Instituto Politécnico Nacional

Escuela Superior de Cómputo

Estructuras de Datos

Tema 10: Árbol binario de búsqueda

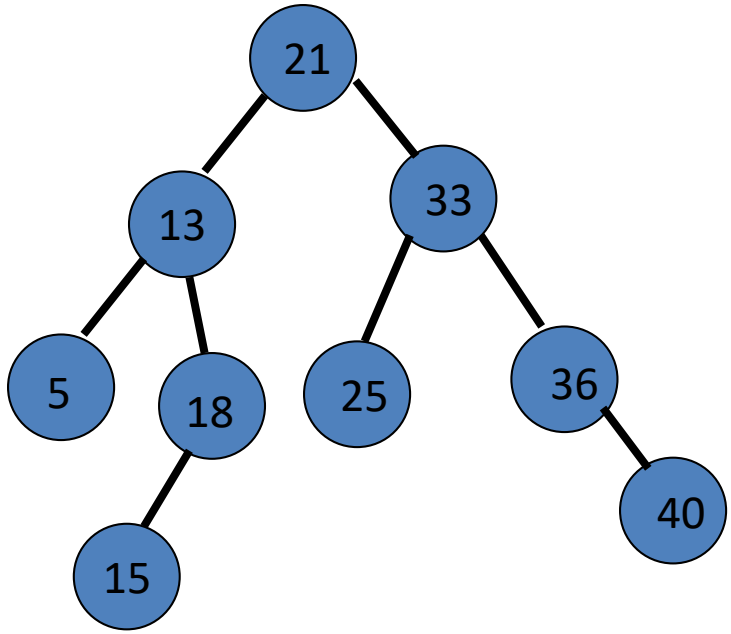
M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom *edgardoadrianfrancom*

Contenido

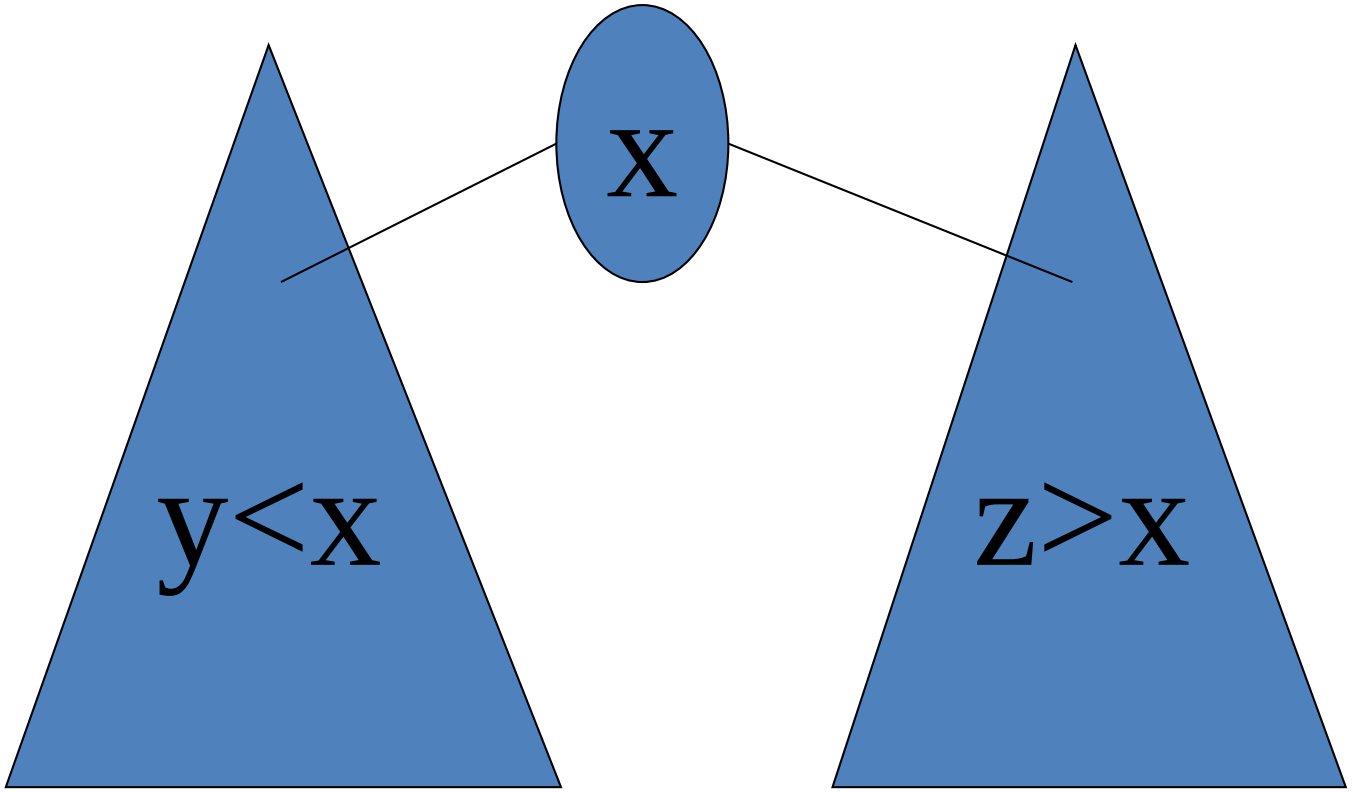
- *Árbol binario de búsqueda*
  - *Búsquedas*
  - *Inserción*
  - *Eliminación*
- *Operaciones útiles en un ABB*
  - *Contar nodos*
  - *Sumar nodos*
  - *Contar hojas*
  - *Calcular la profundidad*
- *Comentarios importantes*
  - *Liga web*

