ÁRBOLES BINARIOS DE BÚSQUEDA.

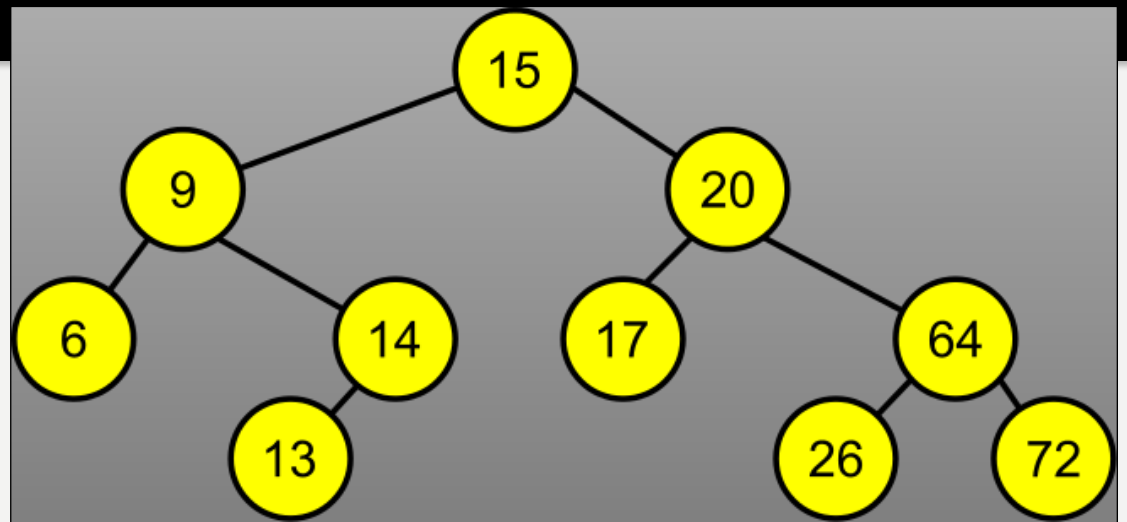
Un árbol binario de búsqueda (ABB) es un árbol binario definido de la siguiente forma:

1. Todo árbol vacío es un árbol binario de búsqueda.

2. Un árbol binario no vacío, de raíz R, es un árbol binario de búsqueda si:

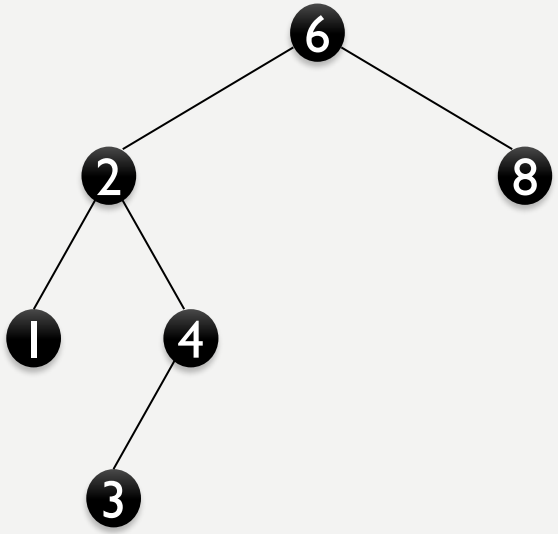
   - En caso de tener subárbol izquierdo, la raíz R debe ser mayor que el valor máximo almacenado en el subárbol izquierdo, y el subárbol izquierdo debe ser un árbol binario de búsqueda.

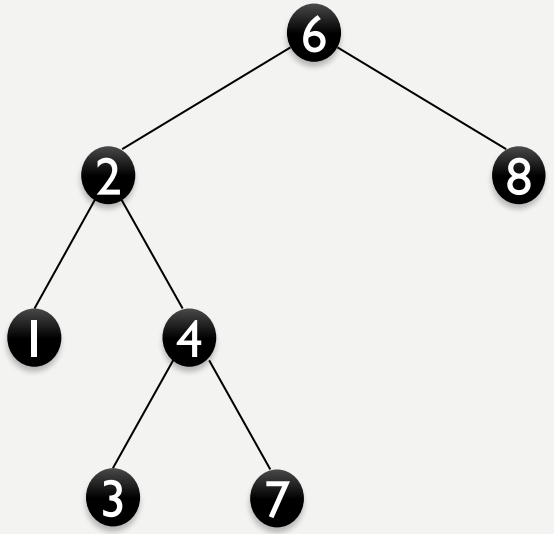
   - En caso de tener subárbol derecho, la raíz R debe ser menor que el valor mínimo almacenado en el subárbol derecho, y el subárbol derecho debe ser un árbol binario de búsqueda.

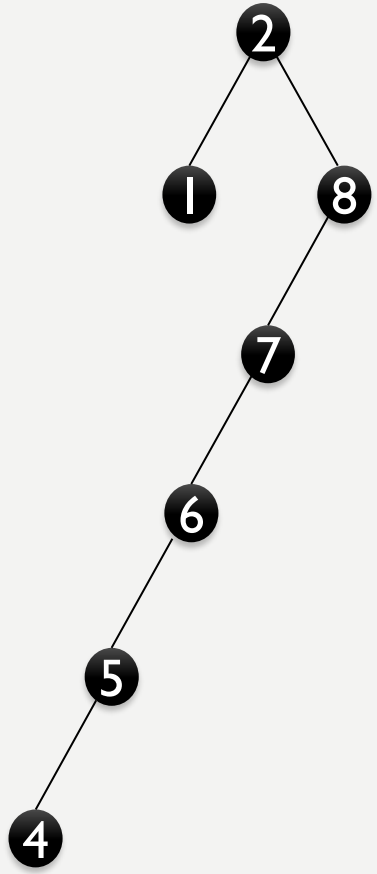


EJEMPLOS

- ¿son todos árboles binarios de búsqueda?



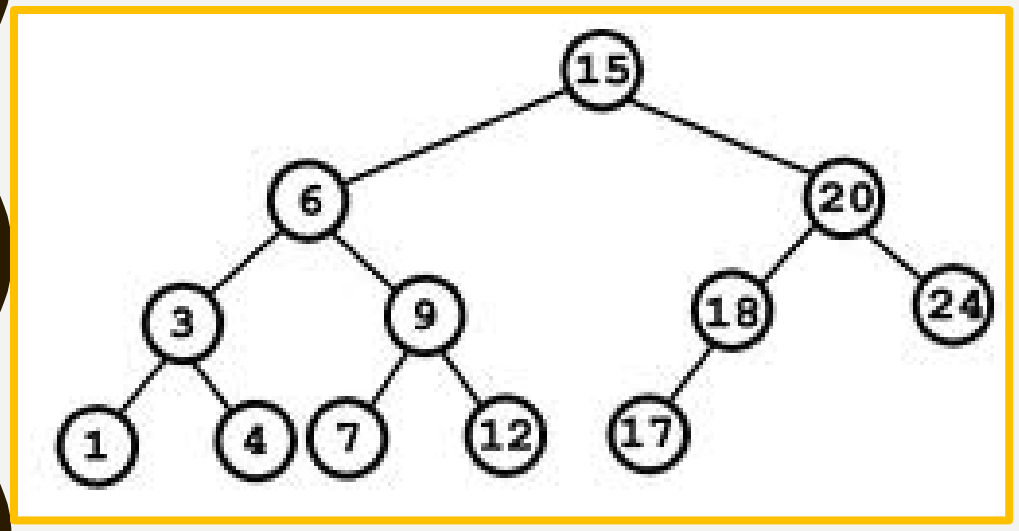




ÁRBOL BINARIO DE BÚSQUEDA

- **Es un árbol:**
  - Una colección de nodos que puede ser vacía, o que en su defecto consiste de un nodo raíz *R* y un número finito de estructuras tipo árbol $T_1,\ldots,T_k$, llamados subárboles, los cuales son disjuntos y sus respectivos nodos raíz están conectados a R.
- **Es binario:**
  - Cada nodo puede tener como máximo dos hijos, en otras palabras, cada nodo sólo puede tener dos subárboles.
- **Es de búsqueda** porque:
  - Los nodos están ordenados de manera conveniente para la búsqueda.
  - Todas las datos del subárbol izquierdo son menores que el dato del nodo raíz, y todas los datos del subárbol derecho son mayores.

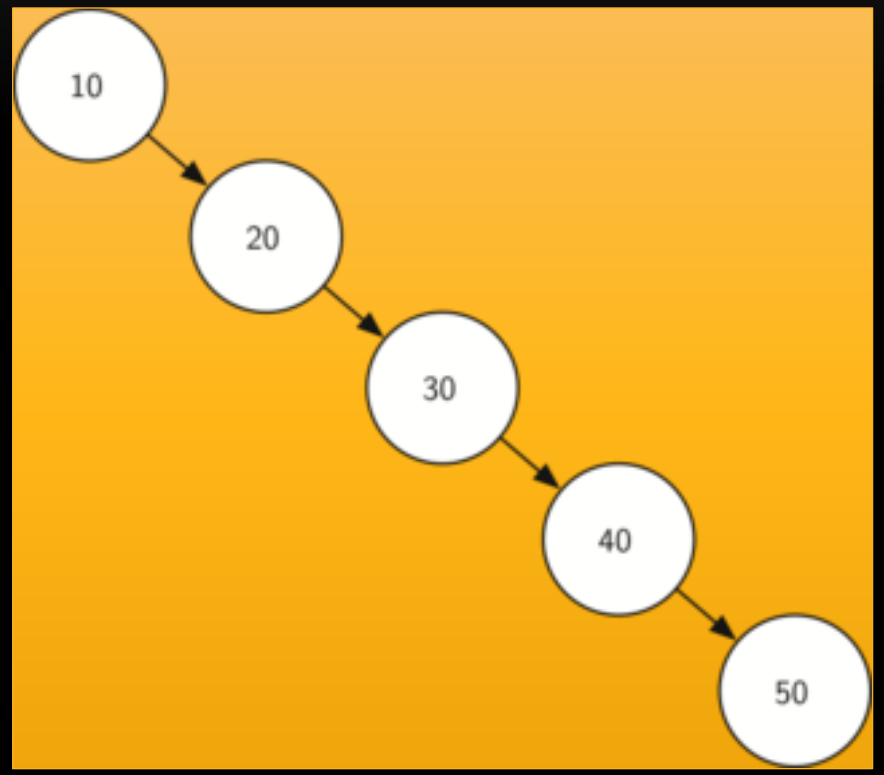
EJEMPLOS DE ABB:



