Programación II
Árboles binarios de búsqueda (ABB)

Definición

Un árbol binario de búsqueda(ABB) a es una estructura de datos de tipo árbol binario en el que para todos sus nodos, el hijo izquierdo, si existe, contiene un valor menor que el nodo padre y el hijo derecho, si existe, contiene un valor mayor que el del nodo padre.
También vale la propiedad para a.izq y a.der.

Es de búsqueda porqué:
Los nodos están ordenados de manera conveniente para la búsqueda (Ver Figura1).

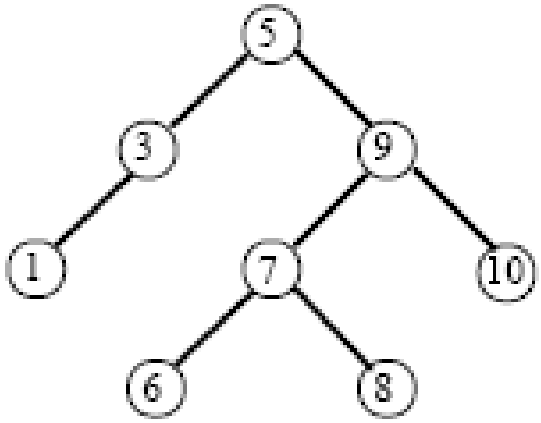


Figura1: Ejemplo de un ABB

Invariante de representación

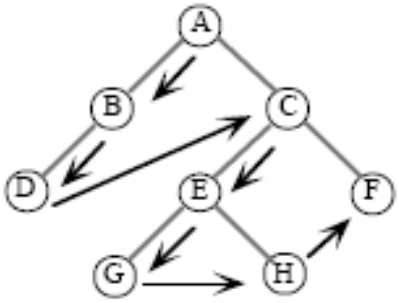
ABB(a) ⇔
a es un AB tal que
-Todos los nodos de a.izq son menos que a.info
-Todos los nodos de a.der son mayores que a.info
-ABB(a.izq)
-ABB(a.der)

Recorrido de Árbol binario

Recorrer un árbol consiste en acceder una sola vez a todos sus nodos.
Esta operación es básica en el tratamiento de árboles y permite, por ejemplo, imprimir toda la información almacenada en el árbol

- Imponiendo la restricción de que el subárbol izquierdo se recorre siempre antes que el derecho, esta forma de proceder da lugar a tres tipos de recorrido, que se diferencian por el orden en el que se realizan estos tres pasos.

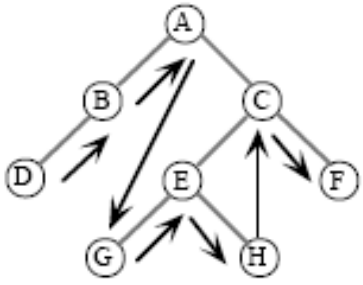
Preorden: primero se accede a la información del nodo, después al subárbol izquierdo y después al derecho(Ver Figura6).



A - B - D - C - E - G - H - F

Figura6: Preorden

Inorden: primero se accede a la información del subárbol izquierdo, después se accede a la información del nodo y, por último, se accede a la información del subárbol derecho (Ver Figura7).



D - B - A - G - E - H - C - F

Figura7: Inorden

Postorden: primero se accede a la información del subárbol izquierdo, después a la del subárbol derecho y, por último, se accede a la información del nodo (Ver Figura8).

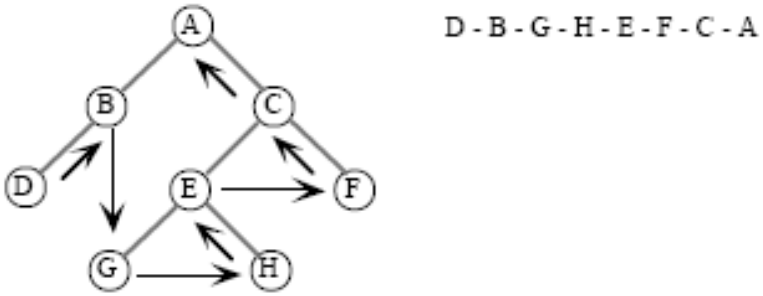


Figura8: Postorden

Si agregamos los nodos más chicos a la izquierda y los más grandes a la derecha;
Y recorremos un árbol *Inorden*, obtenemos los nodos de manera ordenada (Ver Figura9):

