Unidad 5: Árboles

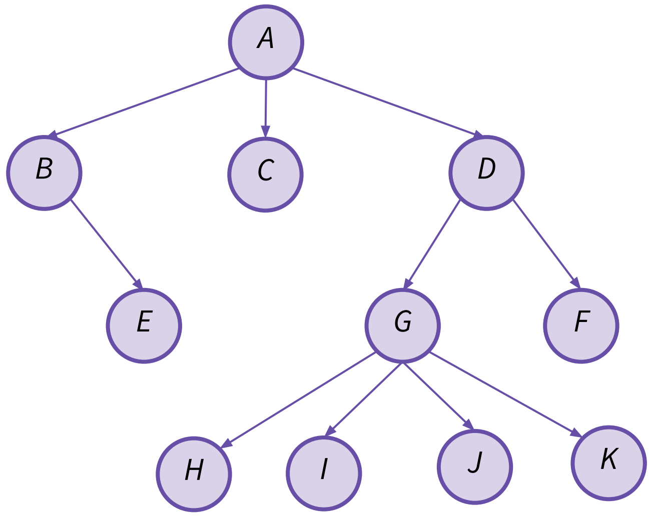
Algoritmos y Estructuras de Datos

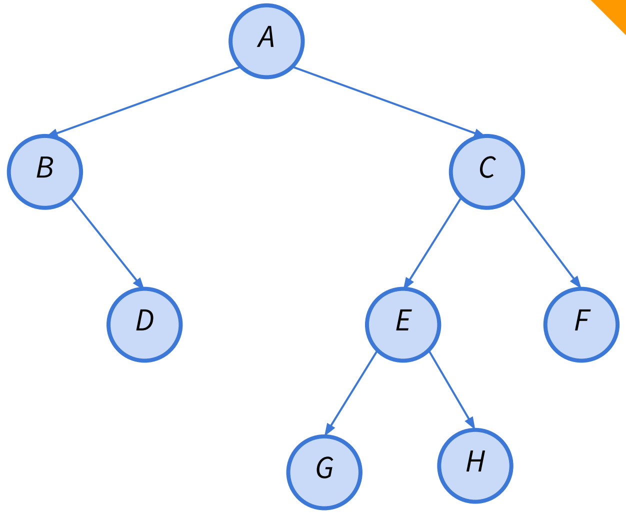
Concepto y propiedades
Árboles Binarios
Árboles Binarios de búsqueda

Ing. Juan Ignacio Iturriaga

Árboles - Concepto y propiedades

- Raíz
- Padre
- Hijo
- Rama
- Hoja
- Profundidad
- Nivel
- Hermano
- Descendiente
- Grado



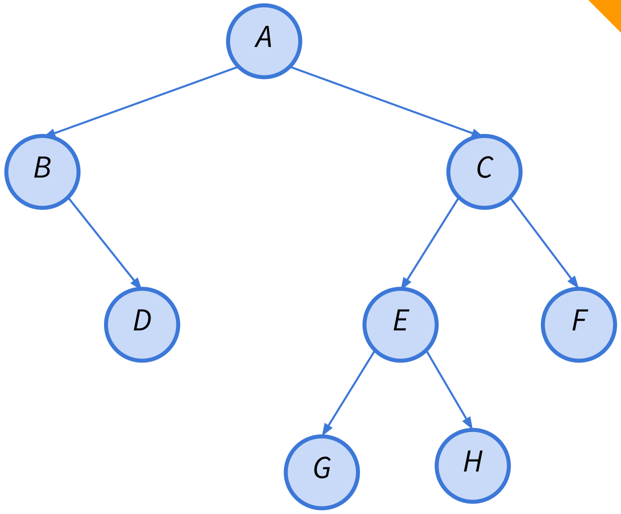


Árboles binarios - Implementación

```
btn {
    datatype value
    btn left
    btn right
}
```

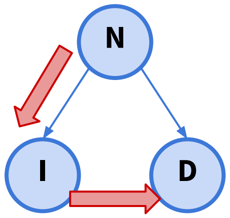
C

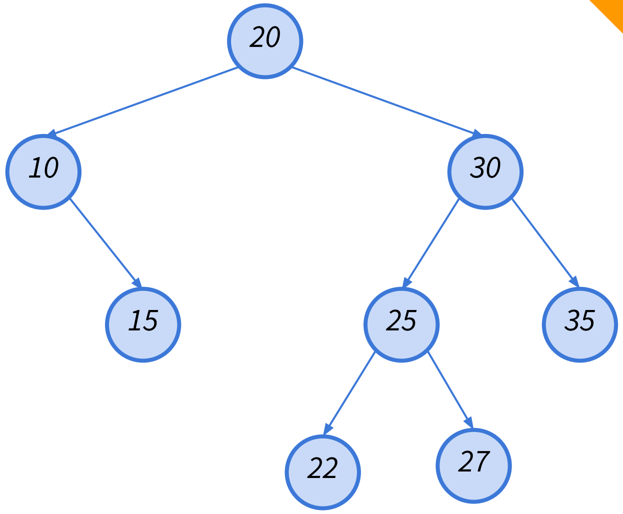
```c
struct btn {  // btn = binary tree node
    int value;
    struct btn *left;
    struct btn *right;
};
```



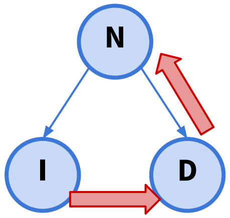
Árboles binarios - Recorrido

Pre-Order

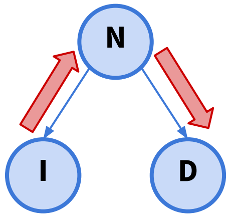




Post-Order



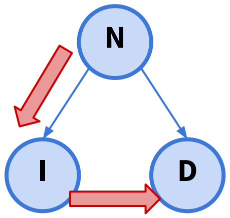
In-Order

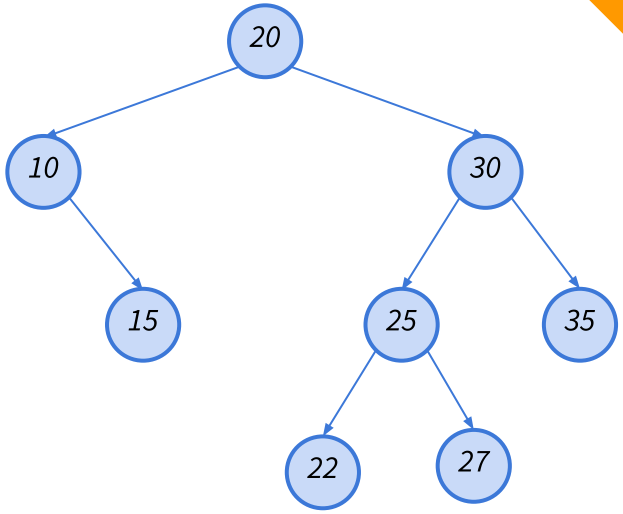


Árboles binarios - Recorrido

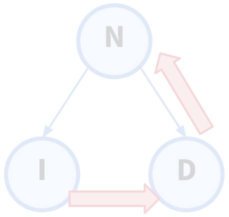
Pre-Order

```
void preOrder(btn node) {



}
```

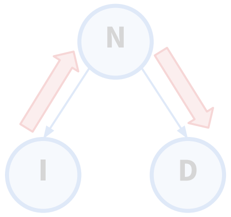




Post-Order



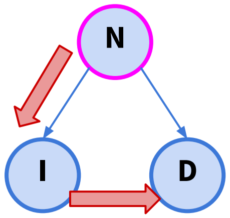
In-Order

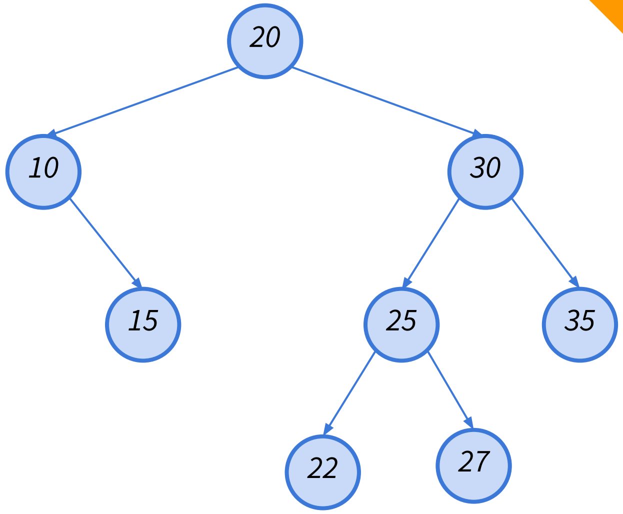


Árboles binarios - Recorrido

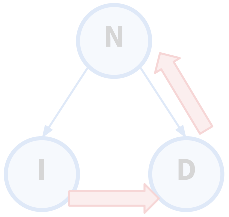
Pre-Order

```
void preOrder(btn node) {

            printNode(node)



}
```

