**Universidad Nacional Autónoma de Honduras**

**Centro Universitario Regional del Centro**



**Ensayo del capítulo 16**

**Algoritmos y estructuras de datos**

**Ingeniero**

Elmer Padilla

**Presentado por:**

Jostin Javier Cruz Castillo

(20221900145)

**Comayagua, Comayagua 7 diciembre del 2023**

**ENSAYO**

En este capítulo, encontramos un resumen general de todo lo hablado en el libro. En la mayoría de los casos en programación, nos contratan en un trabajo para hacer una tarea específica, que puede ser más difícil o más fácil según el caso. ¿Cómo deberíamos analizar estos problemas? Podemos empezar examinando cuáles son sus objetivos o qué procesos requiere para determinar qué tipos de datos debemos utilizar. Ahora que tenemos los datos, ¿qué deberíamos hacer con ellos? Mirarlos, analizarlos. En la mayoría de los casos, hay una gran cantidad de datos que nos puede causar una carga considerable. Debemos definir qué tipos de datos son, o cómo va a ser su entrada, o cómo se accede a ellos. Los datos se agrupan en fragmentos. Así mismo, podemos decir que cada dato es importante, pero no nos debe interesar. La verdadera pregunta es cuántos de estos datos hay. No se ocupa un número exacto, pero sí cuántos decimales. Esto dependerá de si sabemos si los datos son fijos o no. Por ejemplo, tenemos una empresa. Sabemos que en esta empresa hay un determinado tipo de datos fijos. Estos datos no van a aumentar ni disminuir, solo actualizarse. Podemos tener un sistema que no va a procesar más datos, pero, en cambio, si tenemos esta misma empresa que aumenta la cantidad de sus datos, debemos crear programas que tengan una escalabilidad dependiendo del uso que pensamos que vamos a tener, que puede ser de 1000 a millones.

Otra cosa que deberíamos ver es su escalabilidad antes mencionada, ya que no es lo mismo que tenga una escalabilidad x^2, x^3. Si es automatizado o manual, ya que la automatización siempre tiene más escalabilidad que lo manual. En el desarrollo de sistemas informáticos, la eficiencia es la clave para garantizar un rendimiento óptimo. Un elemento fundamental en este proceso es el diseño de operaciones, la frecuencia con la que se lleva a cabo y la elección adecuada de estructuras y algoritmos para optimizar su ejecución. Un buen diseño de sistemas comienza por identificar y enumerar todas las operaciones que se llevarán a cabo, prestando especial atención a la frecuencia con las que estas operaciones se ejecutarán. Tomemos como ejemplo un sistema de contabilidad, donde las nuevas transacciones son algo cotidiano. Cada día llegan informes y pasan a ser semanales, trimestrales y anuales.

Este se vuelve eficiente para garantizar un rendimiento óptimo en el sistema. Para esto, ocupamos un ordenamiento de datos para todas las estructuras de datos.

Hay algoritmos como Dijkstra que encuentra el camino más corto en un grafo y requiere un orden específico para visitar los vértices. Estas restricciones nos pueden ayudar a influir en la elección de las estructuras de datos.

En otras cosas, también debemos ver si somos buenos programadores o no, porque hay algoritmos demasiado complicados. Si se realiza un cambio, ya todo tiene errores fatales. Aunque, en otros casos, si estos programas van dirigidos a empresas con gente capacitada, no debería ser ningún problema y no elegir un algoritmo tan avanzado que el sistema no lo pueda correr. En estos casos, los algoritmos O(n^2) son más fáciles de leer que los O(n × log(n)) y más recomendables en algunos casos. Pero, en cambio, si tenemos una gran cantidad de datos, el O(n × log(n)) es una solución para estos.

