Árboles: Conceptos y Aplicaciones

1. Definición de Árboles

Un árbol es una estructura de datos que puede definirse de forma recursiva o no recursiva.

- Definición no recursiva:

  Un árbol está formado por un conjunto de nodos y un conjunto de aristas dirigidas que conectan pares de nodos.

  En un árbol con raíz:

  - Uno de los nodos es la raíz.
  - Cada nodo, excepto la raíz, tiene un único nodo padre.
  - Existe un único camino desde la raíz hasta cualquier nodo.

- Definición recursiva:

  Un árbol es:

  - Vacío, o
  - Una raíz más un conjunto de subárboles, cada uno también es un árbol.

2. Conceptos Importantísimos

2.1 Nodos y Relaciones

- Raíz: nodo sin padre, el nodo superior.
- Hojas: nodos sin hijos.
- Hermanos: nodos que comparten el mismo padre.
- Camino: secuencia de aristas entre dos nodos.

  Ejemplo: El camino A → B → F tiene longitud 2.

2.2 Métricas

- Profundidad de un nodo: longitud del camino desde la raíz hasta ese nodo.

  Ejemplo: Profundidad de A = 0, B = 1, K = 2.
- Altura de un nodo: longitud del camino más largo desde ese nodo hasta una hoja.

  Ejemplo: Altura de E = 2 (camino E → H → K).
- Tamaño de un nodo: número de descendientes más uno (incluyéndolo a sí mismo).

  Ejemplo: Tamaño de B = 3 (B, F, G), tamaño de A = 11 (todo el árbol).

2.3 Propiedades

- Un árbol con N nodos tiene N - 1 aristas (cada nodo, excepto la raíz, tiene un padre).

3. Implementación de Árboles Generales

3.1 Método Primer Hijo / Siguiente Hermano

- Cada nodo almacena dos referencias:
  - Un enlace al primer hijo (más a la izquierda).
  - Un enlace al siguiente hermano (a la derecha).

- Ventajas:
  - Solo dos punteros por nodo, independientemente del número de hijos.
  - Permite representar árboles con un número variable de hijos.

3.2 Ejemplo Visual

- B es el primer hijo de A.
- C y D son hermanos de B.
- E es primer hijo de B; F es hermano de E.

4. Árboles en Sistemas de Archivos

- La raíz es el directorio principal (ejemplo: mark/ en Unix).
- Los hijos son subdirectorios o archivos (ejemplo: books/, courses/, .login).
- Las rutas representan caminos en el árbol, por ejemplo: mark/books/dsaa/ch1.

4.1 Recorridos

- Preorden: procesa el directorio actual antes que sus hijos (útil para listar rutas).
- Postorden: procesa los hijos antes que el directorio (útil para calcular tamaños totales).

5. Árboles Binarios

5.1 Definición y Estructura

- Un árbol binario es un árbol donde cada nodo tiene como máximo dos hijos: izquierdo (left) y derecho (right).
- Definición recursiva:
  Un árbol binario es vacío o una raíz con un subárbol izquierdo y un subárbol derecho.

5.2 Implementación básica en Java

```
Public class Node<AnyType> {
  T element;
  Node<T> left;
  Node<T> right;
}
```

6. Operaciones Básicas en Árboles Binarios

- Inserción y eliminación dependen del tipo de árbol (por ejemplo, árbol de búsqueda binaria).
- Recorridos principales:
  - Preorden: raíz → izquierdo → derecho.
  - Inorden: izquierdo → raíz → derecho (devuelve elementos ordenados en BST).
  - Postorden: izquierdo → derecho → raíz.

- Métodos auxiliares comunes:
  - size() para contar nodos (recursivo).
  - height() para calcular la altura.

7. Aplicaciones Clave de Árboles Binarios

- Árboles de Expresión:
  - Hojas: operandos (variables o constantes).
  - Nodos internos: operadores (+, -, etc.).
  - Se usan para evaluar expresiones aritméticas.

- Árboles de Búsqueda Binaria (BST):
  - Propiedad: valores a la izquierda < raíz < valores a la derecha.
  - Usados para búsquedas eficientes tiene complejidad promedio O(log n).

- Colas de Prioridad: implementadas con montículos binarios.

8. Recursión en Árboles Binarios: Funciones Clave

8.1 Calcular tamaño (size())

Si nodo es nulo:

    retornar 0

Sino:

    retornar 1 + size(nodo.izquierdo) + size(nodo.derecho)

8.2 Calcular altura (height())

```
Si nodo es nulo:
        retornar -1
Sino:
        izquierda = height(nodo.izquierdo)
        derecha = height(nodo.derecho)
        retornar 1 + máximo(izquierda, derecha)
```

9. Recorridos Iterativos con Pilas y Colas

- Preorden: apilar raíz, luego apilar hijo derecho y luego izquierdo.
- Inorden: usar contador para controlar estados y apilar nodos.
- Postorden: apilar con contador para procesar hijos antes que el nodo.