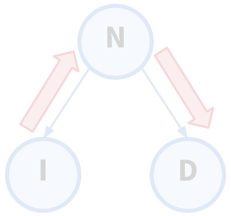




Post-Order



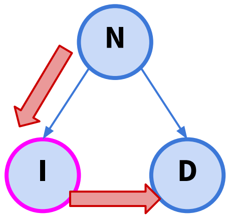
In-Order

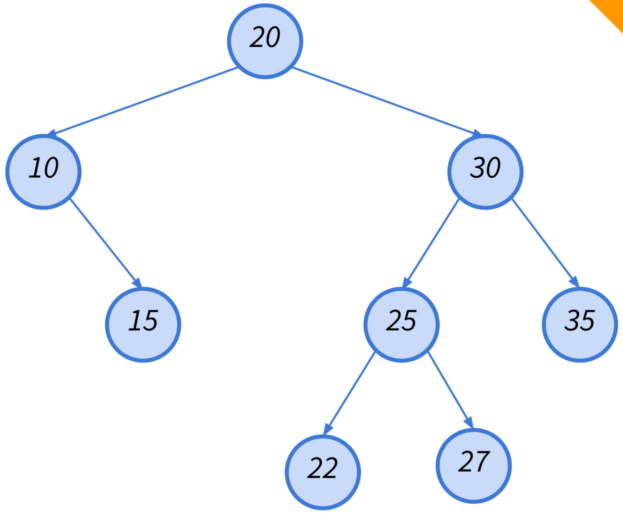


Árboles binarios - Recorrido

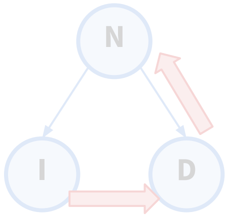
Pre-Order

```
void preOrder(btn node) {

            printNode(node)
            preOrder(node.left)


}
```

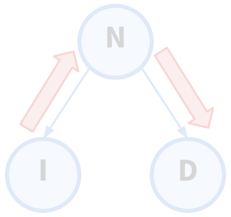




Post-Order



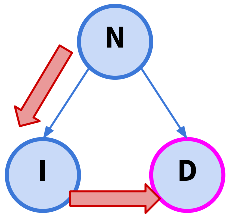
In-Order

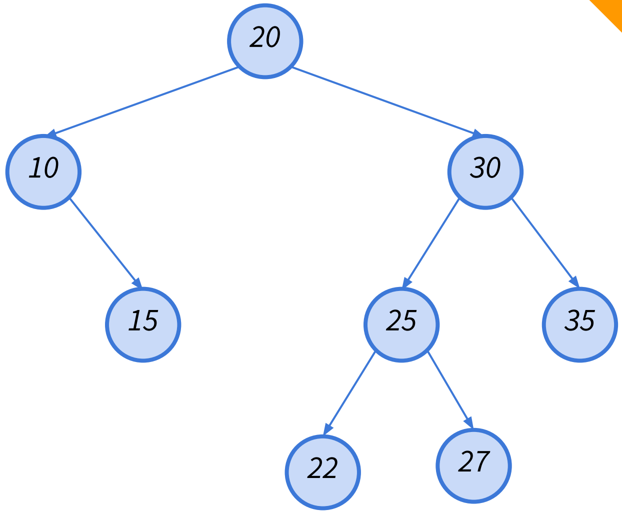


Árboles binarios - Recorrido

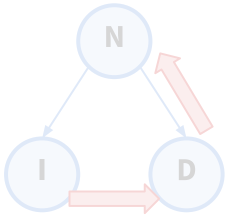
Pre-Order

```
void preOrder(btn node) {

           printNode(node)
           preOrder(node.left)
           preOrder(node.right)

}
```

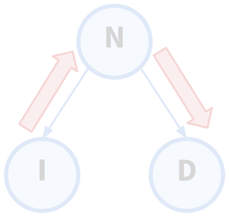




Post-Order



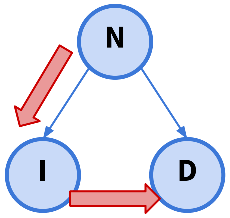
In-Order

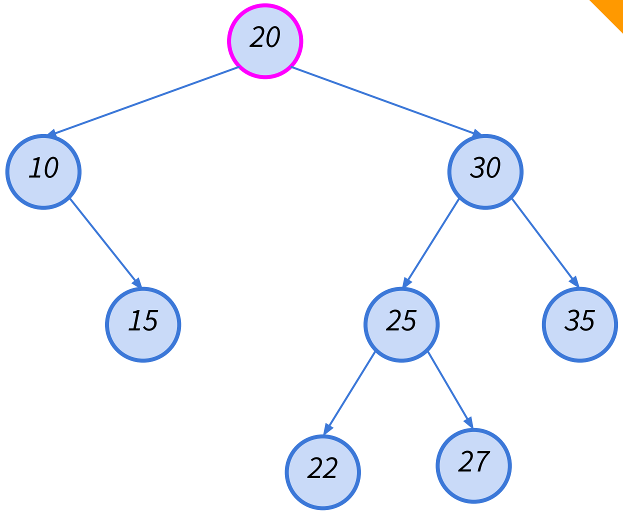


Árboles binarios - Recorrido

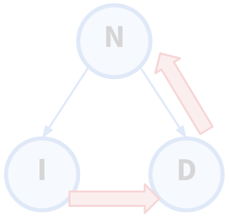
Pre-Order

```
void preOrder(btn node) {

            printNode(node)
            preOrder(node.left)
            preOrder(node.right)

}
```

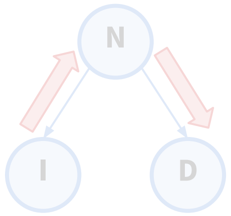




Post-Order



In-Order

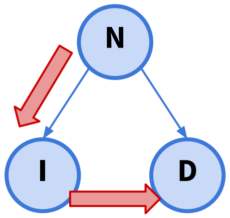


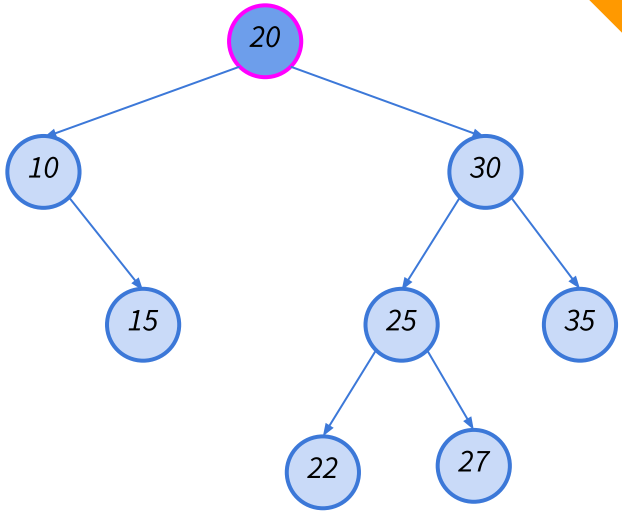
Árboles binarios - Recorrido

Pre-Order

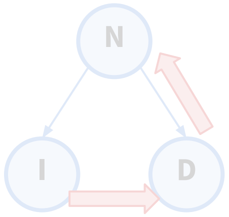
```
void preOrder(btn node) {

          printNode(node)
          preOrder(node.left)
          preOrder(node.right)

}
```

→ 20

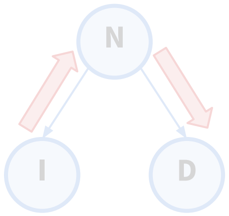




Post-Order



In-Order

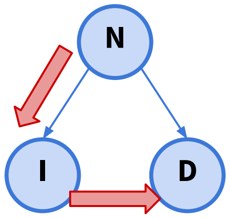


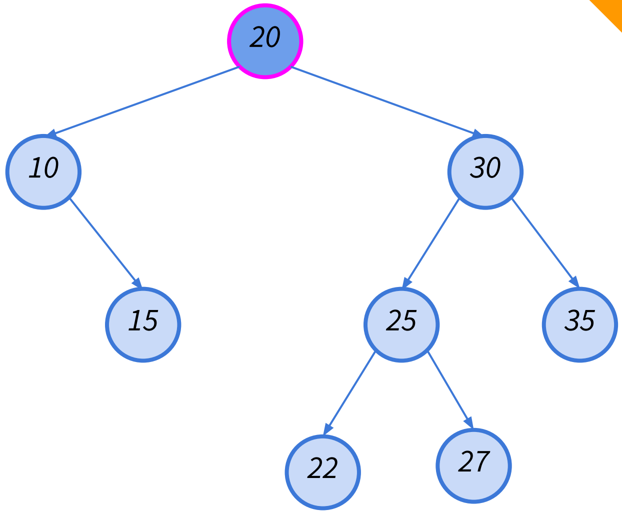
Árboles binarios - Recorrido

Pre-Order

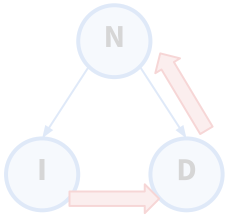
```
void preOrder(btn node) {

        printNode(node)
        preOrder(node.left)
        preOrder(node.right)

}
```

