Programación II
Árboles Binarios de Búsqueda

2C 2023 TN – Ing. Elizabeth Barrera

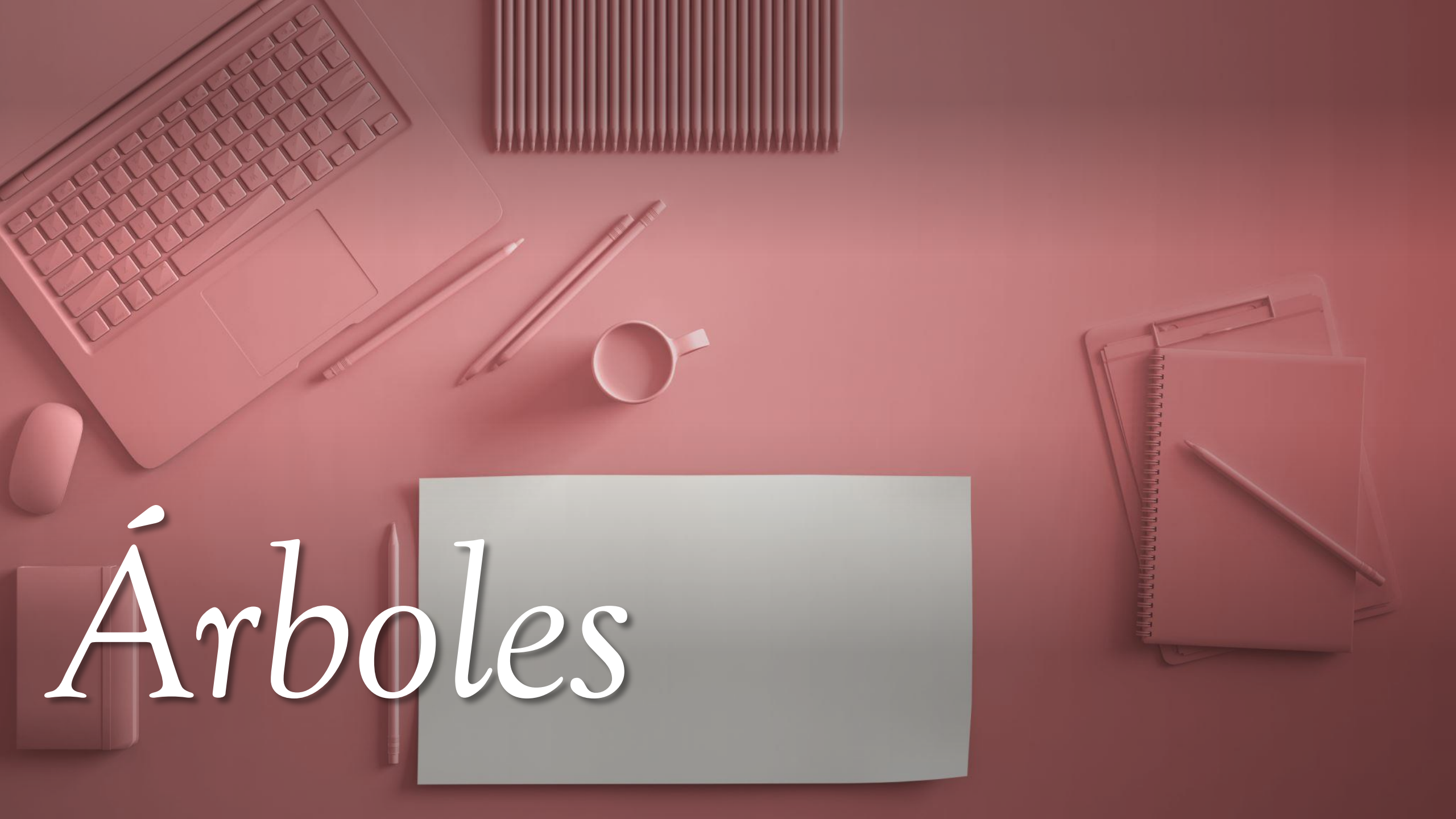
Temas

- TDA
- Árboles - Árboles binarios
- Recursividad
- ABB
- Especificación
- Ejemplos - Recorridos
- Implementación recursiva

TDA

- *Es una abstracción, ignoramos algunos detalles y nos concentramos en los que nos interesan.*
- *A la definición del TDA la llamamos especificación y a la forma de llevar a cabo lo definido lo denominamos implementación.*

Recordar que:
Existen siempre 2 visiones diferentes en el TDA: usuario e implementador.
Son separadas, y una oculta a la otra.


Árboles

Árboles

Las estructuras de datos que hemos visto hasta ahora son lineales (los datos están organizados en secuencia uno a continuación de otro). Un árbol es una ***estructura no lineal jerárquica****.*

Un árbol está compuesto de ***nodos*** *que almacenan información y* ***aristas*** *que conectan unos nodos con otros.*

Árboles

- *Cada nodo está en un **nivel**, o profundidad, bien determinado (la distancia a la raíz). La **raíz** del árbol es el único nodo situado en el nivel superior.*
- *La raíz es el **punto de entrada** a la estructura, pudiendo desde ahí seguir un **camino único** hasta cualquiera de los otros nodos.*
- *Un nodo cualquiera es el origen de un **subárbol**, cuya raíz es el nodo mismo.*

Árboles

- *Los nodos situados en los niveles inferiores son los **hijos** de los nodos ubicados en el nivel superior inmediato. Recíprocamente están los nodos **padres**. Todo nodo tiene uno y sólo un padre, excepto la raíz que no tiene padre.*
- *Un nodo sin hijos se denomina **hoja**.*

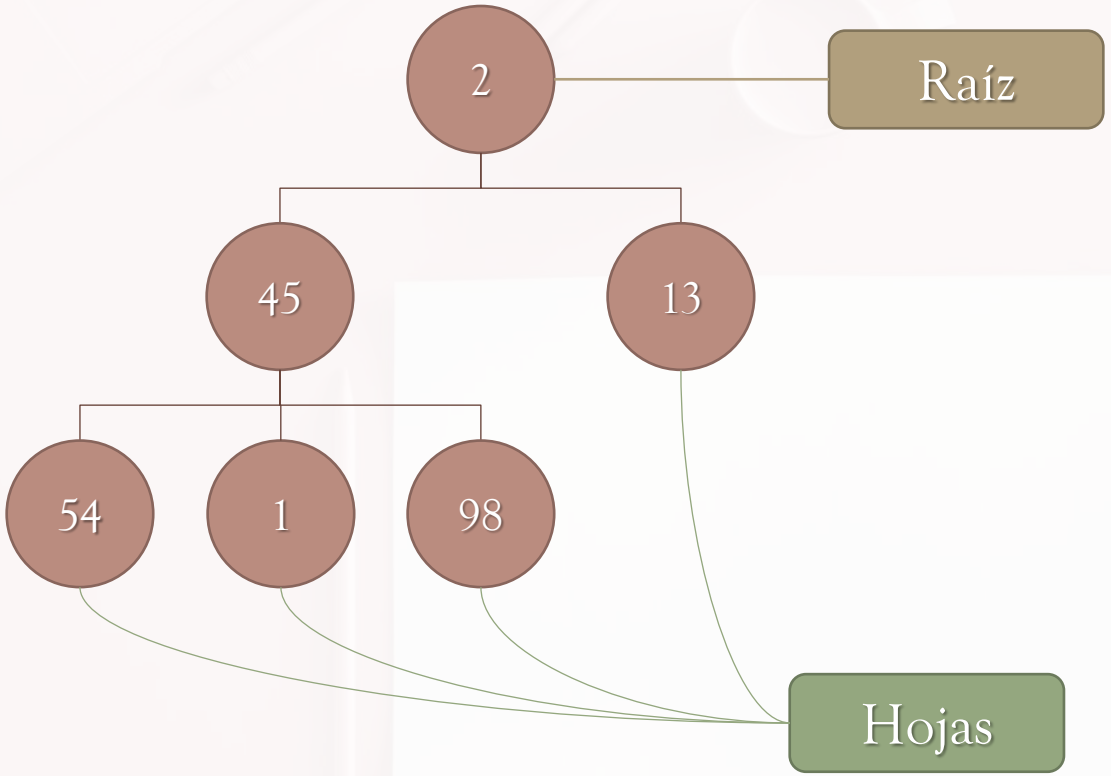
Árboles - algunas definiciones

- La ***longitud de un camino*** es la cantidad de aristas atravesadas.
- La ***altura de un árbol*** es la longitud del máximo camino que une la raíz con una hoja.
- Un árbol es ***balanceado*** o equilibrado si todas las hojas están en el mismo nivel o a lo sumo hay un nivel de diferencia entre dos hojas cualesquiera.

Árboles - algunas definiciones

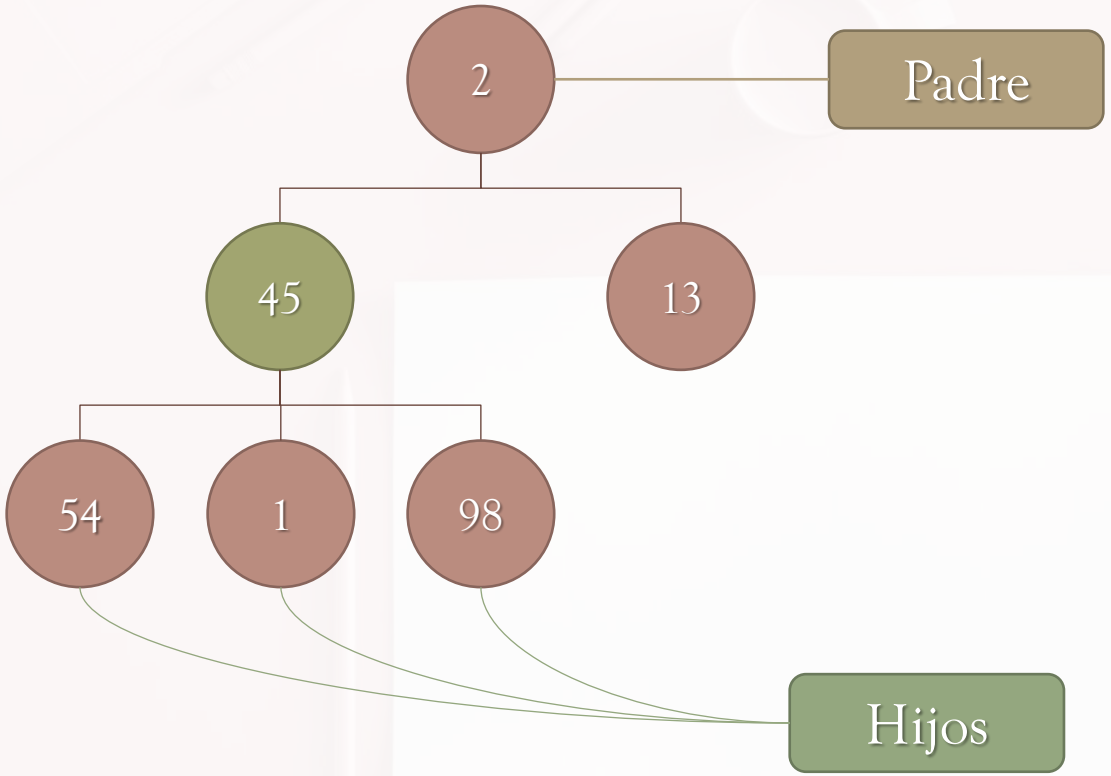
- Los ***ascendientes*** de un nodo son los nodos ubicados por encima de este en el camino hacia la raíz. Los ***descendientes*** son todos los nodos situados por debajo de este en el subárbol cuya raíz es él.
- Todos los nodos que no son ni la raíz ni las hojas se denominan ***nodos internos*** o intermedios.

