

Instituto Tecnológico y de Estudios Superiores de Monterrey



Tecnológico de Monterrey

Programación de Estructuras de Datos y Algoritmos Fundamentales

Ivan Reyez Amezcua

Act 2.3 - Actividad Integral Estructura De Datos Lineales (Evidencia Competencia)

| | |
|--------------------------------------|--------------|
| Santos Alejandro Arellano Olarte | // A01643742 |
| Carlos Iván Armenta Naranjo | // A01643070 |
| Arturo Ramos Martínez | //A01643269 |
| Adair Virgilio Figueroa Medina | //A00572826 |
| Leonardo Mario Alberto Guillén Soria | // A00574110 |
| Daniela Rocha Muñoz | //A00573664 |

2 de Octubre del 2023

En esta actividad, se emplea una base de datos que es organizada mediante vectores con el propósito de lograr una representación más eficiente y permitir modificaciones ágiles. La información se almacena en una lista doblemente ligada, lo que facilita el acceso y la manipulación de los datos.

El enfoque principal consiste en permitir a los usuarios interactuar con la base de datos organizada. Para ello, se solicita un rango inicial y final, y el programa devuelve los datos que se encuentran dentro de esos límites.

La elección de utilizar listas doblemente ligadas agiliza significativamente este proceso, incluso cuando se manejan bases de datos extensas, garantizando un rendimiento óptimo sin un alto consumo de memoria.

Funciones empleadas:

| ordenaMerge() | Descripción | |
|---------------|-------------|---|
| | | Este método de forma recursiva, usa una función que a grandes rasgos va dividiendo el arreglo hasta llegar al caso base, el cual es cuando el arreglo consta de un solo elemento, en ese momento esos valores, van ascendiendo hacia las funciones recursivas previas para ser comparados entre sí e ir ordenando cada pedazo de arreglo. En resumen, cada parte que fue separada, va siendo ordenada después de haber llegado al caso base, esto hasta llegar a la primera llamada de la función, la cual ordena las dos primeras mitades del arreglo. |

| | | |
|------------------------|---------------|--|
| | | Es una operación comúnmente utilizada en algoritmos de ordenamiento y en la manipulación de contenedores como vectores, listas y colas de prioridad (heap). |
| | Entrada | Un vector con los datos enteros, un índice inicial (low) y uno final(high) correspondientes al primer y último índice del vector. |
| | Salida | La salida es el mismo vector, pero ya ordenado |
| | Precondición | El vector debe contener los datos a ordenar y estos deben ser de tipo entero |
| | Postcondición | El vector al final tendrá los datos ya ordenados |
| | Complejidad | La complejidad de esta función se representa como una función logarítmica de base 2 al ir dividiendo el arreglo en mitades e ir comparando las magnitudes: $T(n)=\log_2(n)$ |
| leerAlmacenarOrdenar() | Descripción | Esta es una función que dentro de ella lee las líneas de texto de un archivo llamado bitacora.txt, para después almacenarlas en un |

| | | |
|--|--|--|
| | | <p>vector de manera desordenada y posteriormente ordenarlas convirtiendo su ip a un número entero con ayuda de la función <code>stringToLongIP()</code>, para después ordenar los datos del vector y guardarlos en una lista doblemente ligada, posteriormente con estos datos ya ordenados y los datos de inicio y de final que se tienen para la búsqueda se realice esta misma con ayuda de la función <code>BinarySearch()</code>, la cual nos dirá si encuentra la IP de inicio y de final proporcionadas en la entrada, para sí si se encuentran mostrar desde ahí en adelante mostrarlo, si no se encuentran se mostrarán los datos de la ip más cercana a la proporcionada hasta la más cercana a la ip final proporcionada, ya que no se encuentran esas ip realmente en la lista. Después de esto se llamara a la función <code>storeInTxt()</code> que sirve para guardar los resultados de estas búsquedas en un nuevo archivo .txt que se llama "rango.txt"</p> |
|--|--|--|