Árbol binario de búsqueda

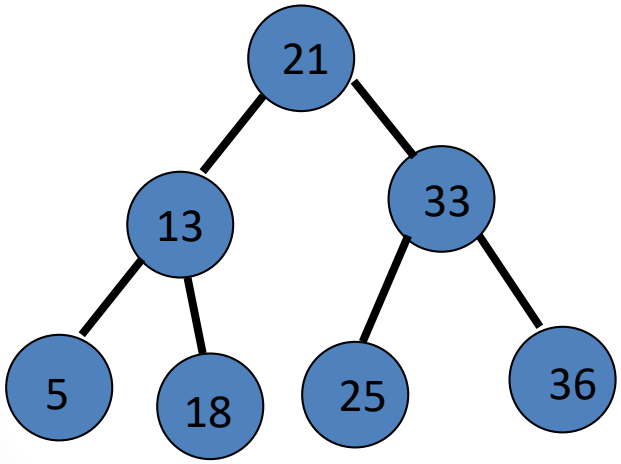
- Un árbol binario de búsqueda es un tipo especial de árbol binario cuya principal característica es que la información se almacena en los nodos cuidando mantener cierto orden.

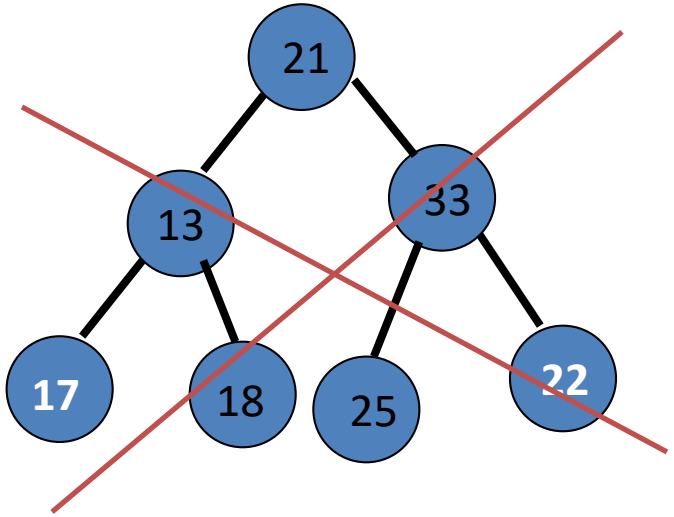


- En un ABB sus nodos también son ABB y contienen información ordenada de tal manera que todos los elementos a la izquierda de la raíz son menores a la raíz y todos los elementos a la derecha de la raíz son mayores a la raíz.



- En un ABB
  - Todos los nodos a la izquierda son menores al padre.
  - Todos los nodos a la derecha son mayores al padre.
  - Un nodo solo puede tener 2 hijos a lo mucho.





- Los árboles binarios de búsqueda se utilizan típicamente como estructura de datos para la representación de **conjuntos de datos** con **una clave única** y **operaciones**:

  - **Inserta:** para insertar un elemento dado en el conjunto.

  - **Suprime:** para suprimir un elemento determinado de el conjunto

  - **Busca:** para saber y obtener si un elemento dado pertenece o no al conjunto.

Árbol binario de búsqueda (Inserción)

- La ***inserción*** es una operación que se puede realizar eficientemente en un árbol binario de búsqueda. **La estructura crece conforme se inserten elementos** al árbol.

- Los pasos que deben realizarse para insertar un elemento a un ABB son los siguientes:

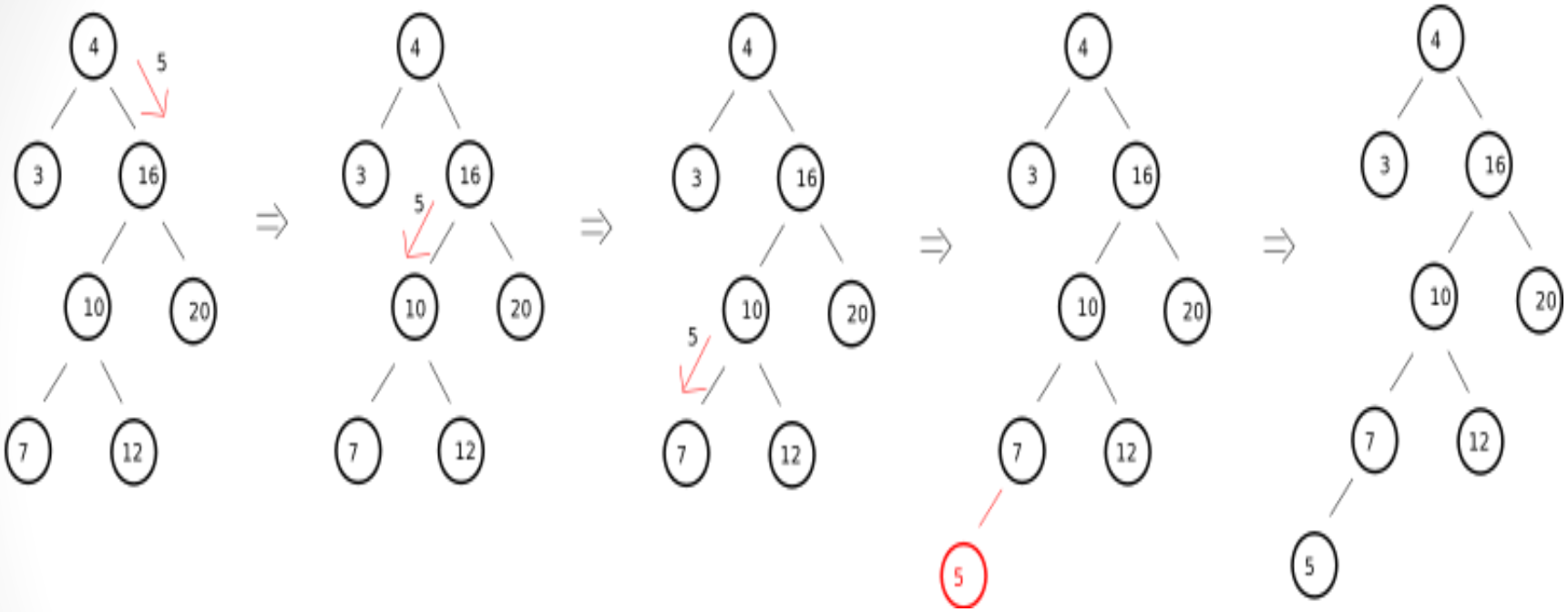
1. Debe **compararse el valor o (clave del elemento)** a insertar con la raíz del árbol. **Si es mayor**, debe avanzarse hacia el **subárbol derecho**. **Si es menor**, debe avanzarse hacia el **subárbol izquierdo**.

2. Repetir sucesivamente el paso 1 hasta que se cumpla alguna de las siguientes condiciones

- El **subárbol derecho es igual a vació**, o el **subárbol izquierdo es igual a vació**; en cuyo caso se procederá a **insertar el elemento en el lugar que le corresponde**.

- El valor o clave que quiere insertarse **es igual a la raíz** del árbol; en cuyo caso **no se realiza la inserción**.

Árbol binario de búsqueda (Inserción)



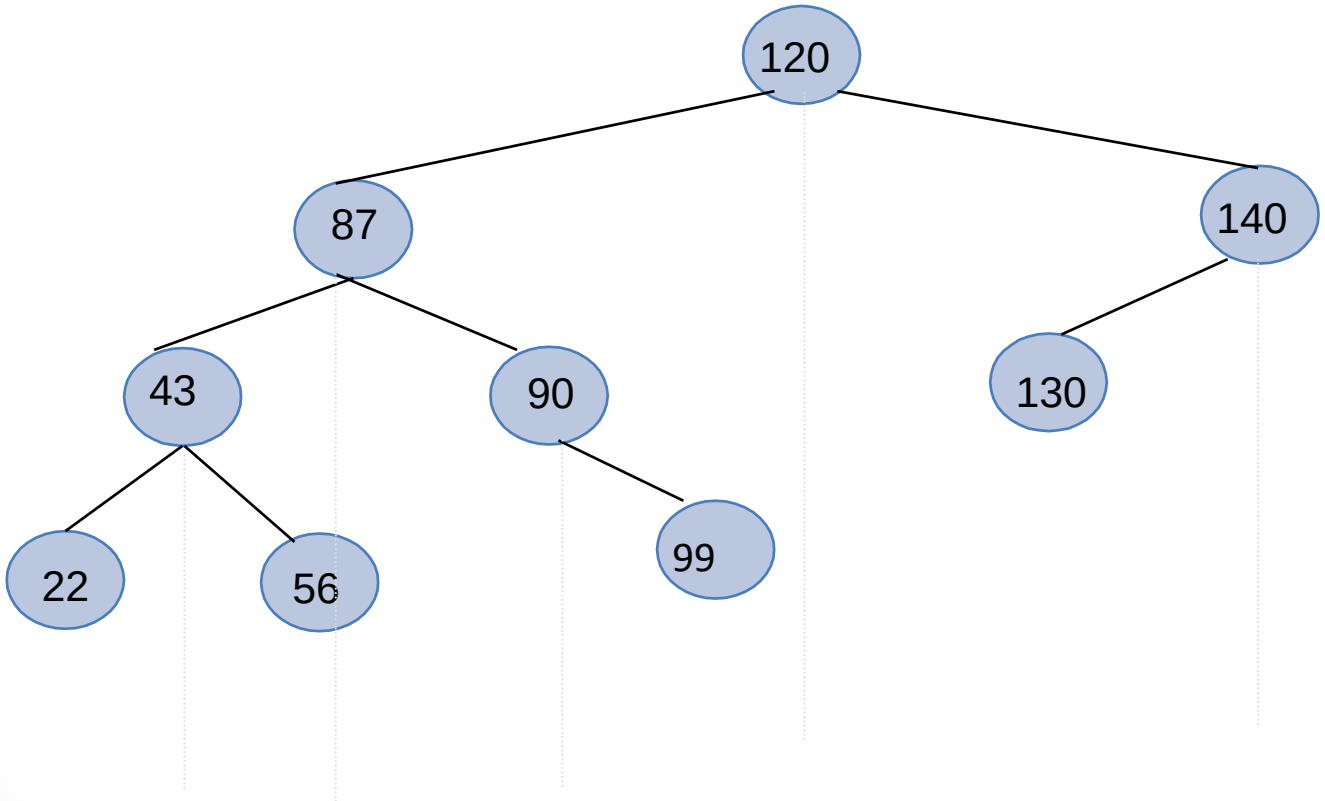
Árbol binario de búsqueda
(Inserción: Ejemplo 01)