→ 20

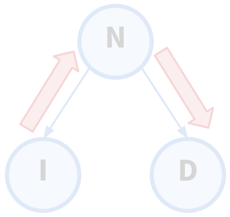




Post-Order



In-Order

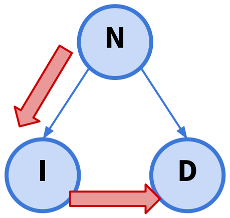


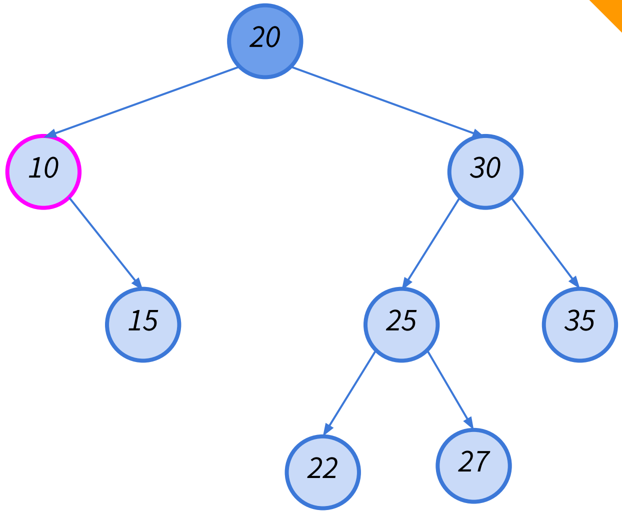
Árboles binarios - Recorrido

Pre-Order

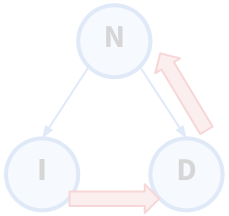
```
void preOrder(btn node) {

            printNode(node)
            preOrder(node.left)
            preOrder(node.right)

}
```

→ 20

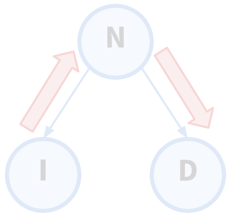




Post-Order



In-Order

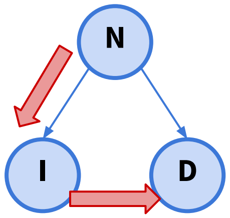


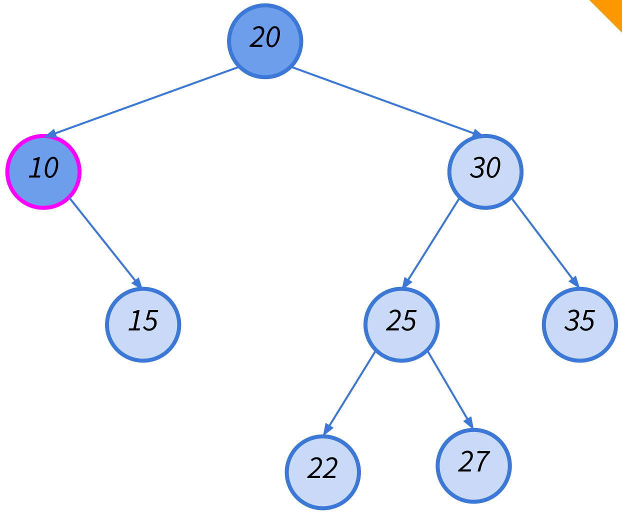
Árboles binarios - Recorrido

Pre-Order

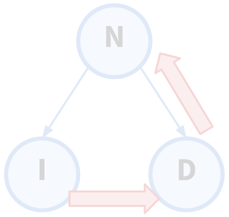
```
void preOrder(btn node) {

            printNode(node)
            preOrder(node.left)
            preOrder(node.right)

}
```

→ 20 10

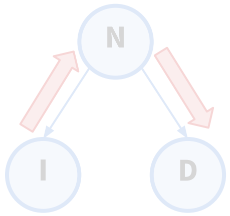




Post-Order



In-Order

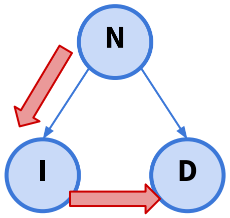


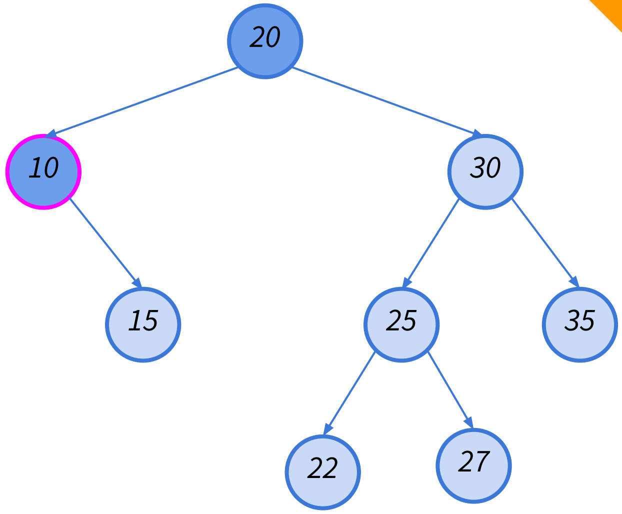
Árboles binarios - Recorrido

Pre-Order

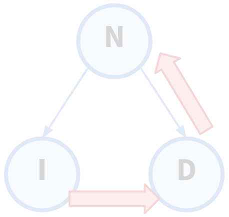
```
void preOrder(btn node) {

            printNode(node)
            preOrder(node.left)
            preOrder(node.right)

}
```

→ 20 10

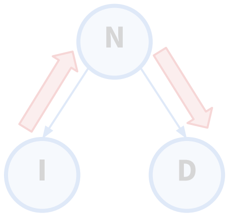




Post-Order



In-Order

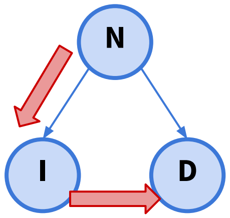


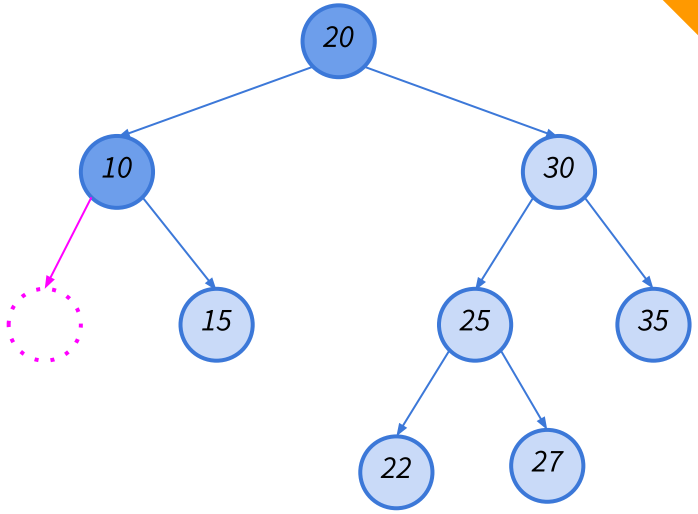
Árboles binarios - Recorrido

Pre-Order

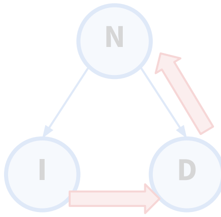
```
void preOrder(btn node) {

            printNode(node)
            preOrder(node.left)
            preOrder(node.right)

}
```