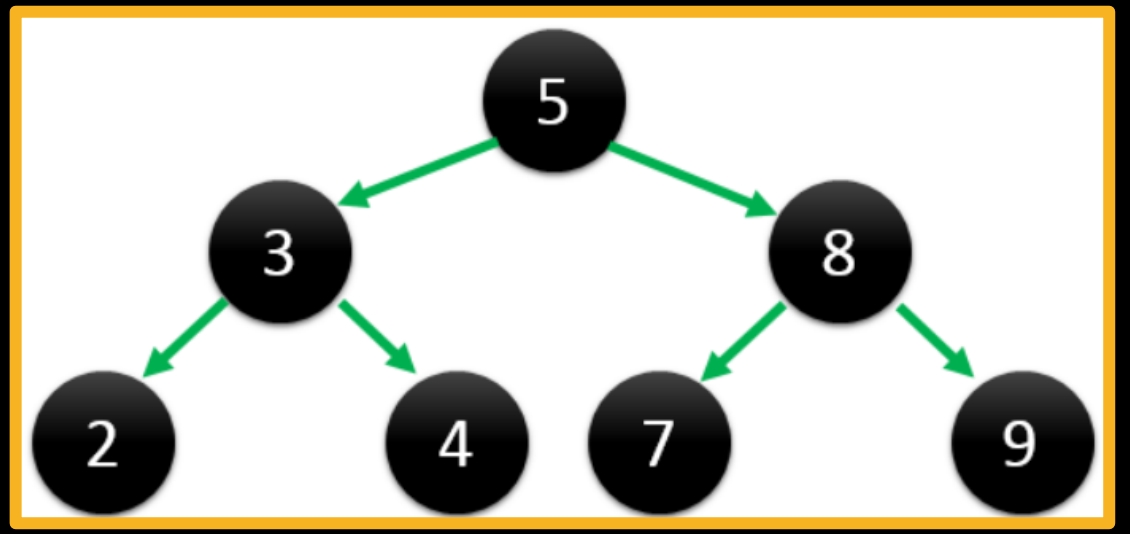
- Preorden: 15, 6, 3, 1, 4, 9, 7, 12, 20, 18, 17, 24
- InOrden: 1, 3, 4, 6, 7, 9, 12, 15, 17, 18, 20, 24.
- PostOrden: 1, 4, 3, 7, 12, 9, 6, 17, 18, 24, 20, 15
- Amplitud: 15, 6, 20, 3, 9, 18, 24, 1, 4, 7, 12, 17

- Se considera que hay una relación de orden establecida entre los elementos de los nodos, sin embargo, puede haber distintos árboles binarios de búsqueda para un mismo conjunto de elementos; estos varían según el orden en que son ingresados los elementos.

- La altura h en el peor de los casos siempre
- el mismo tamaño que el número de elementos
- disponibles.



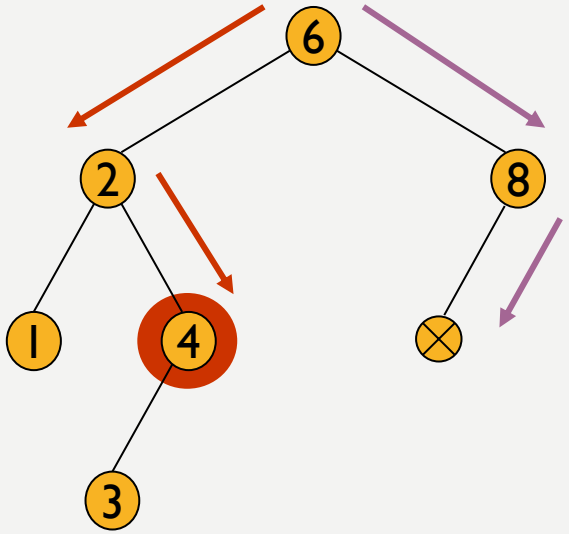
En el mejor de los casos el orden de búsqueda es O(log2N) y corresponde a un árbol completo.



BÚSQUEDA EN ABB:

- La búsqueda consiste acceder a la raíz del árbol.
- Si el elemento a localizar coincide con éste la búsqueda ha concluido con éxito.
- Si el elemento es menor se busca en el subárbol izquierdo y si es mayor en el derecho.
- Si se alcanza un nodo hoja y el elemento no ha sido encontrado se supone que no existe en el árbol.

OPERACIÓN BUSCAR




```
boolean buscar(Arbol *nodo, int elem)
{
     if (nodo == NULL) return FALSE;
     else if (nodo->dato < elem) return buscar(nodo->izq, elem);
               else if (nodo->dato > elem) return buscar(nodo->der, elem);
                         else return TRUE;
 }
```

Buscando 4:VERDADERO
Buscando 7: FALSO

BÚSQUEDA EN ABB:

- La búsqueda en este tipo de árboles es muy eficiente, representa un método logarítmica.

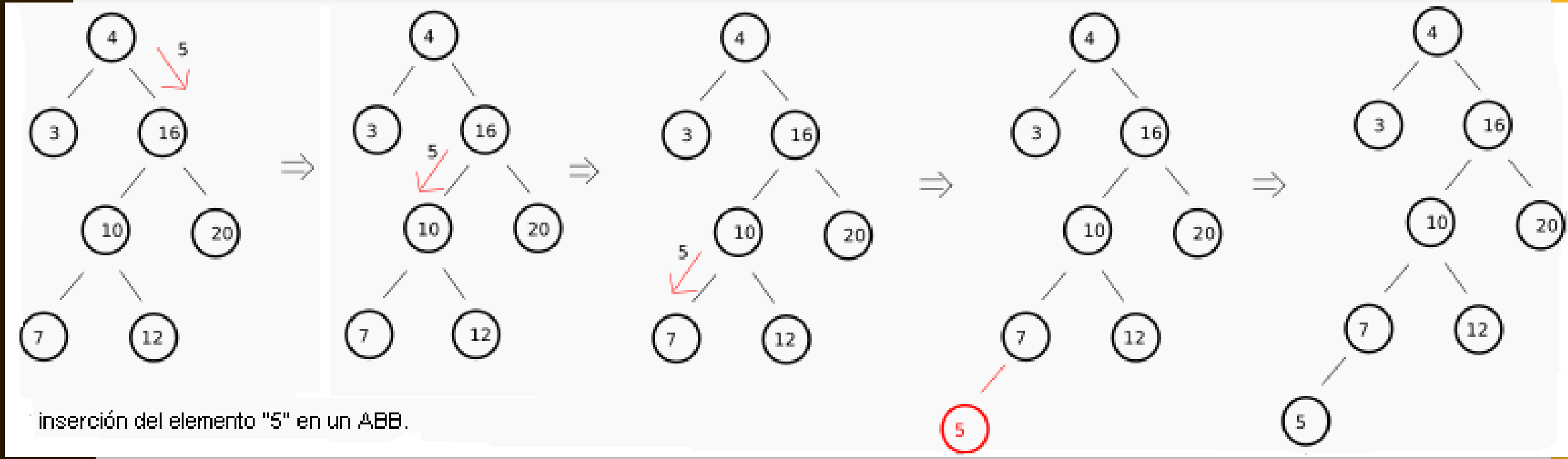
- El máximo número de comparaciones necesario para saber si un elemento se encuentra en un árbol binario de búsqueda estaría entre $[\log_2(N+1)]$ y N, siendo N el número de nodos.

- La búsqueda de un elemento en un ABB (Árbol Binario de Búsqueda) se puede realizar de dos formas, iterativa o recursiva.

INSERCIÓN

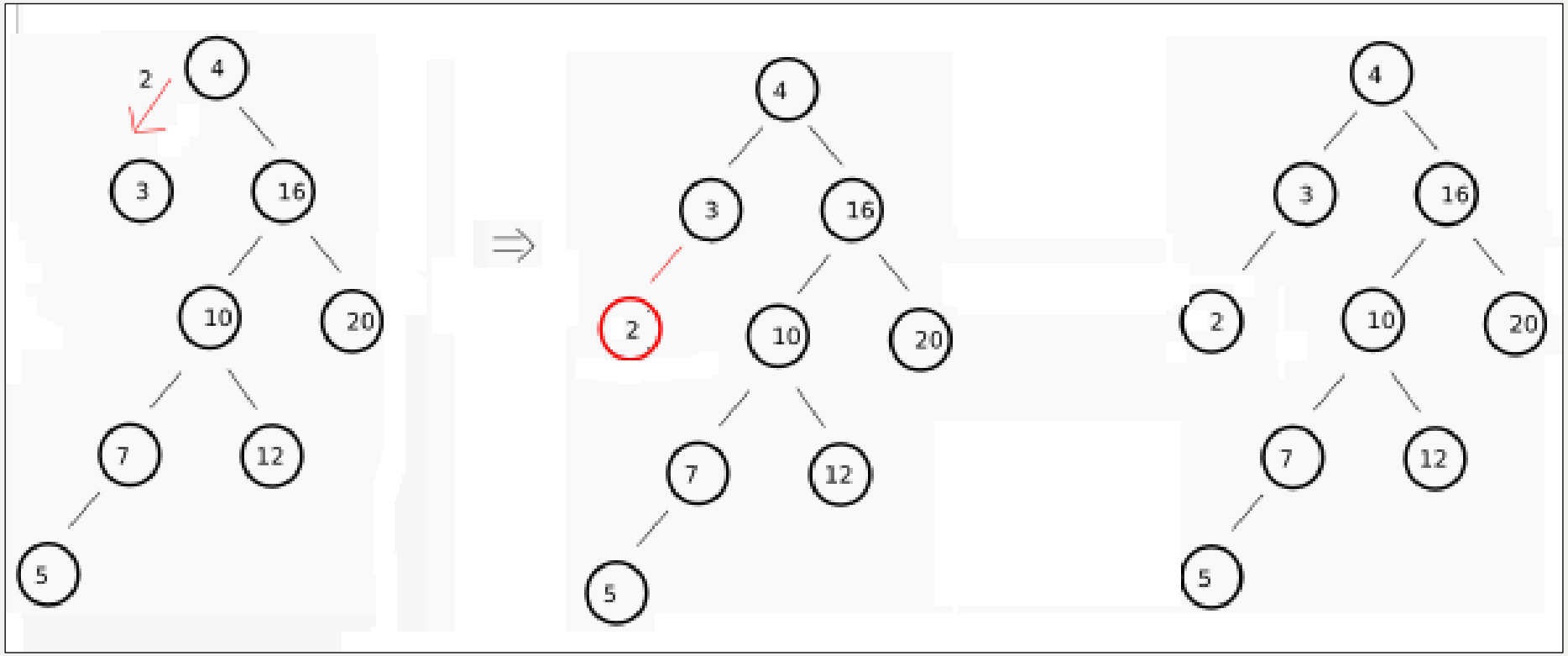
- La inserción es similar a la búsqueda y se puede dar una solución tanto iterativa como recursiva.
- Si inicialmente se tiene un árbol vacío se crea un nuevo nodo como único contenido el elemento a insertar.
- Si no lo está, se comprueba si el elemento dado es menor que la raíz del árbol inicial con lo que se inserta en el subárbol izquierdo y si es mayor se inserta en el subárbol derecho.
- De esta forma las inserciones se hacen en las hojas.