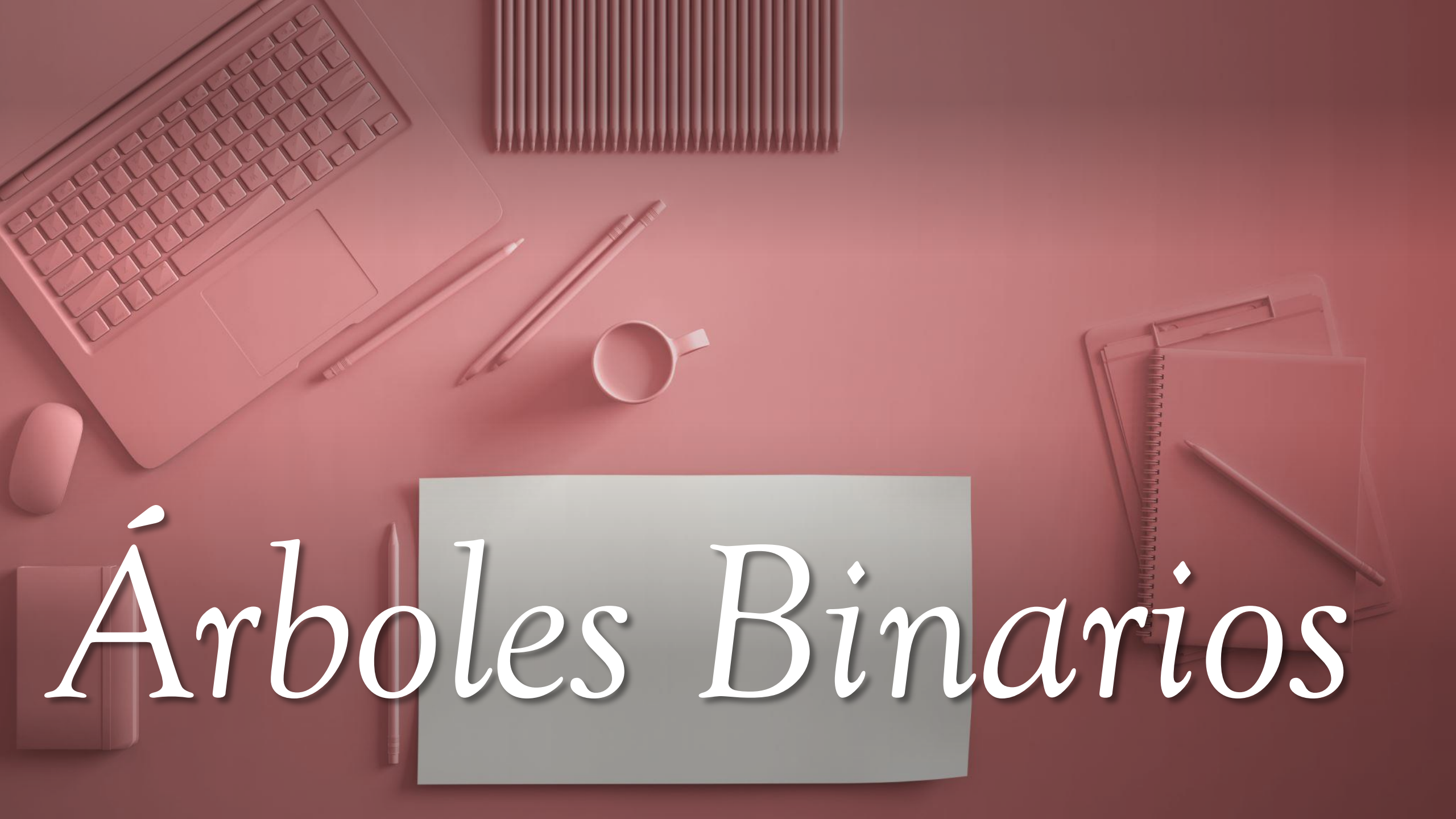
Árboles



Subárbol

Árboles





Árboles Binarios

Árboles binarios

Un caso particular de árboles son los árboles binarios, en los cuales cada nodo puede tener ***a lo sumo dos hijos*** *(o sea cero, uno o dos).*

Los hijos de un nodo de un árbol binario son llamados ***hijo izquierdo*** *e* ***hijo derecho****, subárbol izquierdo y subárbol derecho respectivamente.*


Recursividad

Recursividad

Un método recursivo es aquel que se define en función de sí mismo. Siempre dentro de ese método va a haber una llamada o ***invocación a sí mismo****. Para evitar un método que nunca acabe su ejecución, se debe buscar una forma de cortar esta recursión.*

Recursividad

Así, en todo método recursivo vamos a tener:

- *un **caso base** o condición de corte, en donde se finalice la recursividad,*
- *un **paso recursivo**, o invocación al método en sí mismo.*

Recursividad

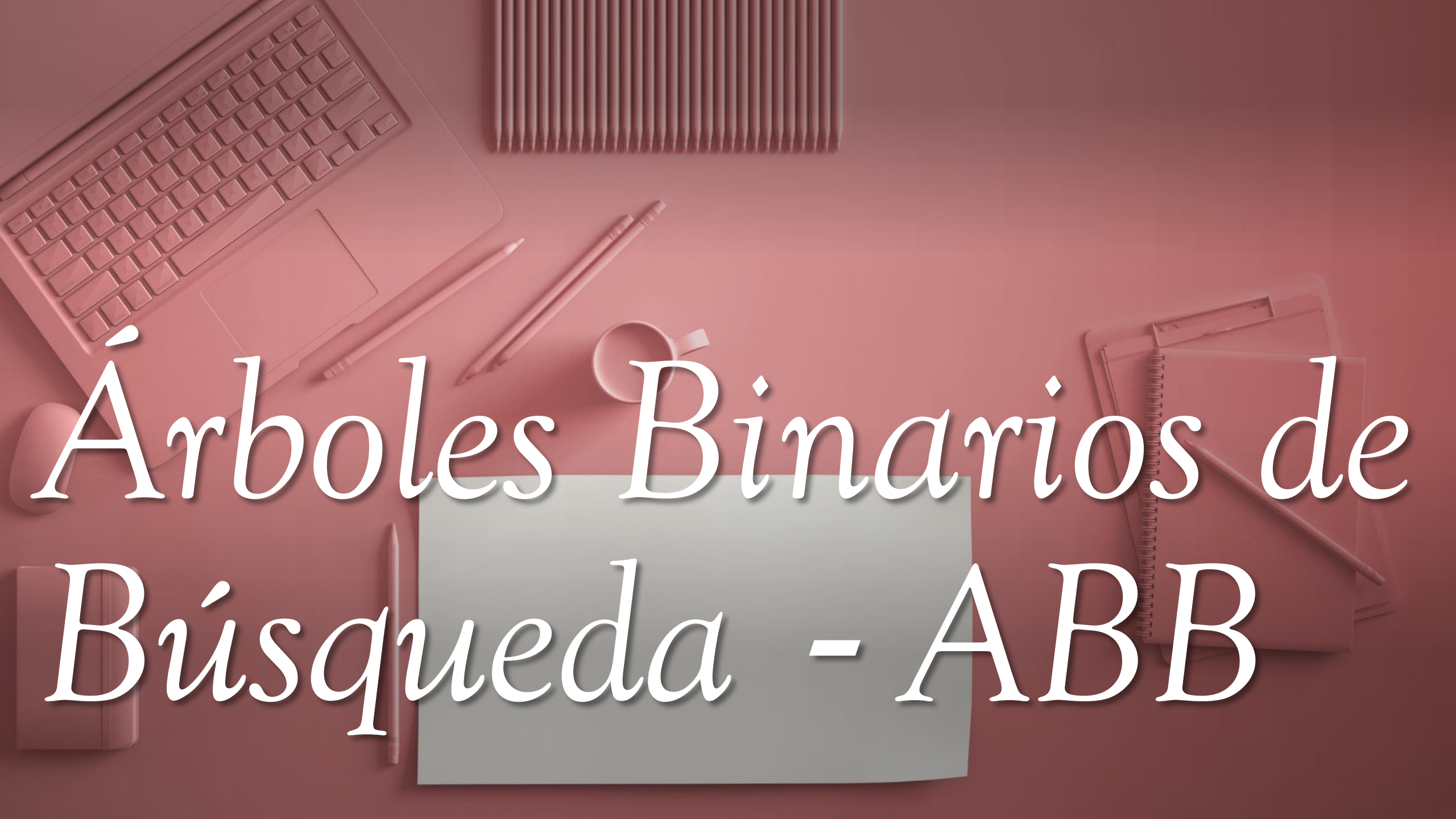
Hay que tener cuidado en que la ejecución del algoritmo alcance siempre la condición de corte, entonces cuando se invoca al método en forma recursiva se debe hacer con los parámetros que tiendan a alcanzar en un punto al caso base que corte la recursividad.

***Ejemplo**: método que calcula el factorial de un número.*

Ejemplo

```
public int factorial (int n) {
        if (n == 0)
                return 1;  //caso base
        else
                return n * factorial (n-1);
                //paso recursivo (se decrementa n
                //hasta llegar al caso base)
}
```

Decremento n



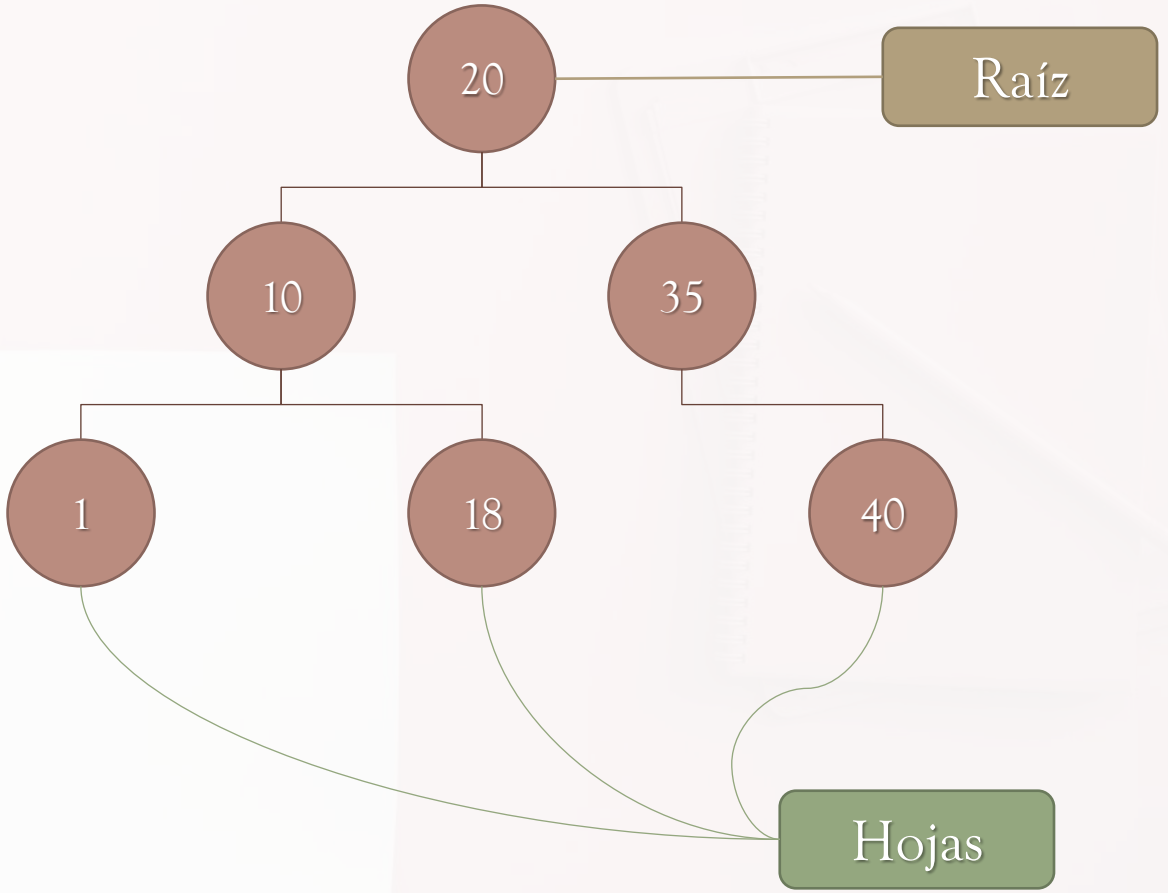
Árboles Binarios de Búsqueda - ABB

ABB

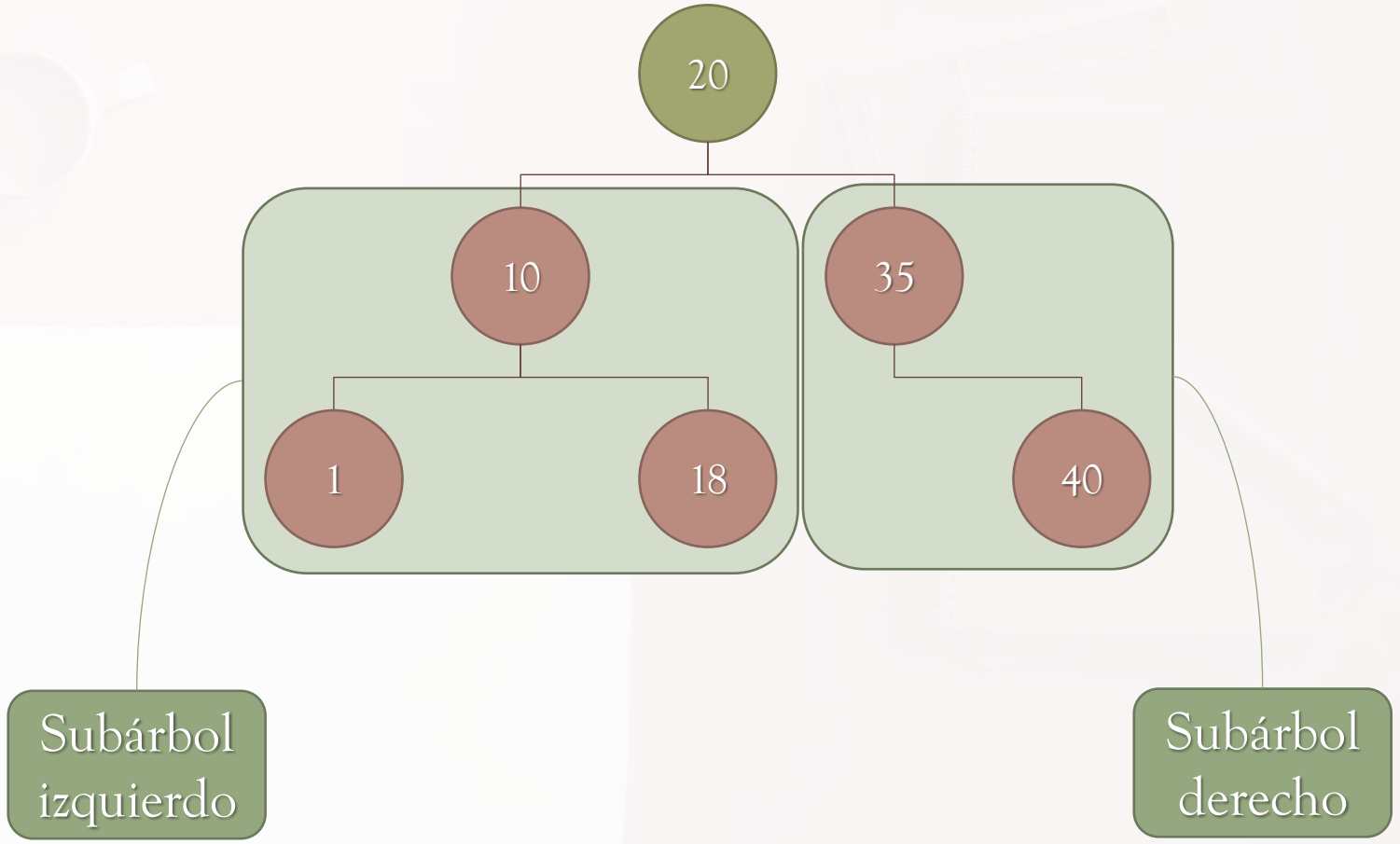
Los ***árboles binarios de búsqueda*** (ABB) son árboles binarios que cumplen con las siguientes condiciones:

- no tiene elementos ***repetidos***;
- para cualquier nodo, todos los elementos de su subárbol izquierdo son ***menores*** y todos los elementos de su subárbol derecho son ***mayores***.

