OBSERVACIONES DE LA PRÁCTICA

Valentina España Cuellar 202414079
Juan Sebastian Cortes Cortes 202411692
Tomas Alarcón Martinez Troncoso 202420126

	Máquina 1
Procesador	Intel Core i7-1255U
Memoria RAM (GB)	8 GB
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	2210470,146	99410,392
0.5	1820936,711	83507,486
0.7	1712145,346	88886,338
0.9	1593599,386	306333,314

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		
4.00		
6.00		
8.00		

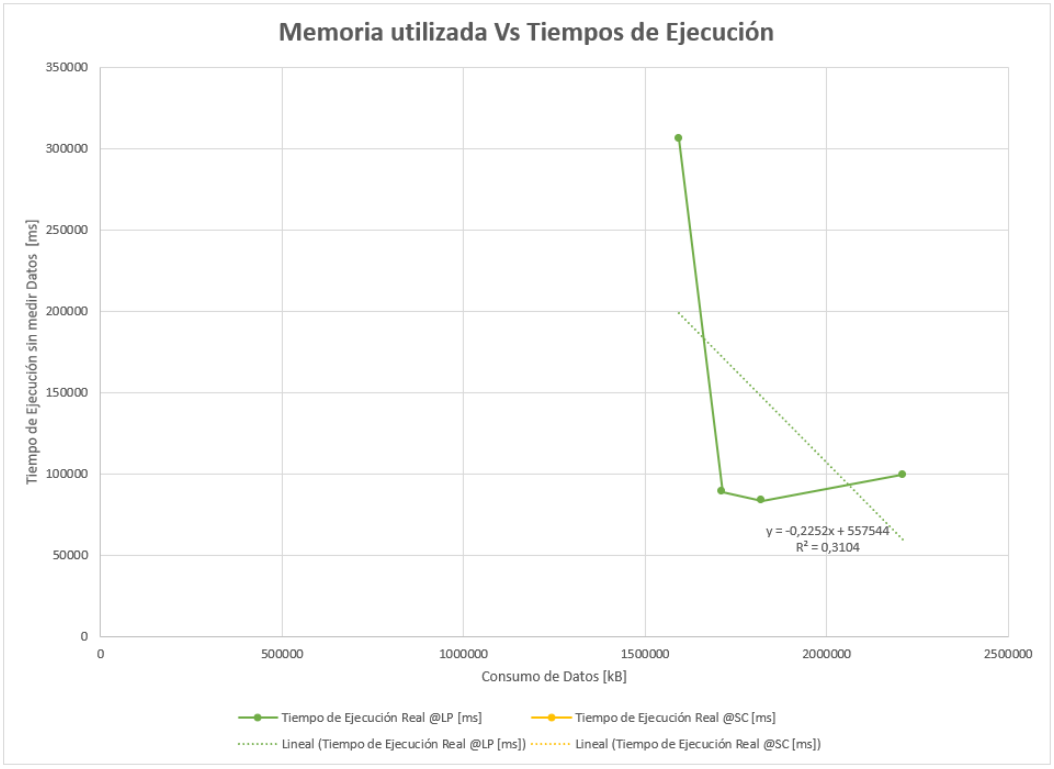
Tabla 3. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING

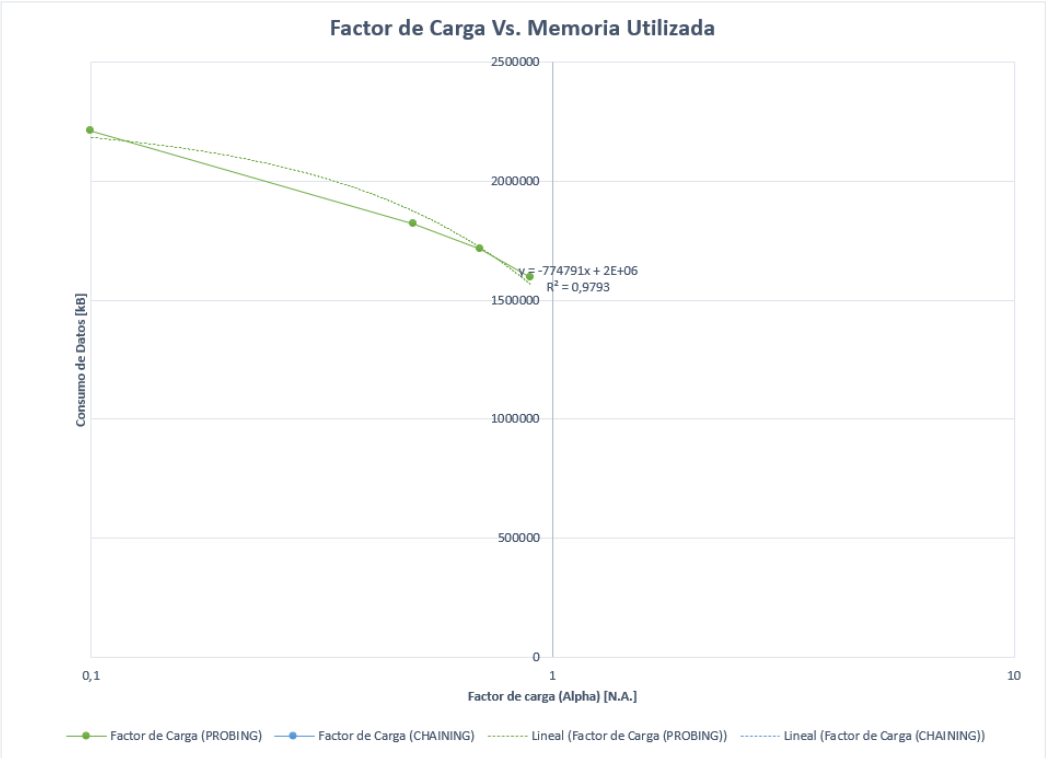
La carga del catalogo con separate chaining supero los 10 minutos.

