OBSERVACIONES DE LA PRÁCTICA

Sebastián Felipe Angarita Rico, 202516784, sf.angarita@uniandes.edu.co
Carolina Ceballos Gómez, 202510267, c.ceballosg@uniandes.edu.co
Andrés Santiago Rodríguez Salazar, 202513997, a.rodriguezs@uniandes.edu.co

	Máquina 1
Procesador	Intel(R) Core(TM) i5-12450HX
Memoria RAM (GB)	24 GB
Sistema Operativo	Windows 11

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

Resultados

Carga de Catálogo LINEAR PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	2691782,16	66113,61
0.5	819181,54	36172,43
0.7	690444,67	31526,31
0.9	608394,04	29224,27

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

Carga de Catálogo SEPARATE CHAINING

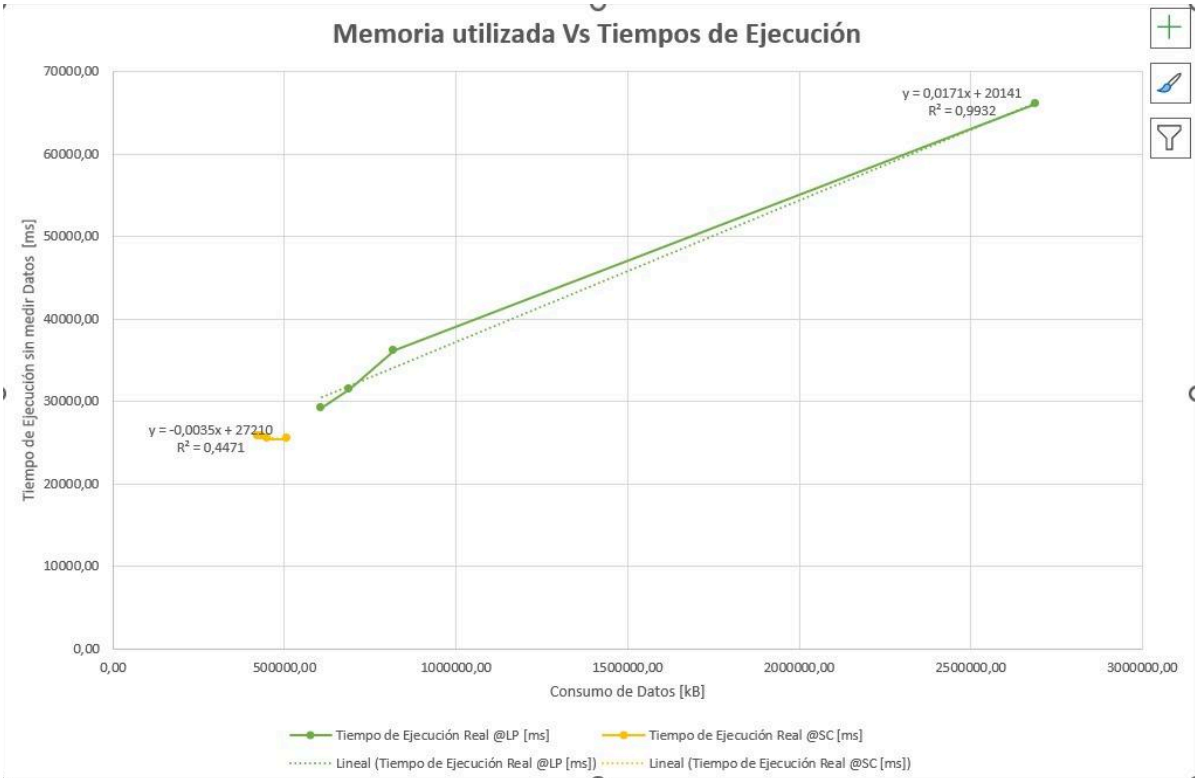
Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	508024,12	25500,74
4.00	451868,93	25439,73
6.00	432739,62	25850,94
8.00	423339,37	25750,38