ABB



ABB



Especificación - Operaciones

- ***inicializarABB***: *inicializa el árbol.*
- ***agregarElem***: *agrega el elemento dado manteniendo la propiedad de ABB (se supone que el árbol está inicializado).*
- ***eliminarElem***: *elimina el elemento dado manteniendo la propiedad de ABB (se supone que el árbol está inicializado).*

Recordar que:
*Las **precondiciones**, son condiciones que deben cumplirse antes de la ejecución de la operación.*

Especificación - Operaciones

- ***raiz***: *recupera el valor de la raíz de un ABB (se supone que el árbol está inicializado y no está vacío).*
- ***hijoIzq***: *devuelve el subárbol izquierdo (se supone que el árbol está inicializado y no está vacío).*
- ***hijoDer***: *devuelve el subárbol derecho (se supone que el árbol está inicializado y no está vacío).*
- ***arbolVacio***: *indica si el árbol está vacío o no (se supone que el árbol está inicializado).*

Recordar que:
*Las **precondiciones**, son condiciones que deben cumplirse antes de la ejecución de la operación.*

Especificación - Interfaz

```
public interface ABBTDA {
    void inicializarArbol( );
    void agregarElem(int x); //árbol inicializado
    void eliminarElem(int x); //árbol inicializado
    int raiz( ); //árbol inicializado y no vacío
    ABBTDA hijoIzq( ); //árbol inicializado y no vacío
    ABBTDA hijoDer( ); //árbol inicializado y no vacío
    boolean arbolVacio( ); //árbol inicializado
}
```

Aclaraciones

- ***Un ABB es vacío, o es un nodo raíz y dos árboles que dependen de esa raíz (el subárbol izquierdo y el subárbol derecho).***
- *Un ABB con un solo nodo será un nodo y dos subárboles vacíos de hijos.*

Uso

Uso - Ejemplos

- Contar la cantidad de nodos de un ABB.

Uso - Recorridos

- Un ABB puede ser recorrido de tres maneras: ***pre-order***, ***in-order*** y ***post-order***.
- Los tres recorridos son recursivos por definición.
- Visitar un nodo puede significar mostrar o cualquier otra acción que se desee realizar.

Recorridos

- ***Pre-order***: *se visita primero el nodo raíz, luego el subárbol izquierdo en pre-order y finalmente el subárbol derecho en pre-order.*

- ***In-order***: *se visita primero in-order el subárbol izquierdo, luego el nodo raíz y por último in-order el subárbol derecho. Este recorrido es el que muestra los elementos* ***ordenados*** *de menor a mayor.*

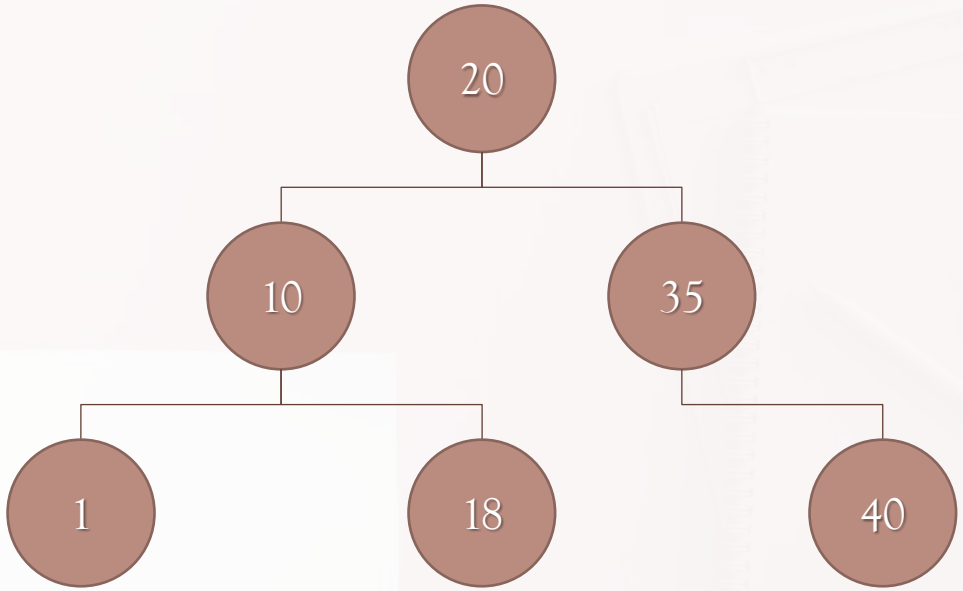
- ***Post-order***: *se visita primero el subárbol izquierdo en post-order, luego el subárbol derecho en post-order y finalmente el nodo raíz.*

```
public void preOrder (ABBTDA a) {
    if (!a.arbolVacio( )) {
        System.out.println(a.raiz( ));
        preOrder(a.hijoIzq( ));
        preOrder(a.hijoDer( ));
    }
}
```

```
public void inOrder (ABBTDA a) {
    if (!a.arbolVacio( )) {
        inOrder(a.hijoIzq( ));
        System.out.println(a.raiz( ));
        inOrder(a.hijoDer( ));
    }
}
```

```
public void postOrder (ABBTDA a) {
    if (!a.arbolVacio( )) {
        postOrder(a.hijoIzq( ));
        postOrder(a.hijoDer( ));
        System.out.println(a.raiz( ));
    }
}
```

ABB



preOrder:
20 - 10 - 1 - 18 - 35 - 40

inOrder:
1 - 10 - 18 - 20 - 35 - 40

postOrder:
1 - 18 - 10 - 40 - 35 - 20

Implementación

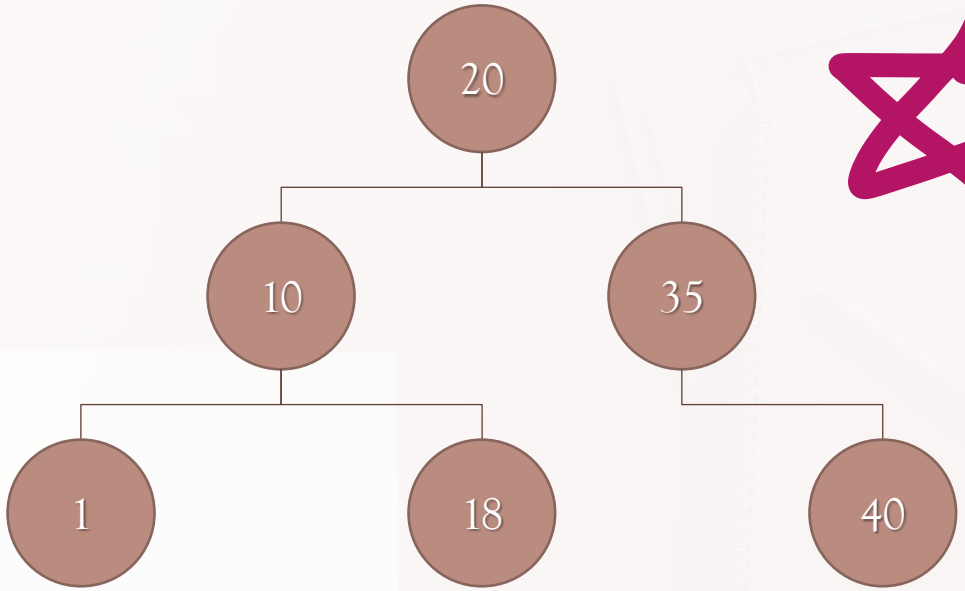
Implementación recursiva

- Se define un nodo, como un valor y sus dos subárboles.
- Se ***busca*** un elemento hasta encontrarlo (éxito) o caer en un elemento vacío (fracaso), tomando el nodo raíz y:
  - si el nodo raíz es el elemento buscado, lo encontramos;
  - si el nodo raíz es mayor que el elemento buscado, seguimos la buscando por el subárbol izquierdo;
  - si el nodo raíz es menor que el elemento buscado, seguimos la buscando por el subárbol derecho.

NodoABB

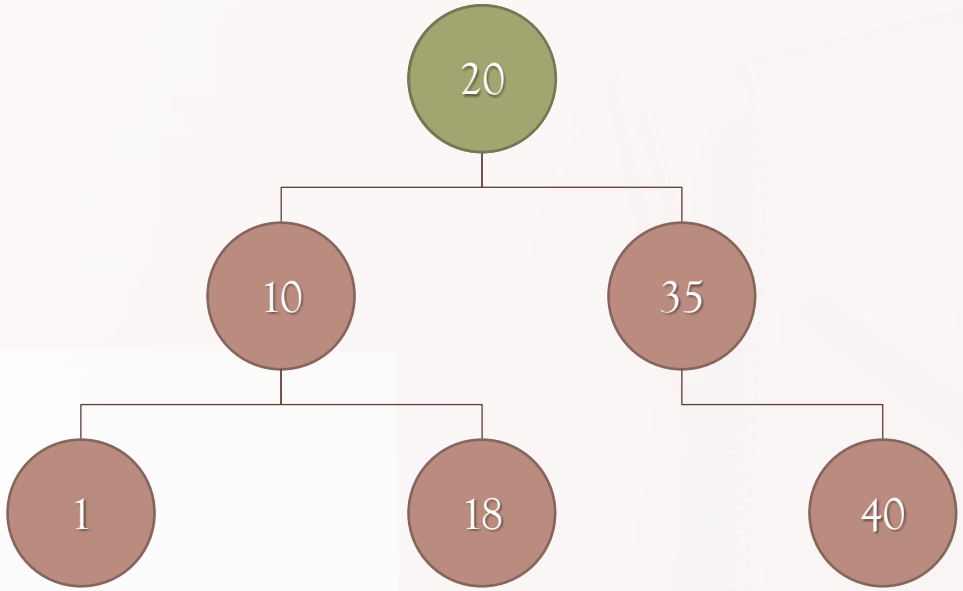
```
public class NodoABB {
    int info;
    ABBTDA hijoIzq;
    ABBTDA hijoDer;
}
```

