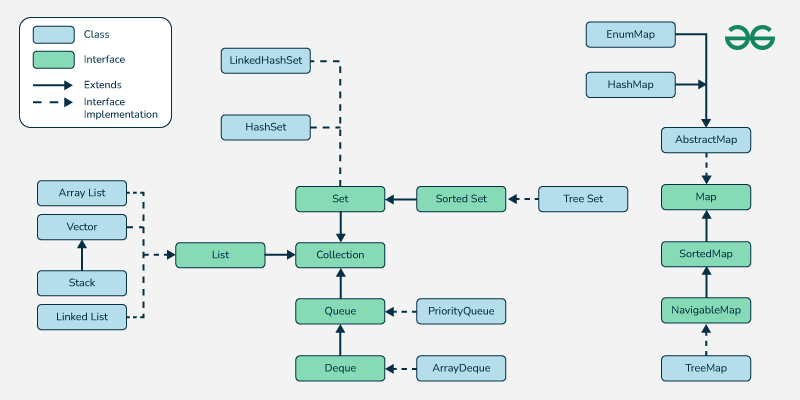
1. **INTRODUCCIÓN**

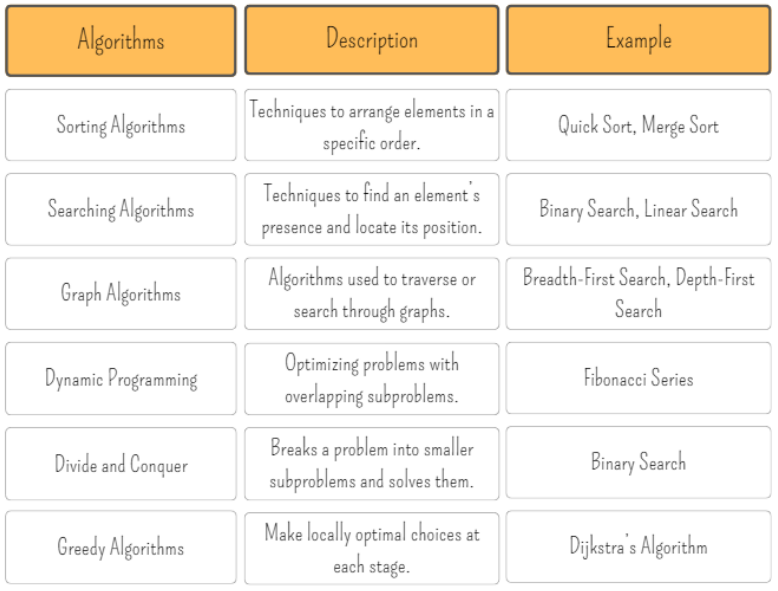
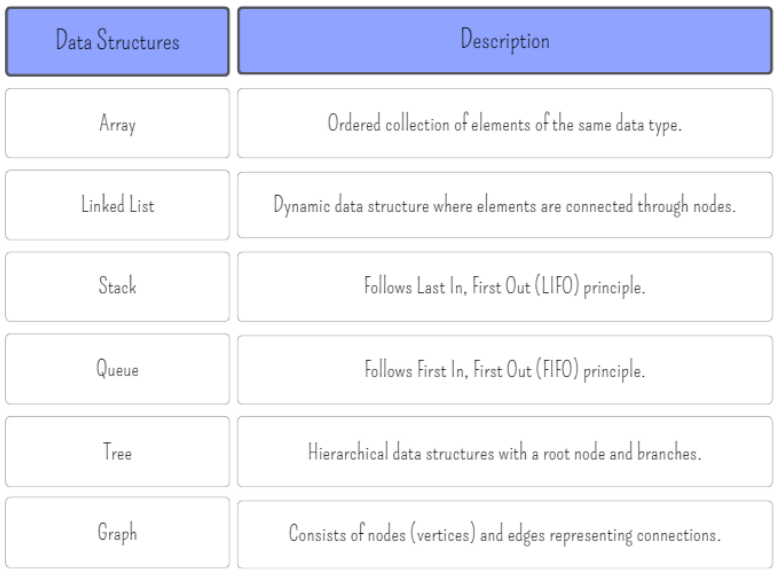
Las colecciones permiten manejar y agrupar datos de forma estructurada y flexible: almacenar, recuperar y manipularlos. Son un conjunto de interfaces, clases concretas y abstractas que definen tipos de datos abstractos en Java que implementan colecciones de elementos.



Pueden tener un orden determinado, que puede ser lineal o no lineal, habiendo diferentes estructuras las cuales cada una podemos usar cuando más las necesitemos.

Al trabajar con aplicaciones Javas del mundo real, se trabaja con grandes cantidades de datos, siendo necesarios de organizar y acceder de forma eficiente. Sin las colecciones, habría que escribir mucho código personalizado para manejar estructuras de datos, siendo difícil de mantener y escalar.

Las colecciones de Java proporcionan una forma estandarizada de manejar estos datos. Cuentan con métodos integrados que simplifican tareas complejas como la ordenación, la búsqueda y el filtrado. De hecho, las colecciones son dinámicas, por lo que pueden crecer o reducirse según sea necesario, lo que supone una ventaja significativa con respecto a las matrices, que tienen un tamaño fijo.



