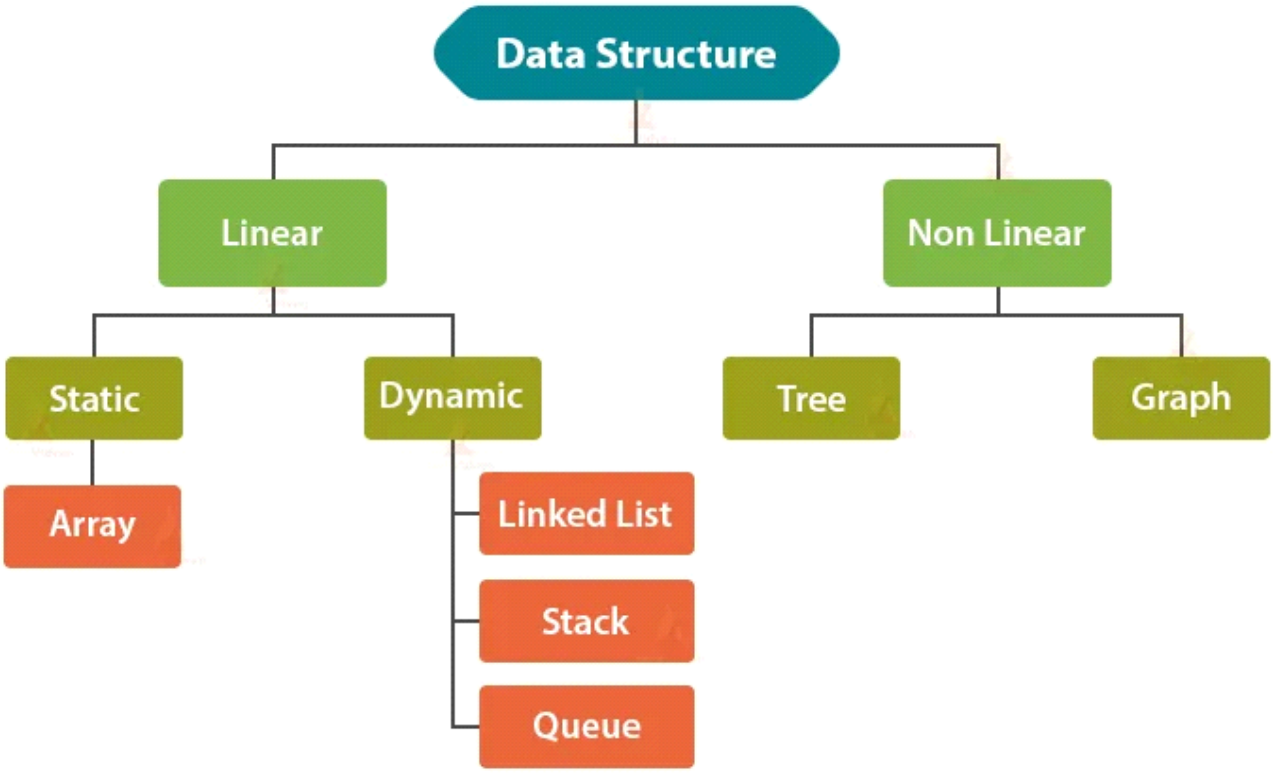
Estructuras de Datos

	Datos	Ventajas	Desventajas	Aplicación
Arrays	Estructura de datos lineal	Acceso aleatorio	El tamaño es fijo	Para almacenar información de forma lineal
	Los elementos se almacenan en ubicaciones de memoria contiguas	Fácil clasificación e iteración	Difícil de insertar y eliminar	Adecuado para aplicaciones que requieren búsquedas frecuentes
	Puede acceder a elementos aleatoriamente usando index	Reemplazo de múltiples variables	Si la capacidad es mayor y la ocupación menor, la mayor parte de la matriz se desperdicia	
	Almacena elementos homogéneos, es decir, elementos similares		Necesita memoria contigua para ser asignada	
Linked List	Estructura de datos lineal	Dinámico en tamaño	Si se pierde el nodo principal, se pierde la lista vinculada	Adecuado donde la memoria es limitada
	Los elementos se pueden almacenar según la disponibilidad de memoria	Sin desperdicio, ya que la capacidad y el tamaño son siempre iguales	No es posible el acceso aleatorio	Adecuado para aplicaciones que requieren inserción y eliminación frecuentes
	Puede acceder a elementos solo de forma lineal	Fácil inserción y eliminación ya que se requiere la manipulación de 1 enlace		
	Almacena elementos homogéneos, es decir, elementos similares	Asignación de memoria eficiente		
	Dinámico en tamaño			
	Fácil inserción y eliminación			
	El elemento inicial o nodo es la clave que generalmente se denomina cabeza.			
Stack	Estructuras de datos lineales usando Java	Mantiene los datos de forma LIFO	La manipulación está restringida a la parte superior de la pila.	Recursividad
	Sigue LIFO: último en entrar, primero en salir	El último elemento está disponible para su uso	No muy flexible	Analizando
	Solo se puede acceder a los elementos superiores	Todas las operaciones son de complejidad O (1)		Navegador
	La inserción y eliminación se realiza desde arriba			Editores
	Toda la operación funciona en tiempo constante, es decir, O (1)			
Queue/Cola	Estructura de datos lineal	Mantiene los datos en forma FIFO	Depende la implementación	Planificación
	Sigue a FIFO: primero en entrar, primero en salir	La inserción desde el principio y la eliminación desde el final toma O (1) tiempo		Manteniendo la lista de reproducción
	La inserción puede realizarse desde la parte trasera.			Interrumpir el manejo
	La eliminación puede tener lugar desde la interfaz.			
	Toda la operación funciona en tiempo constante, es decir, O (1)			