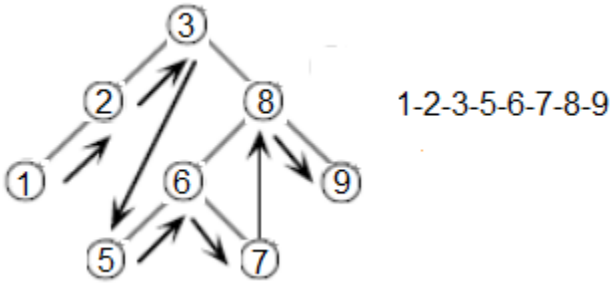


Figura9: Nodos ordenados

Búsqueda

1. Si el valor del nodo actual es igual al valor buscado, lo hemos encontrado.
2. Si el valor buscado es menor que el del nodo actual, deberemos inspeccionar el subárbol izquierdo.
3. Si el valor buscado es mayor que el del nodo actual, deberemos inspeccionar el subárbol derecho.

Para continuar la búsqueda en el subárbol adecuado se aplica el mismo razonamiento.

```
boolean pertenece(Integer x, Nodo<Integer> p) {
      if (p == null)
            return false;
      if (x < p.val)
            return pertenece(x, p.izq);
      if (x > p.val)
            return pertenece(x, p.der);
      return (x == p.val);
}
```

Inserción

La operación de inserción de un nuevo nodo en un árbol binario de búsqueda consta de tres fases básicas:
1. Creación del nuevo nodo.
2. Búsqueda de su posición correspondiente en el árbol. Se trata de encontrar la Posición que le corresponde para que el árbol resultante siga siendo de búsqueda.
3. Inserción en la posición encontrada. Se modifican de modo adecuado los enlaces de la estructura.

La versión recursiva es muy similar al buscar. Una vez que se llega a una hoja, se inserta el elemento ahí:

```
	protected Nodo<Integer> insertar(Integer x, Nodo<Integer> p)
{
		if (p == null)
			return new Nodo<>(x);
		if (x < p.val)
			p.izq = insertar(x, p.izq);
		else if (x > p.val)
			p.der = insertar(x, p.der);
		return p;
	}
```

A continuación se muestra como queda el Árbol luego de insertar cada nodo:

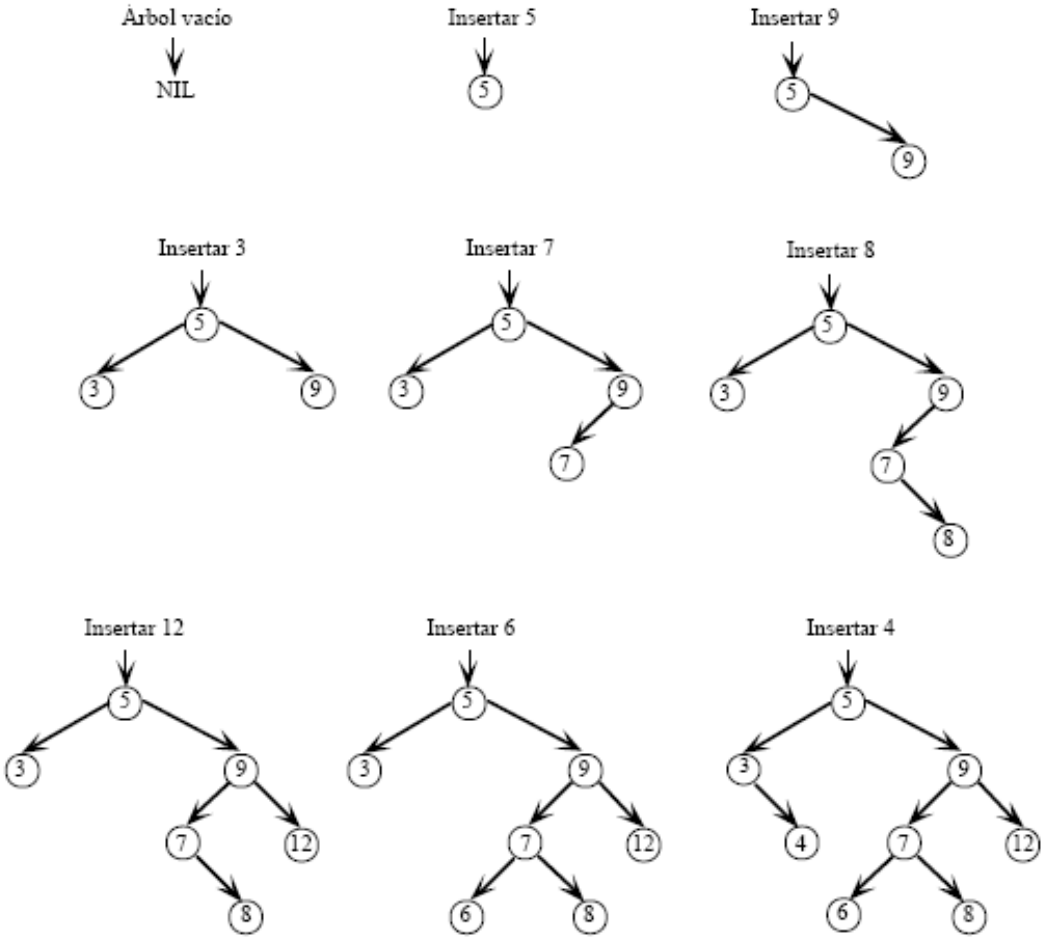


Figura2: Inserción en un ABB de 8 nodos.

Como cambia el mínimo de ABB respecto de AB

Notar que mínimo es una función ***parcial****, por lo cual no opera sobre todos los árboles, en particular, no opera sobre áboles con 0 nodos.

Asumir que T == Integer para este ejercicio.

Class **AB**<T>

```
int minimo()
if (raiz != null) return minimo(raiz)

private int minimo(Nodo nodo)
if (nodo == null)
        return MAX_INT
else
        return Math.min(nodo.info, minimo(nodo.izq), minimo(nodo.der)  )
```

Notar que hay que traducir el pseudocódigo a java. Ejercicio: implementar en java la función Integer min(Integer a, Integer b, Integer c) que recibe tres argumentos y devuelve el más chico.

Utilizamos el irep de ABB a nuestro favor:

Class **ABB**<T> extends AB<T>

```
private int minimo(Nodo nodo)
if (nodo.izq == null)
        return nodo.info
else
        return minimo(nodo.izq)
```

*Funciones Totales/Parciales

Totales: Operan sobre todo el dominio
Parciales: Operan sobre una parte del dominio.

Eliminar

Existen cuatro distintos escenarios:

1. Intentar eliminar un nodo que no existe.
   - No se hace nada, simplemente se regresa FALSE.
2. Eliminar un nodo hoja.
   - Caso sencillo; se borra el nodo y se actualiza el apuntador del nodo padre a NULL.
3. Eliminar un nodo con un solo hijo.
   - Caso sencillo; el nodo hijo se convierte en el padre.
4. Eliminar un nodo con dos hijos.
   - Caso complejo, es necesario mover más de una referencia.
   - Se busca el máximo de la rama izquierda o **el mínimo de la rama derecha**.

Eliminar, Caso2 y caso3 (Figura3a y Figura3b):

