UNIVERSIDAD DEL VALLE DE ORIZABA

UniVO®

Universidad del Valle de Orizaba

Reporte de practica

Trabajo de investigación estructura de Datos

Miguel Angel Tress Ruiz

Docente. Flor Denisse Arenas Cortes

Licenciatura en Ingeniería en Sistemas computacionales

Asignatura. Estructura de Datos

UNIVERSIDAD DEL VALLE DE ORIZABA

Índice

UNIVERSIDAD DEL VALLE DE ORIZABA

Introducción

En el desarrollo de software, elegir la estructura de datos adecuada y diseñar algoritmos eficientes es fundamental para resolver problemas de manera óptima. Este trabajo explora tres conceptos clave:

1. Estructuras de datos: Formas de organizar información (arreglos, listas, árboles, grafos, etc.).

2. Tipos abstractos de datos (TAD): Modelos teóricos que definen operaciones sin especificar su implementación.

3. Complejidad algorítmica (Big-O): Medida de eficiencia de algoritmos según su tiempo y uso de memoria.

A través de esta investigación, buscamos establecer una comprensión integral de cómo estos elementos interrelacionados forman la base para el desarrollo de software eficiente y mantenible, capaz de resolver problemas complejos en diversos dominios de aplicación.

1. ¿Qué es estructura de Datos?

Una estructura de datos es una forma organizada de almacenar, procesar y recuperar información en un programa de computadora. Permite que los datos sean accesibles y manipulados de forma eficiente. En esencia, es una manera de organizar la información para facilitar su uso en aplicaciones informáticas.

1.1 ¿Para qué sirven las estructuras de datos?

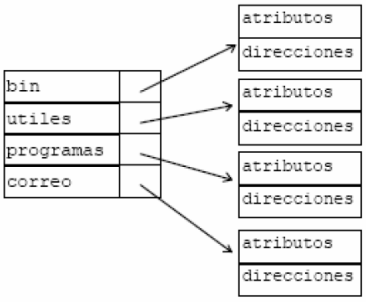
En el ámbito de la informática, las estructuras de datos son aquellas que nos permiten, como desarrolladores, organizar la información de manera eficiente, y en definitiva diseñar la solución correcta para un determinado problema.

Ya sean las más utilizadas comúnmente -como las variables, arrays, conjuntos o clases- o las diseñadas para un propósito específico -árboles, grafos, tablas, etc.-, una estructura de datos nos permite trabajar en un algo nivel de abstracción almacenando información para luego acceder a ella, modificarla y manipularla.

1.2 ¿Cuáles son los tipos de estructuras de datos?

Las estructuras de datos estáticas son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa, mientras que una estructura de datos dinámica es aquella en la que el tamaño ocupado en memoria puede modificarse durante la ejecución del programa.Cada tipo de estructura dependerá del tipo de aplicación que se requiera.

2.0 Tipos abstractos de datos (TAD)



Un Tipo Abstracto de Datos (TAD) es un modelo matemático que define:

- Un conjunto de datos (estructura).
- Un conjunto de operaciones que se pueden realizar sobre esos datos.

La clave de un TAD es que oculta los detalles de implementación, permitiendo trabajar con la estructura a través de una interfaz bien definida sin preocuparse por cómo está almacenada en memoria.

2.1 Características

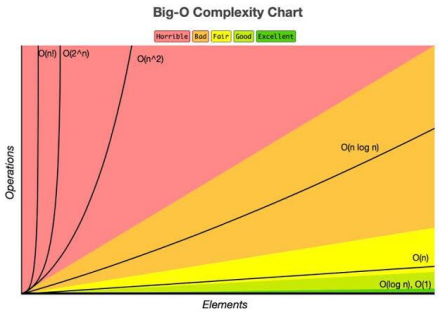
- Abstracción: Solo importa qué hace, no cómo lo hace.
- Encapsulamiento: La estructura interna es privada; solo se interactúa mediante operaciones.
- Reutilización: Puede implementarse de distintas formas sin cambiar su uso.

2.2 Importancia e la programación

- Permiten organizar datos de manera eficiente.
- Facilitan el diseño modular de software.
- Optimizan el rendimiento según el problema (ej: una cola es ideal para procesos en espera).

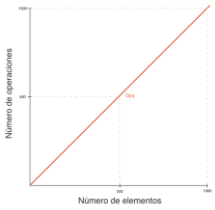
Los TADs son la base de estructuras más complejas y librerías estándar (como las colecciones de Java).