EJEMPLO:



inserción del elemento "5" en un ABB.

EJEMPLO:

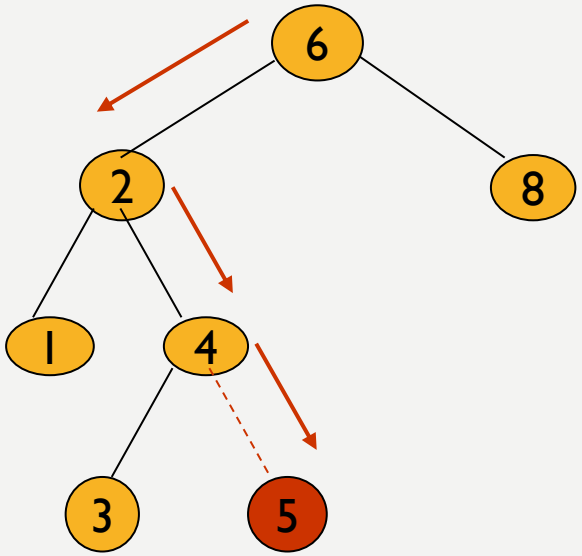


IMPLEMENTACIÓN EN C:

```
1. typedef struct tarbol
2. {
3.   int clave;
4.   struct tarbol *hizq;
5.   struct tarbol *hder;
6. } tarbol;
```

OPERACIÓN INSERTAR

Insertando 5




```
void insertanodonuevo(ARBOL *rarbol,int nuevo)
{
if(*rarbol==NULL){
        *rarbol=(NODO *)malloc(sizeof(NODO));
        if(*rarbol!=NULL){
                        (*rarbol)->info=nuevo;
                        (*rarbol)->izqnodo =NULL;
                        (*rarbol)->dernodo=NULL;
          }
          else{ printf("\n Memoria No Disponible !!!!\n");}
  }
 else
        if(nuevo<(*rarbol)->info) insertanodonuevo(&((*rarbol)->izqnodo),nuevo);
        else if(nuevo>(*rarbol)->info) insertanodonuevo(&((*rarbol)->dernodo),nuevo);
}
```

INSERCIÓN EN ABB:

- Como en el caso de la búsqueda puede haber varias variantes a la hora de implementar la inserción en el Árbol Binario de Búsqueda, y es una decisión a tomar cuando el elemento (o clave del elemento) a insertar ya se encuentra en el árbol: puede que éste sea modificado o que sea ignorada la inserción.

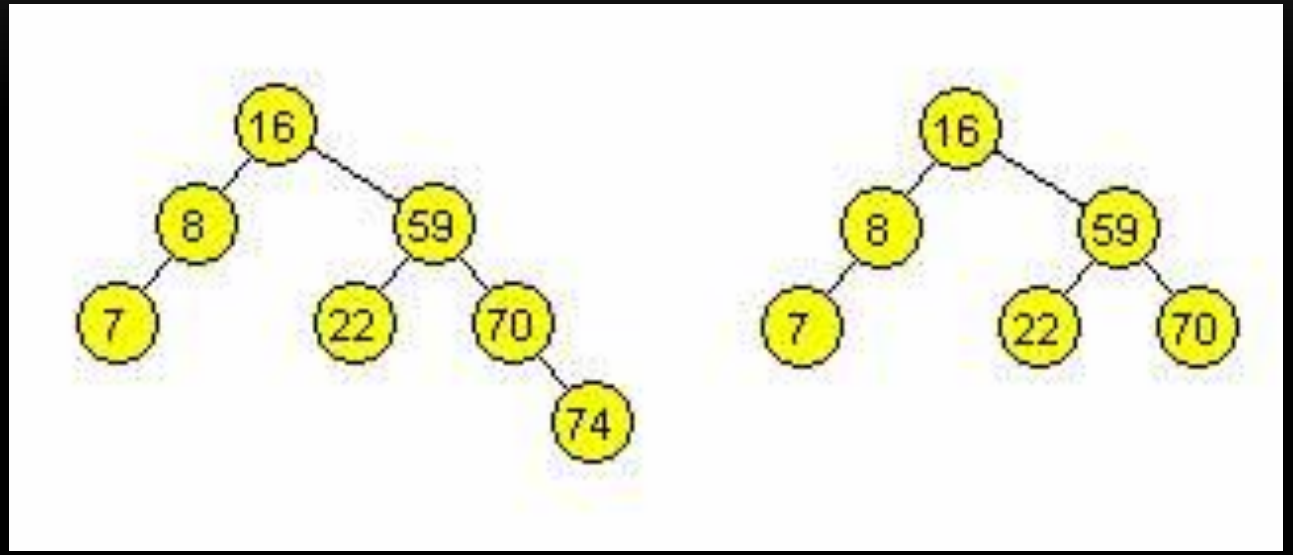
OPERACIÓN ELIMINAR

- 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 puntero del nodo padre a NULL.
  3. **Eliminar un nodo con un solo hijo.**
     - Caso sencillo; el nodo padre del nodo a borrar se convierte en el padre del único nodo hijo.
  4. **Eliminar un nodo con dos hijos.**
     - Caso complejo, es necesario mover más de un puntero.

ELIMINACIÓN: NODO HOJA

- Simplemente se borra y se establece a nulo el puntero de su padre.

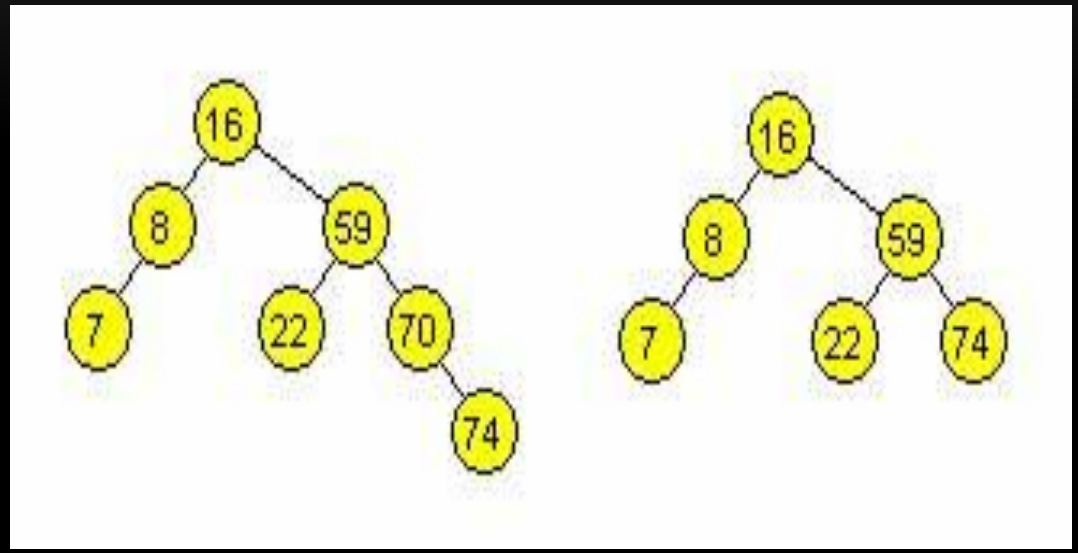
Nodo a eliminar 74



BORRAR UN NODO CON UN SUBÁRBOL HIJO

- se borra el nodo y se asigna su subárbol hijo como subárbol de su padre.

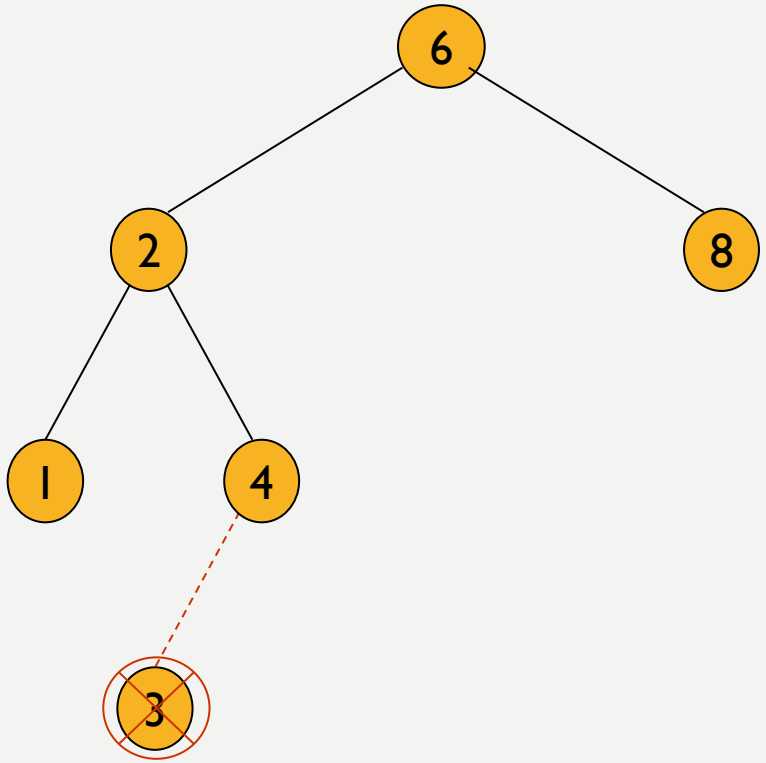
Nodo a eliminar 70



ELIMINAR (CASOS SENCILLOS)