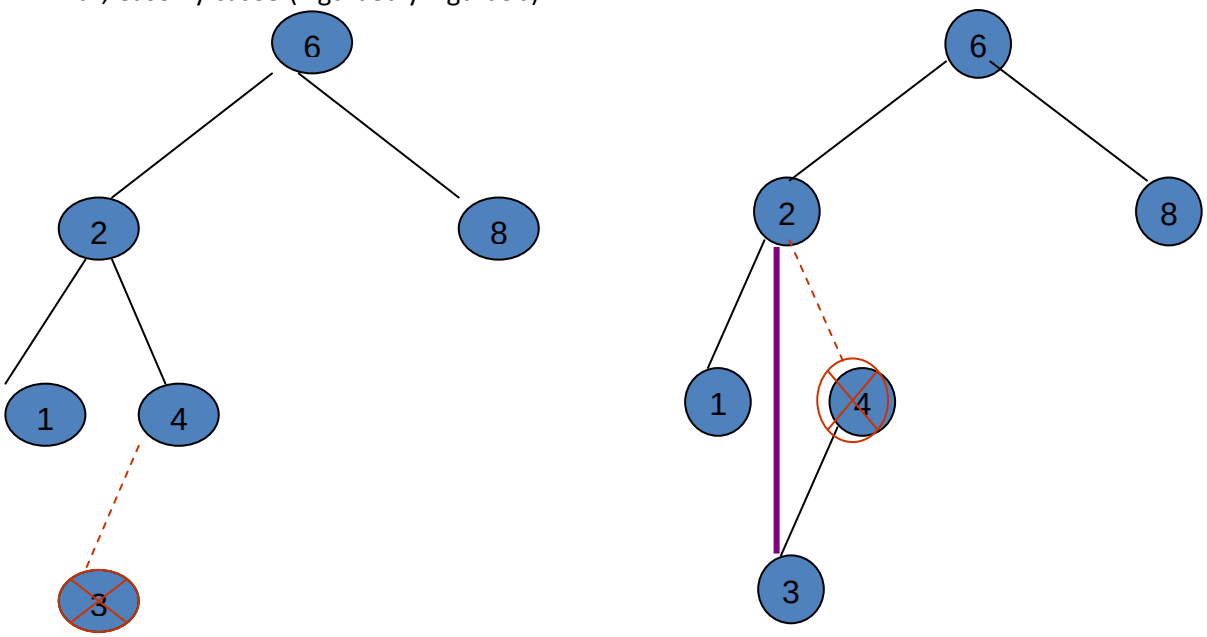


Figura3a: Eliminar Nodo hoja y Figura3b: Eliminar nodo con un hijo

Eliminar, Caso4 (Figura4):

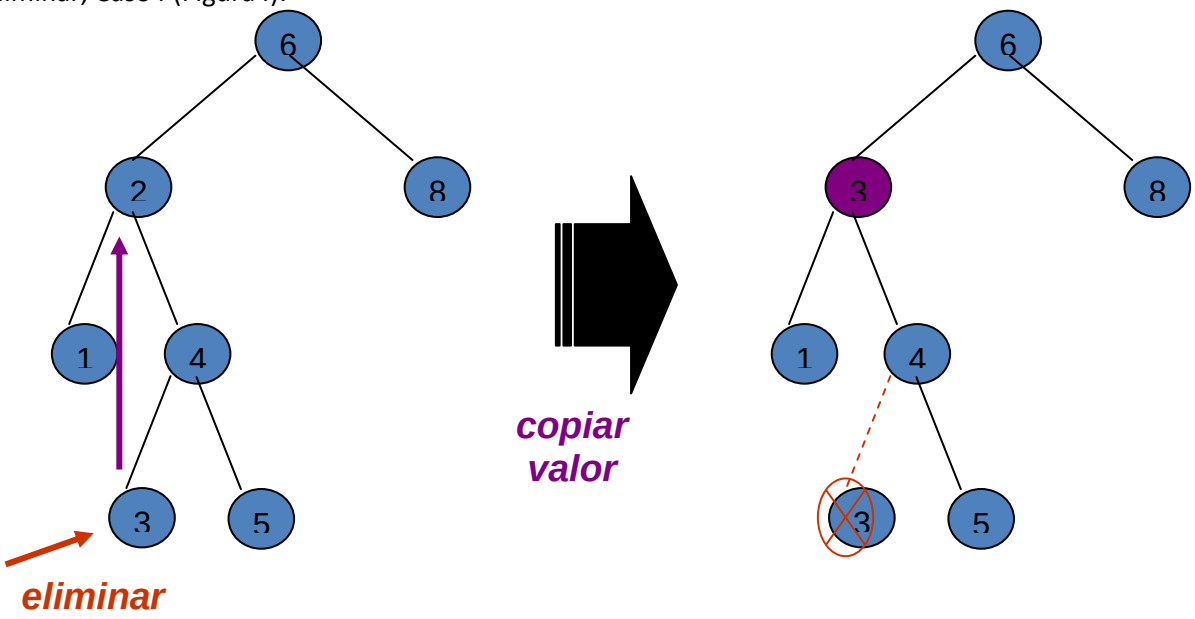


Figura 4: Eliminar nodo con dos hijos (Ejemplo1)