→ 20 10

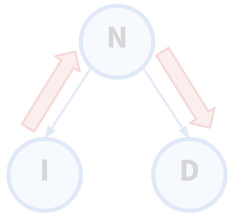




Post-Order



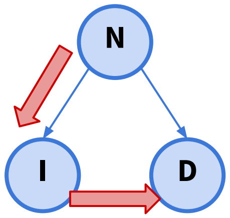
In-Order

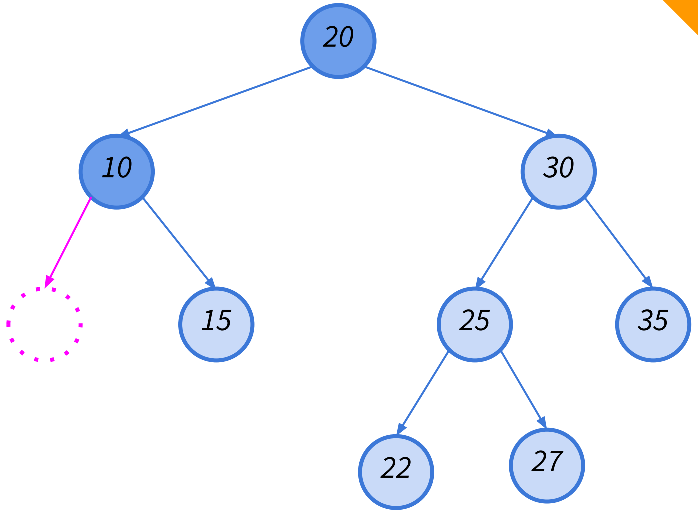


Árboles binarios - Recorrido

Pre-Order

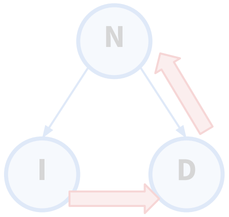
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```



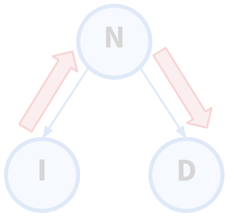


→ 20 10

Post-Order



In-Order



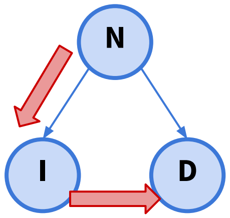
Árboles binarios - Recorrido

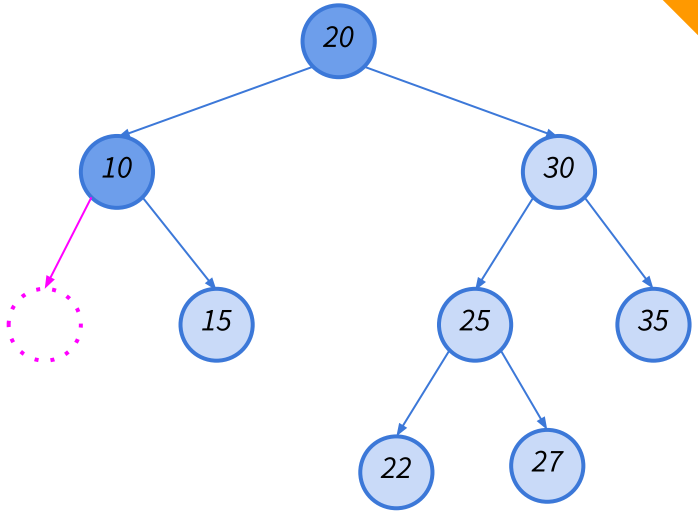
Pre-Order

node != NULL

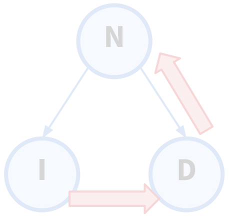
```
void preOrder(btn node) {
        if (notEmpty(node)){
                printNode(node)
                preOrder(node.left)
                preOrder(node.right)
        }
}
```

→ 20 10

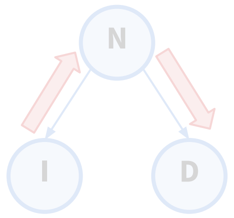




Post-Order



In-Order

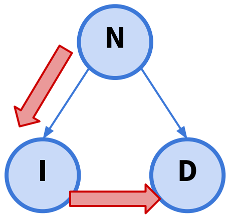


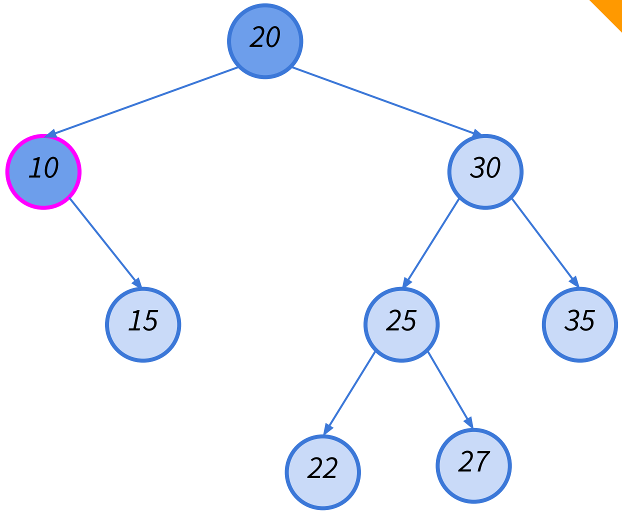
Árboles binarios - Recorrido

Pre-Order

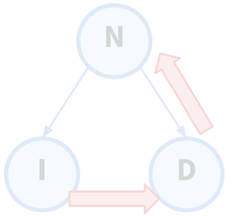
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10

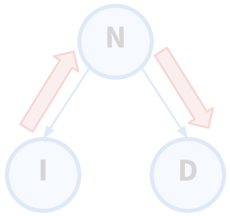




Post-Order



In-Order

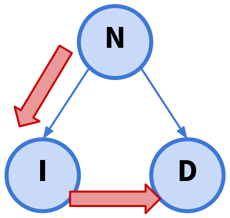


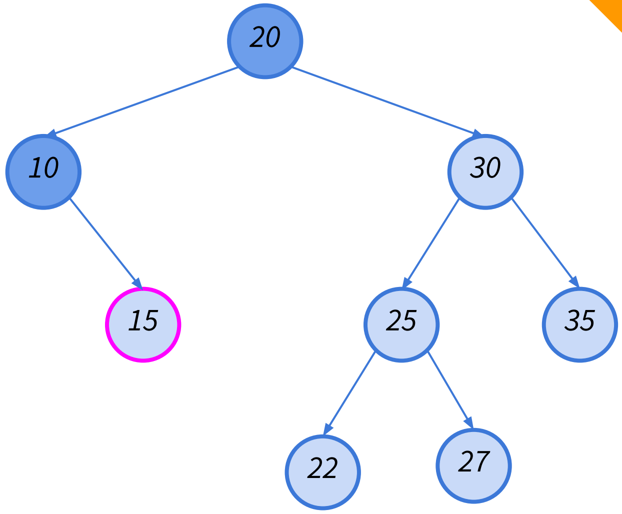
Árboles binarios - Recorrido

Pre-Order

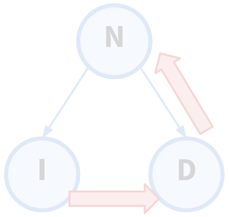
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10

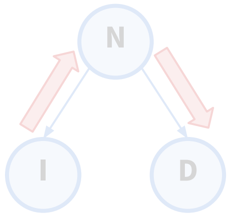




Post-Order



In-Order

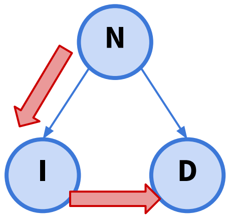


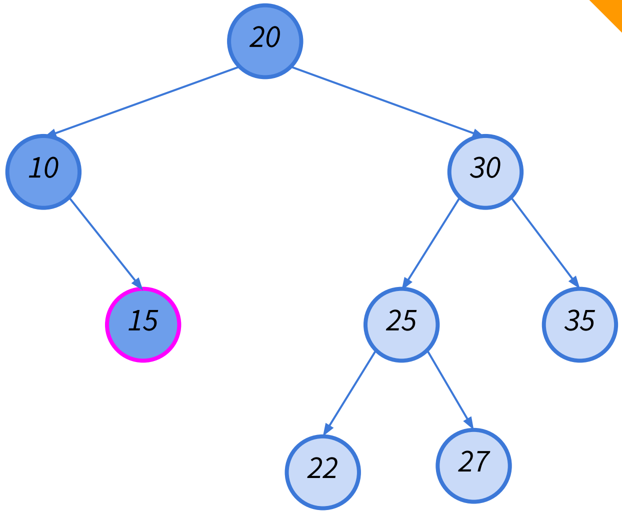
Árboles binarios - Recorrido

Pre-Order

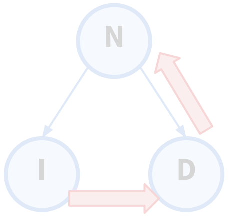
```
void preOrder(btn node) {
       if (notEmpty(node)){
              printNode(node)
              preOrder(node.left)
              preOrder(node.right)
       }
}
```

