Capítulo 5 ESTRUCTURAS DE DATOS

Nombre: Jean Carlos Iñiguez

* **MATRICES (ARRAYS), PILAS Y COLAS:**
* Una matriz es una estructura de datos que almacena un número fijo de elementos del mismo tipo, en posiciones contiguas de memoria.
* En una matriz unidimensional, se accede a cada elemento usando un único índice (ej. tab[1] a tab[50]).
* Se puede calcular la dirección de cualquier elemento en tiempo constante O(1), por eso leer o escribir en una posición es muy eficiente.

**Ejemplo:**  
Si los enteros ocupan 4 bytes y la matriz empieza en 5000,  
la dirección de tab[k] se calcula con: 4996 + 4k.

* Operaciones que recorren toda la matriz (como buscar el máximo o inicializar valores) tienen tiempo lineal O(n).

**Implementación de Pilas con Arrays**

* Una pila (stack) sigue el principio LIFO: *Último en entrar, primero en salir*.
* Se usa una matriz + contador para implementarla:
  + push (apilar): aumenta el contador y agrega el elemento.
  + pop (desapilar): toma el elemento actual y reduce el contador.
* Hay que controlar desbordamientos (stack overflow) y subdesbordamientos (vaciar una pila vacía).

**Implementación de Colas con Arrays**

* Una cola (queue) sigue el principio FIFO: *Primero en entrar, primero en salir*.
* Operaciones:
  + enqueue (poner): agrega un elemento al final.
  + dequeue (quitar): elimina el elemento del frente.
* También se implementa con una matriz fija y punteros para el inicio y el final.
* El problema: se necesita prever un tamaño máximo, lo que puede desperdiciar espacio o causar límites innecesarios.
* **ARBOLES:**

Un árbol es una estructura fundamental en informática y matemáticas que puede definirse como un grafo no dirigido, conexo y sin ciclos. Esto significa que, entre cualquier par de nodos en un árbol, hay exactamente un único camino, lo que evita redundancias o caminos alternativos.

Los árboles poseen propiedades importantes que los distinguen de otros grafos, como la relación entre el número de nodos y aristas, y cómo se comportan al añadir o eliminar conexiones. Además, en la mayoría de las aplicaciones prácticas, trabajamos con árboles con raíz, que tienen un nodo especial (la raíz) desde donde se organiza toda la estructura de manera jerárquica, facilitando su representación y uso.

**Puntos clave:**

* Un árbol es un grafo no dirigido, conexo y sin ciclos.
* Entre dos nodos cualesquiera existe un único camino.
* Un árbol con *n* nodos tiene *n - 1* aristas.
* Añadir una arista crea un ciclo único.
* Eliminar una arista desconecta el grafo.
* En un árbol con raíz, la raíz es un nodo especial ubicado en la parte superior.
* La estructura jerárquica de los árboles con raíz es similar a la de un árbol genealógico.
* La mayoría de los ejemplos prácticos se basan en árboles con raíz por su utilidad.
* **MONTICULOS (HEAPS):**

Un montículo o heap es un tipo especial de árbol con raíz que se puede representar de forma muy eficiente usando un arreglo, sin necesidad de punteros para los nodos. Esto permite implementaciones rápidas y compactas. Los montículos son útiles en muchas aplicaciones, destacando la ordenación por montículos (heapsort) y la gestión eficiente de listas dinámicas de prioridades, como las colas de prioridad usadas en sistemas operativos o simulaciones.

Un montículo es un árbol binario esencialmente completo, lo que significa que casi todos los niveles del árbol están completamente llenos, excepto quizás el último nivel, que se llena de izquierda a derecha. Esta propiedad hace que el heap sea especialmente adecuado para ser almacenado en un arreglo porque los nodos se organizan de forma compacta y ordenada.

Además, la altura del árbol binario esencialmente completo con nnn nodos es aproximadamente ⌊log⁡2n⌋\lfloor \log\_2 n \rfloor⌊log2​n⌋, lo que es importante para analizar la eficiencia de las operaciones sobre el heap.

**Puntos clave:**

* Un montículo (heap) es un árbol con raíz que puede implementarse eficientemente con un arreglo.
* No necesita punteros explícitos para representar las relaciones entre nodos.
* Se usa en la ordenación por montículos (heapsort) y para manejar listas de prioridad dinámicas.
* Un heap es un árbol binario esencialmente completo:
  + Todos los nodos internos (excepto uno posible) tienen dos hijos.
  + El último nivel se llena de izquierda a derecha.
* La estructura del heap asegura que los nodos estén “compactados” hacia arriba y a la izquierda.
* Si el heap tiene altura k, entonces:
  + Nivel k: 1 nodo (la raíz)
  + Nivel k−1: 2 nodos
  + Nivel k−2: 4 nodos, etc.
* La altura k de un heap con n nodos cumple: k=⌊log2n⌋
* Esto implica que las operaciones sobre montículos son eficientes en tiempo logarítmico.