Remplazar el dato del nodo que se desea eliminar con el dato del nodo más pequeño del subárbol derecho.
Después, eliminar el nodo más pequeño del subárbol derecho (caso fácil)

Eliminar, Caso4 (Figura5):

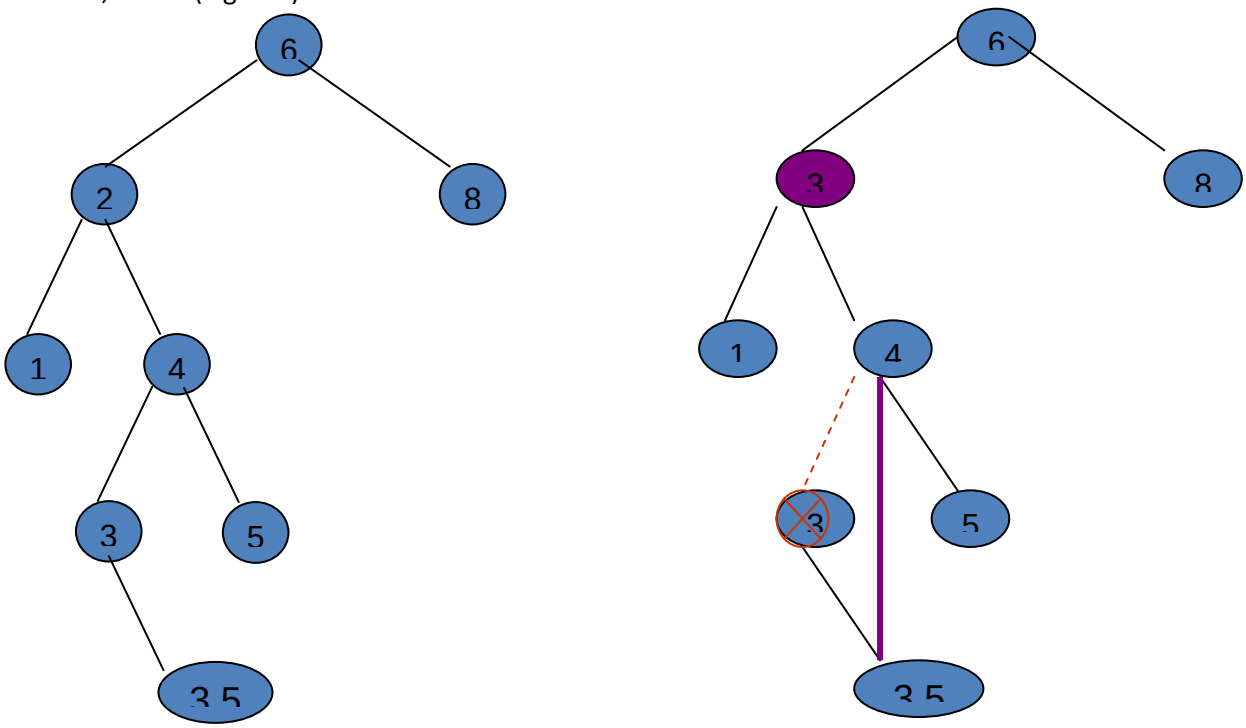


Figura5: (Ejemplo2)

Varios ejemplos de eliminación de un nodo (Figura6):

a. Si el nodo a borrar no tiene hijos, simplemente se libera el espacio que ocupa
b. Si el nodo a borrar tiene un solo hijo, se añade como hijo de su padre (p), sustituyendo la posición ocupada por el nodo borrado.
c. Si el nodo a borrar tiene los dos hijos se siguen los siguientes pasos:
i. Se busca el máximo de la rama izquierda o **el mínimo de la rama derecha**.
ii. Se sustituye el nodo a borrar por el nodo encontrado.

