OBSERVACIONES DE LA PRÁCTICA

Ileanne Valderrama Ocampo 202514355 – Est 1
Juan Pablo Peñaranda Cely 202422285-

Ambientes de pruebas

ESTUDIANTE 1

	Computador 1 – Estudiante 1
Procesador	Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz (2.59 GHz)
Memoria RAM (GB)	
Sistema Operativo	Sistema operativo de 64 bits, procesador x64

Tabla . Especificaciones del computador para ejecutar las pruebas de rendimiento.

Para realizar las siguientes pruebas, es importante que las ejecuten utilizando el archivo de datos **large**, ya que es con este conjunto donde realmente se puede observar el comportamiento y la eficiencia de los algoritmos en contextos más exigentes.

Carga de Catálogo (PROBING)

Factor de Carga (PROBING)	Consumo de Datos (kB)	Tiempo de Ejecución Real @LP [ms]
0.1	46,875 kB	0.2 ms
0.5	5,208 kB	0.6 ms
0.7	4,1783 kB	0.8 ms
0.9	10,293 kB	1.7 ms

Carga de Catálogo CHAINING

Factor de Carga (CHAINING)	Consumo de Datos (kB)	Tiempo de Ejecución Real @LP [ms]
2.00	5,078 kB	0.4 ms
4.00	4,988 kB	0.6 ms
6.00	4,8879 kB	1.2 ms
8.00	4,785 kB	1.3 ms

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

El factor de carga óptimo para probing es de 0.5 o de 0.7, mientras que el de chaining son valores como 2 o 4,es importante aclarar, que esto funciona dependiendo del tipo de archivo y su capacidad tota.

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

A nivel de tiempo, probing tiene más tiempo de ejecución porque se generan más colisiones, mientras que chaining aumenta de una manera más gradual pero por el recorrido de los datos, probing tiene un tiempo enorme al momento de ejecución cuando se trata de ingresar lo datos, porque necesita hacer rehash.

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

Que probing es mucho más ineficiente para cargar datos, porque no llega al factor de carga máximo, porque si se llega a la mitad, hace rehash, mientras que chaining, no. Por esto, probing reserva más espacio, mientras que chaining, lo hace con el mismo espacio siempre.

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

Probing tiene tiempos de ejecución mucho más rápidos a comparación de chaining, por el tipo de búsqueda, solo que cuando se sobre carga, se cae por la necesidad de hacer rehash al pasar del límite de ubicaciones (50%) que tiene, mientras que el chaining, es mucho más estable porque no tiene la necesidad de hacer rehash al pasar el 50%.

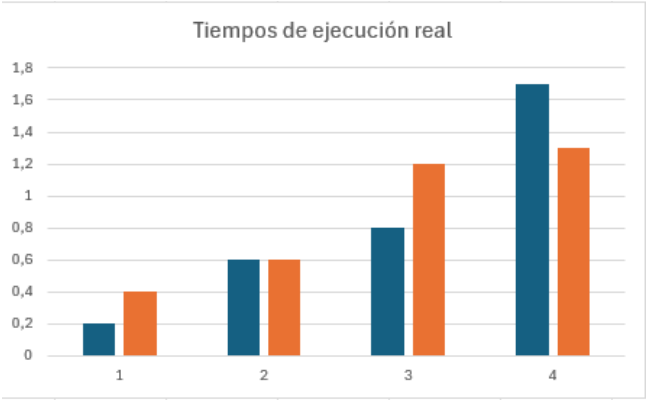
e. ¿Qué cambios percibe en el consumo de memoria al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.

   Según lo evaluado, linear probing tiene un uso más eficiente de la memoria, pues solo requiere un arreglo con posiciones fijas, no necesita nada adicional, mientras que el chaining utiliza listas enlazadas para almacenar la informacion, lo que cuesta más en memoria.

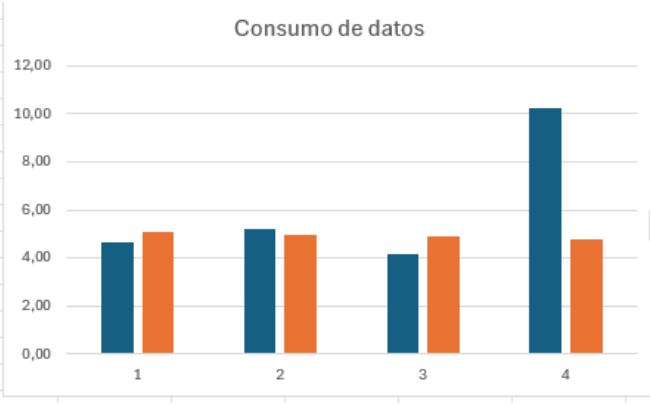
Gráficas

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()**?