Los arrays son una estructura muy básica. Estos nos permiten acceso a través de índices. Se les puede asignar un tamaño. También tenemos las listas enlazadas y los árboles. Estos vinculan fragmentos de datos. Las vinculaciones sirven para ordenar los datos. En cambio, tenemos las tablas hash que son muy parecidas a los arrays, pero usan claves para su indexación. Hay un punto a destacar y es sobre la asignación de la memoria. Cada tipo de estructura la asigna de manera diferente, mientras que los arrays lo asignan en bloques continuos. En cambio, a las listas enlazadas y los árboles su asignación es mediante referencias dinámicas. Esto hace que asignen memoria según la vayan ocupando. Igual con las tablas hash, aunque tengan su indexación, también su espacio en memoria es de manera dinámica. Mediante esto, podemos asignar cada estructura de datos mediante los siguientes parámetros: velocidad de acceso, asignación de memoria y las operaciones requeridas.

Siempre está el tema de qué será mejor, la velocidad vs la simplicidad. La velocidad de los arrays y las listas enlazadas es bastante lenta por su especificación en claves. En el otro lado, tenemos a los árboles, que son rápidos, pero las tablas hash son más rápidas. Pero, como ya sabemos, también dependemos de la memoria y de la complejidad. Actualmente, tenemos computadoras que tienen más RAM y mejores procesadores. Esto hace que busquemos estructuras que sean más eficientes.

Por otro lado, tenemos a las librerías. Estas tienen algoritmos y estructuras de datos que nos simplifican el trabajo y, a la vez, reducen el código.

Tenemos la primera estructura de datos, que son los arrays. Estos destacan por su simplicidad. En ellos, podemos manipular y almacenar los datos. Esta estructura es muy buena cuando los datos que se desean almacenar son muy pequeños y considerables. Estos están diseñados para ser eficientes cuando sus elementos son por índices. La velocidad de los arrays es el punto más fuerte que tienen, pero tienen limitaciones cuando los datos se deben insertar o eliminar.

Las listas enlazadas son una elección natural cuando los datos no se pueden predecir fácilmente y minimizar el uso de memoria es crucial. A diferencia de su hermano, el array, estas se pueden expandir dinámicamente a medida que se agregan elementos. Estas solo ocupan el espacio necesario. Es necesario comprender que no es una solución óptima para todos los escenarios. La eliminación y búsqueda por claves son más lentas en comparación con los arrays.

Los árboles de búsqueda binaria ofrecen una solución eficiente para situaciones específicas donde la abordabilidad y la ordenación de datos son fundamentales. La capacidad de ordenar datos de manera eficiente los convierte en una clave algo valioso.

Vamos con las tablas hash, que son la opción de rendimiento más estable. Su capacidad para abordar cualquier tipo de datos que pueda considerar un índice, incluyendo enteros, las hace versátiles. Tienen un rendimiento de O(1) en búsqueda e inserción, el único problema es que ocupan más memoria que el mínimo asignado.

Las pilas nos pueden ayudar mucho y es recomendable usarlas si lo que quieres es acceder al último dato insertado. Estas siguen el principio de último entrar, primero en salir. Por otro lado, tenemos las colas, que son al revés, es el primero entrar, es el primero en salir. Estas se usan para acceder rápidamente a los primeros elementos insertados, que priorizan al elemento más antiguo.

Estas dos estructuras de datos se pueden implementar mediante un array o lista enlazada, que ofrecen eficiencias en inserciones o eliminaciones.

Las colas de prioridad nos dan acceso al elemento seleccionado con mayores prioridades y se presentan como la elección perfecta. Este elemento se determina por la clave más grande, siendo el foco central.

Los grafos son una herramienta hecha para reflejar y modelar cosas del mundo real. La estructura de los grafos refleja la complejidad de los problemas, como en redes, transporte y comunicación. Las operaciones especiales que los grafos manejan incluyen la búsqueda de rutas, árboles y subgrafos. Las búsquedas en profundidad y en amplitud para un vértice o tipo de vértice específico toman O(v^2) en tiempo para representación con matriz de adyacencia, mientras que en la representación con listas de adyacencia toman O(v+e), donde e es el número de aristas.

En conclusión, hay que elegir bien la estructura de datos que vamos a utilizar, ya que la más adecuada es esencial para el rendimiento eficiente y la resolución efectiva de los problemas de programación y ciencia de la computación. Cada estructura tiene su fortaleza o debilidad, y la clave radica en las características del problema en cuestión.