Ejemplo búsqueda



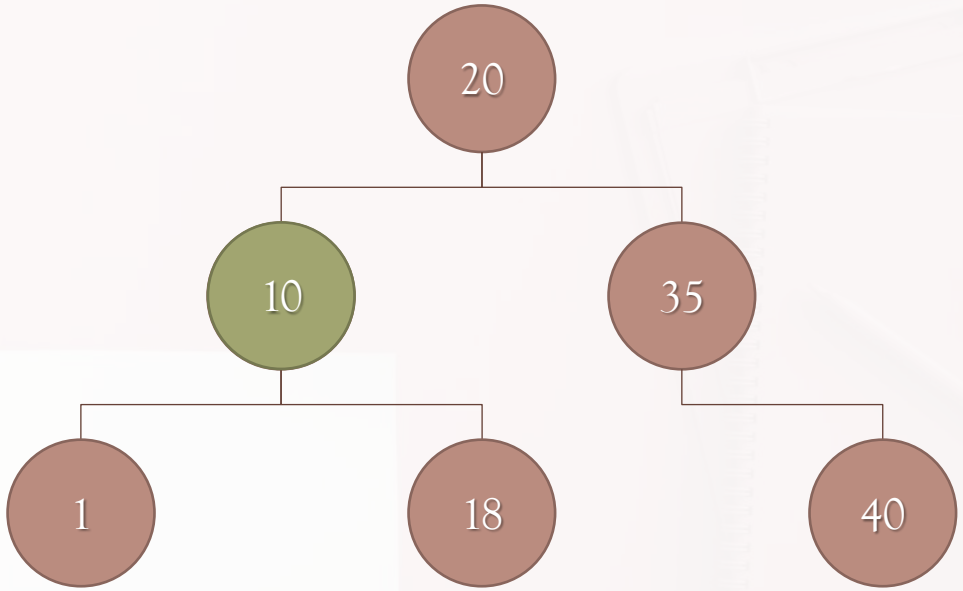
Buscar:
• 18

Ejemplo búsqueda



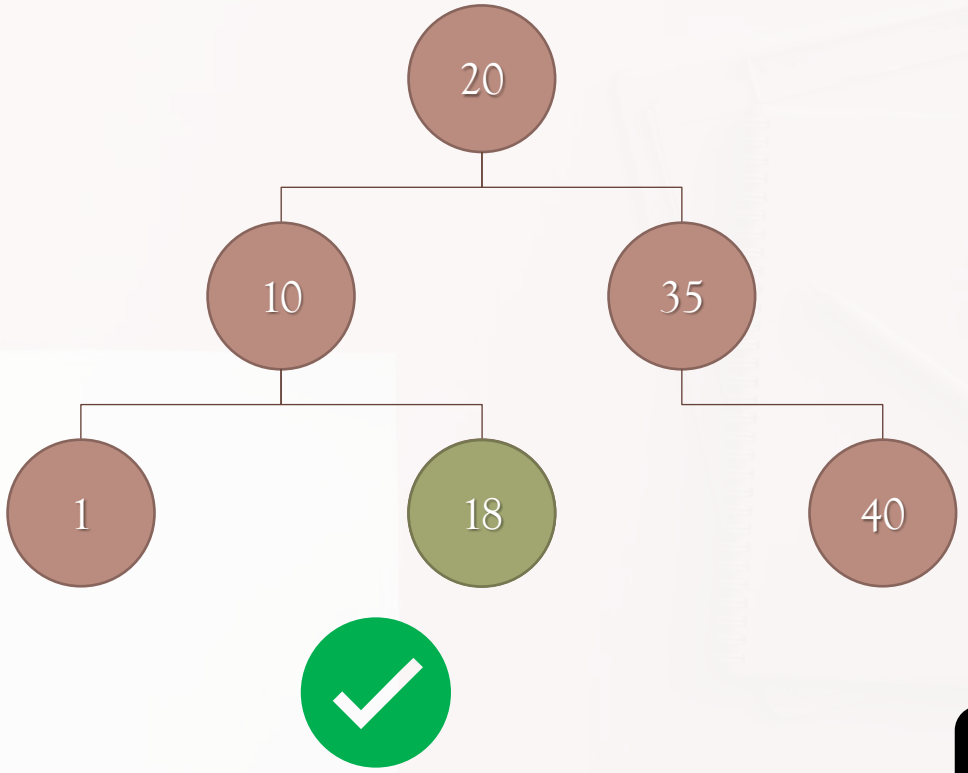
Buscar:
- 18

Ejemplo búsqueda



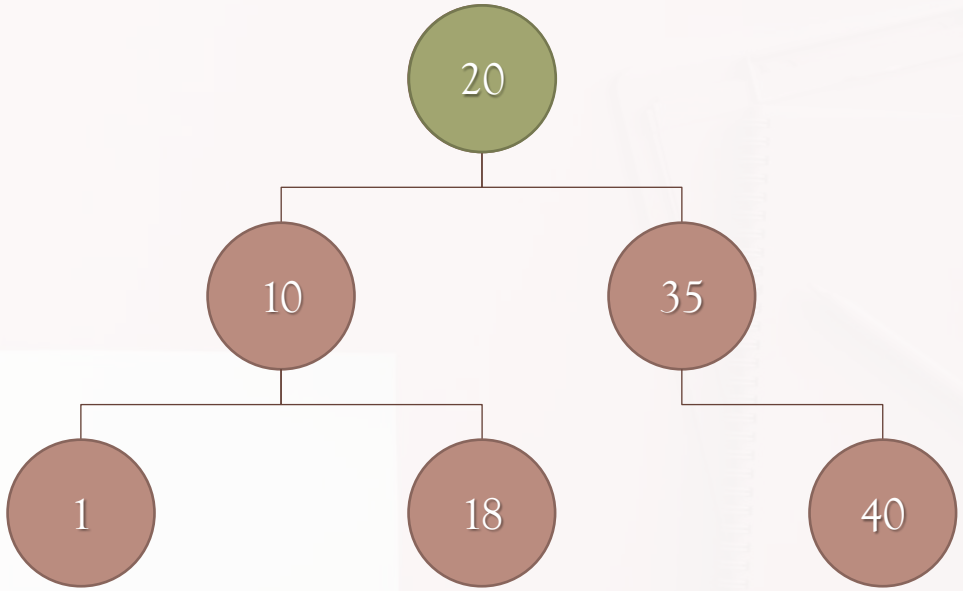
Buscar:
- 18

Ejemplo búsqueda



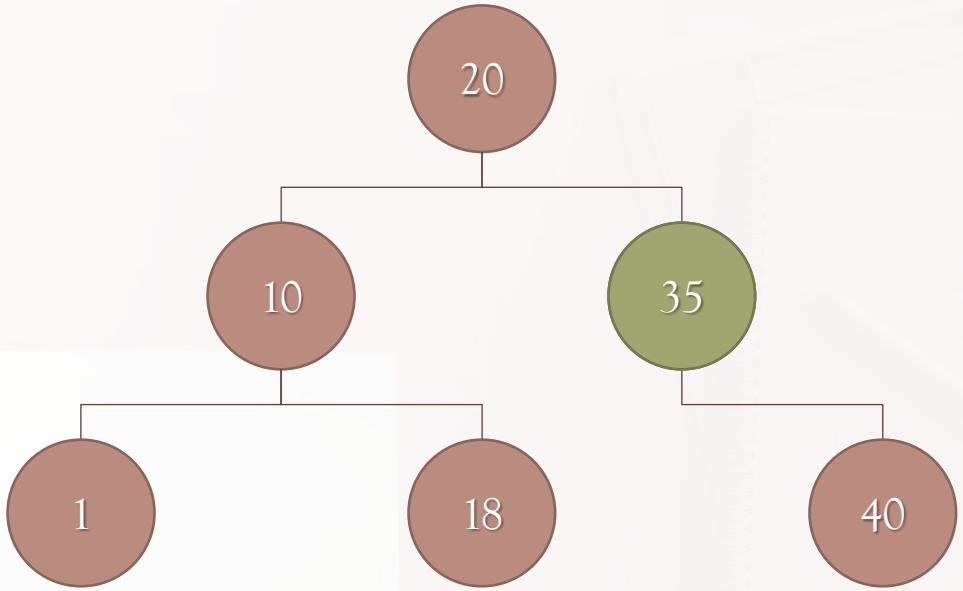
Buscar:
• 18

Ejemplo búsqueda



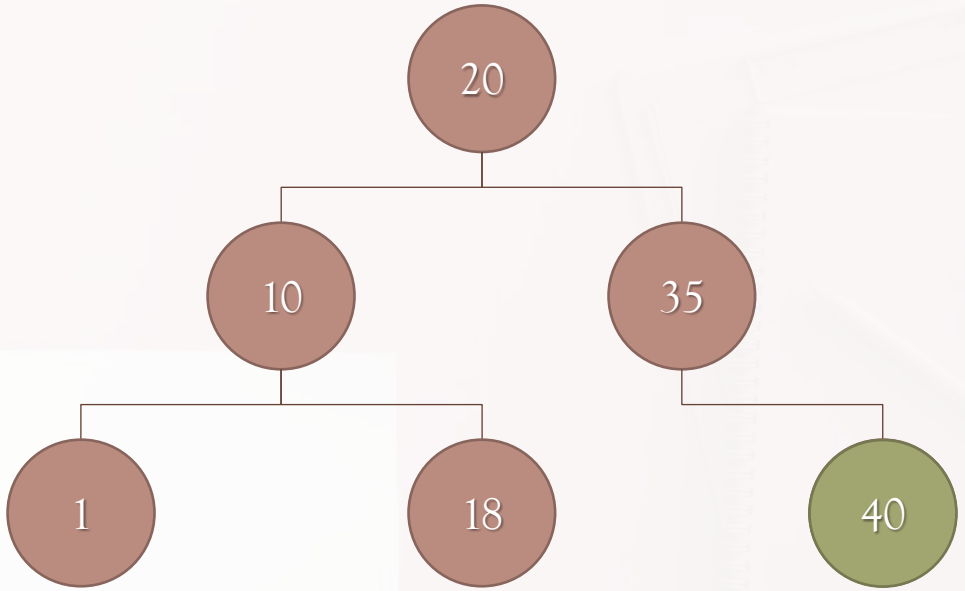
Buscar:
- 37

Ejemplo búsqueda



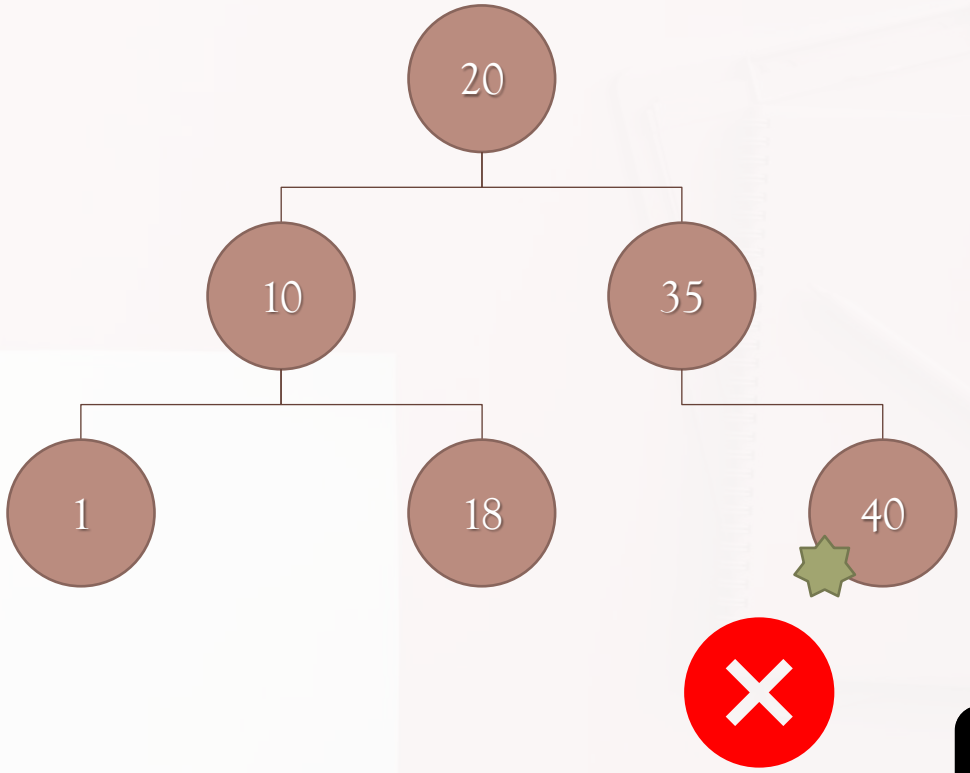
Buscar:
• 37

Ejemplo búsqueda



Buscar:
• 37

Ejemplo búsqueda



Buscar:
• 37

Implementación recursiva

- Para **agregar** un elemento, lo hacemos en la primera posición libre.
- Se busca el elemento, al no encontrarlo, lo colocamos en la posición en que nos "caeríamos" del árbol.
- Siempre se agregan los nuevos elementos como hojas.

Implementación recursiva

- Para ***eliminar*** un elemento, al encontrarlo, se lo reemplaza con el mayor elemento de su subárbol izquierdo (el mayor de los menores).
- Si no tiene subárbol izquierdo, lo reemplazamos con el menor elemento de su subárbol derecho (el menor de los mayores).
- Si tampoco tiene subárbol derecho, es una hoja y lo reemplazamos por null.