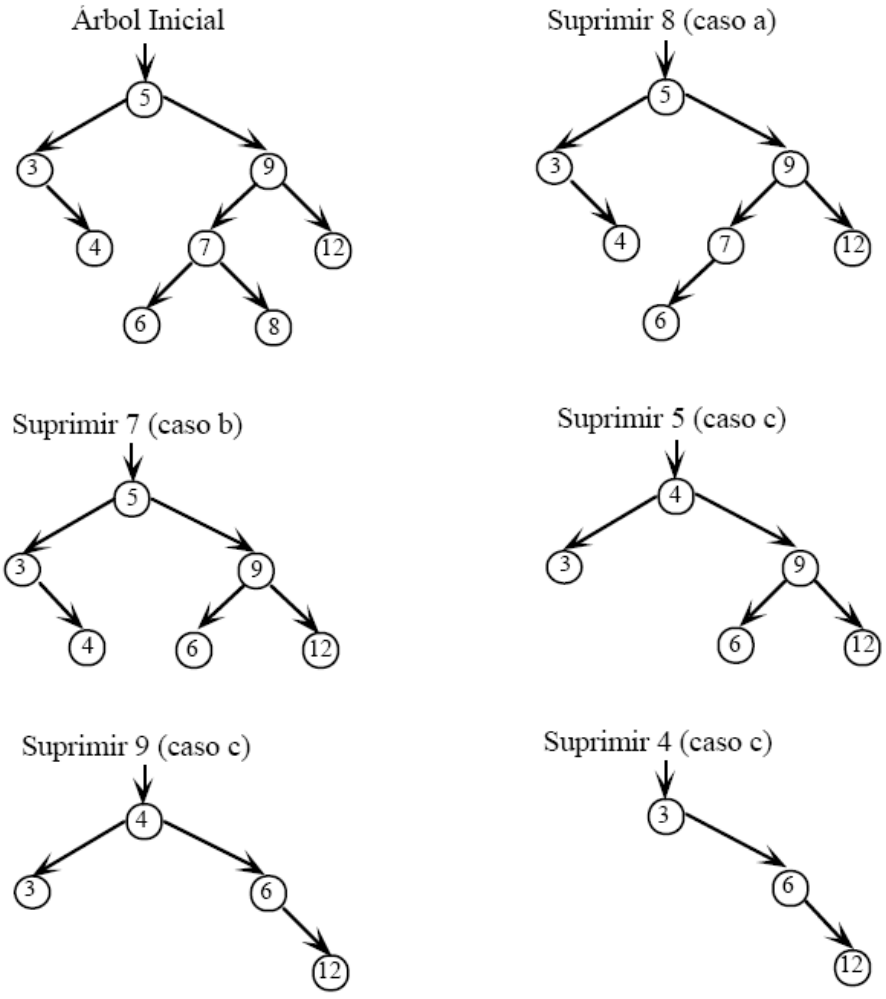


Figura6: Representación grafica

```
	protected Nodo<Integer> eliminar(Integer x, Nodo<Integer> p) {
		if (p == null)
			return null;
		if (x == p.val) {
			// Es una hoja o tiene un solo hijo: devolver el otro.
			// (No hace falta recursión cuando hay un solo hijo.)
			if (p.izq == null)
				return p.der;
			if (p.der == null)
				return p.izq;
			// Tiene dos hijos: intercambiar por el maximo de la
izquierda.
			p.val = maxVal(p.izq);
			p.izq = eliminar(p.val, p.izq);
		}
		else if (x < p.val) {
			p.izq = eliminar(x, p.izq);
		}
		else if (x > p.val) {
			p.der = eliminar(x, p.der);
		}
		return p;
	}

	private int maxVal(Nodo<Integer> p) {
		while (p.der != null) {
			p = p.der;
		}
		return p.val;
	}
```

Ejercicio1:

Cuál será el orden de complejidad de insertar un nodo para el peor caso sí:

a) El Árbol no está balanceado

b) El Árbol esta balanceado

Ejercicio2:

Cuál será el orden de complejidad de buscar un nodo para el peor caso sí:

a) El Árbol no está balanceado

b) El Árbol esta balanceado

¿Cuál es la mejora respecto del AB?

Ejercicio3:

Implementar el método boolean irep() que es verdadero si un AB es ABB
Ayuda: Utilizar acumuladores booleanos

```
bool irep (Nodo n)
booleean ret = true  // Si es n es null, cumple ABB

if n != null
…

return ret
```