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


Memoria utilizada Vs Tiempos de Ejecución
Tiempo de Ejecución sin medir Datos [ms]
350000
300000
250000
200000
150000
100000
50000
0
0
500000
1000000
1500000
2000000
2500000
Consumo de Datos [kB]
y = -0,2252x + 557544
R² = 0,3104
Tiempo de Ejecución Real @LP [ms]
Tiempo de Ejecución Real @SC [ms]
Lineal (Tiempo de Ejecución Real @LP [ms])
Lineal (Tiempo de Ejecución Real @SC [ms])


Factor de Carga Vs. Memoria Utilizada
Consumo de Datos [kB]
2500000
2000000
1500000
1000000
500000
0
y = -774791x + 2E+06
R² = 0,9793
0,1
1
10
Factor de carga (Alpha) [N.A.]
Factor de Carga (PROBING)
Factor de Carga (CHAINING)
Lineal (Factor de Carga (PROBING))
Lineal (Factor de Carga (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()**?

La funcion time.perf_counter() mide el tiempo con la mayor precisión posible desde que el sistema arrancó y no se ve afectado por cambios en el reloj del sistema. Es ideal para medir tiempos de ejecución porque incluye tanto el tiempo de CPU como el tiempo de espera. En cambio, time.process_time() solo mide el tiempo de CPU consumido por el proceso, sin contar tiempos de espera. Como en las pruebas de rendimiento interesa el tiempo total transcurrido, time.perf_counter() es más adecuado.

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

Estas funciones permiten medir el uso de memoria durante la ejecución del programa.

- tracemalloc.start inicia el monitoreo del uso de memoria.
- tracemalloc.stop detiene el monitoreo.

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

Porque las mediciones de tiempo time.perf_counter y las de memoria tracemalloc tienen diferentes métodos de captura de datos y pueden interferir entre sí, ya que la primera mide el tiempo total de ejecución, pero no captura el impacto exacto del consumo de memoria en el rendimiento y la segunda monitorea asignaciones de memoria en Python y puede añadir una sobrecarga en el rendimiento.

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

Para linear probing, la capacidad de la tabla se define como el siguiente número primo mayor a num_elem / factor de carga. Si usamos la función next_prime para calcular la capacidad, obtenemos:

- 0,1 = 0,099
- 0,5 = 0,499
- 0,7 = 0,699
- 0,9 = 0,899

Para separate chaining:

- 2 = 1,9
- 4 = 3,9
- 6 = 5,9
- 8 = 7,9

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

Cuando el factor de carga es alto (>0.7 o 0.9), aumenta la cantidad de colisiones y el tiempo de ejecución puede crecer significativamente debido a la cantidad de sondeos lineales necesarios para encontrar un espacio vacío. Esto se ve reflejado en la tabla de resultados, donde a 0.9 el tiempo de ejecución crece más de 3 veces en comparación con 0.7

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

A medida que el factor de carga aumenta, la tabla aprovecha mejor el espacio, pero a costa de un mayor número de colisiones. Un factor de carga bajo genera más posiciones vacías y desperdicio de memoria, mientras que un factor alto (>0.7) reduce el desperdicio pero puede degradar el rendimiento. Si se alcanza el límite del factor de carga, se produce un rehash, lo que temporalmente incrementa el consumo de memoria.

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

En linear probing, el tiempo de ejecución crece conforme aumenta el factor de carga, ya que las colisiones obligan a recorrer más posiciones en la tabla. A bajas cargas, las operaciones son rápidas, pero cuando el factor de carga se acerca a 0.9, la cantidad de comparaciones aumenta drásticamente, empeorando el rendimiento.

8. ¿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 linear probing, el consumo de memoria es más eficiente porque todos los elementos se almacenan en un solo arreglo sin estructuras adicionales, lo que reduce la sobrecarga. Sin embargo, al aumentar el factor de carga, la fragmentación de la tabla y la necesidad de rehash pueden provocar picos de consumo de memoria. En contraste, otros métodos como chaining requieren memoria extra para listas enlazadas, lo que puede ser menos eficiente en términos de uso de espacio.