Repasar representación gráfica de las estructuras de datos.

Repasar representación de algoritmos

1. **ESTRUCTURAS DE DATOS**

Las estructuras de datos son formatos especiales para organizar y almacenar datos en un ordenador. Son como contenedores que almacenan información de forma específica. Mantienen nuestra información en orden. Son importantes porque nos ayudan a manejar y organizar grandes cantidades de datos. Hacen más fácil y rápida la búsqueda y uso de información.

* 1. **Arrays**

Un array es una colección de ítems del mismo tipo almacenados en orden uno detrás del otro. Cada ítem tiene un índice para encontrarlo fácilmente. Tienen un tamaño fijo que no se modifica tras su creación. Son útiles cuando sabes cuántos ítems se necesitan almacenar y cuando quieres acceder a ítems de forma rápida por su posición.

Access: O(1) Search: O(n) Insertion: O(n) Deletion: O(n)

* + 1. ***ArrayList***

Array redimensionable que permite almacenar una lista de elementos. Automáticamente ajusta su tamaño al añadir o eliminar elementos. Provee métodos para hacer operaciones de forma fácil, y mantiene el orden en el que los elementos han sido insertados. Son buenos para almacenar y acceder a datos de forma rápida. No son buenos para añadir o eliminar ítems en la mitad del array.

Access: O(1) Search: O(n) Insertion (at end): O(1) Insertion (at specific index): O(n) Deletion (at end): O(1) Deletion (at specific index): O(n)

* + 1. ***LinkedList***

Consiste en una secuencia de elementos donde cada elemento está enlazado a siguiente y al anterior, formando una cadena. Cada elemento es almacenado en un nodo separado que contiene dicho elemento y referencias al nodo siguiente y al anterior. Están hechas para ítems enlazados entre sí. Son eficientes para inserciones y eliminaciones en cualquier posición, pero más lentos para acceder a elementos por un índice concreto.

Access: O(1) Search: O(n) Insertion (at beginning/end): O(1) Insertion (at specific index): O(n) Deletion (at beginning/end): O(1) Deletion (at specific index): O(n)

* 1. **Stacks**

Siguen el principio LIFO. Permiten operaciones como push, pop, peek… Se usan en escenarios como backtracking, mecanismos para deshacer, y expresiones de parseo. Útiles para tareas que necesitan un orden inverso o llevar un seguimiento sobre lo último que ha pasado.

Push: O(1) Pop: O(1) Peek: O(1) Search: O(n)

* 1. **Queues**

Siguen el principio FIFO. En Java, una queue provee operaciones como offer (añadir elementos), poll (eliminar y devolver el primer elemento), o peek (ver el primer elemento sin eliminarlo). También es útil para manejar tareas en orden, como trabajos de impresión o solicitudes de servicio al cliente.

Enqueue: O(1) Dequeue: O(1) Peek: O(1) Search: O(n)

* 1. **Trees**

Un árbol en Java es una estructura de datos jerárquica en la que cada elemento, llamado nodo, está conectado a otros en una relación padre-hijo. El nodo superior se llama raíz y cada nodo puede tener cero o más nodos secundarios. Los árboles son útiles para representar datos jerárquicos, como sistemas de archivos, estructuras organizativas o procesos de toma de decisiones. En Java existen tipos específicos de árboles, como los árboles binarios, en los que cada nodo tiene como máximo dos hijos, y árboles equilibrados como TreeMap o TreeSet, que mantienen los elementos en orden ordenado para realizar búsquedas e inserciones rápidas.

* 1. **Binary Searh Trees (BST)**

Es un tipo de árbol binario en el que cada nodo tiene como máximo dos hijos y el árbol mantiene un orden específico. El hijo izquierdo contiene valores menores que el valor del nodo y el hijo derecho contiene valores mayores que el valor del nodo. Los árboles binarios de búsqueda se utilizan comúnmente para buscar, ordenar y mantener un conjunto dinámico de datos ordenados.

En el ejemplo de los apuntes se usa la clase Comparable. Esta interfaz impone un ordenamiento total a los objetos de cada clase que la implementa. Este ordenamiento se refiere como el ordenamiento natural de la clase, y el método compareTo de la clase se refiere como su método de comparativa natural.

En el cuaderno y en Eclipse tengo un ejemplo de árbol.



* 1. **Hash Tables**

Estructura de datos que almacena parejas clave-valor, permitiendo una recuperación rápida de valores en función de sus claves. Se utilizan ampliamente en muchas aplicaciones donde el acceso rápido a los datos es crucial. Si varias claves se asignan al mismo índice (colisión), técnicas adicionales como encadenamiento o direccionamiento abierto son usadas para resolver el conflicto. Las hash tables se implementan en java a través de clases como HashMap o Hashtable.

Insert: O(1) average case, O(1) worst case Delete: O(1) average case, O(1) worst case Search: O(1) average case, O(1) worst case