Ejemplo agregado - eliminado

Agregar:
- 20

Ejemplo agregado - eliminado

Ejemplo agregado - eliminado

20

Agregar:
- 10

Ejemplo agregado - eliminado

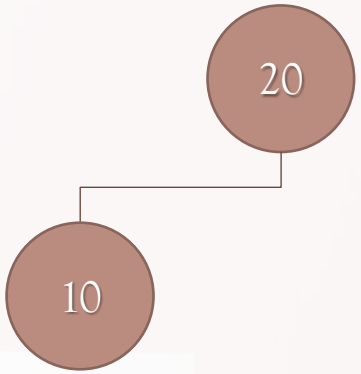
20

Agregar:
- 10

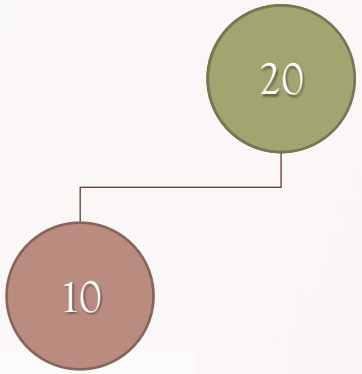
Ejemplo agregado - eliminado

Agregar:
- 10

Ejemplo agregado - eliminado

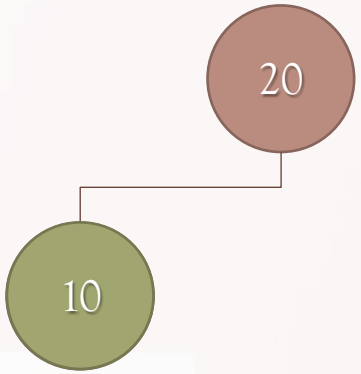


Ejemplo agregado - eliminado



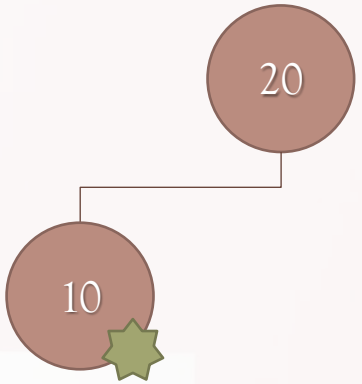
Agregar:
- 18

Ejemplo agregado - eliminado



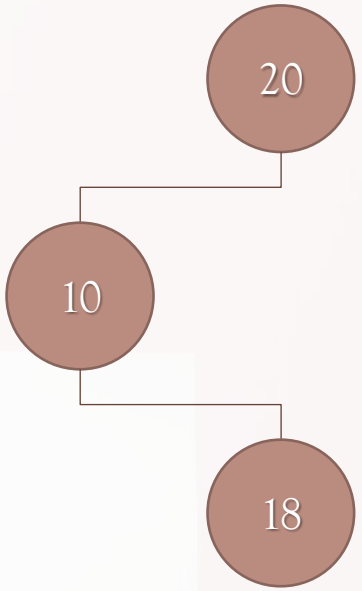
Agregar:
- 18

Ejemplo agregado - eliminado

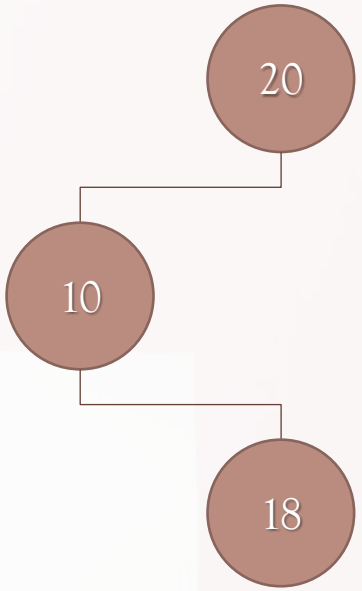


Agregar:
- 18

Ejemplo agregado - eliminado

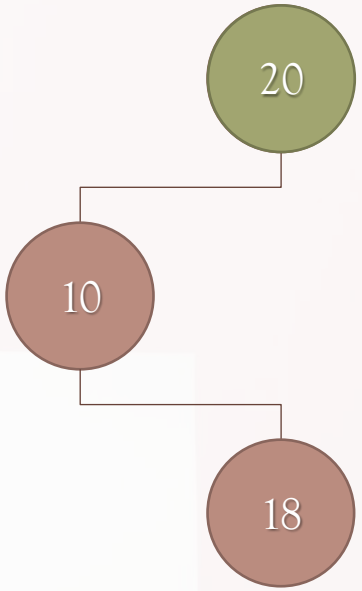


Ejemplo agregado - eliminado



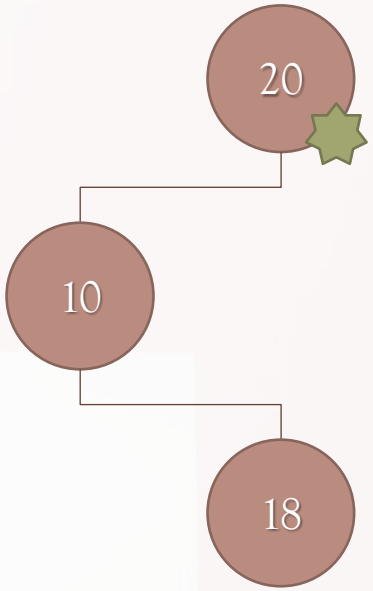
Agregar:
• 35

Ejemplo agregado - eliminado



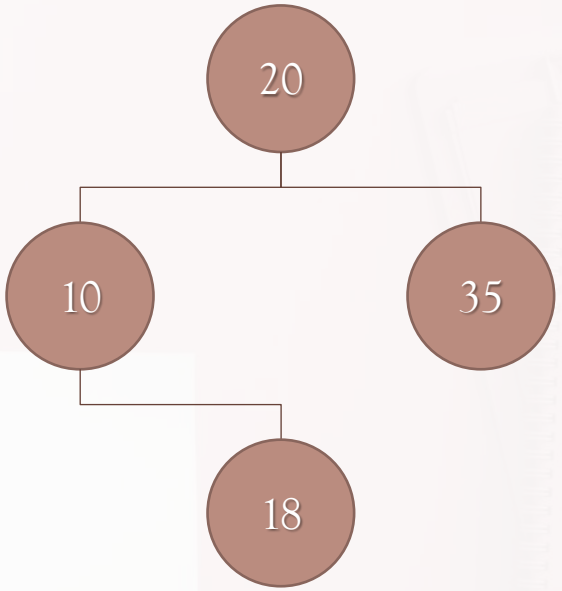
Agregar:
• 35

Ejemplo agregado - eliminado

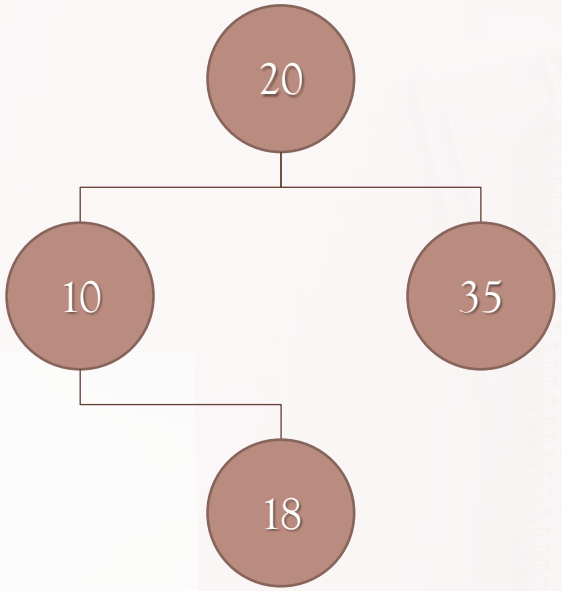


