OBSERVACIONES DE LA PRÁCTICA

Juliana Rodríguez Morales 202421552
María Clara Quijano 202420069
Juan Andrés Lozada Barragán 202510410

	Máquina 1
Procesador	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, 2803 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria RAM (GB)	32.0 GB
Sistema Operativo	Sistema operativo de 64 bits, PC basado en x64

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

Resultados

No pudimos hacer los resultados porque al hacer main() nos daba un loop infinito y no cargaba los datos del catálogo. Todas las respuestas del análisis se basan en la teoría del linear probing y separate chaining vistos en clase.

Carga de Catálogo LINEAR PROBING

Factor de Carga (PROBING)	**Consumo de Datos [kB]**	**Tiempo de Ejecución Real @LP [ms]**
0.1		
0.5		
0.7		
0.9		

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		
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 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**

Aunque no tenemos gráfica, por teoría dado que el Linear Probing mantiene espacios libres es más costoso en memoria que Separate Chaining, que solo guarda las llaves-valor sin espacios entre ellos. De esta forma, Linear Probing ocupa más memoria al tener que mantener por lo menos la cantidad de espacios vacíos en el array que permita el limit factor.

Por otra parte, en tiempo de ejecución Linear Probing y Separate Chaining tienen todas sus funciones con complejidad O(n) como máximo. Linear Probing oscila entre O(1) y O(n) mientras que Separate Chaining

entre O(1) y O(n/N). Sin los datos a analizar no podemos dar una respuesta precisa de cuál es más rápido, sin embargo, por complejidades inferimos que Linear Probing es más rápido, aunque sea por milisegundos.

- **Comparación de factor de carga y memoria para LINEAR PROBING y SEPARATE CHAINING**

Entre mayor el factor de carga, se desperdicia menos memoria en Linear Probing y Separate Chianing pues se permite almacenar más parejas llave-valor antes de tener que hacer un rehash. Lo anterior se debe al factor de carga, que es $\alpha = n/N$, entre mayor sea n mayor será el current factor, y si el factor de carga máximo es grande entonces el aprovechamiento del espacio sería mejor que con un factor de carga máximo más pequeño. Sin embargo, esto aumenta las operaciones que se hacen en el manejo de colisiones.

Preguntas de análisis

1. ¿Por qué en la función **getTime()** se utiliza **time.perf_counter()** en vez de otras funciones como **time.process_time()**?
2. ¿Por qué son importantes las funciones **start()** y **stop()** de la librería **tracemalloc**?
3. ¿Por qué no se puede medir paralelamente el **uso de memoria** y el **tiempo de ejecución** de las operaciones?
4. Dado el número de elementos de los archivos (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?
5. ¿Qué cambios percibe en el tiempo de ejecución al modificar el factor de carga máximo?
6. ¿Qué cambios percibe en el consumo de memoria al modificar el factor de carga máximo?
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.
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.

Respuestas:

1. En la función getTime() se usa time.perf_counter() porque mide el tiempo real transcurrido desde que inicia hasta que termina una operación, incluyendo pausas del sistema o tareas externas. Es ideal para medir el rendimiento total del programa. En cambio, time.process_time() solo mide el tiempo de CPU usado, sin contar esperas ni otros procesos, por lo que no refleja el tiempo real de ejecución.
2. Las funciones start() y stop() de tracemalloc son importantes porque permiten controlar cuándo inicia y termina la medición del uso de memoria. Con start() se comienza a registrar las asignaciones de memoria, y con stop() se detiene el seguimiento para no sobrecargar el sistema. Así se pueden medir solo las partes del código que interesan.
3. No se puede medir al mismo tiempo el uso de memoria y el tiempo de ejecución porque una medición afecta a la otra. Al usar tracemalloc, el sistema gasta más recursos, lo que altera el tiempo real. Del mismo modo, medir tiempo mientras se rastrea memoria genera datos poco precisos. Por eso se recomienda hacerlo por separado.
4. Como el archivo large tiene 10000 elementos entonces N inicial debería ser 10000, pues la tabla debería ser capaz de mantener esa cantidad de elementos. Como en las tablas se guardan diferentes categorías como autor-lista(libros), autor-año(lista(libros)), tag name-tag, etc; entonces n depende de la cantidad de parejas llave-valor que se puedan formar con los datos. En conclusión, el factor de carga es n/N donde n puede ser igual o menor a N. Por otra parte, el factor límite varía según LP o

SC, si es Linear Probing este debe ser menor a 1 ($\alpha<1$) mientras que en Separate Chaining puede ser mayor a 1.

5. Si se tiene un mayor factor de carga máximo, entonces se requiere hacer menos rehash, por lo que requiere menos tiempo para cargar los elementos. Adicionalmente, también significa que hay más colisiones por lo que hay menos espacios vacios.
6. Si aumenta el factor de carga máximo entonces en Linear Probing la lista tendría la capacidad de albergar más parejas llave-valor sin necesidad de hacer rehash tan seguido, lo mismo sucedería con Separate Chaining pues aumenta el promedio de parejas llave-valor que puede contener cada "caja" del array list. Sin embargo, la memoria utilizada en Linear Probing mejoraría ya que eso significa que necesita mantener menos espacios vacíos.
7.
8. Como se ha dicho anteriormente, Linear Probing debe asegurar espacios vacíos para poder manejar las colisiones, por lo que dependiendo del esquema de manejo de colisiones utilizado se logra ver una diferencia entre la memoria que se utiliza. Siempre Linear Probing será más costoso en memoria que Separate Chaining (si estos tienen los mismos n y N). Lo anterior debido a que la forma en que Separate Chaining maneja las colisiones es poniendo aquellos con mismo valor hash en un mismo "bucket" (que es la posición del array_list que guarda un single_linked); mientras que Linear Probing ocupa un espacio y si hay colisión entonces busca el primer espacio vacío (None o Empty) y esto solo se logra siempre que se asegure la existencia de espacios vacíos (esto lo hace el rehash y la comparación del limit factor y current factor). En conclusión, LP es más costoso en memoria porque debe su capacity debe ser mayor a la cantidad de parejas existentes, mientras que SC solo guarda lo necesario.