asignaciones), ayuda a monitorear y comparar el uso que se tiene de la memoria antes y después (dado los fragmentos de tiempo de uso) de ejecutar cualquier línea o fragmento de código

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

No es que no se pueda medir, es que la medición de es exacta ni simultanea dado que herramientas como lo pueden ser la librería tracemalloc o la función perf\_counter, causan una sobrecarga en cuanto a consumir recursos mientras miden (RAM o CPU por ejemplo), y esto puede generar colisiones y distorsionar un poco los resultados .

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

Separate chaining (encadenamiento) tiene un factor de carga ideal de 4(lo cual no quiere decir que 4 elementos tienen que ir por bucket), dado que este tipo de funciones benefician la memoria y ejecución en cuanto a almacenamiento de datos.

Por otro lado, el open adressing (linear probing), el factor ideal de carga tiene que ser 0.5 (generalmente mitad de la tabla vacía y su otra mitad con elementos) dado que el almacenamiento de su algoritmo (arreglos) debe de prevenir colisiones excesivas para su procesamiento de datos.

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

En el caso que se llegue a aumentar el factor de carga máximo, lo más probable que se presenten más colisiones en cuanto a la búsqueda de inserción.

En el caso de que se reduzca el factor de carga máximo, lo más probable que suceda es el mejoramiento de rendimiento de inserción, dado que no hay una cantidad grande de elementos los cuales procesar.

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

Al tener un mayor factor de carga, la tabla dispuesta para la insercion de datos se llena más, lo cual disminuye el rendimiento de ejecución.

Al tener un menor factor de carga, en la tabla habrán más espacios vacios, lo que genera una mejora en el rendimiento y causa que hayan menos colisiones en cuanto a la ejecución.

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.

Cuando hay cambios en el esquema de colisiones, pueda que haya cambios en los tiempos de ejecucion en el mecanismo.

Por un lado, en el algoritmo separate chaining ocurren colisiones en el momento que se almacenan listas que contienen una gran carga de datos dentro de cada “bucket”. Esto significa que el tiempo de inserción va a ser mayor. No obstante, esta estructura de algoritmo coincide con el beneficio de tener un mayor factor de carga, lo que lo hace más resistente.

En el caso de open adressing (de tipo linear probing), es un tipo de algoritmo en el cual cuando ocurre una colisión, el rendimiento disminuye, dado a que cuando la tabla esta más ocupada de elementos, la búsqueda e inserción aumenta su tiempo, creando un fenómeno que se conoce en el Machine Learning como *Clustering,* un conflicto que sufre el sistema de inserción, dado que la ejecución tarda más en buscar un espacio vacío en la tabla.

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.

Al compararel consumo de memoria entre Linear Probingy Separate Chaining se observa que LP utiliza más memoria a medida que aumenta el factor de carga, debido a la necesidad de mantener espacio adicional para minimizar colisiones y realizar rehashing. En cambio, SC maneja las colisiones mediante listas encadenadas, l que reduce la necesiad de redimensionar la tabla y resulta enun uso de memoria más eficiente en factores de carga altos. Para factores de carga bajos o moderados, lp puede ser máseficiente en memoria, pero en escenarios de alta carga, S presenta un menor consumo de memoria al distribuir las colisiones en cadenas.