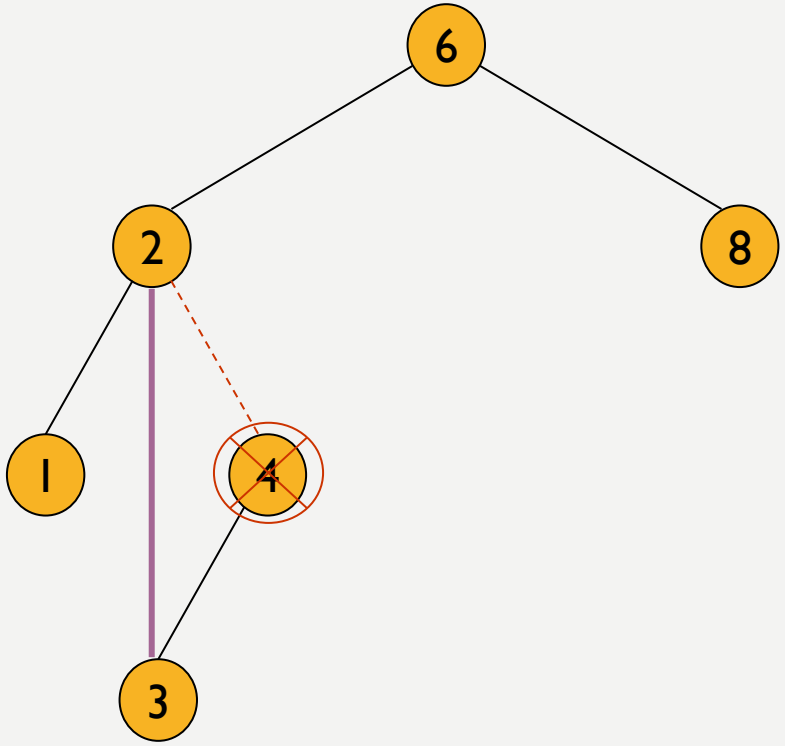
Eliminar nodo hoja

Eliminar 3



Eliminar nodo con un hijo

Eliminar 4

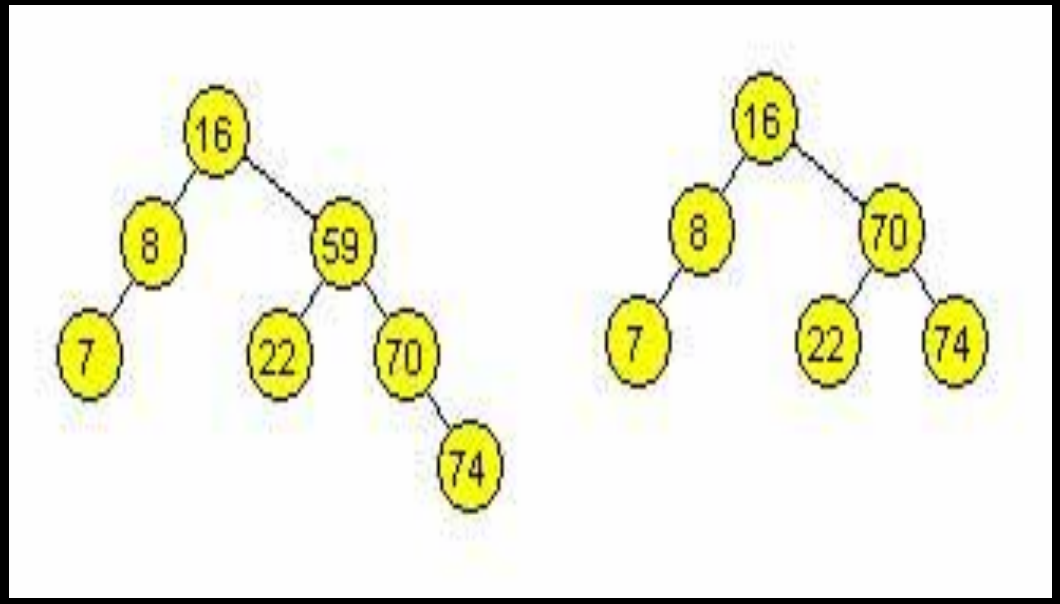


BORRAR UN NODO CON DOS SUBÁRBOLES HIJO:

- la solución está en reemplazar el valor del nodo por el de su predecesor o por el de su sucesor en inorden y posteriormente borrar este nodo.

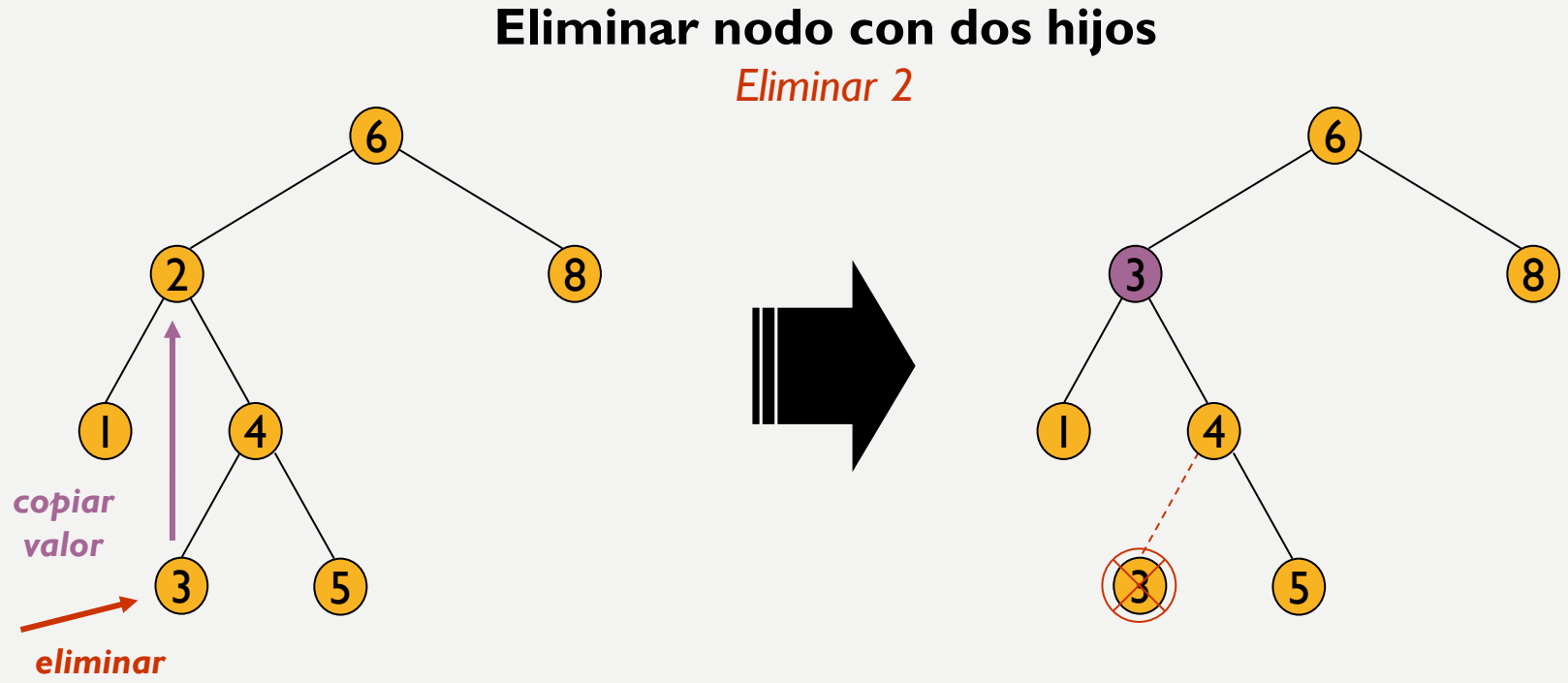
- Su predecesor en inorden será el nodo más a la derecha de su subárbol izquierdo (mayor nodo del subarbol izquierdo), y su sucesor el nodo más a la izquierda de su subárbol derecho (menor nodo del subarbol derecho).

- En la siguiente figura se muestra cómo existe la posibilidad de realizar cualquiera de ambos reemplazos:



Nodo a eliminar: 59

ELIMINAR (CASO COMPLEJO)