→ 20 10 15

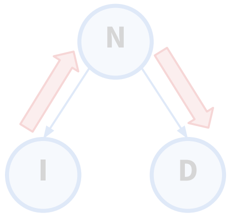




Post-Order



In-Order

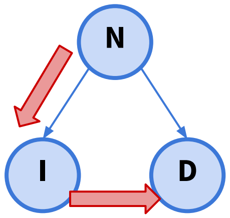


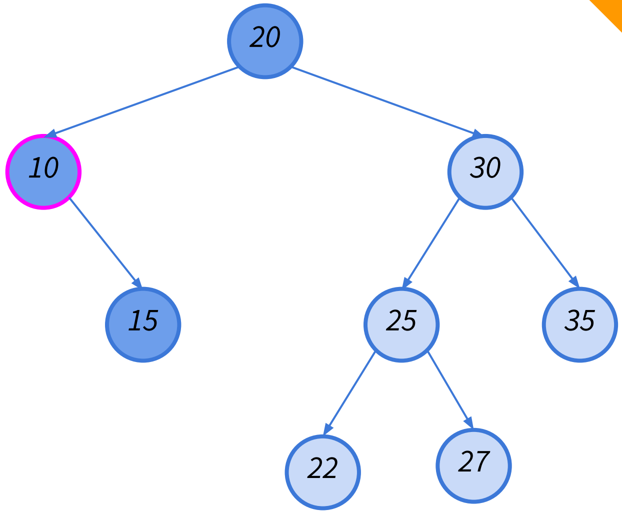
Árboles binarios - Recorrido

Pre-Order

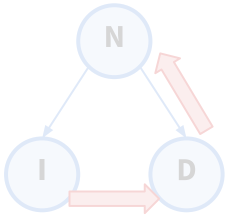
```
void preOrder(btn node) {
        if (notEmpty(node)){
                printNode(node)
                preOrder(node.left)
                preOrder(node.right)
        }
}
```

→ 20 10 15

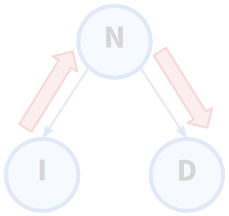




Post-Order



In-Order

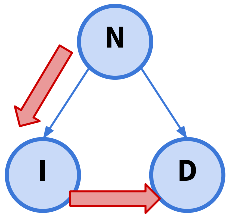


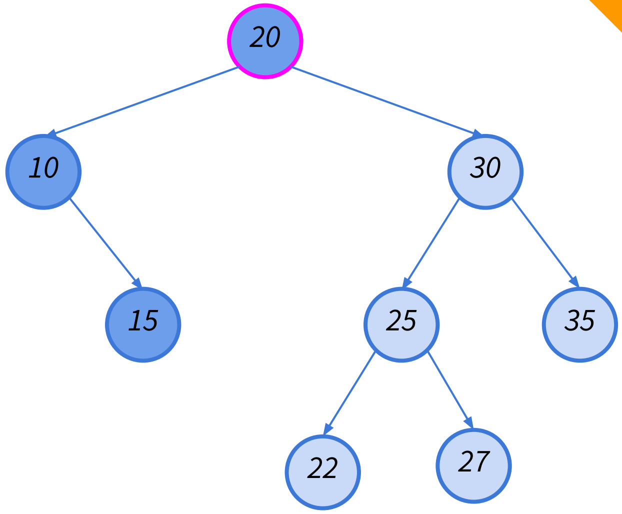
Árboles binarios - Recorrido

Pre-Order

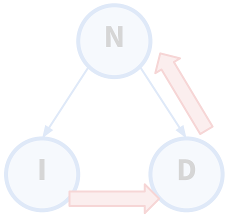
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15

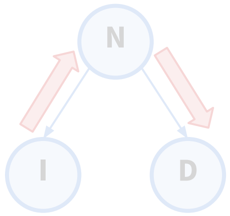




Post-Order



In-Order

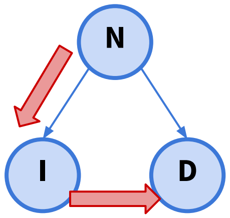


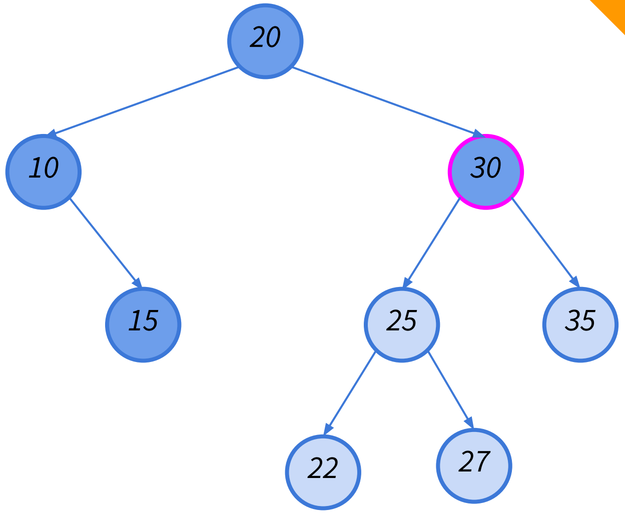
Árboles binarios - Recorrido

Pre-Order

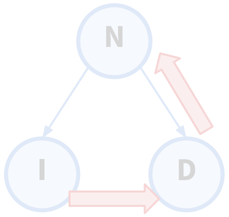
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30

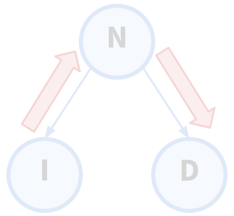




Post-Order



In-Order

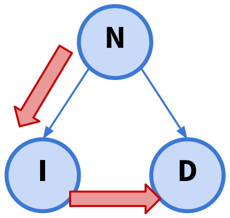


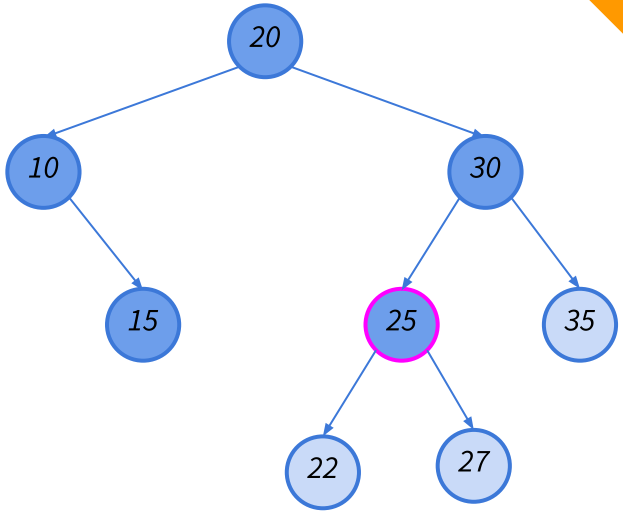
Árboles binarios - Recorrido

Pre-Order

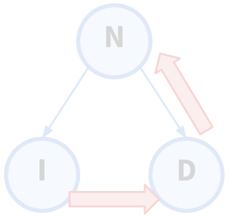
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25

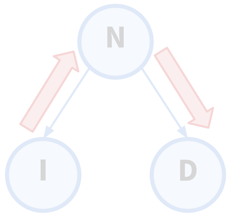




Post-Order



In-Order

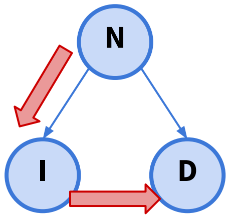


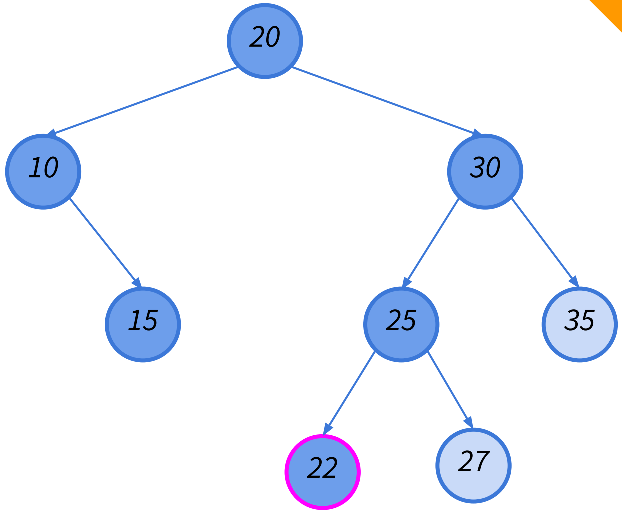
Árboles binarios - Recorrido

Pre-Order

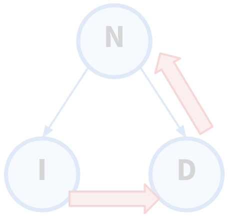
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22