- Supóngase que quieren insertarse elementos con las siguientes claves, en un árbol binario de búsqueda que se encuentra vació.

120 –87 – 43 – 90 – 140 – 99 – 130 – 22 – 56

Árbol binario de búsqueda
(Inserción: Ejemplo 02)

- Si se tiene la siguiente secuencia de inserción: 120 – 87 – 43 – 90 – 140 – 99 – 130 – 22 – 56



Árbol binario de búsqueda (Búsquedas)

- El árbol de búsqueda facilita el almacenamiento y búsqueda de información contenida en sus nodos, por medio de un campo clave.

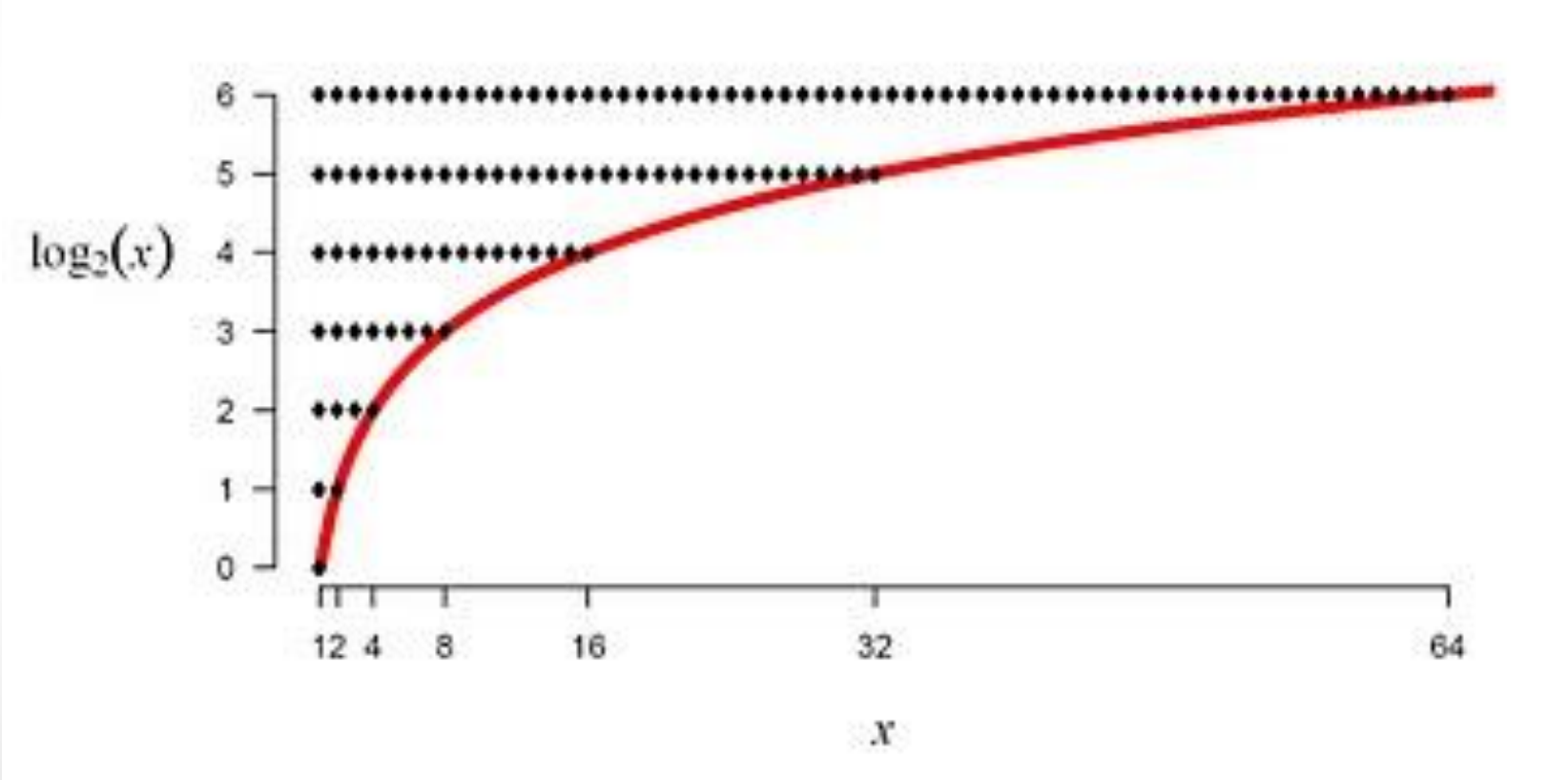
- La búsqueda consiste en acceder a la raíz del árbol, si la clave del elemento a localizar coincide con ésta la búsqueda ha concluido con éxito, si la clave del elemento es menor se busca en el subárbol izquierdo y si es mayor en el derecho. Si se alcanza un nodo hoja y la clave no ha sido encontrado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