Árbol binario	Estructura de datos jerárquica	Puede representar datos con alguna relación	Clasificar es difícil	Jerarquía del sistema de archivos
	El elemento superior se conoce como la raíz del árbol.	La inserción y la búsqueda son mucho más eficientes	No muy flexible	Varias variaciones del árbol binario tienen una amplia variedad de aplicaciones
	Cada nodo puede tener como máximo 2 hijos en el árbol binario			
	Puede acceder a elementos aleatoriamente usando index			
Heap	El montón binario se puede visualizar como una matriz como un árbol binario completo	Puede ser de 2 tipos: min heap y max heap	No es posible el acceso aleatorio	Adecuado para aplicaciones que se ocupan de la prioridad
	El elemento Arr [0] se tratará como raíz	El montón mínimo mantiene el más pequeño y el elemento y la parte superior y el máximo mantiene el más grande	Solo el elemento mínimo o máximo está disponible para accesibilidad	Algoritmo de programación
	longitud (A) - tamaño de la matriz	O (1) para tratar con elementos mínimos o máximos		almacenamiento en caché
	heapSize (A) - tamaño del montón			
	Usado generalmente cuando se trata de elementos mínimos y máximos			
Hashing	Para el i-ésimo nodo	La función hash ayuda a buscar elementos en tiempo constante.	La resolución de colisiones aumenta la complejidad	Adecuado para la aplicación que necesita un tiempo constante de búsqueda.
	Utiliza la función hash especial	Una forma eficiente de almacenar elementos.		
	Una función hash mapea un elemento a una dirección para su almacenamiento			
	Esto proporciona acceso en tiempo constante			
	La colisión se maneja mediante técnicas de resolución de colisiones.			
	Técnica de resolución de colisiones			
	Encadenamiento			
	Direccionamiento abierto			
Graph	Básicamente es un grupo de aristas y vértices.	encontrar conectividad	El almacenamiento del gráfico (lista de adyacencia y matriz de adyacencia) puede generar complejidades	Adecuado para una red de circuitos
	Representación gráfica	Camino más corto		Adecuado para aplicaciones como Facebook, LinkedIn, etc.
	G (V, E); donde V (G) representa un conjunto de vértices y E (G) representa un conjunto de aristas	coste mínimo para pasar de 1 pt a otro		Ciencia médica
	El gráfico puede ser dirigido o no dirigido	Árbol de expansión mínimo		
	El gráfico puede estar conectado o disjunto.			

Data Structure Classification in Java



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)