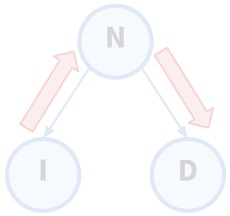




Post-Order



In-Order

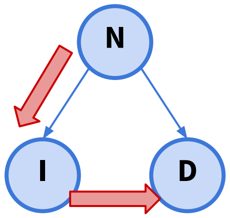


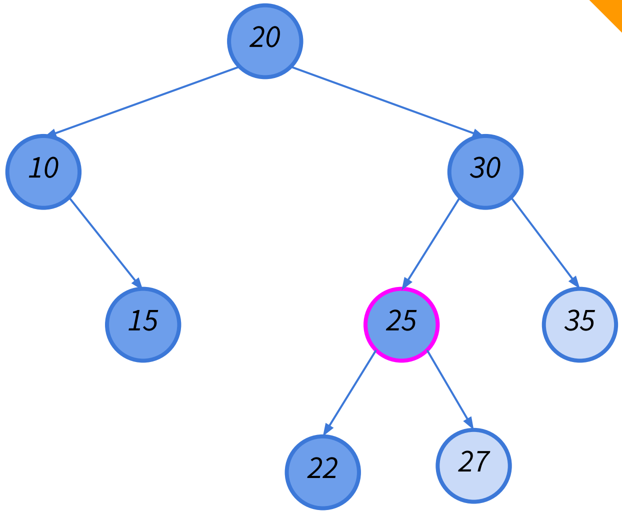
Árboles binarios - Recorrido

Pre-Order

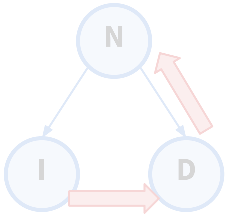
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22

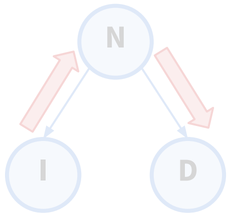




Post-Order



In-Order

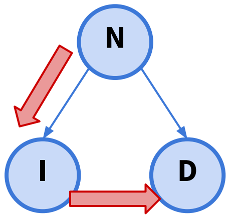


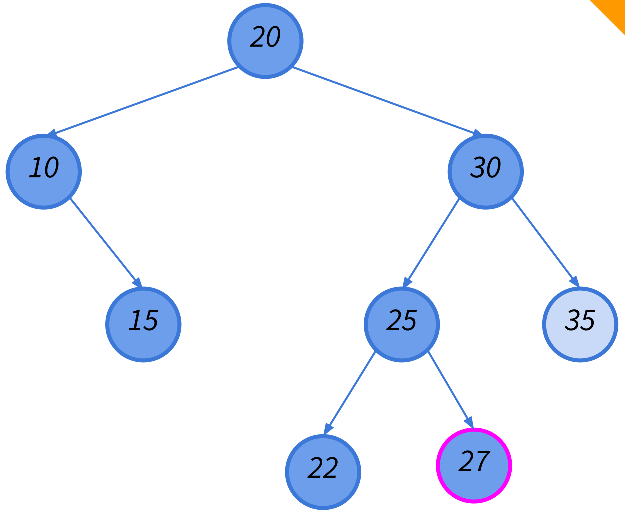
Árboles binarios - Recorrido

Pre-Order

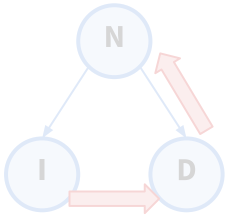
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27

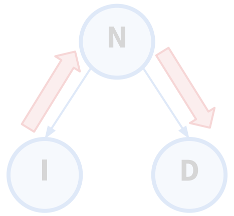




Post-Order



In-Order

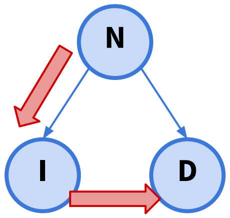


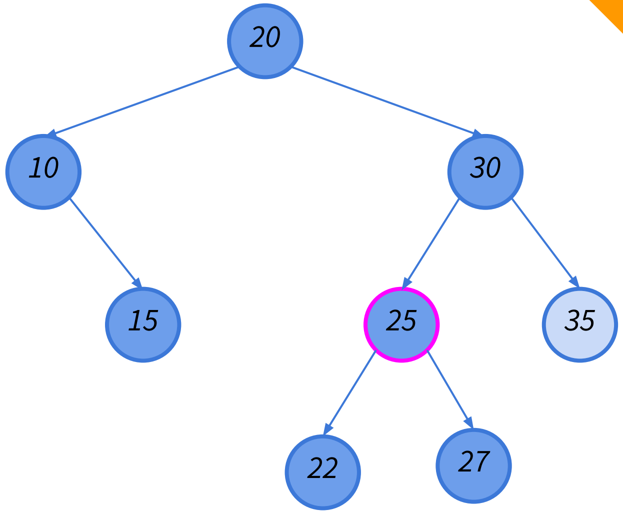
Árboles binarios - Recorrido

Pre-Order

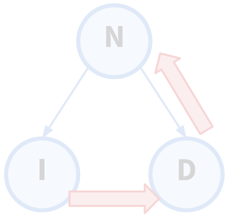
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27

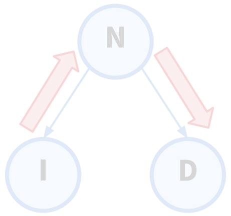




Post-Order



In-Order

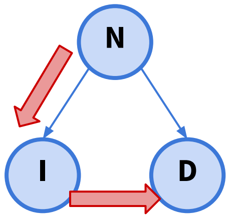


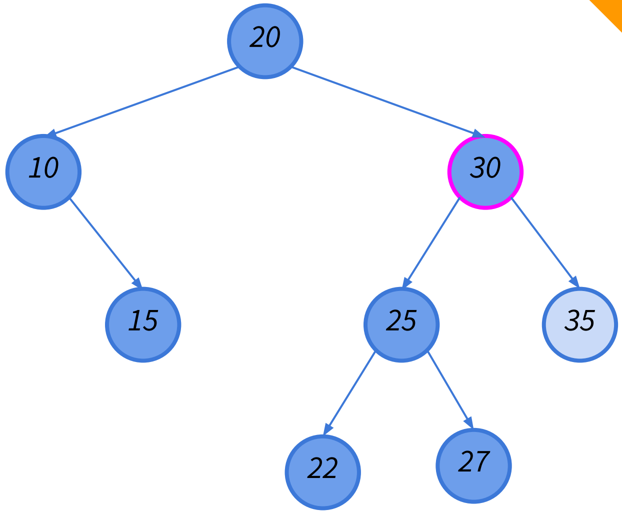
Árboles binarios - Recorrido

Pre-Order

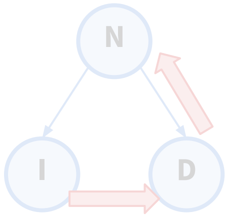
```
void preOrder(btn node) {
       if (notEmpty(node)){
              printNode(node)
              preOrder(node.left)
              preOrder(node.right)
       }
}
```

→ 20 10 15 30 25 22 27

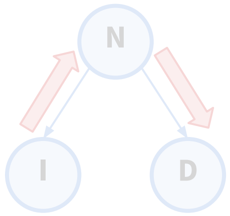




Post-Order



In-Order

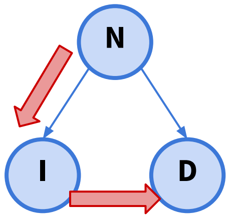


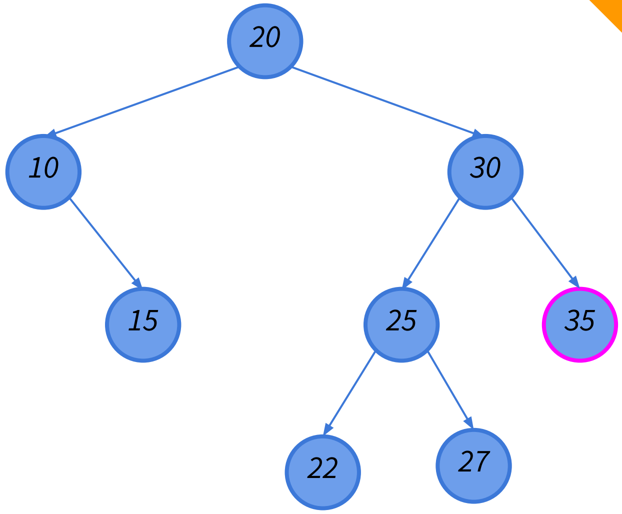
Árboles binarios - Recorrido

Pre-Order

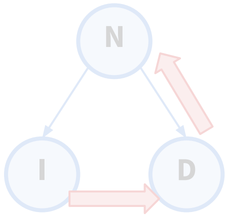
```
void preOrder(btn node) {
       if (notEmpty(node)){
              printNode(node)
              preOrder(node.left)
              preOrder(node.right)
       }
}
```