3.0 Complejidad algorítmica (Big-O)



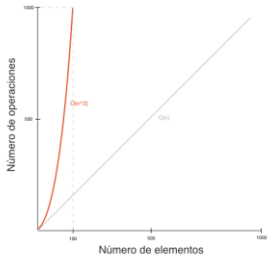
La complejidad algorítmica, o complejidad de algoritmos, es una medida de cuán eficiente es un algoritmo para resolver un problema. Evalúa la cantidad de recursos (tiempo y espacio/memoria) que el algoritmo requiere para funcionar, especialmente en relación al tamaño del problema a resolver. Esta medida permite comparar diferentes algoritmos y elegir el más eficiente para una tarea específica.

3.1 Complejidad lineal – O(n)



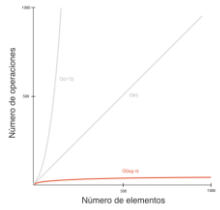
A medida que aumentan los datos el número de operaciones aumenta de forma proporcional. Por ejemplo, si necesitamos 1 operaciones por cada dato, vamos a necesitar 10 operaciones para 10 datos y 100 operaciones para 100 datos.

3.2 Complejidad exponencial – O(n^2), O(n^3), etc



A medida que aumenta la cantidad de datos, el número de operaciones crece de forma exponencial. Por ejemplo, una complejidad cuadrática (O(n^2)) si para un dato necesitamos 2 operaciones, para 2 datos vamos a necesitar 4 operaciones, para 3 datos 9 operaciones y así sucesivamente.

3.3 Complejidad logarítmica O(log n)



A medida que aumenta la cantidad de datos, el número de operaciones aumenta, pero no de forma proporcional a los datos.

En la vida real la complejidad logarítmica se da en casos en los que no es necesario recorrer todos los elementos. Por ejemplo, para adivinar un número de 1 a 100 podemos intentar con 50 y preguntar si el número es mayor o menor. Si es mayor intentamos con 75 y repetimos el proceso acercándonos cada vez más a la respuesta.

3.4 Complejidad constante – O(1)

A medida que aumenta la cantidad de datos el número de operaciones se mantiene constante. O(1) significa que sólo se ejecuta una operación, O(2) significa que se ejecutan dos operaciones, O(3) tres operaciones y así sucesivamente. Pero el número de operaciones no es importante, lo importante es que, independiente del número de datos de entrada, el rendimiento va a ser constante.

Una operación puede ser una asignación de una variable, una comparación, o algún cálculo aritmético (suma, resta, multiplicación, división, etc.), entre otros.

3.5 Complejidad espacial

La complejidad espacial es la cantidad de memoria que requiere el algoritmo. Algunas veces es posible reducir la complejidad temporal incrementando la complejidad espacial y viceversa. Por ejemplo, un algoritmo que utiliza una variable adicional es O(1). Un algoritmo que utiliza una lista adicional es O(n).

4.0 Clase

Una clase en programación actúa como una plantilla o un plano para crear objetos. Puedes pensar en ella como un molde que define cómo construir algo. En términos de programación, una clase especifica los atributos (datos) y los métodos (funciones) que los objetos creados a partir de esta clase tendrán.

Por ejemplo, considera una aplicación que gestiona vehículos. Podríamos definir una clase "Coche" que tenga atributos como "marca", "modelo" y "color", y métodos como "arrancar" y "parar".

En Java, las clases son una parte fundamental de la POO y son la base para la creación de objetos, que son instancias de esas clases. Una clase en Java es un plano o un modelo que define la estructura y el comportamiento de los objetos que se pueden crear a partir de ella.

4.1 Conceptos fundamentales de las clases

- Atributos: Las clases pueden tener atributos, también conocidos como variables de instancia, que representan las características o propiedades del objeto. En el ejemplo del coche los atributos serian el modelo, la velocidad máxima, etc.

- Métodos: Las clases contienen métodos que representan el comportamiento del objeto. Los métodos son funciones que pueden realizar acciones y manipular los atributos de la clase. En nuestro caso sería acelerar frenar, girar, etc.

- Constructores: Las clases pueden tener constructores, que son métodos especiales utilizados para inicializar objetos cuando se crean. Los constructores tienen el mismo nombre que la clase y pueden tener diferentes parámetros para configurar el estado inicial del objeto.

- Encapsulación: La encapsulación es un principio de la POO que se refiere a la ocultación de los detalles de implementación de una clase y la exposición controlada de sus atributos y métodos. En Java, puedes utilizar modificadores de acceso como private, protected, public, etc., para controlar el acceso a los miembros de una clase.

- Herencia: Las clases pueden heredar atributos y métodos de otras clases. La herencia permite la creación de jerarquías de clases, donde una clase (subclase o clase derivada) puede extender o heredar características de otra clase (superclase o clase base).