- La búsqueda en un ABB es muy eficiente, representa una función logarítmica.

- El máximo número de comparaciones que necesitaríamos 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 una clave de elemento** en un ABB (Árbol Binario de Búsqueda) se puede realizar de dos formas, **iterativa** o **recursiva**.

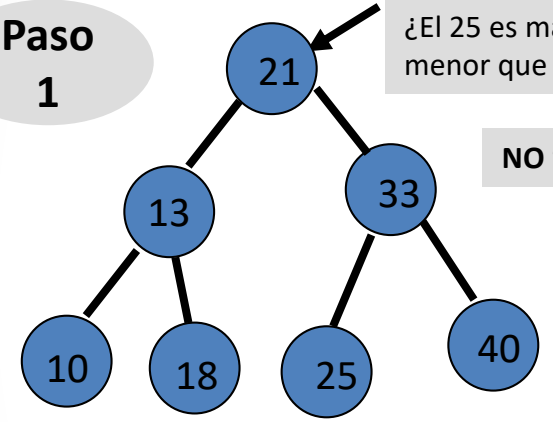
- Si se habla de un árbol binario completamente lleno este tendrá 2^{n-1} elementos, por lo que el orden de complejidad de la búsqueda será ideal ***O(log$_2$(N+1))***.



Árbol binario de búsqueda
(Búsquedas: Ejemplo 01)

Buscar el 25

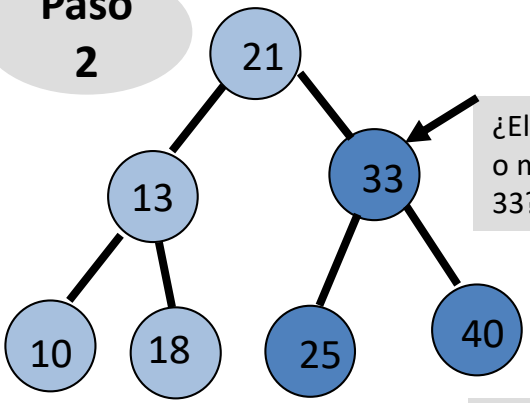
Paso 1



¿El 25 es mayor o menor que el 21?

NO "Ir por la **derecha**"

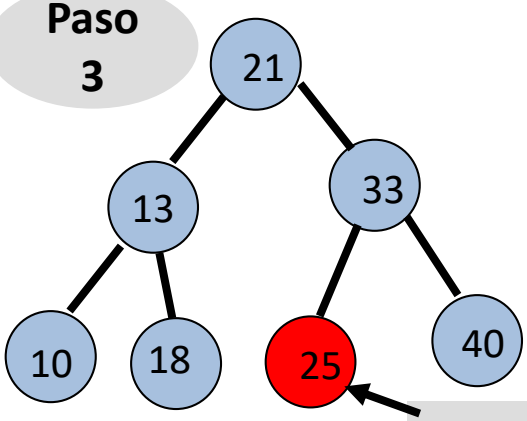
Paso 2



¿El 25 es mayor o menor que el 33?

SI "Ir por la **izquierda**"

Paso 3



Encontrado

Árbol binario de búsqueda (Eliminación)