La función pref_counter time se utiliza porque da la medida precisqa del tiempo que dura en realizarse, además de que tiene en cuenta cuando son demoras por cula de cosas externas, mientras que process_time time solo mide el tiempo general que toma todo, ignorando lo demás.

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

Porque start es la que inicia el conteo del tiempo de las funciones del programa, mientras que stop lo detiene.

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

Por qué se podria colapsar, se ejecutaría más lento, lo que no daría una buena respuesta, o una concreta ante lo que se evalua.

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

En probing, 0.5, 0.7 o 0.9, mientras que en chaining, 4, 6 u 8.

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

El tiempo depende de la inserción y el recorrido de las listas, por lo que en probing es un efecto mucho más fuerte porque necesita recorrer más posiciones hasta encontrar un espacio y que además no supere el 50%, el cual es necesario para que no se haga un rehash, mientras que en chaining, tiene una menor complejidad o cambio, porque a pesar de que se tiene que mover de nodo a nodo, no tiene que volver a hacer rehash hasta llegar a cierto punto.

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

El consumo de memoria en probing cuando se reduce el factor de carga, aumenta, mientras que en chaining, se mantiene constante y no afecta mucho, solo mejora el tiempo.

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.

Cuando se trata de probing y los esquemas de colisiones, el tiempo es mucho menor, pero mientras la tabla se llena, se demora muchiiiiisimo más, mientras que en chaining se mantiene el tiempo y es mucho mejor que con factores de carga altos.

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.

Probing utiliza un arreglo de tamaño fijo en el que cada posición es llenada por una entrada, el consumo de memoria puede llegar a ser menor, además de que es más predecible, mientras que el probing es mucho más flexible, pero necesita mucho más espacio.

ESTUDIANTE 2

	Computador 2 – Estudiante 2
Procesador	13th Gen Intel(R) Core(TM) i5-13450HX (2.40 GHz)
Memoria RAM (GB)	16 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador x64

Tabla 1. Especificaciones del computador para ejecutar las pruebas de rendimiento.

Para realizar las siguientes pruebas, es importante que las ejecuten utilizando el archivo de datos **large**, ya que es con este conjunto donde realmente se puede observar el comportamiento y la eficiencia de los algoritmos en contextos más exigentes.

Carga de Catálogo (PROBING)

Factor de Carga (PROBING)	**Consumo de Datos (kB)**	**Tiempo de Ejecución Real @LP [ms]**
0.1	46,87500 kB	0.1
0.5	5,92833 kB	0.6
0.7	4.188453 kB	0.8

0.9	9,99343 kB	1.6

Carga de Catálogo CHAINING

Factor de Carga (CHAINING)	Consumo de Datos (kB)	Tiempo de Ejecución Real @LP [ms]
2.00	5,14516 kB	0.4
4.00	47624 kB	0.6
6.00	47892.73 kB	1.2
8.00	4,67845 kB	1.3

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

Para el mecanismo de linear probing, el factor de carga óptimo se encuentra generalmente entre 0.5 y 0.7, ya que valores superiores aumentan notablemente las colisiones y provocan rehash más frecuentes, afectando el rendimiento.
En cambio, en separate chaining es posible manejar factores de carga más altos, entre 2.0 y 4.0, porque las colisiones se gestionan mediante listas enlazadas y no requieren reorganizar la tabla. En todo caso, el valor ideal depende del tamaño total del archivo y de la cantidad de registros cargados, ya que un mayor volumen de datos puede exigir ajustar estos límites para equilibrar rendimiento y uso de memoria.

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

Al aumentar el factor de carga, el tiempo de ejecución tiende a crecer en ambos mecanismos, pero de manera distinta.
En linear probing, el tiempo aumenta de forma considerable cuando se superan ciertos umbrales de ocupación, debido al incremento de colisiones y al costo del *rehash*. En cambio, en separate chaining el tiempo crece de manera más progresiva, ya que las colisiones se resuelven con listas y no requieren reestructurar toda la tabla. Sin embargo, a factores de carga muy altos, el recorrido de las listas enlazadas también genera demoras perceptibles.

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

En linear probing, el consumo de memoria se incrementa más rápidamente porque requiere crear nuevas tablas durante el *rehash* cuando se alcanza el factor de carga máximo. Además, mantiene espacios vacíos que deben reservarse para futuras inserciones. Por otro lado, separate chaining usa la memoria de manera más flexible: mantiene la misma tabla base, pero distribuye las colisiones en listas enlazadas. Esto hace que su consumo sea más estable, aunque un poco mayor por la sobrecarga de punteros y estructuras adicionales en cada lista.