- Instanciación: Para utilizar una clase, primero debes crear una instancia u objeto de esa clase. Puedes hacerlo utilizando el operador new, seguido del nombre de la clase y los argumentos necesarios para el constructor si lo tiene.

5.0 Objetos

Los objetos en programación representan cosas del mundo real, así como conceptos abstractos con sus características y comportamientos específicos. Un objeto cuenta con su estructura interna que combina variables, funciones y estructuras de datos. Usando el nombre del objeto y la sintaxis según el lenguaje de programación, puedes visualizar los valores del objeto y llamar las funciones que tiene predefinidas.

Los elementos de un objeto se dividen en dos categorías principales: propiedades y métodos.

- Las propiedades, también conocidas como atributos, incluyen información sobre el objeto. Por ejemplo, si consideramos un objeto Coche, algunas de sus propiedades serán: el color, la marca, el modelo o el año de fabricación.

- Los métodos definen las operaciones que se pueden realizar con el objeto. Por ejemplo, para el objeto Coche, los métodos podrían ser acelerar, frenar o girar.

6.0 Listas

En programación, una lista (o secuencia) es una estructura de datos que organiza elementos en un orden específico. Es una colección ordenada de valores donde un mismo valor puede aparecer más de una vez. Las listas son ampliamente utilizadas para almacenar y manipular datos en diversos lenguajes de programación.

```
mi_lista = [1, "texto", 3.14, True]
```

6.1 Características

- Ordenadas: Los elementos en una lista tienen un orden específico (cada elemento ocupa una posición única y predecible en la secuencia de la lista). Este orden se mantiene, lo que significa que puedes acceder a los elementos en un secuencia predecible.

- Indexadas: Cada elemento en una lista tiene un índice asociado que representa su posición en la lista. La indexación generalmente comienza desde 0 (en algunos lenguajes puede comenzar desde 1).

- Mutables: En muchos lenguajes de programación, las listas son estructuras de datos mutables, lo que significa que puedes cambiar, añadir o eliminar elementos después de haber creado la lista.

- Homogéneas o Heterogéneas: Las listas pueden contener elementos del mismo tipo (homogéneas) o elementos de diferentes tipos (heterogéneas).

- Capacidad Dinámica: Las listas pueden crecer o disminuir dinámicamente en tamaño para adaptarse a la cantidad de elementos que contienen.

- Operaciones de Acceso Eficientes: Puedes acceder rápidamente a cualquier elemento en una lista utilizando su índice, lo que proporciona operaciones de acceso eficientes.

- Operaciones de Inserción y Eliminación Eficientes: Las listas permiten la inserción y eliminación eficientes de elementos en diferentes posiciones.

7.0 Conclusión

Los Tipos Abstractos de Datos (TAD) proporcionan una base teórica esencial para el manejo eficiente de información en programación, definiendo operaciones sin depender de implementaciones concretas. Al combinarlos con las estructuras de datos adecuadas (como listas, pilas, colas y árboles) y analizar su complejidad algorítmica (Big-O), podemos diseñar soluciones óptimas para distintos problemas.

En resumen, dominar TADs, estructuras de datos y su implementación en Java es clave para desarrollar software escalable, mantenible y de alto rendimiento. Este conocimiento no solo optimiza recursos, sino que también abre puertas a soluciones más elegantes en el desarrollo de aplicaciones complejas.

UNIVERSIDAD DEL VALLE DE ORIZABA

8.0 Bibliografía

Henry Blog. (s.f.). ¿Qué es una estructura de datos en programación? Soy Henry. https://blog.soyhenry.com/que-es-una-estructura-de-datos-en-programacion/

UNO. (s.f.). Tipo de dato abstracto. Google Sites. https://sites.google.com/site/programacioniiuno/temario/unidad-2---tipo-abstracto-de-dato/tipo-de-dato-abstracto

Make It Real. (s.f.). Complejidad algorítmica. Guías Make It Real. https://guias.makeitreal.camp/docs/algoritmos/complejidad

SAP. (s.f.). Tipo de datos abstractos. SAP Help Portal. https://help.sap.com/docs/EAD_HANA/0e60f05842fd41078917822867220c78/b3e01e20f33f45699fe21f6e9fec9e21.html?locale=es-ES

EBAC. (s.f.). ¿Qué es un objeto en programación? EBAC Blog. https://ebac.mx/blog/objeto-en-programacion

Apinem. (s.f.). ¿Qué son las listas en programación y para qué sirven? Apinem. https://www.apinem.com/listas-programacion-que-son-y-para-que-sirven/