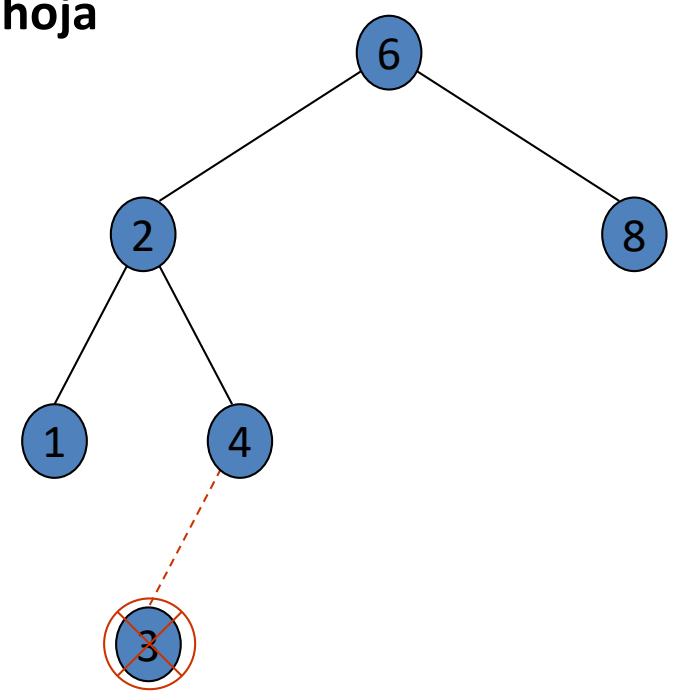
- Debido a que **al eliminar un nodo**, **no se desea eliminar al subárbol de debajo** de este, es necesario realizar los siguientes casos:

1. Si el elemento a borrar es terminal (hoja)
2. Si el elemento a borrar tiene un solo hijo
3. Si el elemento a borrar tiene los dos hijos

- **Caso 1**
  - Si el elemento a borrar es terminal (hoja), simplemente se elimina.

Eliminar nodo hoja

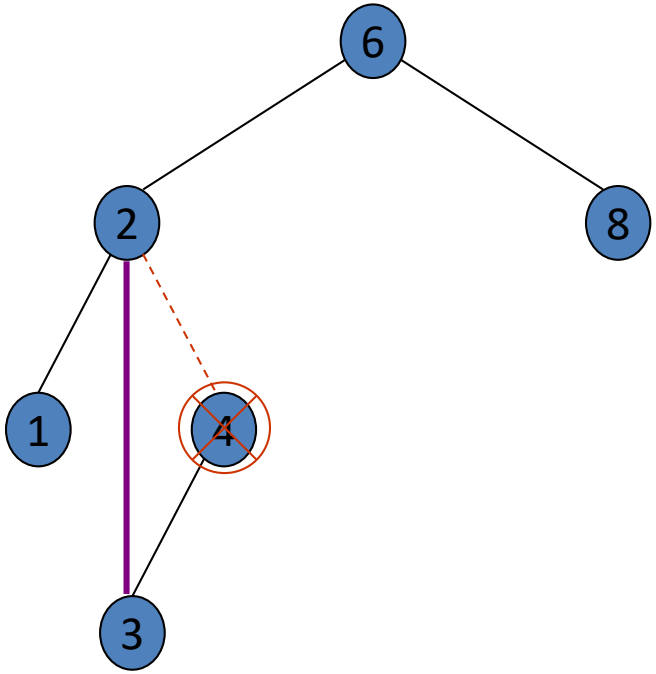
Eliminar 3



- **Caso 2**
  - Si el elemento a borrar tiene un solo hijo, entonces tiene que ser sustituido por el hijo

Eliminar nodo con un hijo

Eliminar 4



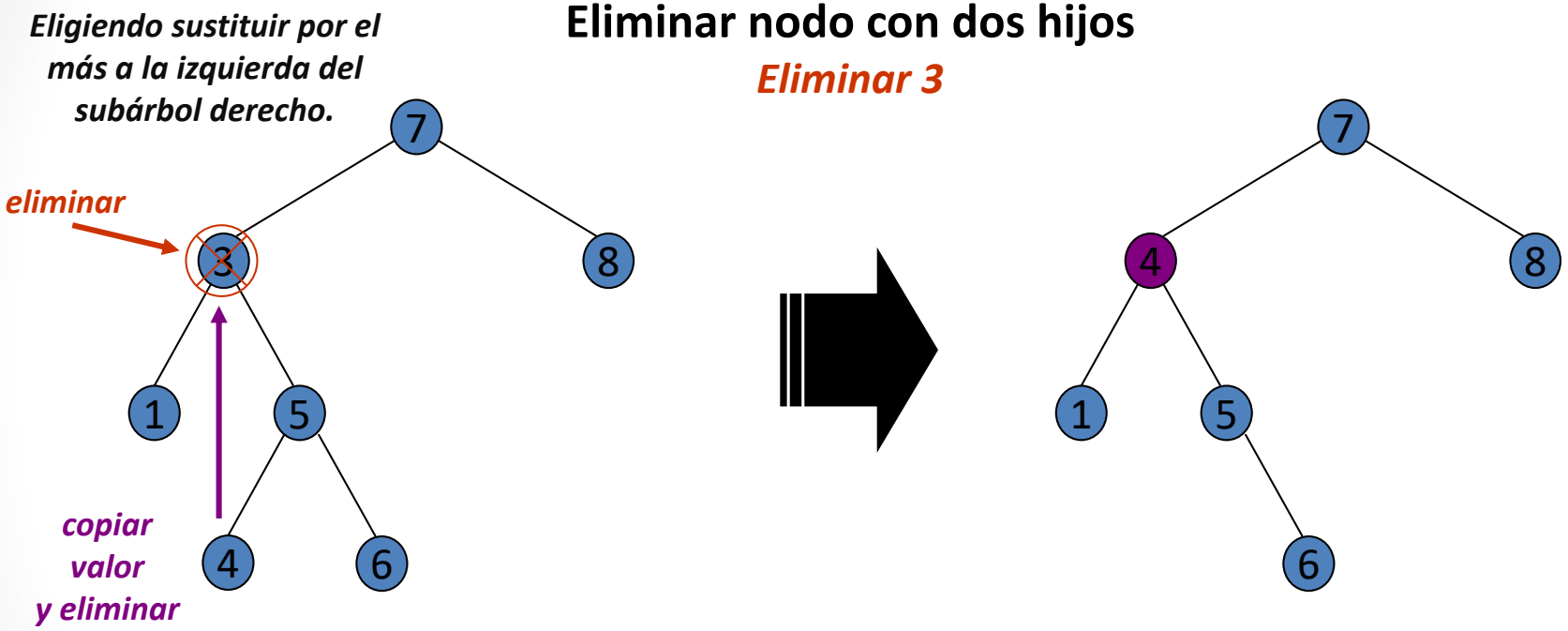
- **Caso 3**
  - Si el elemento a borrar tiene los dos hijos, entonces se tienen que sustituir por el nodo:

    1. Que se encuentra mas a la derecha en el subárbol izquierdo, **o por el nodo**

    2. Que se encuentra mas a la izquierda en el subárbol derecho.

       - **Nota**: Si el nodo sustituto tuviera un hijo, lo colocamos en lugar de este.

Árbol binario de búsqueda
(Eliminación: Ejemplo 01)



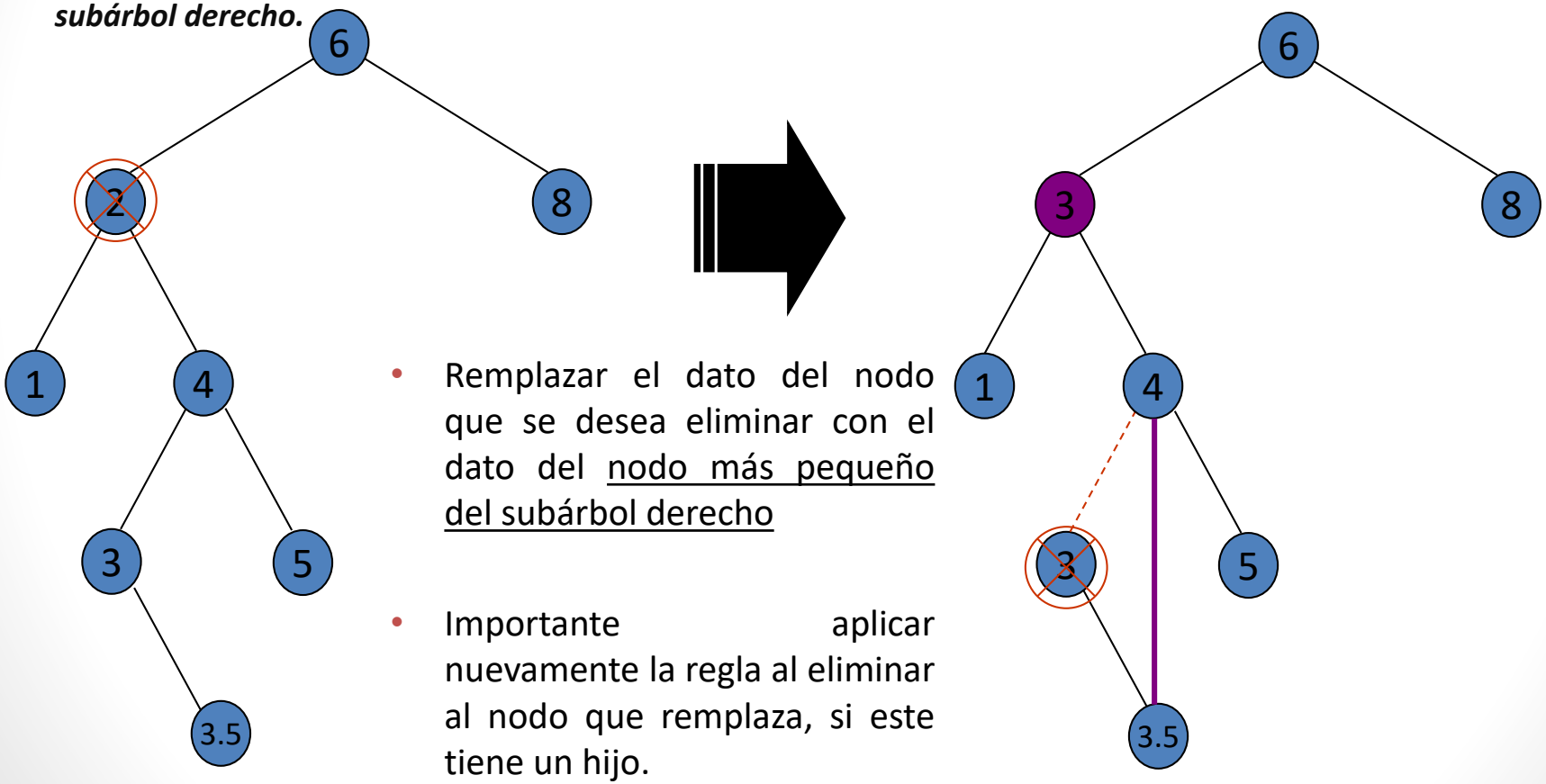
- Remplazar el dato del nodo que se desea eliminar con el dato del nodo más pequeño del subárbol derecho
- Después, eliminar el nodo más pequeño del subárbol derecho (caso fácil)