→ 20 10 15 30 25 22 27 35

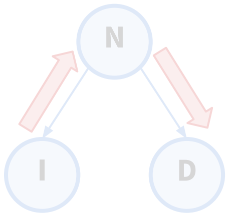




Post-Order



In-Order

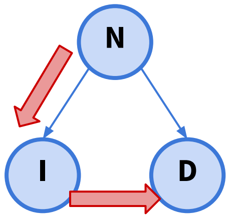


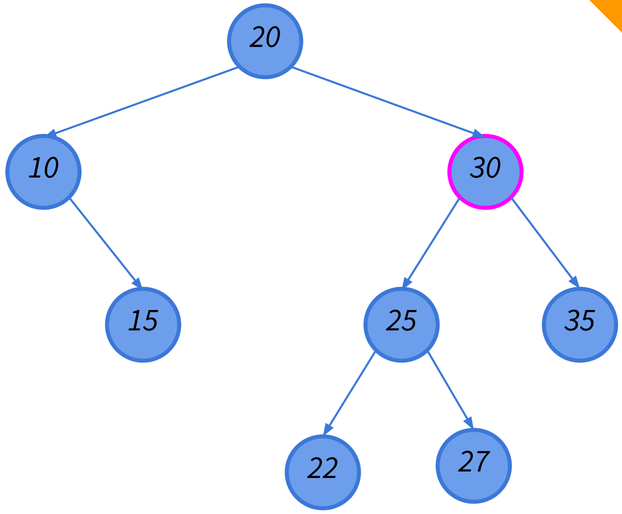
Árboles binarios - Recorrido

Pre-Order

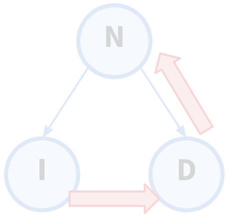
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

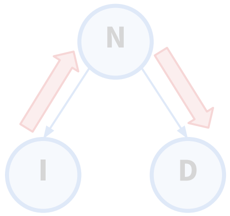




Post-Order



In-Order

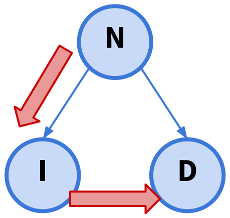


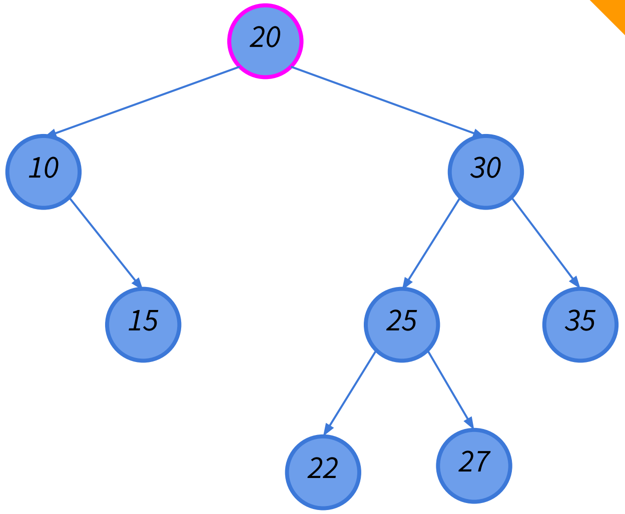
Árboles binarios - Recorrido

Pre-Order

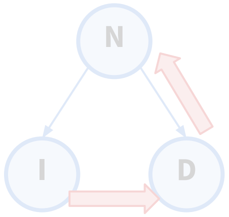
```
void preOrder(btn node) {
       if (notEmpty(node)){
              printNode(node)
              preOrder(node.left)
              preOrder(node.right)
       }
}
```

