UNIVERSIDAD DEL VALLE DE ORIZABA

UniVO®

Universidad del Valle de Orizaba

Reporte semanal

Miguel Angel Tress Ruiz

Docente. Flor Denisse Arenas Cortes

Licenciatura en Ingeniería en Sistemas computacionales

Asignatura. Estructura de datos II

Clase 1

Introducción

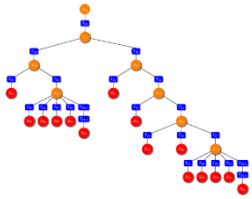
En la materia se enfatiza que las estructuras de datos son fundamentales porque permiten organizar la información de manera eficiente, facilitando su manipulación y optimizando tanto el tiempo de ejecución como el uso de memoria. Se trabaja bajo el concepto de Tipo Abstracto de Dato (TAD), que define cómo debe comportarse una estructura sin importar cómo se implemente, y se refuerza el uso de la notación Big-O, que sirve para analizar el rendimiento de los algoritmos según crece el tamaño de los datos.

Estructuras de Datos Lineales

Dentro de las estructuras lineales, se revisaron varias muy comunes y útiles:

- **Arrays o Arreglos:** son estructuras de tamaño fijo donde los elementos se almacenan de manera contigua en memoria. Su principal ventaja es el acceso directo por índice, lo que permite llegar a cualquier elemento de forma inmediata. Sin embargo, su desventaja es que no pueden crecer o reducirse dinámicamente.
- **ArrayList:** en Java, este tipo de lista es similar a los arreglos, pero con la ventaja de que pueden crecer de manera dinámica según se necesite. Son muy usados cuando no sabemos de antemano cuántos datos se van a almacenar.
- **Listas enlazadas:** a diferencia de los arreglos, no ocupan posiciones contiguas en memoria. Están formadas por nodos que contienen un dato y un puntero al siguiente (en el caso de las listas simples) o punteros al anterior y siguiente (en el caso de listas doblemente enlazadas). Este tipo de listas es muy eficiente cuando se requiere insertar o eliminar elementos con frecuencia.
- **Pilas (Stack):** funcionan bajo el principio LIFO (Last In, First Out), es decir, el último elemento en entrar es el primero en salir. Sus operaciones principales son *push* (insertar), *pop* (eliminar) y *peek* (consultar el elemento superior). Ejemplos de uso son los sistemas de deshacer/rehacer en editores o la gestión de llamadas recursivas en un programa.
- **Colas (Queue):** siguen el principio FIFO (First In, First Out), donde el primer elemento en entrar es el primero en salir. Se utilizan en contextos como sistemas de turnos, colas de impresión o planificación de procesos. Existen variantes como la cola circular, la doble cola (deque) o la cola de prioridad, donde algunos elementos pueden adelantarse en función de su importancia.
-

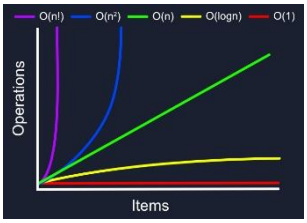
Estructuras de Datos No Lineales



Las estructuras no lineales permiten organizar la información en jerarquías o redes más complejas:

- **Tablas Hash:** almacenan pares clave-valor y permiten acceder a los datos de manera muy rápida. Suelen usarse en autenticación de usuarios, almacenamiento en caché o conteo de frecuencias.
- **Árboles:** organizan los datos de forma jerárquica. Dentro de ellos se encuentran varias variantes:
    - Árbol binario: cada nodo tiene como máximo dos hijos.
    - Árbol binario de búsqueda (BST): mantiene ordenados los elementos, facilitando las búsquedas.
    - Árbol AVL: son árboles balanceados, que evitan que las ramas se vuelvan demasiado largas en comparación con otras.
    - Árboles B y B+: se utilizan principalmente en bases de datos, ya que permiten manejar grandes cantidades de información en disco.
- **Grafos:** son estructuras formadas por nodos (vértices) y conexiones (aristas). Se dividen en dirigidos o no dirigidos, ponderados o no ponderados. Son muy importantes en algoritmos como búsqueda en profundidad (DFS), búsqueda en anchura (BFS) y en la búsqueda de rutas más cortas con Dijkstra, Kruskal o Prim. Ejemplos claros de su uso son Google Maps o las redes sociales.
- **Heaps:** son árboles binarios completos con propiedades de prioridad, donde el valor de cada nodo se organiza en función de sus hijos. Existen MinHeap y MaxHeap, y son esenciales en la implementación de colas de prioridad o en algoritmos como Dijkstra.
- **Tries:** se conocen como árboles digitales para cadenas, muy útiles en sistemas de autocompletado o validación de prefijos, como ocurre en los buscadores.
- **Bloom Filters:** estructuras probabilísticas que permiten verificar rápidamente si un dato está presente o no, aunque con la posibilidad de falsos positivos. Se usan mucho en sistemas distribuidos.
- **Segment Trees y Fenwick Trees:** estructuras avanzadas que permiten responder consultas en rangos (sumas, mínimos o máximos). Son muy utilizadas en bases de datos, videojuegos y análisis de grandes volúmenes de datos en tiempo real.

Complejidad Algorítmica (Big-O)



Finalmente, se repasó la clasificación de algoritmos según su eficiencia:

- **O(1) constante:** siempre tarda lo mismo, como acceder a un elemento de un arreglo.
- **O(log n) logarítmica:** divide el problema por la mitad en cada paso, como en la búsqueda binaria.
- **O(n) lineal:** depende directamente del tamaño de la entrada.
- **O(n log n) cuasi lineal:** común en algoritmos de ordenamiento eficientes como *merge sort* o *quick sort*.
- **O(n^2) cuadrática:** crece muy rápido, como los algoritmos de burbuja.
- **O(2^n) exponencial:** se dispara en tiempo, como en la recursión del Fibonacci.
- **O(n!) factorial:** completamente impráctico en entradas grandes, como en la generación de todas las permutaciones posibles.

Este análisis es fundamental porque permite elegir la estructura y el algoritmo más adecuado dependiendo del problema que se quiera resolver.

Conclusión

En conclusión, esta semana el curso reforzó la importancia de comprender las distintas estructuras de datos y su papel en el desarrollo de software. No se trata únicamente de aprender definiciones, sino de entender en qué contexto se debe usar cada una y cómo afectan al rendimiento general de un programa. También quedó claro que las aplicaciones que usamos diariamente, desde buscadores hasta redes sociales y videojuegos, funcionan gracias a estas estructuras.