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

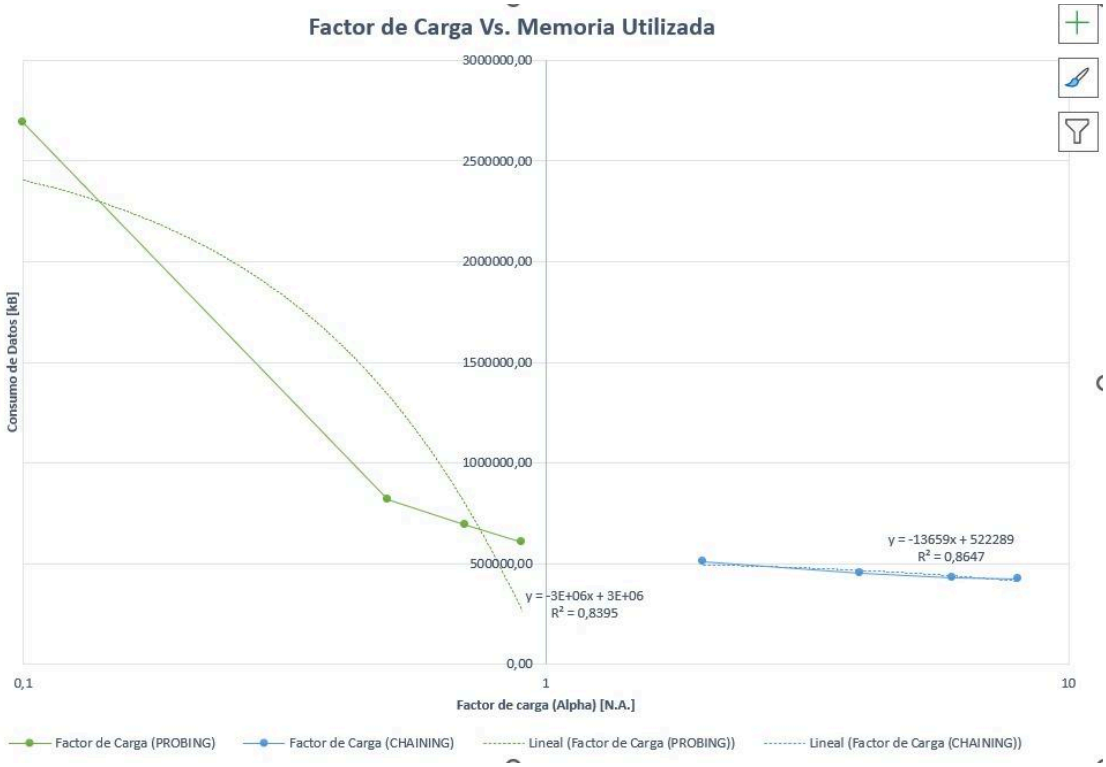
Gráficas

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

- Comparación de memoria y tiempo de ejecución para LINEAR PROBING y SEPARATE CHAINING



- Comparación de factor de carga y memoria 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()**?
   En la función getTime se utiliza time.perf_counter() pues esta función mide el tiempo real que pasa desde que empieza a ejecutarse hasta que termina una tarea, incluyendo absolutamente todo, por ejemplo, incluye las pausas, los accesos a memoria, etc. Es decir, que da un estimado de tiempo muchísimo más real. Mientras que la función time.process_time() mide solo el tiempo que el procesador estuvo trabajando, osea que no podríamos saber cuánto se demoró en ejecutar realmente la operación completa.

2. ¿Por qué son importantes las funciones **start()** y **stop()** de la librería **tracemalloc**?
   Son fundamentales, pues son estas las que nos ayudarán a contabilizar la memoria, es decir, nos dirán cuándo iniciar a contar (start())y cuándo parar (stop()). Sin start() nunca empieza a contar, y sin stop() el programa seguiría guardando datos que podrían terminar siendo innecesarios, lo que nos puede llevar a resultados inexactos. Gracias a estas es que podemos medir cuánta memoria ocupan las operaciones de hashing sin mezclar esta operación con el resto del programa, y es gracias a estas que se activa el registro de la memoria en la aplicación, pues sin ellos, la librería no sería capaz de registrar la snapshot tomada con la función getMemory. Además, si no se hace el stop(), la función que haga start() puede durar mucho más tiempo del que en verdad se demoraría, ya que no se parará manualmente el proceso de medir la memoria.

3. ¿Por qué no se puede medir paralelamente el **uso de memoria** y el **tiempo de ejecución** de las operaciones?
   No se puede medir al tiempo, pues usar la memoria de por sí cuesta tiempo, así que si los medimos los dos a la vez no vamos a obtener los resultados esperados sino unos variados y aumentados, pues se estaría dando gran peso al uso de memoria lo que podría aumentar significativamente el tiempo de ejecución de las operaciones. Por eso lo óptimo es medir ambas por separado y así concluir el uso de memoria y a la vez observar el tiempo de ejecución sin operaciones o funciones que obstaculicen su rendimiento.