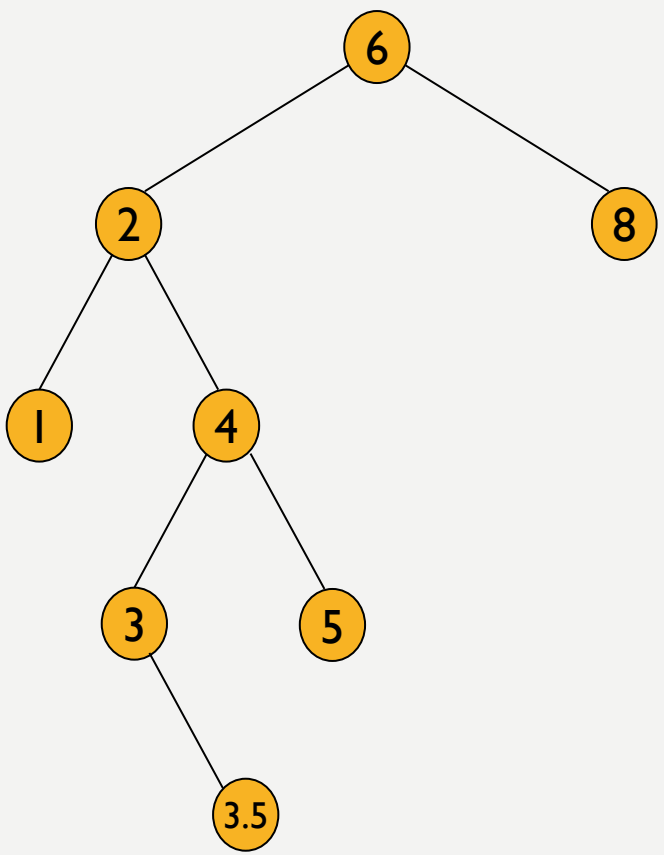


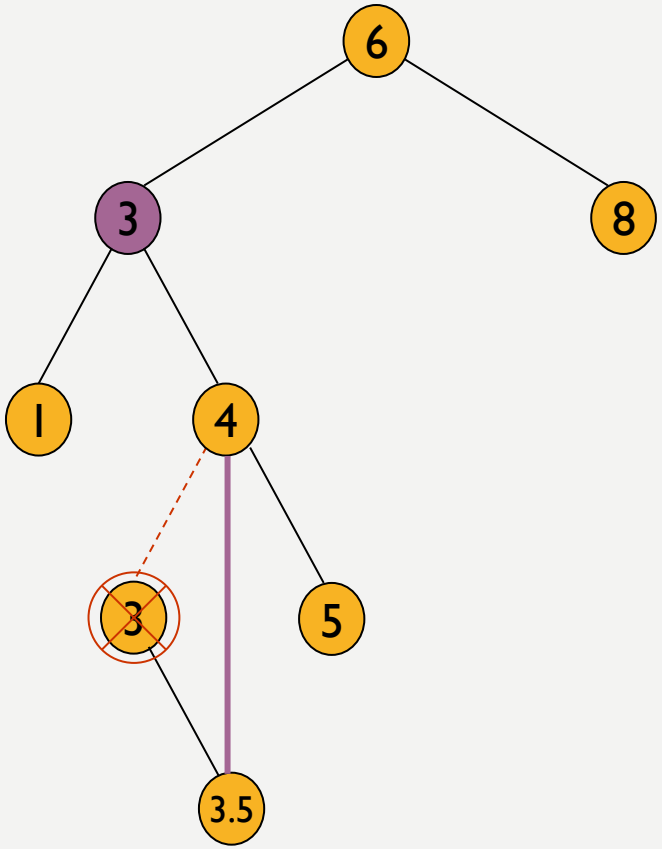
- Remplazar el dato del nodo que se desea eliminar con el dato del nodo más pequeño del subárbol derecho (predecesor inorden).
- Después, eliminar el nodo más pequeño del subárbol derecho (caso fácil)

OTRO EJEMPLO (CASO COMPLEJO)

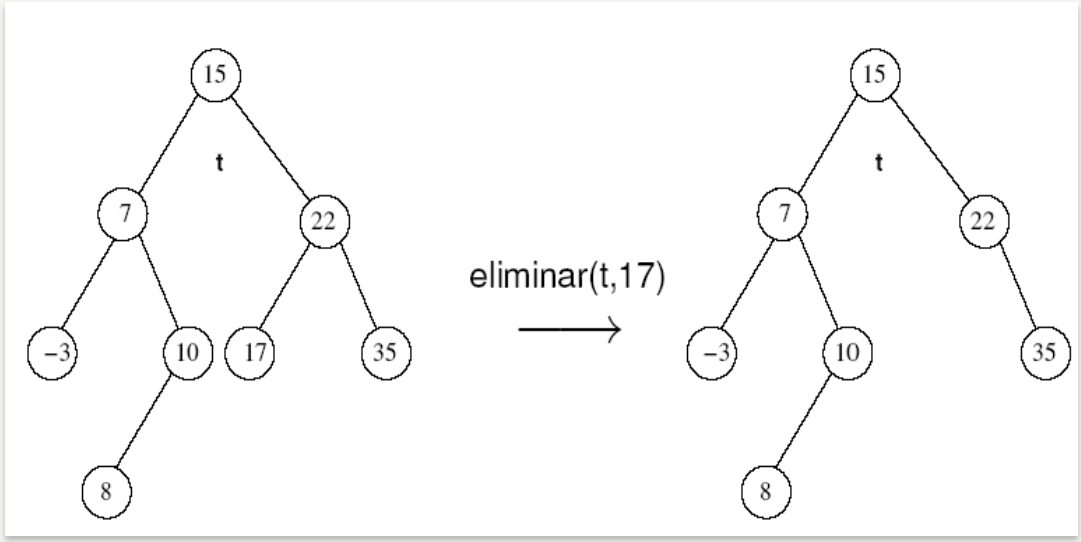
Eliminar nodo con dos hijos

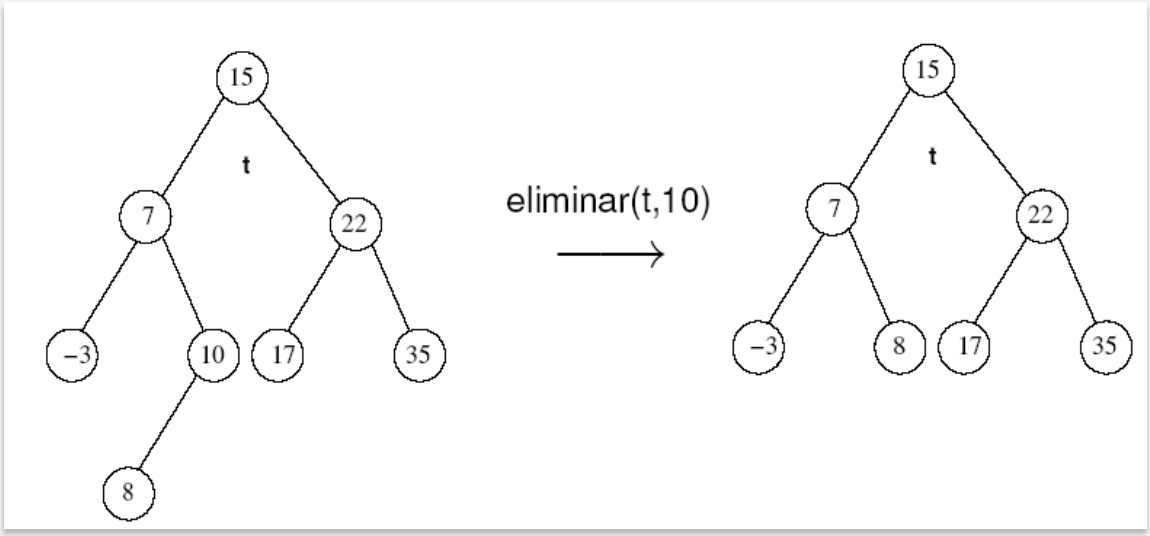
Eliminar 2

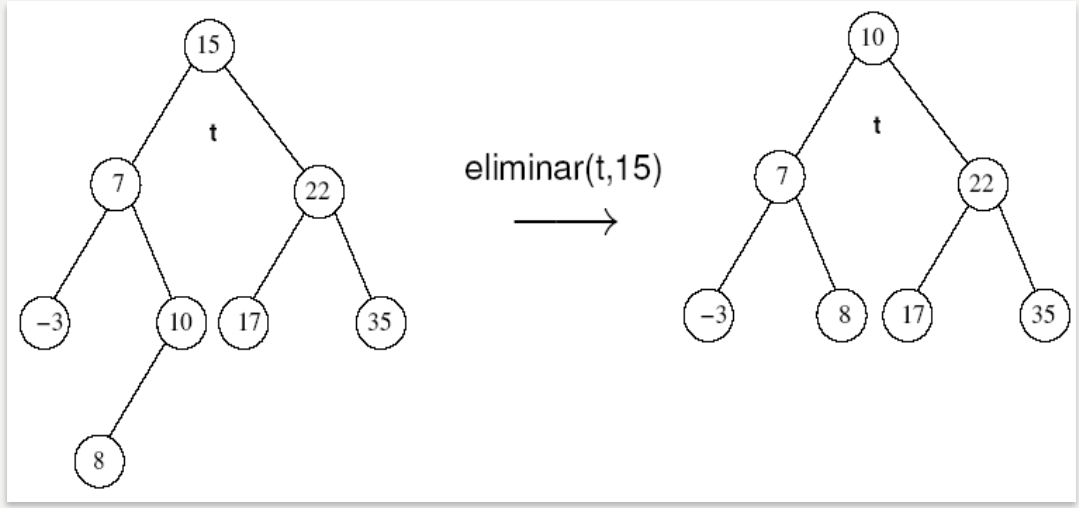


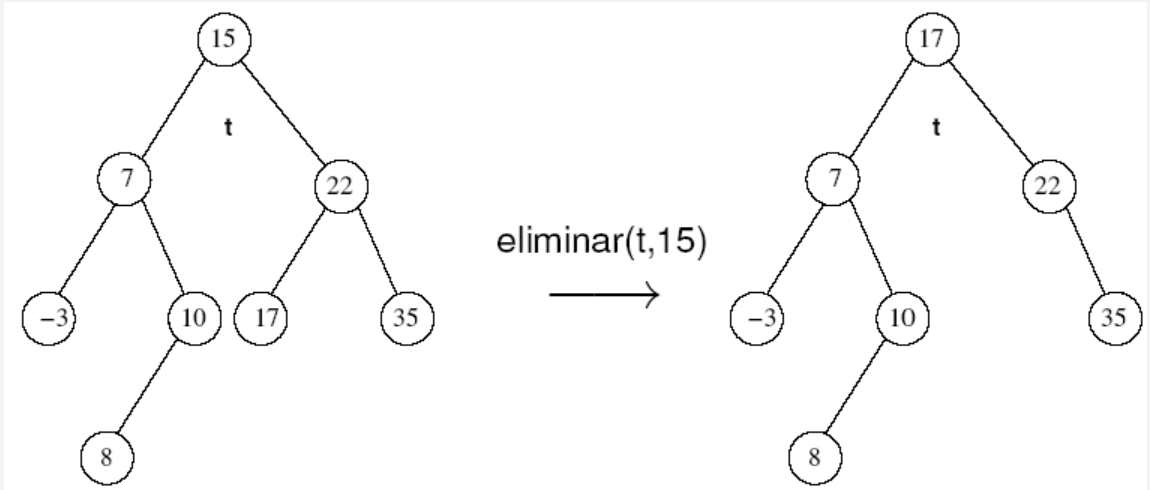


EJEMPLO:


15
t
7
22
−3
10
17
35
8
eliminar(t,17)
15
t
7
22
−3
10
35
8


15
t
7
22
−3
10
17
35
8
eliminar(t,10)
15
t
7
22
−3
8
17
35


15
t
7
22
−3
10
17
35
8
eliminar(t,15)
10
t
7
22
−3
8
17
35


15
t
7
22
−3
10
17
35
8
eliminar(t,15)
17
t
7
22
−3
10
35
8

EJERCICIO:

1. Mostrar el resultado de insertar los valores 3, 1, 4, 6, 9, 2, 5 y 7 en un ABB inicialmente vacío.
2. Repetir la operación insertando los valores en orden creciente.
3. Repetir de nuevo la operación insertándolos en el orden siguiente: 5, 2, 4, 3, 7, 9, 6, 1.