Agregar:
- 35

Ejemplo agregado - eliminado

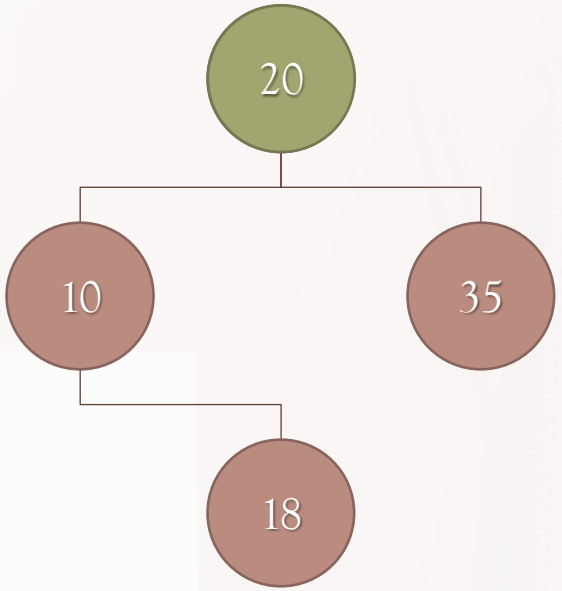


Ejemplo agregado - eliminado



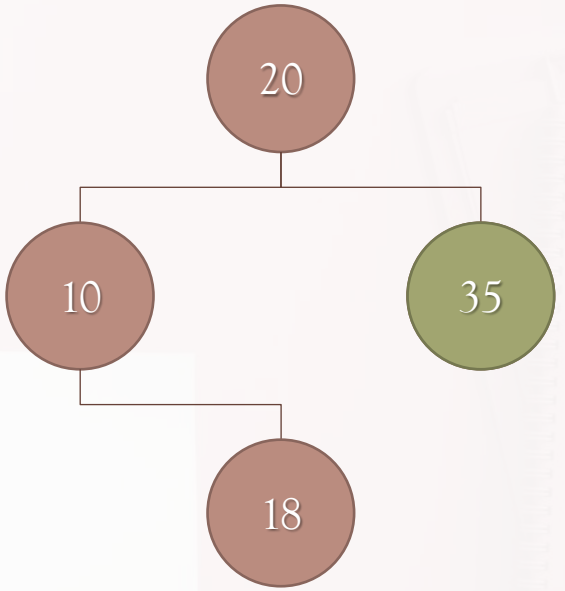
Agregar:
- 40

Ejemplo agregado - eliminado



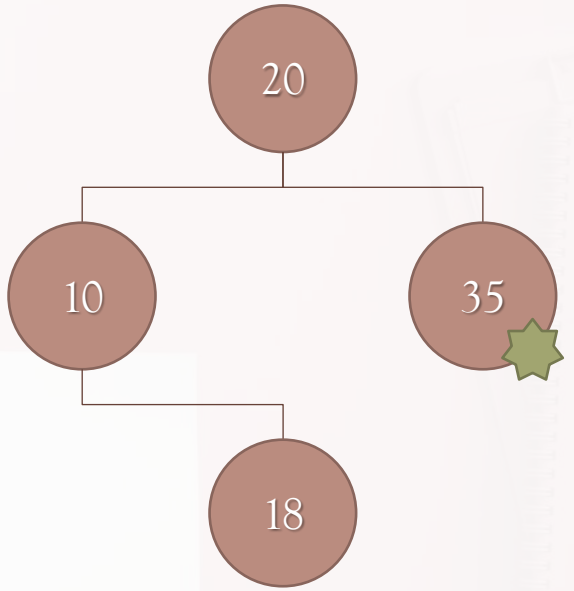
Agregar:
- 40

Ejemplo agregado - eliminado



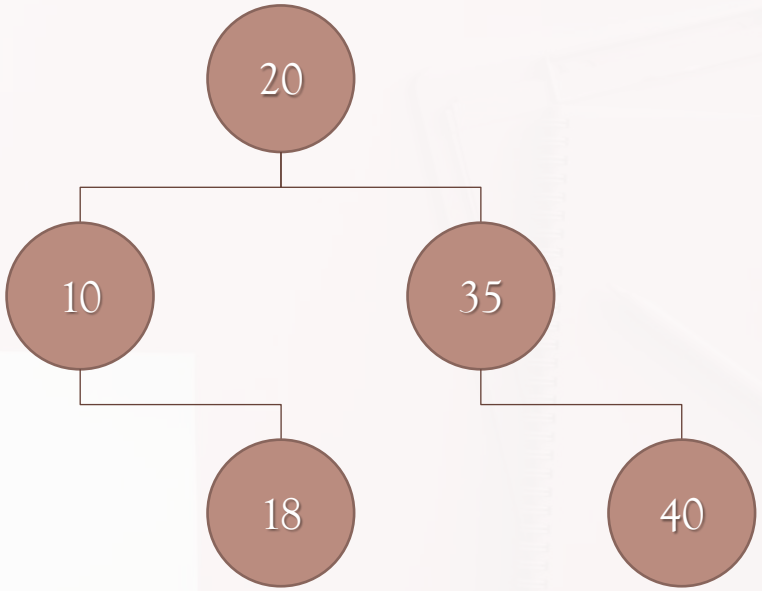
Agregar:
- 40

Ejemplo agregado - eliminado

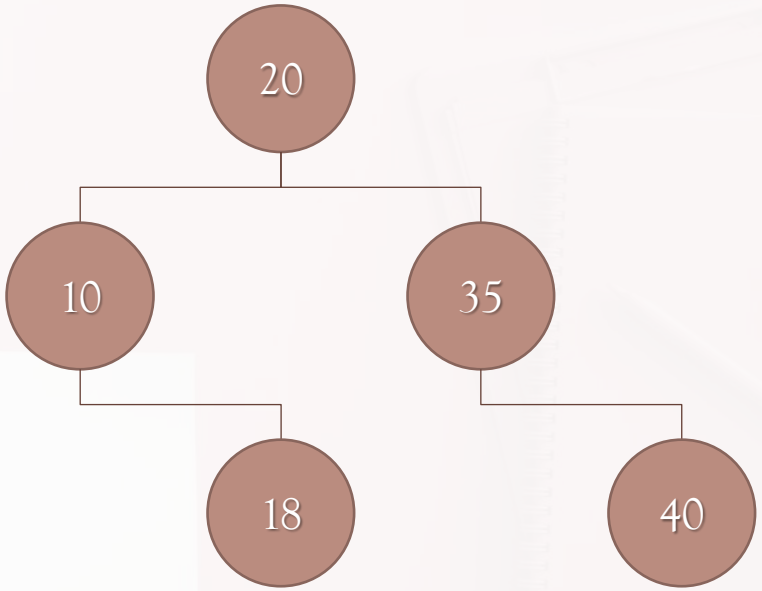


Agregar:
- 40

Ejemplo agregado - eliminado

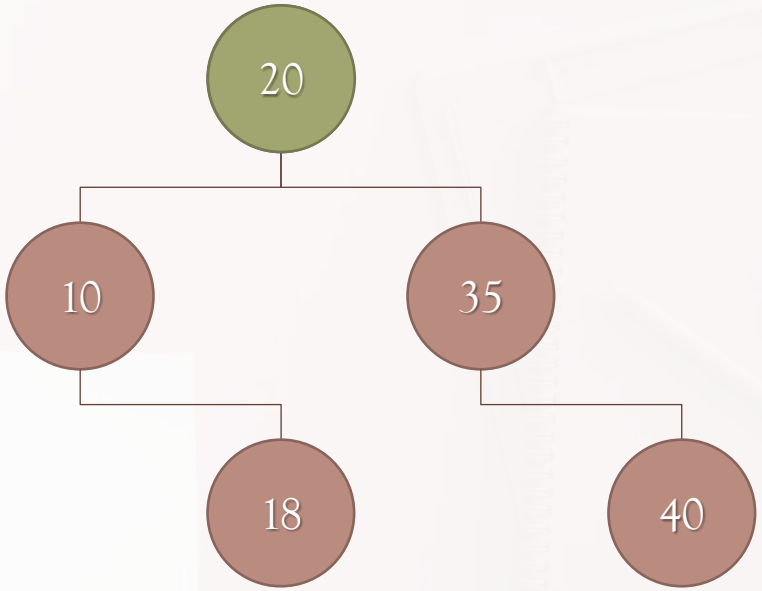


Ejemplo agregado - eliminado



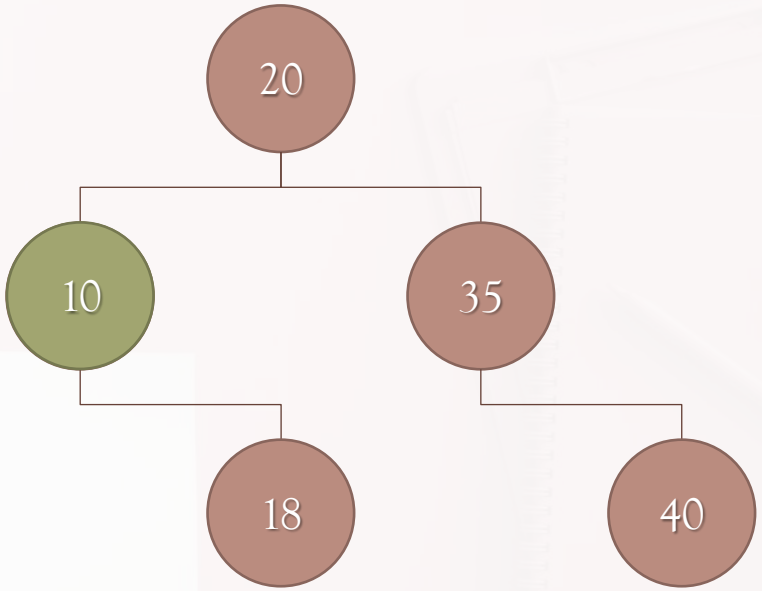
Agregar:
- 1

Ejemplo agregado - eliminado



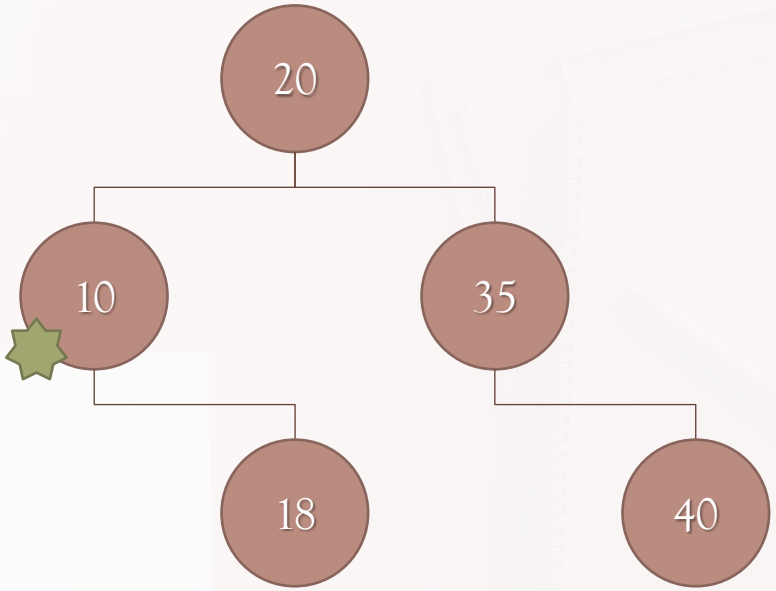
Agregar:
- 1

Ejemplo agregado - eliminado



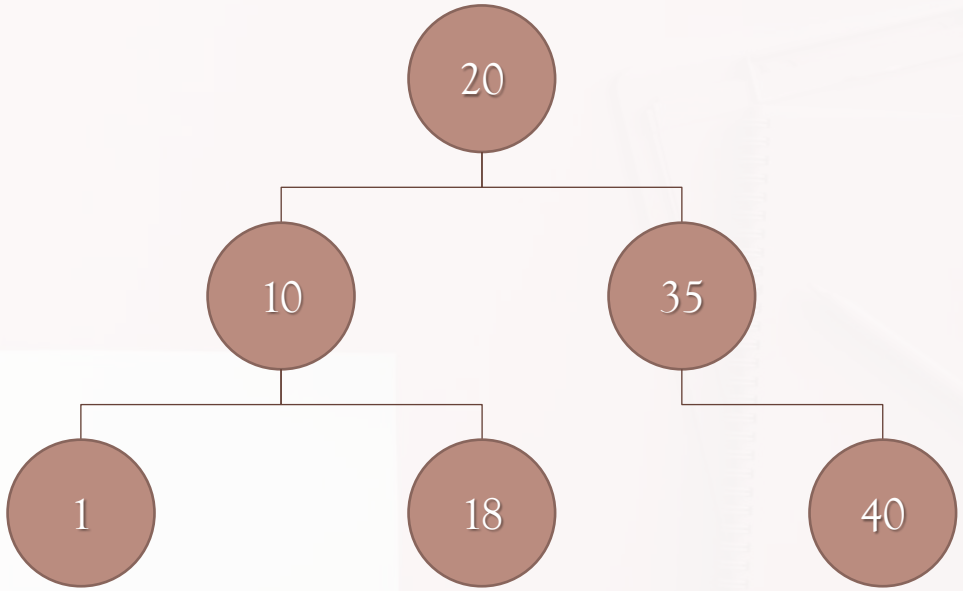
Agregar:
- 1

Ejemplo agregado - eliminado

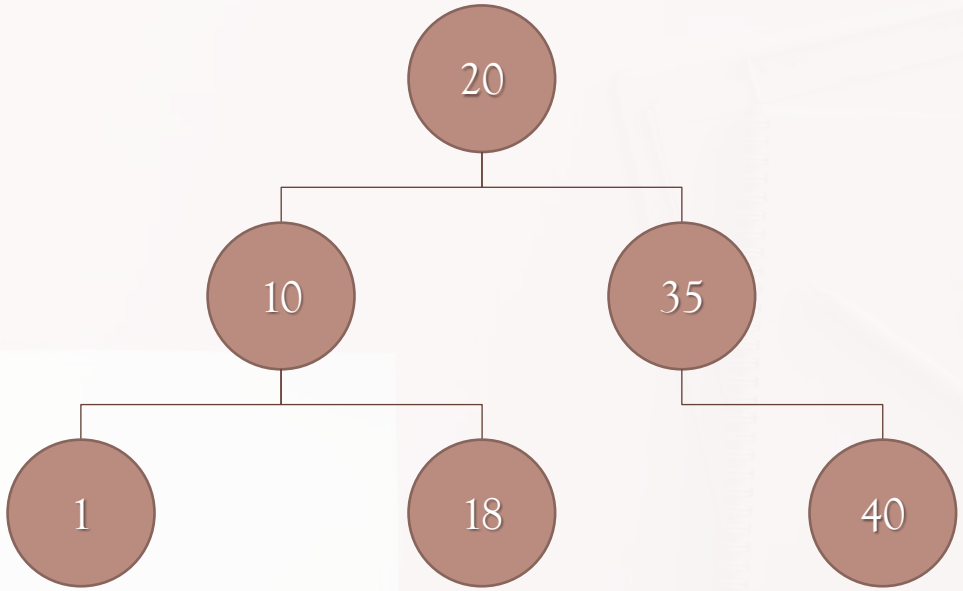


Agregar:
- 1

Ejemplo agregado - eliminado

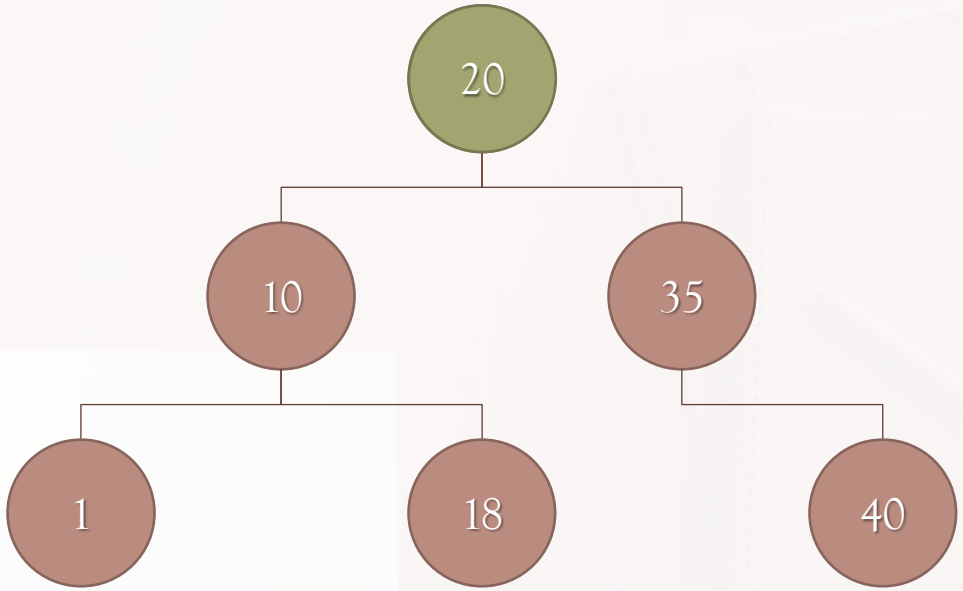