| | | |
|--------------|---------------|---|
| | Entrada | La IP de inicio y la IP final del rango en el que desea buscar. |
| | Salida | Esta función de salida imprime en pantalla la lista de datos ordenados por IP |
| | Precondición | El archivo que va a leer y ordenar se debe de llamar bitácora.txt y debe de estar en la misma carpeta además debe de contar con el formato establecido en las instrucciones. |
| | Postcondición | La lista doblemente ligada debe tener los datos ordenados por IP y el rango de búsqueda debe ser almacenado correctamente en el archivo txt resultante. |
| | Complejidad | Al ordenar los datos por IP con el MergeSort que tiene complejidad $n\log(n)$ y hacer la búsqueda en el mejor de los casos por un binary search tendría una complejidad de $n\log(n)$, pero en el peor de los casos en el que la búsqueda la realiza con un rango aproximado y hacerlo con un while se tendría una complejidad lineal $T(n)$ |
| binarySearch | Descripción | Esta función, como su nombre lo indica, nos |

| | | |
|--|---------------|---|
| | | <p>ayuda a encontrar el nodo donde se encuentra la ip que se le pasa como parámetro. Antes de empezar la búsqueda, verifica si la llave está más cerca de la cabeza o de la cola, para así, comenzar a recorrer desde ese nodo; esto permite que se le dé uso a una de las ventajas de usar la lista doblemente ligada, la cual es la de avanzar de manera bidireccional.</p> |
| | Entrada | <p>Recibe: Apuntadores de tipo nodo a la cabeza y cola y la key de tipo long long int.</p> |
| | Salida | <p>Retorna un puntero de tipo nodo que contiene el valor o la llave buscada. Por otro lado, si no se encuentra esta, retorna un nullptr.</p> |
| | Precondición | <p>Que los punteros no sean nullptr.</p> |
| | Postcondición | <p>Un puntero de tipo nodo que contiene el valor buscado, o en la otra opción, un puntero a nullptr.</p> |
| | Complejidad | <p>Como se le está aplicando a una lista doblemente ligada, la complejidad no sería $\log n$, como si lo es en un simple array que nos</p> |

| | | |
|-----------------------|---------------|---|
| | | <p>permite el acceso directo. En la lista doblemente ligada, a pesar de saber dónde está la mitad del en cada iteración, se necesita pasar por cada nodo hasta llegar al nodo que se encuentra en la mitad. Es por eso que la complejidad sería de $O(n)$ en el peor de los casos.</p> |
| <i>StringToLongIP</i> | Descripción | Esta función convierte una dirección IP en formato de cadena a un número entero largo (long long). |
| | Entrada | const string& full_ip: La dirección IP en formato de cadena |
| | Salida | long long: El número entero largo (long long) que representa la dirección IP. |
| | Precondición | El formato de la dirección IP debe ser válido (por ejemplo, "###.##.###.###:####") |
| | Postcondición | La función devuelve un número entero largo que representa la dirección IP. |
| | Complejidad | La función no contiene bucles ni procesos complejos, por lo que tiene una complejidad $O(1)$. La conversión de |

| | | |
|-------------------|---------------|---|
| | | cadena a entero es una operación con complejidad $O(1)$ |
| <i>storeInTxt</i> | Descripción | Esta función guarda los resultados de la búsqueda u¿ en un archivo .txt |
| | Entrada | La dirección de memoria de la línea que va a guardar, y el nombre del archivo |
| | Salida | Un archivo .txt con los datos guardados |
| | Precondición | Nada |
| | Postcondición | Nada |
| | Complejidad | Lineal $T(n)$ |

Reflexiones:

Daniela:

En el mundo de la seguridad informática, la elección de la estructura adecuada para organizar la información es crucial. Las listas doblemente ligadas, lo que tiene de especial es que te permiten hacer cambios en cualquier punto sin tener que reorganizar todo. Por ejemplo, si hay un nuevo evento importante, lo puedes añadir sin que eso afecte a los demás, lo que ahorra mucho tiempo. Esto es especialmente útil cuando la llegada de registros es impredecible. En el mundo de la seguridad, donde los eventos ocurren en cualquier momento y la información llega a diferentes velocidades, esta flexibilidad es clave. Las listas doblemente ligadas permiten adaptarte rápidamente a estos cambios, asegurando que no se escape nada importante. Además, son muy buenas para lidiar con diferentes tipos de información, ya que pueden manejarla de manera eficaz sin importar su complejidad. Esto garantiza que todos los datos, sin importar qué tan complicados sean, se guarden, organicen y encuentren fácilmente.