EJERCICIO: VERIFIQUE SI LA FUNCIÓN ENTREGA EL RESULTADO CORRECTO (PREDECESOR INORDEN DEL NODO CON CLAVE "C" EN UN ABB).

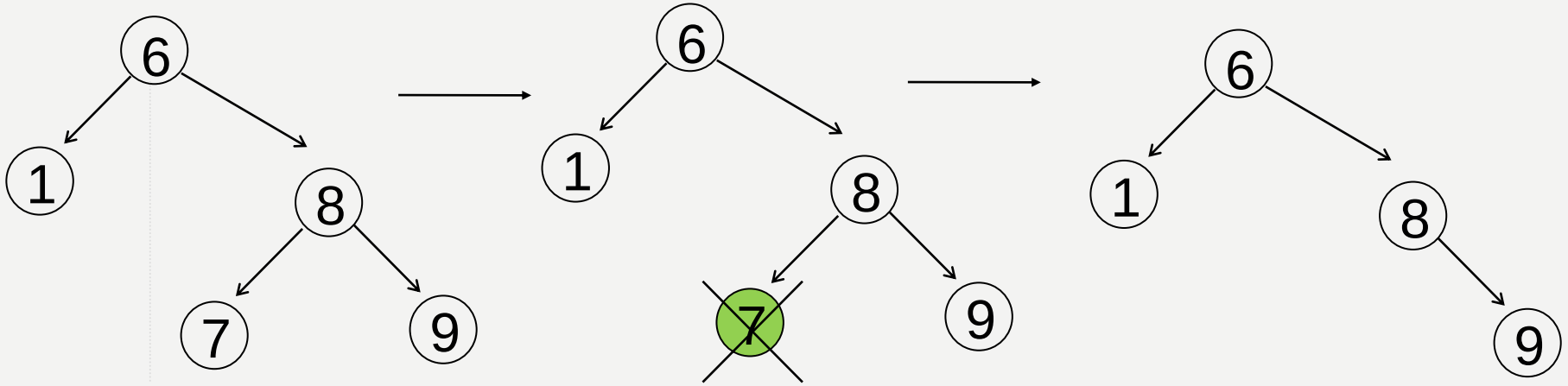
```
1.  void predecesorInorden(tarbol *a,int c)
2.  {   int flag=0;
3.      tarbol *aux=a, *aux1=NULL;
4.      while(aux->clave != c)if(aux->clave > c)aux=aux->hizq;
5.                          else aux=aux->hder;
6.
7.    if (aux->hizq != NULL){
8.              aux1=aux->hizq;
9.              while(aux1->hder!=NULL)aux1=aux1->hder;
10.             printf("\n Predecesor InOrden de %c  es %c",aux->clave, aux1->clave);
11.             }
12.   else if (aux ==a)printf("El nodo no tiene predecesor");
```

```
•  else if (aux ==a)printf("El nodo no tiene predecesor");
•       else  {aux1=aux;
•             while(flag==0){
•                   aux1=padreDe(a,aux1->clave);
•                   if(aux1->clave < aux->clave){
•                       printf("El predecesor es %c", aux1->clave);
•                       flag=1;
•                       }
•                   else if(aux1==a){flag=1;
•                                   printf("El nodo no tiene predecesor");
•                                   }
•             }
•             }
•  }
•  /***************/
```

REVISIÓN: ELIMINACIÓN DE NODOS EN ÁRBOLES BINARIOS DE BÚSQUEDA.

Si el elemento a borrar es terminal (hoja), simplemente se elimina.

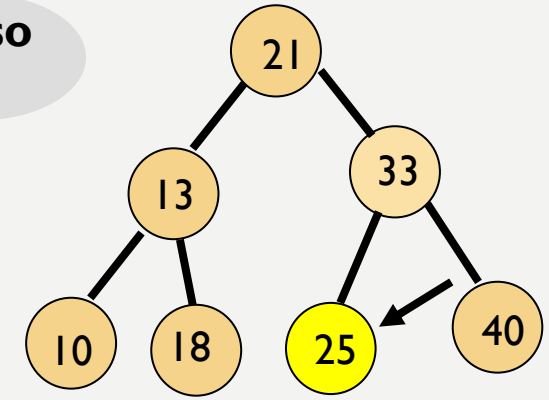
Ejemplo eliminar nodo 7



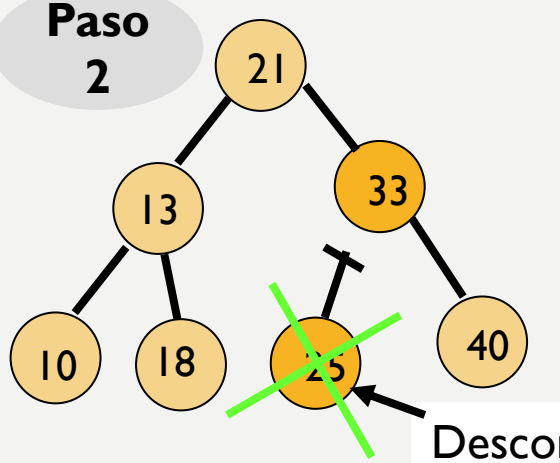
CASO 1: ELIMINAR NODO HOJA

Eliminar el valor 25

Paso 1



Paso 2

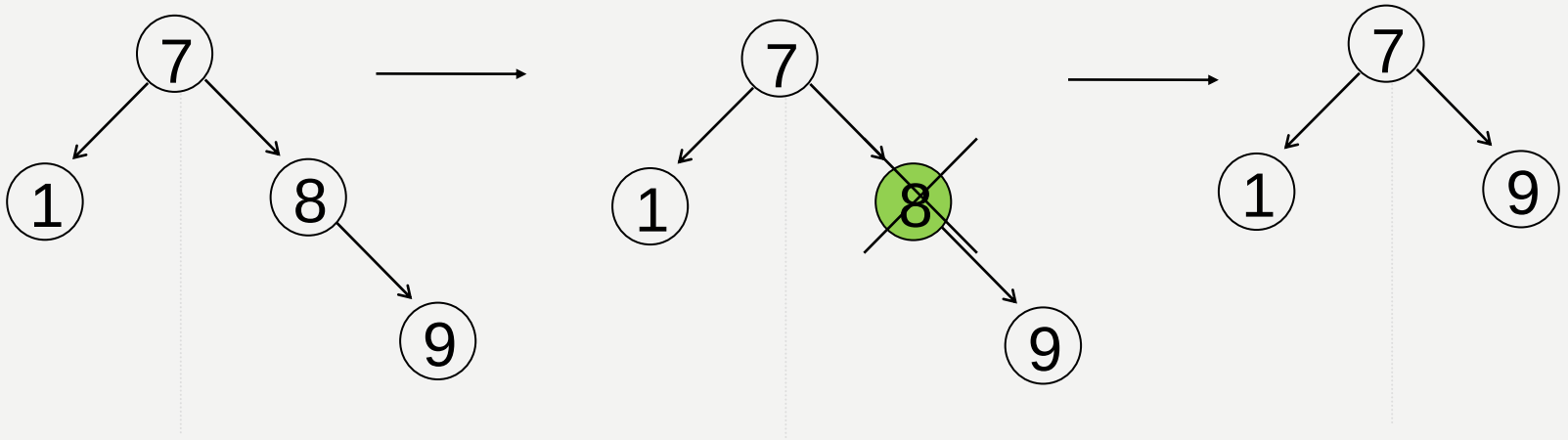


Desconectarlo y liberar el nodo

CASO 2: ELIMINAR UN NODO CON UN HIJO.

- Si el elemento a borrar tiene un solo hijo, entonces tiene que sustituirlo por el hijo