Ejemplo agregado - eliminado



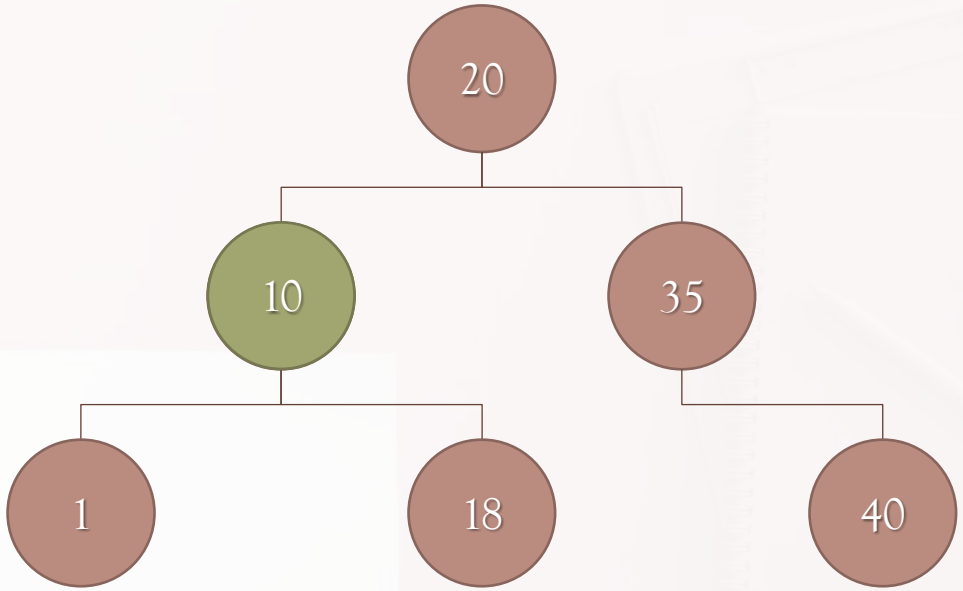
Eliminar:
- 18

Ejemplo agregado - eliminado



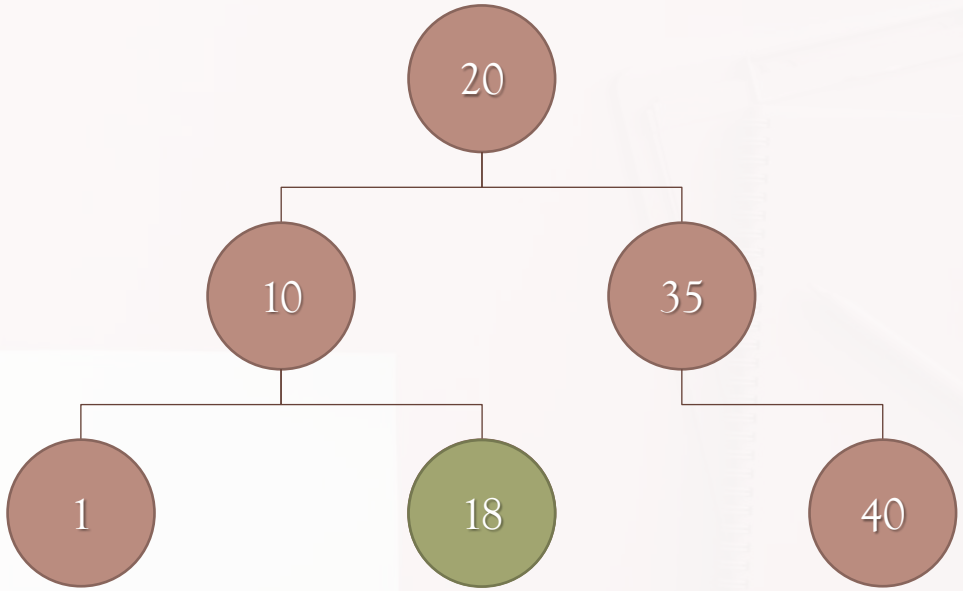
Eliminar:
- 18

Ejemplo agregado - eliminado



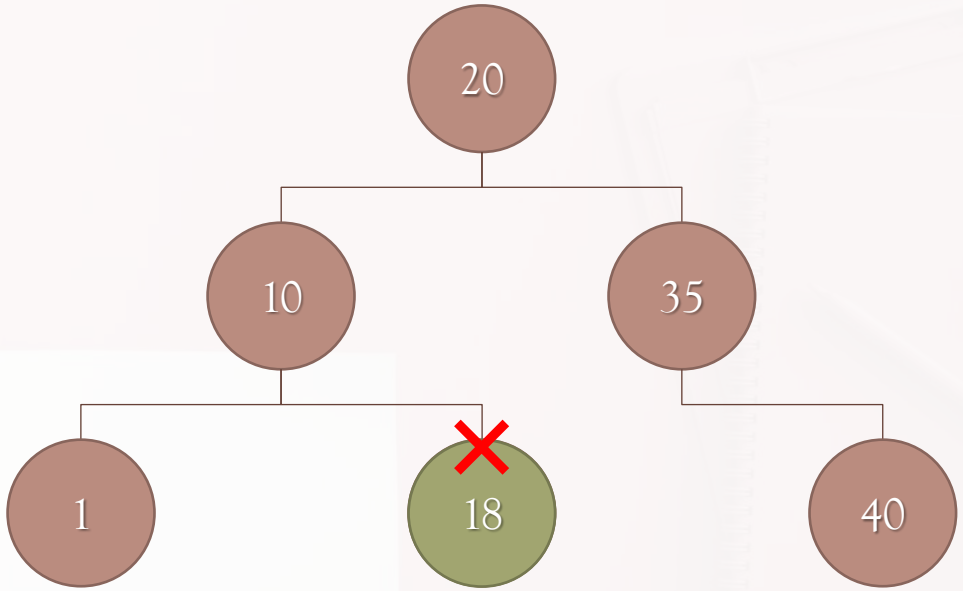
Eliminar:
- 18

Ejemplo agregado - eliminado



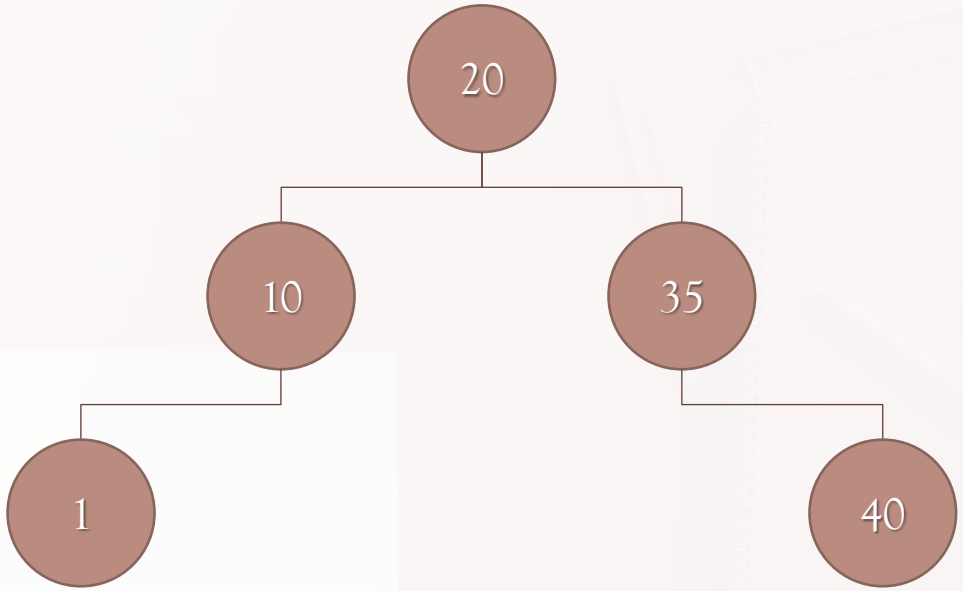
Eliminar:
- 18

Ejemplo agregado - eliminado

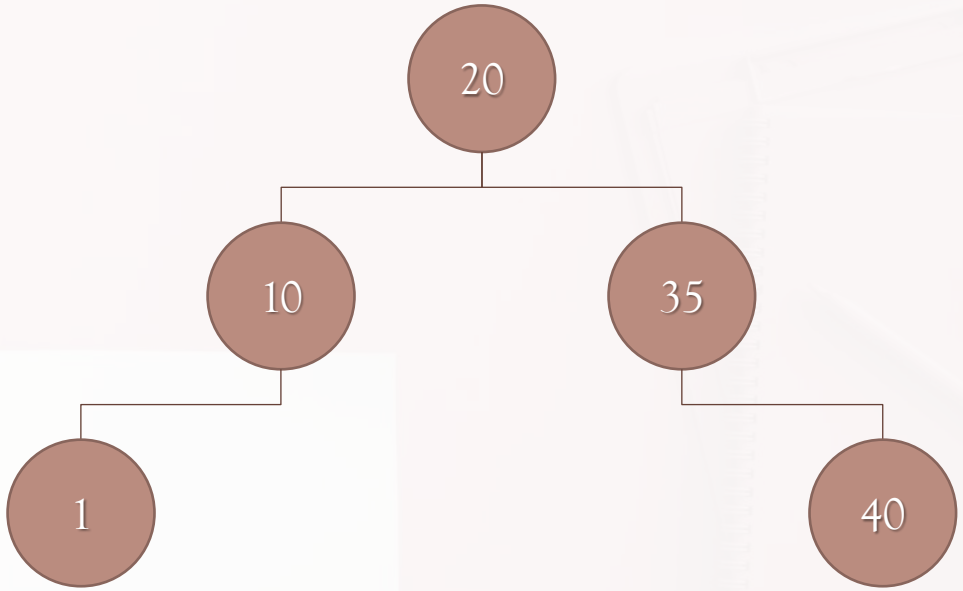


Eliminar:
- 18

Ejemplo agregado - eliminado

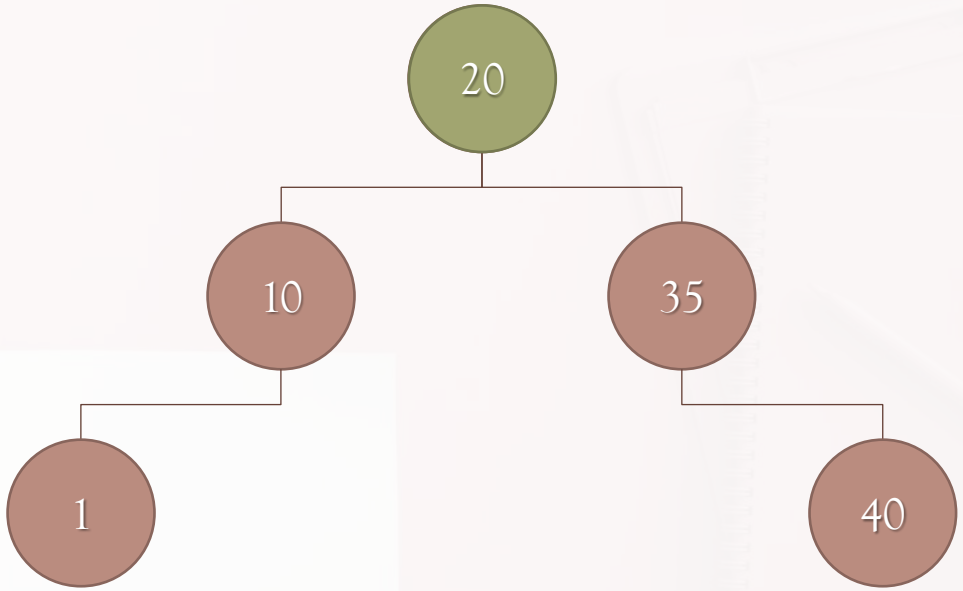


Ejemplo agregado - eliminado



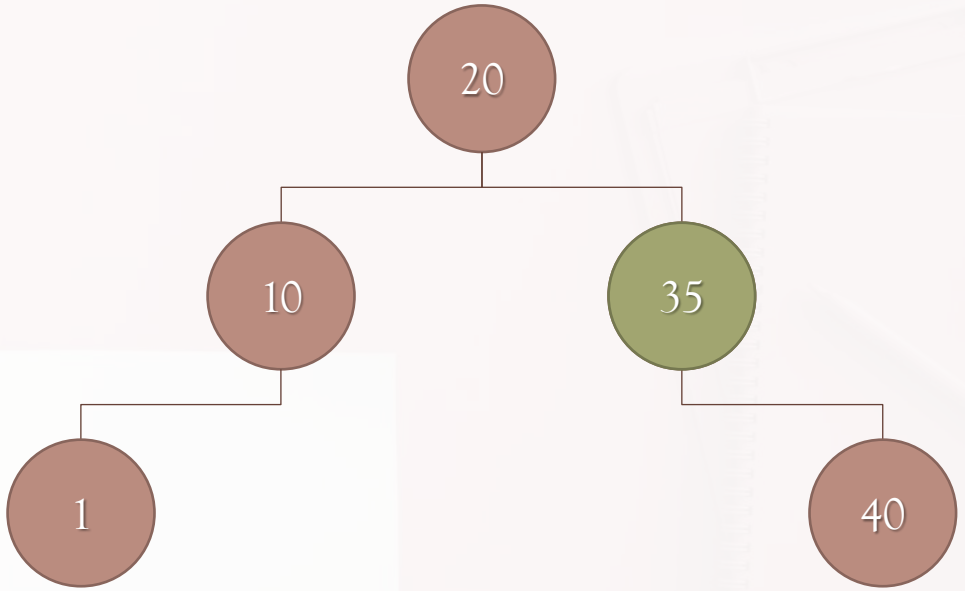
Eliminar:
- 35

Ejemplo agregado - eliminado



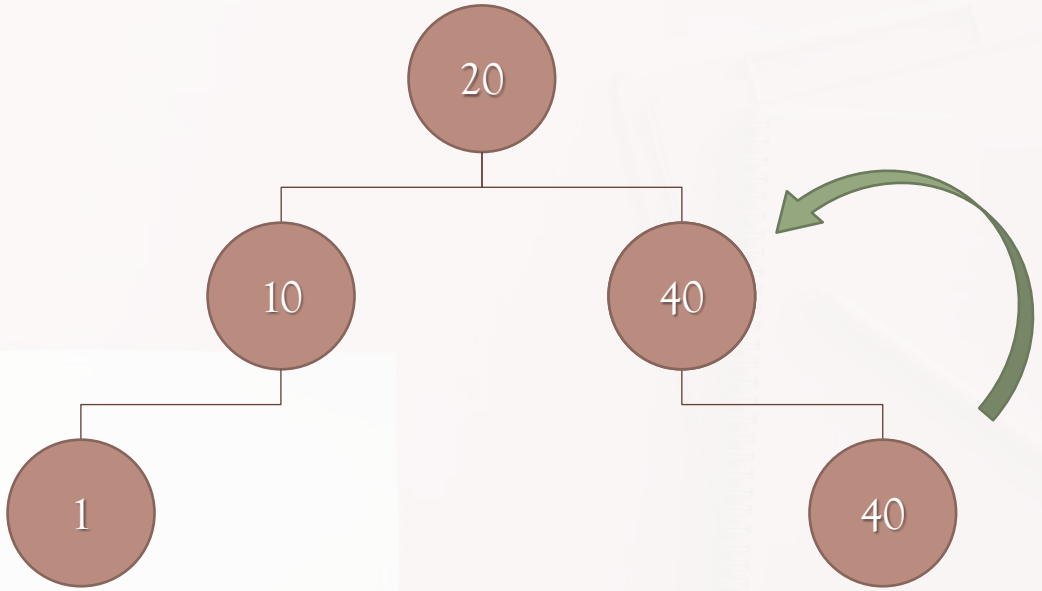
Eliminar:
- 35

Ejemplo agregado - eliminado



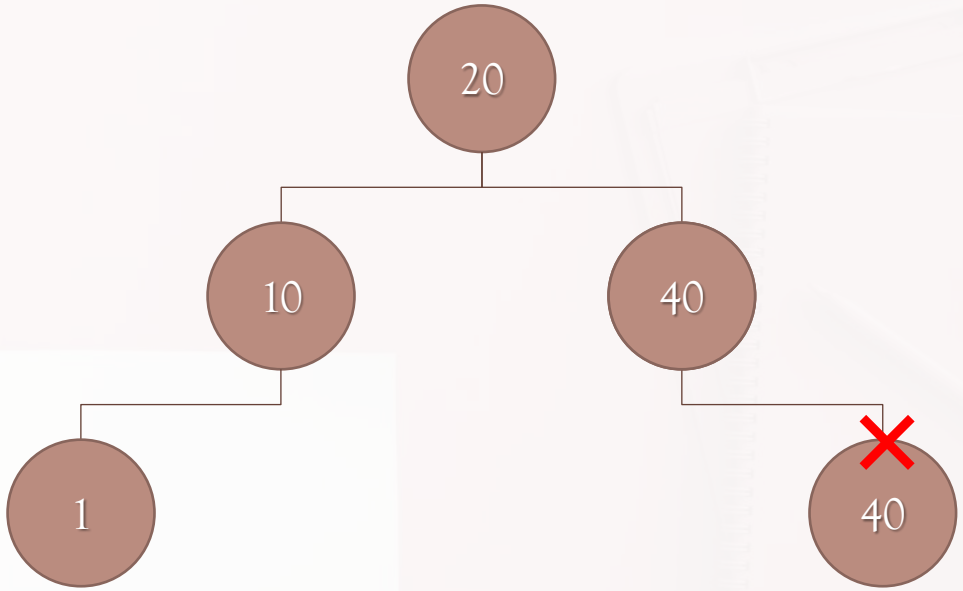
Eliminar:
- 35

Ejemplo agregado - eliminado



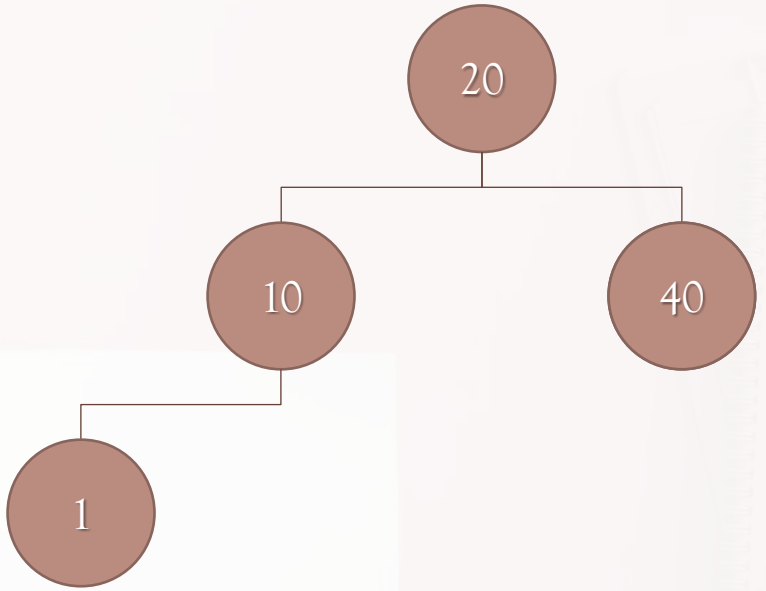
Eliminar:
- 35

Ejemplo agregado - eliminado

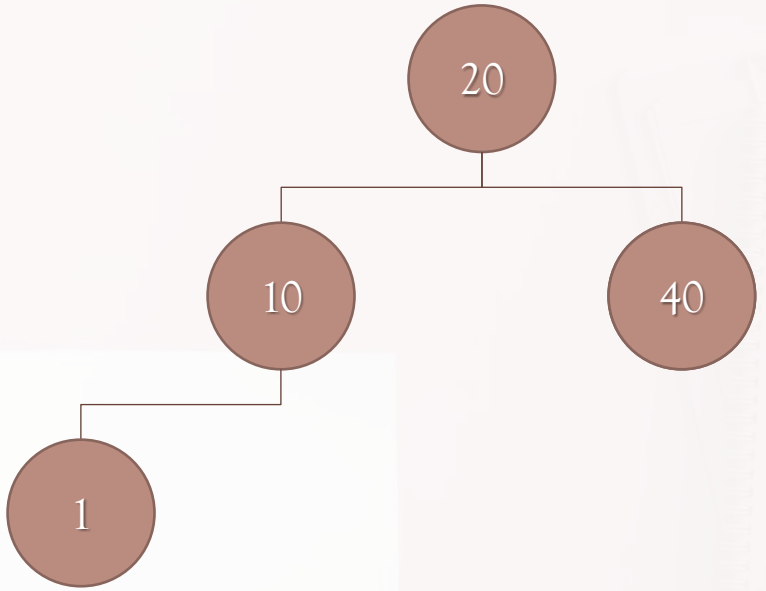


Eliminar:
- 35