Árbol binario de búsqueda (Eliminación)

Eligiendo sustituir por el más a la izquierda del subárbol derecho.

Eliminar nodo con dos hijos

Eliminar 2



- Remplazar el dato del nodo que se desea eliminar con el dato del nodo más pequeño del subárbol derecho
- Importante aplicar nuevamente la regla al eliminar al nodo que remplaza, si este tiene un hijo.

Operaciones útiles en un ABB (Contar nodos del ABB)

- El conocer el número de nodos en un árbol binario de búsqueda puede resultar útil para distintas aplicaciones de los mismos.

Si retomamos las operaciones hechas en C del árbol binario…

```
//Cuenta los nodos que hay a partir de una posición P
int CuentaNodos(arbol *a,posicion p)
{
   if(NullNode(a,p))
   {
      return 0;
   }
   else
   {
      return (1+ CuentaNodos(a,RightSon(a,p)) + cuentaNodos(a,LeftSon(a,p));
   }
}
```

Operaciones útiles en un ABB
(Sumar las claves de los nodos del ABB)

- El conocer la suma de los valores de las claves de los nodos en un árbol binario de búsqueda puede resultar útil para distintas aplicaciones de los mismos.

```
//Suma  el valor de las claves de los nodos a partir de una posición
int SumaNodo(arbol *a, posicion p)
{
        elemento e;
        if(NullNode(a,p))
        {
                return 0;
        }
        else
        {
                e=ReadNode(a,p);
                return e.clave+SumaNodo(a,RightSon(a,p))+SumaNodo(a,LeftSon(a,p));
        }
}
```

Operaciones útiles en un ABB
(Calcular la profundidad del ABB)

- El conocer la profundidad de un árbol o subárbol binario de búsqueda puede resultar útil para distintas aplicaciones de los mismos.

```
//Calcula la profundidad de un árbol a partir de una posicion p
int Profundidad(arbol *a, posicion p)
{
        if(NullNode(a,p))
                return 0;
        //Retornar la mayor profundidad (Derecha o Izquierda)
        if (Profundidad(a,RightSon(a,p)) > Profundidad(a,LeftSon(a,p)))
                return Profundidad(a,RightSon(a,p)) + 1;
        else
                return Profundidad(a,LeftSon(a,p) + 1;
}
```

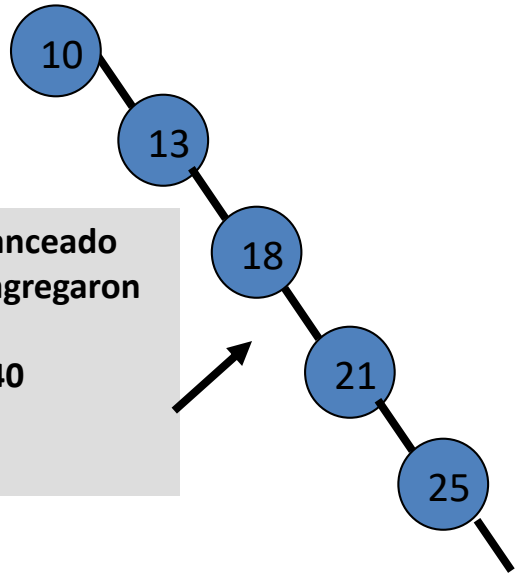
Operaciones útiles en un ABB
(Contar las hojas del ABB)

- El contar las hojas de un árbol binario de búsqueda puede resultar útil para distintas aplicaciones de los mismos.

```
//Cuenta  hojas de un árbol a partir de una posición p
int ContarHojas(arbol *a, posicion p)
{
          if(NullNode(a,p))
                    return 0;
          else if(NullNode(a,LeftSon(a,p))) && (NullNode(a,RightSon(a,p)))
                    return 1;
          else
                    return ContarHojas(a,LeftSon(a,p)))+ContarHojas(a,RightSon(a,p)));
}
```

Comentarios importantes

- El orden de inserción de los datos, determina la forma del ABB.
- Se tendrá el peor caso si se insertan los datos en forma ordenada
- La forma del ABB determina la eficiencia del proceso de búsqueda.



Este árbol está desbalanceado porque los valores se agregaron en el siguiente orden:
10, 13, 18, 21, 25, 33, 40

Liga Web

- Inserta elimina y busca en un árbol binario

https://www.cs.usfca.edu/~galles/visualization/BST.html