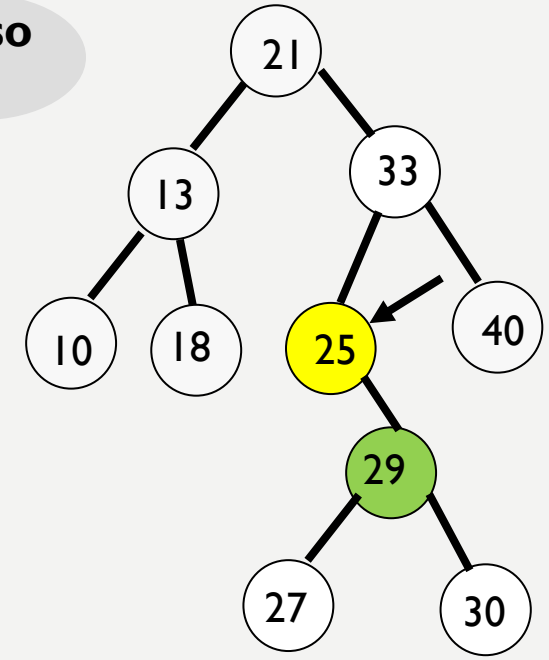
Ejemplo: eliminar nodo 8



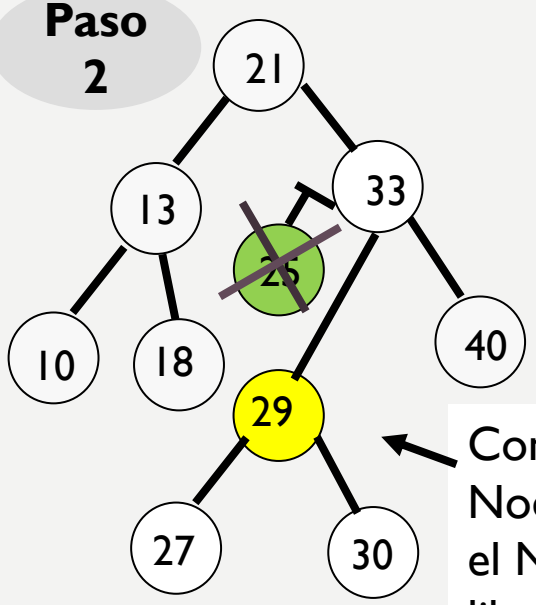
CASO2: ELIMINAR NODO CON UN HIJO

Eliminar el valor 25

Paso 1



Paso 2

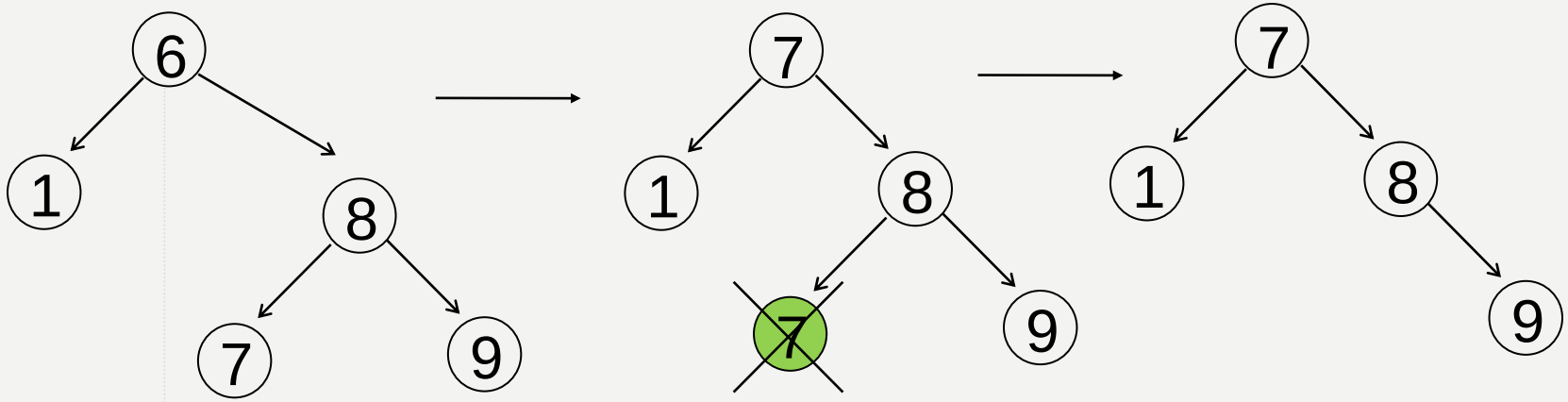


Conectar el Nodo Padre con el Nodo Hijo y liberar el nodo.

CASO 3: ELIMINAR UN NODO CON DOS HIJOS.

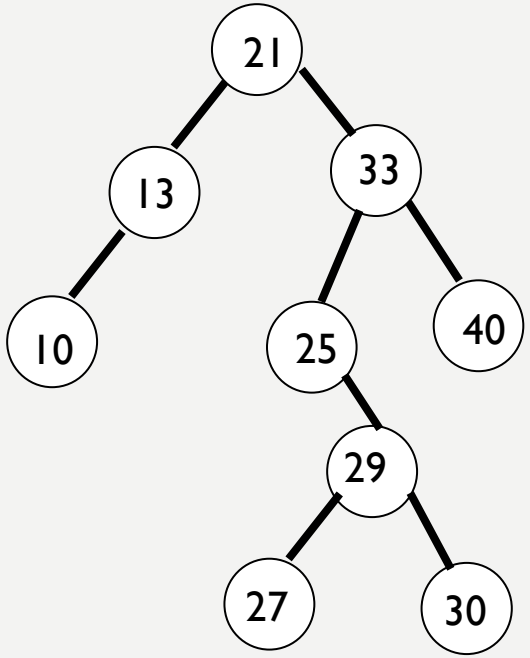
•Si el elemento a borrar tiene los dos hijos, entonces se tienen que sustituir por sucesor o predecesor inorden.

Ejemplo: eliminar el 6



PREDECESOR

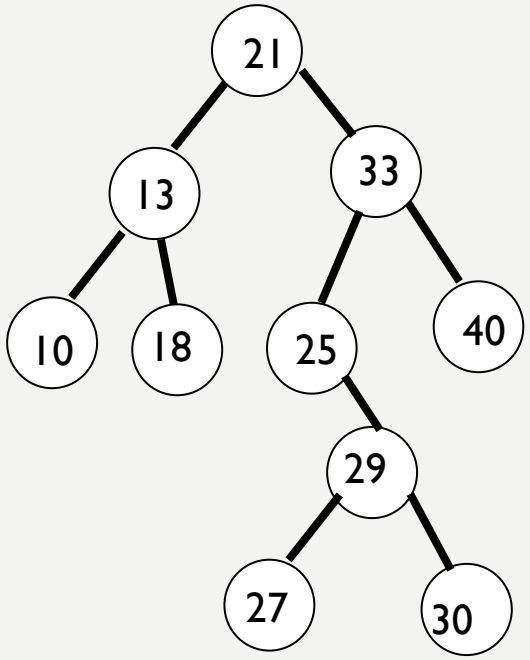
Uno a la IZQUIERDA y todo a la DERECHA



El predecesor de:	Es:
33	30
21	13
29	27

SUCESOR

Uno a la DERECHA y todo a la IZQUIERDA

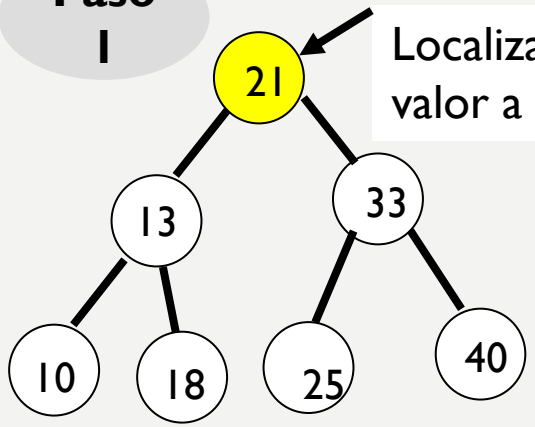


El sucesor de:	Es:
21	25
33	40
29	30

CASO3: ELIMINAR NODO CON DOS HIJOS

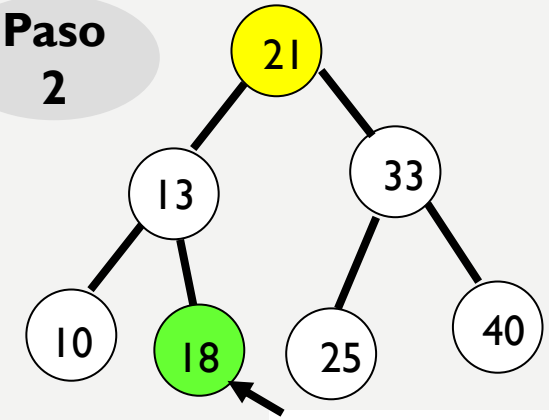
Eliminar el valor 21 utilizando el predecesor

Paso 1



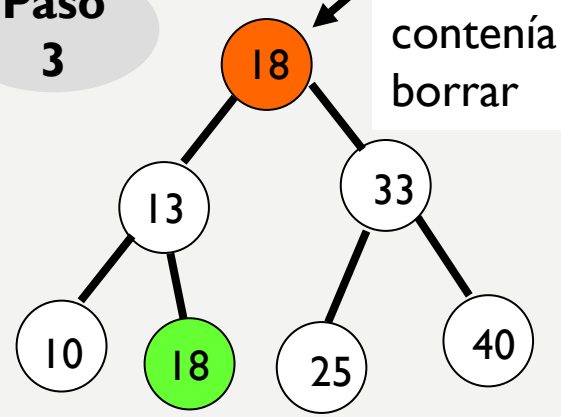
Localizar el valor a borrar

Paso 2



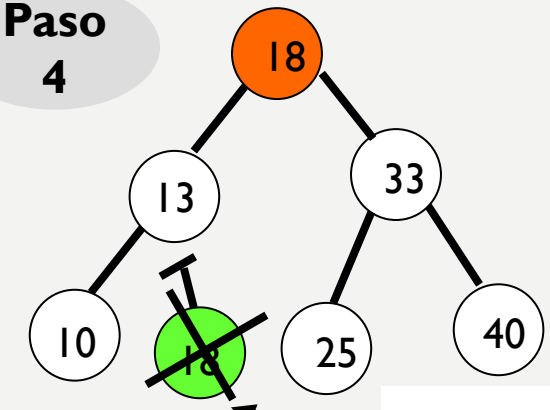
Localizar el Predecesor

Paso 3



Copiar el valor del Predecesor al nodo que contenía el valor a borrar

Paso 4

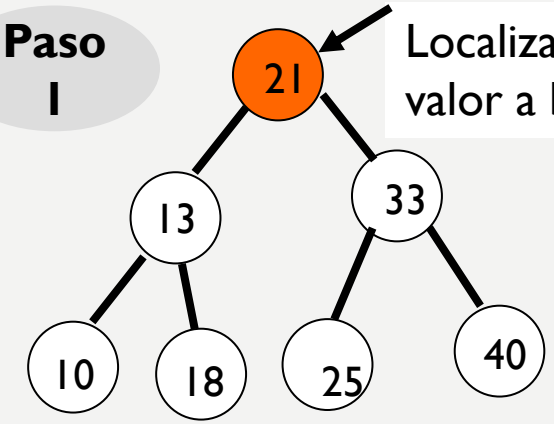


Desconectar y liberar el nodo del Predecesor

CASO3: ELIMINAR NODO CON DOS HIJOS

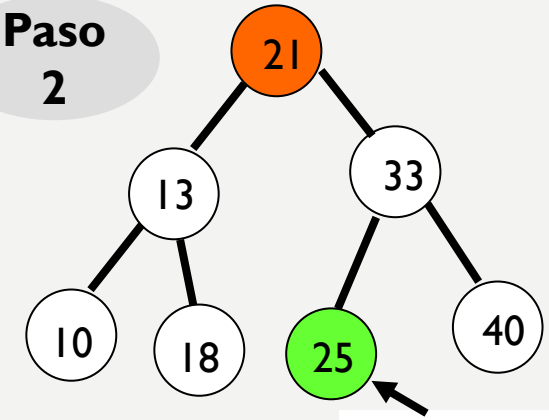
Eliminar el valor 21 utilizando el Sucesor