1. Dado el número de elementos de los archivos (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?
   Si buscamos el mejor uso de memoria, y considerando las estadísticas, el mejor valor para linear probing sería $\alpha=1$, dado que, aunque aumentáramos el tiempo de ejecución, ya que al final los clusters cubrirán la mayor parte de la lista, esto nos permitiría la mayor eficiencia de espacio, dado

que solo estaríamos reservando en memoria la información que requerimos. Tomando un acercamiento parecido, podemos argumentar que el α más eficiente para separate chaining, es infinito, o sea, un tamaño de lista de 1, pues solo usaríamos la memoria que necesitemos, pues la linked list solo crearía nuevos nodos si la necesitaría, lo que decrementaría la eficiencia temporal, pues implicaría tiempos O(n) para la mayoría de operaciones.

Si, en cambio, buscamos la mayor eficiencia temporal, tenemos que usar un enfoque más experimental. Si tomamos, los datos de las estadísticas, los α más eficientes, serían 0.9 y 4, para linear probing y separate chaining respectivamente. Esta medida se ve alterada, pues se extrae de únicamente agregar datos a la estructura, esto hace que tengamos preferencia por α más altos para linear probing, pues hacemos menos operaciones de relocalización de memoria, lo que impulsa el tiempo de ejecución, como también reduce el tamaño de los clusters, lo que disminuye el tiempo del "put" individual. A su vez, separate chaining minimiza el tamaño de los buckets, lo que reduce el tiempo de revisión para cada uno de los puts, pero si redujera mucho el α, estaríamos desperdiciando la capacidad de los buckets, por eso la mayor eficiencia termina en un valor medio, 4 en este caso.

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

   Al modificar el factor de carga, puedo ver que para Linear Probing, con un factor de carga más bajo (ej 0.1 o 0.2) el tiempo de ejecución es mucho mayor a factores de carga mayores como 0.7 o 0.9, donde el tiempo de ejecución es menor.

   Por el lado de separate chaining, observamos que modificando el factor de carga, los tiempos no varían mucho y todos los tiempos (sin diferencias sustanciales entre factores de carga entre 2 a 8) rondan entre los 25000 ms.

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

   En el consumo de memoria con linear probing percibimos que cuando es un factor de carga menor como 0.1 se desperdicia bastante memoria y puede llegar a tener un consumo de 2691782,16 kB, mientras que si aumentamos el factor de carga a por ejemplo 0.9, el consumo de datos se reduce y llega hasta 608394,04 kB.

   Para separate chaining no percibimos un gran cambio pues para un factor de carga de 2 el consumo de datos está en 508024,12 kB, para un factor de carga de 4 el consumo está en 451868,93 kB. Ya si se modifica más el factor de carga hasta llegar a un factor de 8, el consumo de datos está en 423339,37 kB, que si lo analizamos realmente, no hay gran cambio o variación.

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

   Lo que observamos es que al cambiar el esquema de colisiones es decir, al mirar entre linear probing y separate chaining vemos que para el tiempo de ejecución para el primer esquema mencionado este tiempo si depende bastante del factor de carga y que entre este sea más grande entonces menor será el tiempo de ejecución o viceversa.

   Sin embargo, para el otro esquema de colisiones vemos que el tiempo de ejecución es casi independiente del factor de carga, pues al variar este no encontramos mayor diferencia entre los tiempos, es mucho más constante al variar este factor. Así que podemos llegar a la conclusión de que

separate chaining es más estable y predecible en tiempo que el linear probing, pues linear probing es más sensible al factor de carga.

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

   Los cambios en el consumo de memoria que observamos es que al cambiar el esquema de colisiones, es decir, al mirar entre linear probing y separate chaining, son similares a los cambios que encontramos en el tiempo de ejecución.

   Para linear probing, cuando tenemos un factor de carga pequeño, observamos un mayor gasto o consumo de memoria, mientras que si aumentamos el factor de carga, este consumo disminuye (siendo así inversamente proporcional), pero siendo también sensible a las modificaciones del factor de carga.

   Por el otro lado, con separate chaining el consumo de datos es prácticamente constante, aun teniendo factores de datos cambiantes (entre 2, 4, 6 y 8) y además gasta menos memoria, pues la tabla no necesita crecer tanto ya que se almacenan en listas dentro de cada bucket.

   Así que, como en la respuesta anterior, también podemos concluir que separate chaining es mucho más estable y predecible en consumo de datos, comparándolo con linear probing que sí es sensible y dependiente del factor de carga.