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

El esquema de linear probing suele ser más rápido en búsquedas e inserciones cuando la tabla no está muy llena, ya que aprovecha la localización espacial del arreglo y evita saltos de memoria. Sin embargo, cuando el factor de carga es alto, las colisiones sucesivas degradan el rendimiento drásticamente y el proceso de *rehash* aumenta los tiempos. En contraste, separate chaining mantiene un tiempo de ejecución más constante, ya que las colisiones no requieren desplazamiento dentro del arreglo principal, sino que se manejan en listas independientes, haciendo que su rendimiento sea más estable bajo alta carga.

e. ¿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 esquema de linear probing es más eficiente en términos de uso de memoria, porque almacena todos los elementos dentro de un único arreglo y no necesita estructuras adicionales. En cambio, separate chaining requiere más memoria debido a la creación de múltiples listas enlazadas para manejar las colisiones, lo que implica un gasto adicional en punteros y objetos. Aun así, este consumo extra de memoria le permite mantener una mejor estabilidad y evitar los costos de *rehash* que tiene linear probing.

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

¿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` porque ofrece la mayor precisión disponible para medir intervalos de tiempo en Python, incluyendo todo el tiempo transcurrido desde el inicio hasta el final de una operación, sin importar si el proceso estuvo inactivo o esperando recursos del sistema. A diferencia de `time.process_time`, que solo mide el tiempo de CPU, `perf_counter` mide el tiempo real de ejecución (wall-clock time), lo que permite capturar el rendimiento total de la operación, incluyendo demoras por lectura de archivos o manejo de estructuras de datos, tal como se requiere en el laboratorio.

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

Las funciones `start` y `stop` (usadas para iniciar y detener las mediciones) son fundamentales porque permiten registrar una muestra precisa del uso de memoria antes y después de ejecutar una operación. Esto permite determinar cuánta memoria se consumió realmente durante la ejecución de una función, sin incluir el uso previo del programa. En otras palabras, aseguran que las mediciones sean coherentes y que solo se considere la memoria utilizada por la función evaluada.

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

No se pueden medir al mismo tiempo porque las mediciones de memoria y de tiempo interfieren entre sí. El uso de `tracemalloc` introduce una sobrecarga adicional que altera el tiempo de ejecución real, y viceversa. Si se intentan medir en paralelo, las operaciones de seguimiento de memoria afectan la velocidad del código. Por eso, en el laboratorio se recomienda medir primero el tiempo y luego la memoria de forma independiente, para obtener resultados más precisos.

Dado el número de elementos de los archivos (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?
Para el mecanismo de linear probing, el factor de carga óptimo se encuentra entre 0.5 y 0.7, ya que un valor superior incrementa la probabilidad de colisiones y obliga a realizar rehash más frecuentes. En cambio, el mecanismo de separate chaining puede operar correctamente con factores de carga más altos, entre 2.0 y 4.0, debido a que las colisiones se resuelven mediante listas enlazadas, sin necesidad de redimensionar la tabla. El valor ideal depende del tamaño del archivo y la cantidad total de registros cargados.

¿Qué cambios percibe en el tiempo de ejecución al modificar el factor de carga máximo?
A medida que aumenta el factor de carga, el tiempo de ejecución también crece, aunque de manera distinta en cada mecanismo. En linear probing, el tiempo aumenta de forma más brusca debido a la mayor cantidad de colisiones y a los procesos de rehash. En cambio, en separate chaining el

incremento del tiempo es más gradual, ya que las colisiones se manejan en listas enlazadas sin necesidad de reorganizar toda la tabla.

¿Qué cambios percibe en el consumo de memoria al modificar el factor de carga máximo?
En linear probing, el consumo de memoria aumenta porque se requiere crear nuevas tablas al realizar rehash, reservando más espacio del necesario. En separate chaining, el consumo de memoria se mantiene más estable, ya que solo se añaden nodos en las listas de colisiones sin duplicar la tabla. Sin embargo, las estructuras adicionales (punteros y listas) hacen que chaining use más memoria total.

¿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, las operaciones de búsqueda e inserción son más rápidas mientras la tabla no esté muy llena, gracias a la localización continua en memoria. Sin embargo, cuando el factor de carga aumenta, las colisiones sucesivas y los rehash degradan notablemente el rendimiento. En separate chaining, los tiempos se mantienen más constantes, ya que las colisiones no implican desplazamientos dentro del arreglo principal, sino operaciones dentro de listas. Esto hace que chaining sea más estable bajo alta carga.

¿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 esquema de linear probing es más eficiente en memoria porque todos los elementos se almacenan en un solo arreglo. Sin embargo, para evitar colisiones debe mantener más posiciones vacías. En separate chaining, el uso de memoria es mayor porque necesita estructuras adicionales (listas enlazadas) para almacenar las colisiones. Aun así, este esquema permite manejar mejor grandes volúmenes de datos sin realizar rehash, ofreciendo una mayor estabilidad.