Paso 1



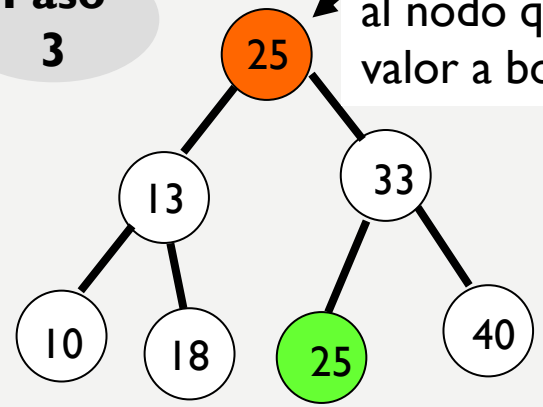
Localizar el valor a borrar

Paso 2



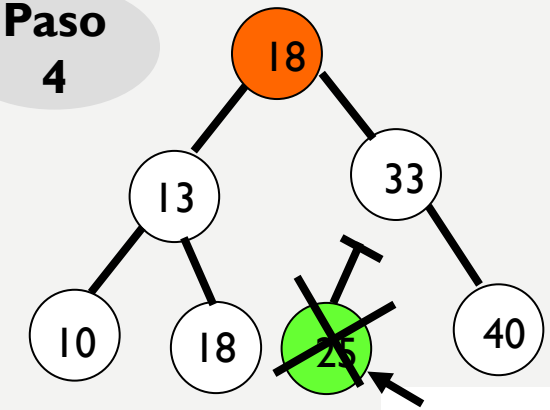
Localizar el Sucesor

Paso 3



Copiar el valor del Sucesor al nodo que contenía el valor a borrar

Paso 4

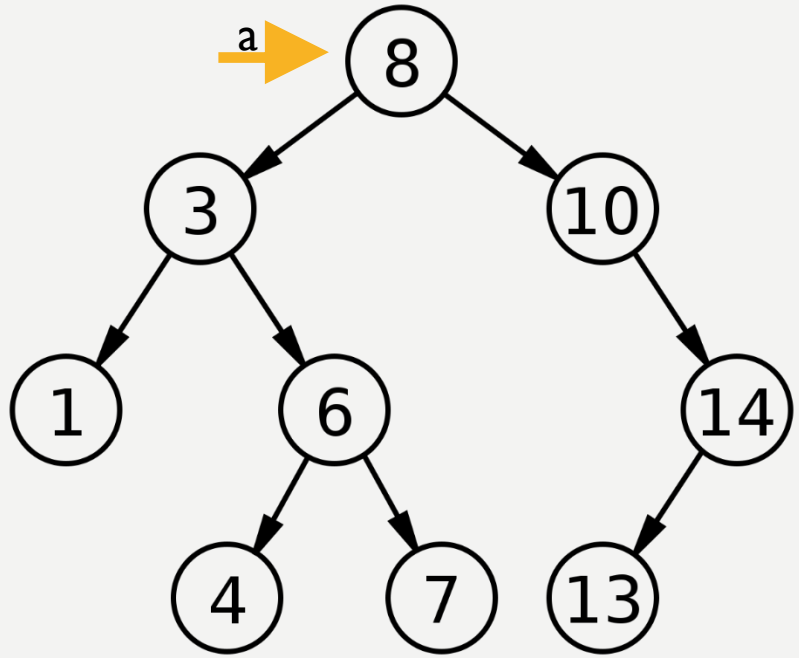


Desconectar y liberar el nodo del Sucesor

EJERCICIO: VERIFIQUE LA FUNCIÓN QUE RETORNA UN PUNTERO AL PADRE DEL NODO CON CLAVE K.

```
1.  tarbol *padreDe(tarbol *a, int k)
2.  {int flag=0;
3.  do{
4.  if((a->hizq!=NULL)&&(a->hizq->clave == k)){return(a);
5.                                          flag=1;
6.                                          }
7.  else if((a->hder!=NULL)&&(a->hder->clave == k)){return(a);
8.                                         flag=1;
9.                                         }
10.       else if(a->clave < k) a=a->hder;
11.            else a=a->hizq;

12. }while (flag !=1);
13. }
```

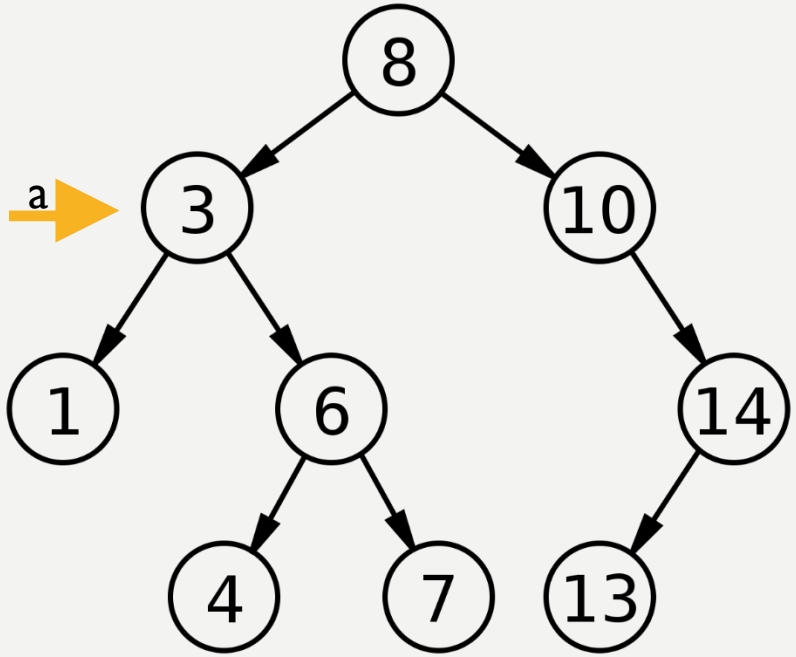


Paso 1, k=7

EJERCICIO: VERIFIQUE LA FUNCIÓN QUE RETORNA UN PUNTERO AL PADRE DEL NODO CON CLAVE K.

```
1.  tarbol *padreDe(tarbol *a, int k)
2.  {int flag=0;
3.  do{
4.  if((a->hizq!=NULL)&&(a->hizq->clave == k)){return(a);
5.                                          flag=1;
6.                                          }
7.  else if((a->hder!=NULL)&&(a->hder->clave == k)){return(a);
8.                                         flag=1;
9.                                         }
10.       else if(a->clave < k) a=a->hder;
11.            else a=a->hizq;

12. }while (flag !=1);
13. }
```

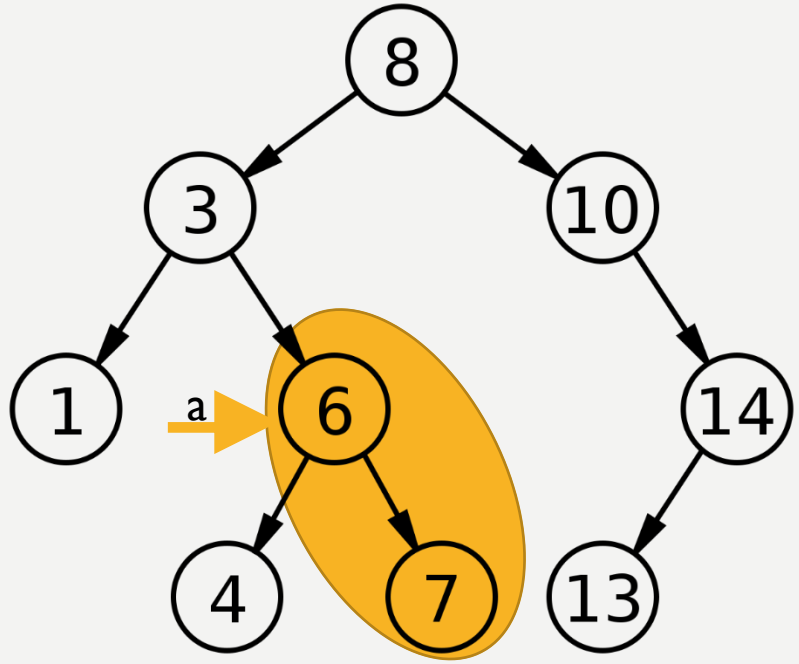


Paso 1, k=7

EJERCICIO: VERIFIQUE LA FUNCIÓN QUE RETORNA UN PUNTERO AL PADRE DEL NODO CON CLAVE K.

```
1.  tarbol *padreDe(tarbol *a, int k)
2.  {int flag=0;
3.  do{
4.  if((a->hizq!=NULL)&&(a->hizq->clave == k)){return(a);
5.                                          flag=1;
6.                                          }
7.  else if((a->hder!=NULL)&&(a->hder->clave == k)){return(a);
8.                                         flag=1;
9.                                         }
10.       else if(a->clave < k) a=a->hder;
11.            else a=a->hizq;

12. }while (flag !=1);
13. }
```

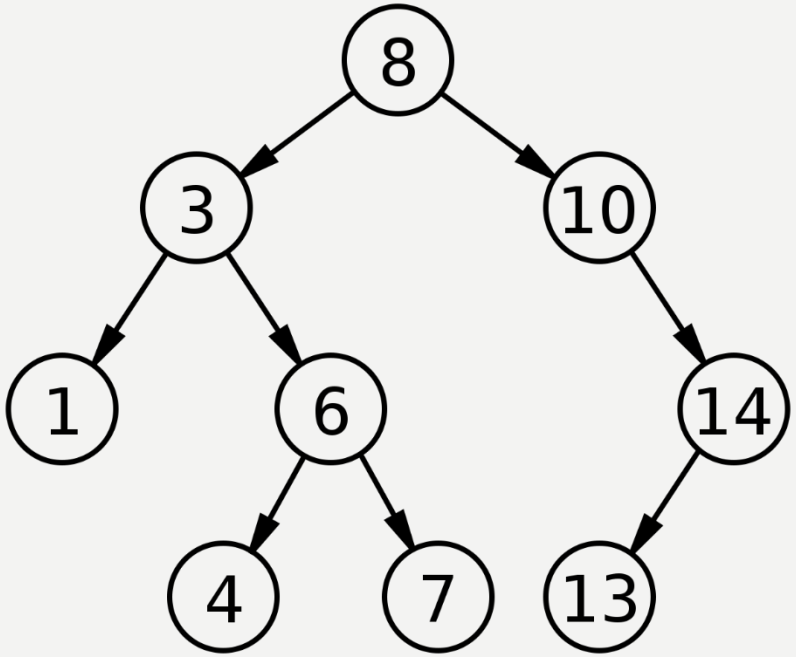


Paso 3, k=7

EJERCICIO:

Escribir una función C que tome como entrada un ABB T y dos claves k1 y k2 (k1 < k2), e imprima la secuencia de elementos x tales que k1 < clave(x) < k2.

- Ejemplo: si k1 =3 y k2 = 10, debería imprimir los valores:
- 4, 6, 7, 8



ELIMINAR NODOS…

- Se les pide eliminar sucesivamente, del ABB presentado, las claves:
- 22, 99, 87, 120, 140, 135, 56.