Ejemplo agregado - eliminado

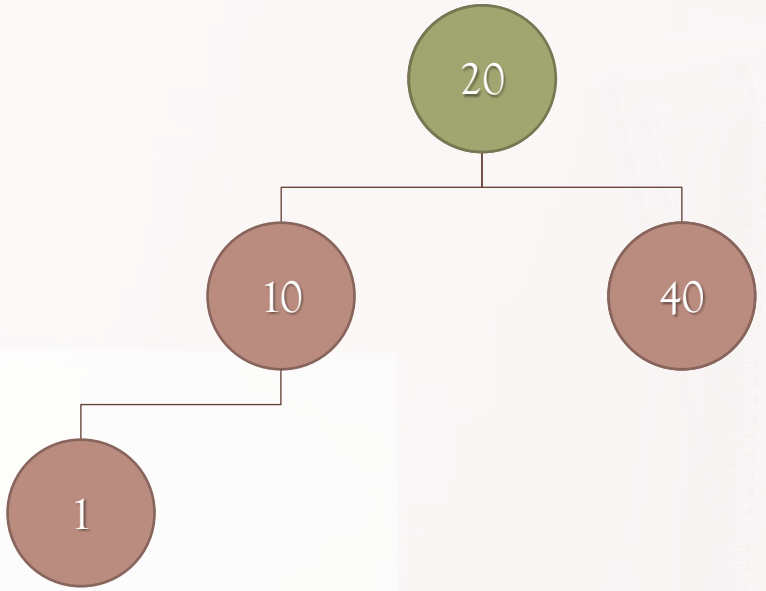


Ejemplo agregado - eliminado



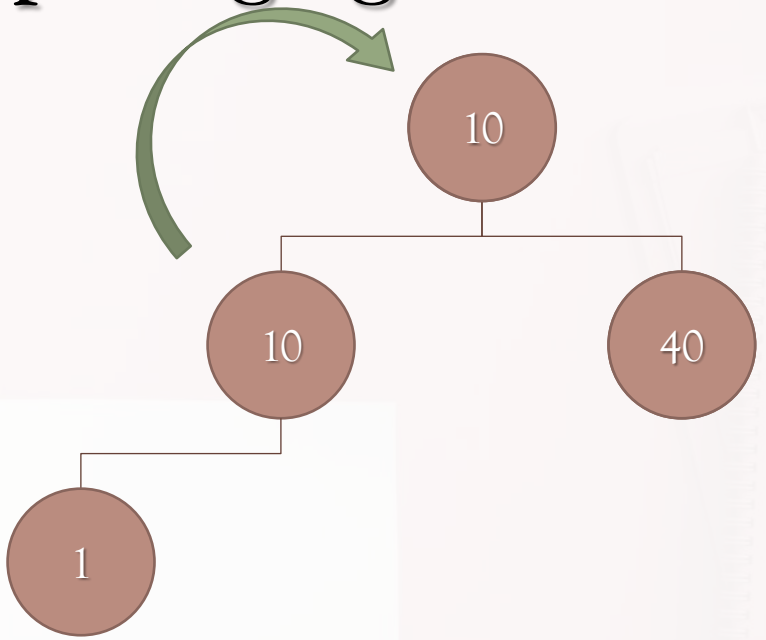
Eliminar:
- 20

Ejemplo agregado - eliminado



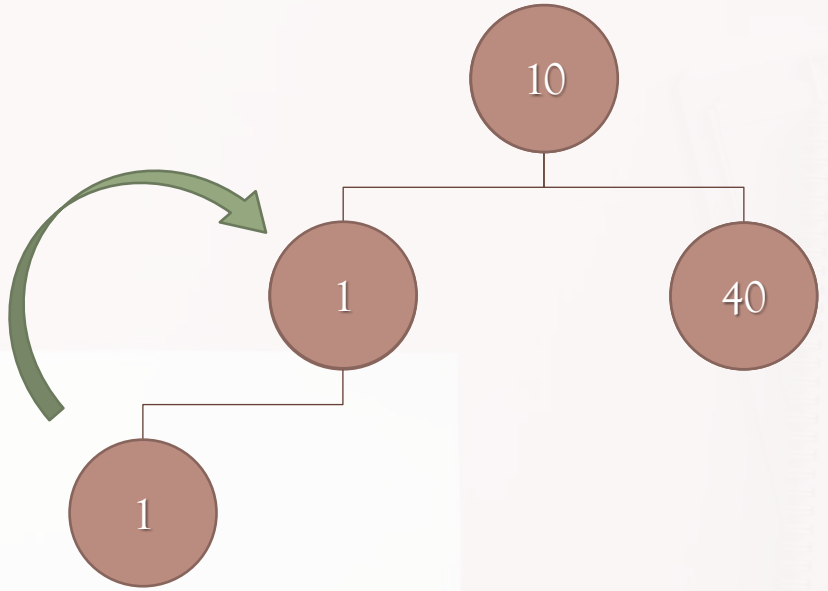
Eliminar:
- 20

Ejemplo agregado - eliminado



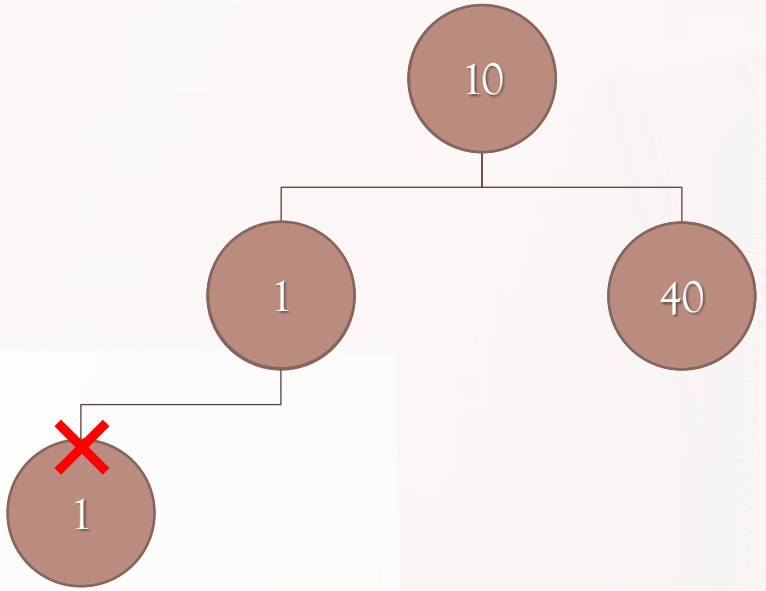
Eliminar:
- 20

Ejemplo agregado - eliminado



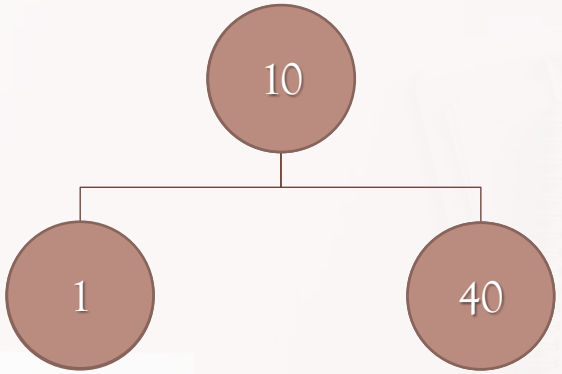
Eliminar:
- 20

Ejemplo agregado - eliminado



Eliminar:
- 20

Ejemplo agregado - eliminado



Implementación Aclaraciones

- Para encontrar el menor elemento, descendemos hacia la izquierda mientras sea posible; para encontrar el mayor elemento, descendemos hacia la derecha mientras sea posible.
- Para eliminar un elemento, si lo buscamos y no lo encontramos, no hacemos nada.


¡Muchas Gracias!

Bibliografía

- *Programación II – Apuntes de Cátedra – V1.3 – Cuadrado Trutner – UADE*
- *Programación II – Apuntes de Cátedra – Wehbe – UADE*