Desde mi punto de vista, el impacto positivo de las listas doblemente ligadas en la gestión de bitácoras de seguridad es muy claro. No sólo simplifican el proceso, sino

que también ofrecen una ventaja invaluable en un entorno donde la precisión y la eficacia son fundamentales. En comparación con otras estructuras de datos, las listas doblemente ligadas destacan por su combinación de flexibilidad, eficiencia y acceso en ambas direcciones. Esto las convierte en la elección óptima en la gestión de bitácoras de seguridad.

Leonardo:

Las listas doblemente ligadas son importantes y nos sirven para realizar cosas de manera más eficiente, tales como buscar en una base de datos grande y en casos como cuando se necesita filtrar en un rango ya sea predefinido o definido por un usuario.

Esto es gracias a su capacidad de inserción, eliminación eficiente, búsqueda rápida y flexibilidad. Algo malo es que dependiendo de cómo sean aplicadas estas podrían consumir más memoria y requerir un proceso de ordenamiento inicial. Por ello esto deberá de ser aplicado solo a bases de datos adecuadas, para que funcione de una manera ágil y correcta.

Sin duda, trabajar a más bajo nivel como lo es con memoria tiene muchísimas ventajas, pero también tiene que ser aplicado de la mejor manera posible para que sea eficiente y no se termine consumiendo más memoria que de la manera tradicional como lo son los vectores o arrays.

Carlos:

Después de haber realizado esta actividad puedo entender un poco mejor el funcionamiento de las listas doblemente ligadas, además de que al ir realizando la actividad nos dimos cuenta de que la si bien el acceso a elementos de la lista es más fácil en una doblemente ligada que en una lista ligada normal, no son la mejor opción a implementar para actividades como esta ya que por su estructura algoritmos como el binary search o el merge sort no están optimizados para estos casos y su implementación se complica bastante a diferencia de si se usan vectores, por lo que el usar listas doblemente ligadas no es una gran mejora en complejidades respecto a otras opciones.

Por lo que el uso de listas ligadas es un recurso que puede ser utilizado para el manejo de muchos datos, ya que su flexibilidad y la posibilidad de acceder de manera bidireccional hace que sea una muy buena elección a la hora de la gestión de datos, por lo que sí es bien implementada puede contribuir a la optimización del programa y mejora de complejidades de distintos procesos.

Adair:

En esta actividad pudimos aprender de la importancia de las listas doblemente ligadas debido a su flexibilidad y la eficiencia que tienen en la manipulación dinámica de datos. Nos permiten hacer intersecciones y eliminaciones eficientemente, aprovechando de una manera eficaz la memoria. Su capacidad para representar las relaciones entre elementos hacen que estas sean esenciales en el desarrollo de software

Santos Arellano:

Trabajar con listas doblemente ligadas en esta actividad me mostró lo valiosas que son en el ámbito de la seguridad informática. Su flexibilidad y eficiencia en la gestión de registros de eventos son impresionantes. Poder añadir o eliminar datos rápidamente sin afectar el rendimiento es esencial en un entorno donde la información fluye constantemente. Las listas doblemente ligadas se han convertido en una herramienta clave para mantener la integridad de las bitácoras de seguridad y garantizar que nada importante se escape.

Arturo Ramos Martínez:

Esta actividad me ayudó a entender mejor el funcionamiento de una lista doblemente ligada, pero también comprendí la dificultad que implica usarla, ya que por lo menos en este caso, era mucho más sencillo trabajar con arreglos o vectores. Y es que nos dimos cuenta de la complejidad que implica aplicar a una lista doblemente ligada algoritmos como Binary Search o Mergesort, realmente no están tan optimizados como para los arreglos, o son más difíciles de implementar, debido a la naturaleza de las estructuras de datos (acceso directo en los arrays contra acceso lineal en las listas ligadas). Dado lo anterior, puedo decir que usar una lista ligada, no es la mejor opción, ya que le agrega complejidad de desarrollo y a la hora de buscar u ordenar, no se agrega una ventaja significativa o incluso presenta complejidades similares a la misma implementación con un array.