→ 20 10 15 30 25 22 27 35

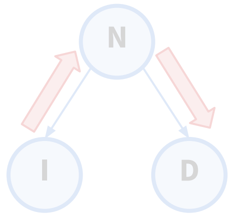




Post-Order



In-Order

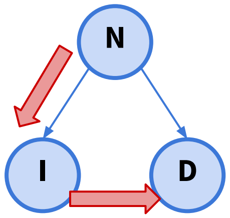


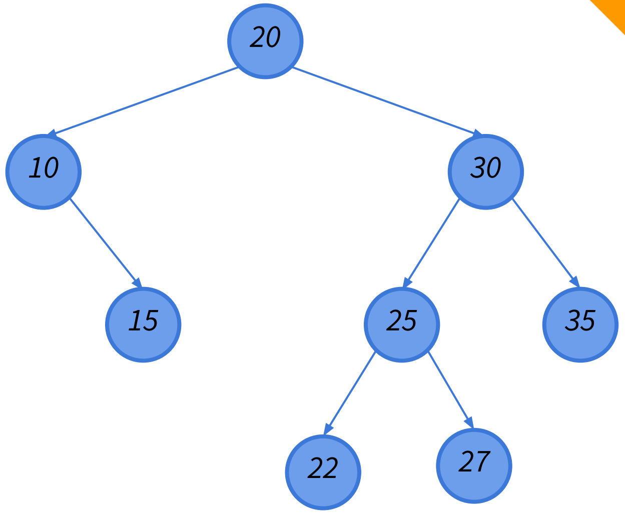
Árboles binarios - Recorrido

Pre-Order

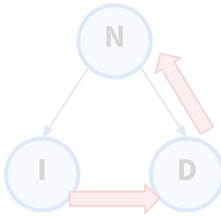
```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

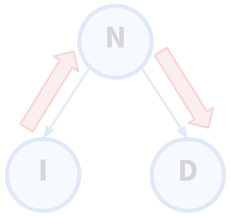




Post-Order



In-Order

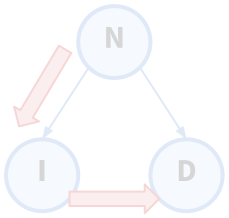


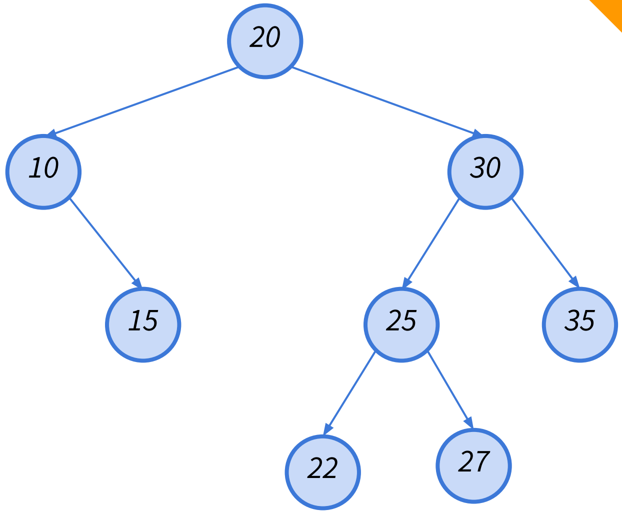
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

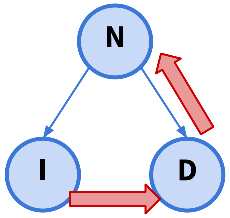




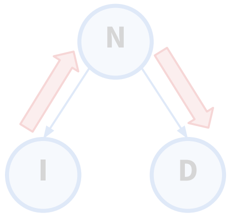
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→



In-Order

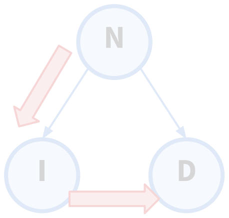


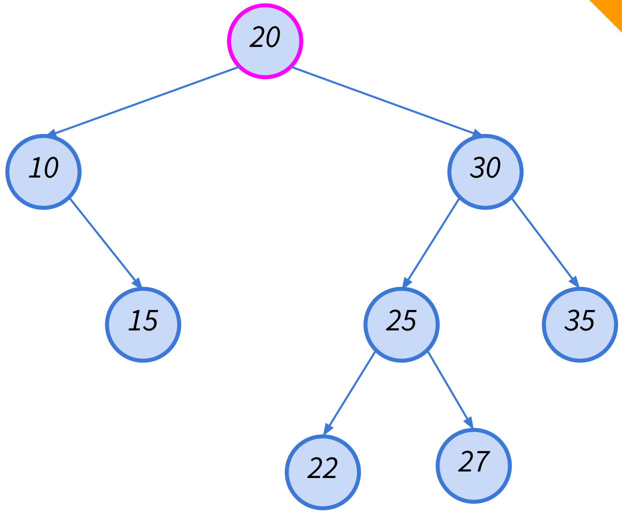
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

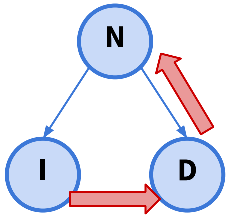




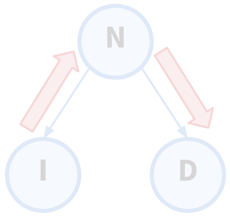
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→



In-Order

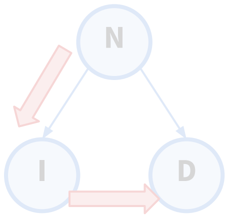


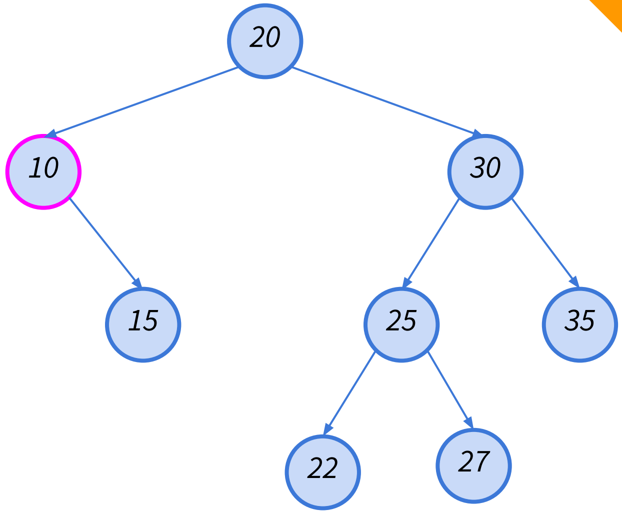
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

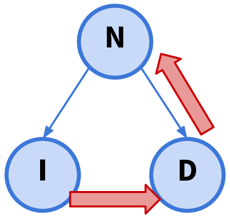




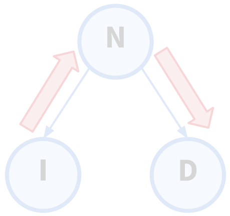
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→



In-Order

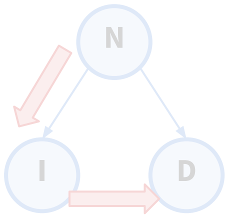


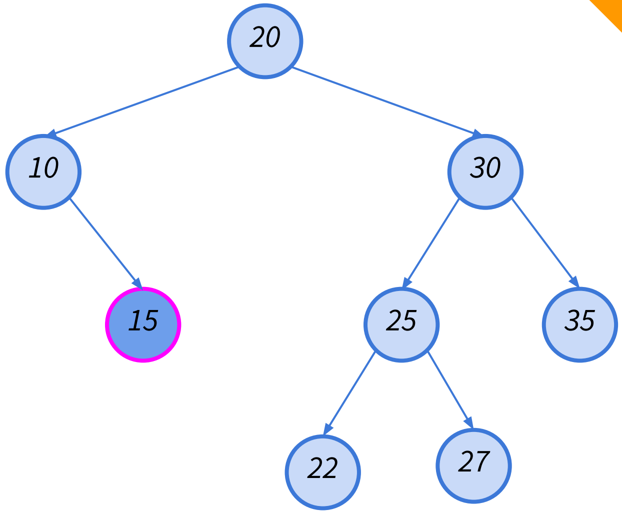
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

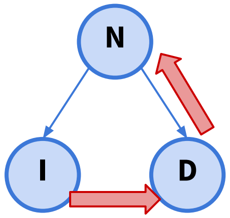




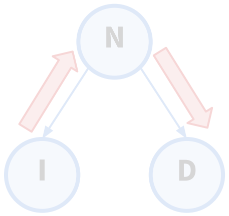
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15



In-Order

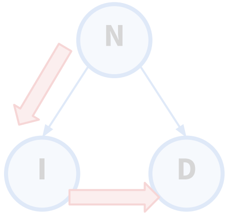


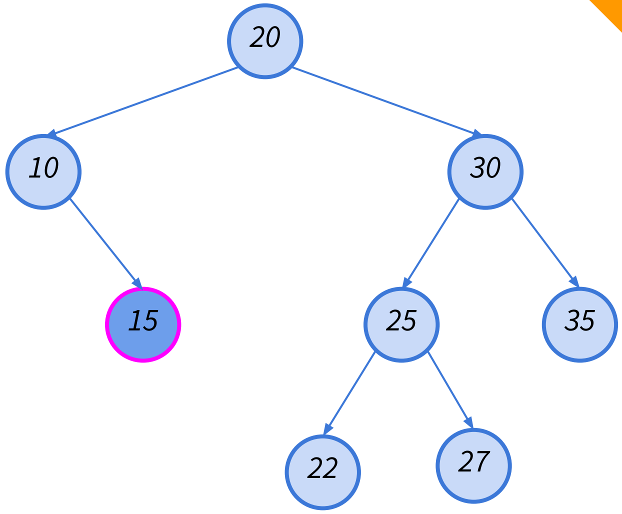
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
        if (notEmpty(node)){
                printNode(node)
                preOrder(node.left)
                preOrder(node.right)
        }
}
```

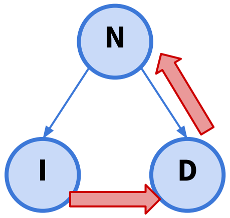
→ 20 10 15 30 25 22 27 35





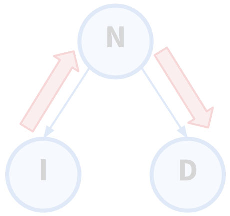
Post-Order

```
void postOrder(btn node) {
        if (notEmpty(node)){
                postOrder(node.left);
                postOrder(node.right);
                printNode(node);
        }
}
```



→ 15

In-Order

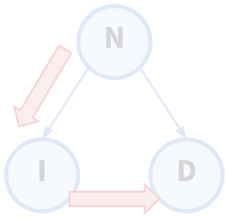


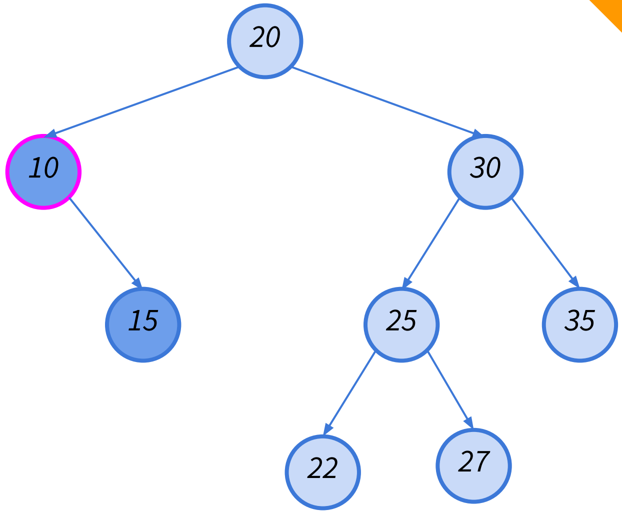
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

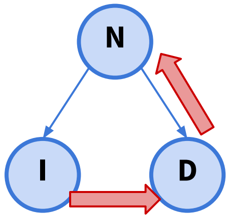




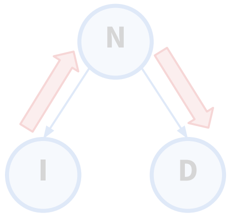
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10



In-Order

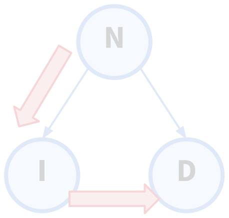


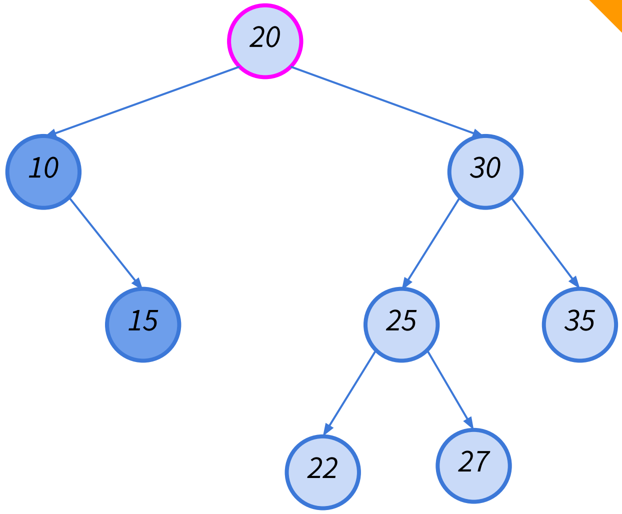
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

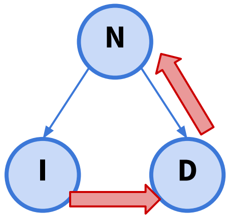




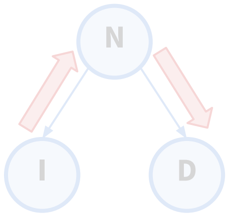
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10



In-Order

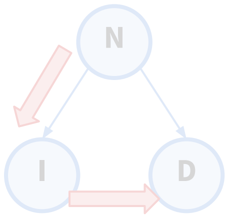


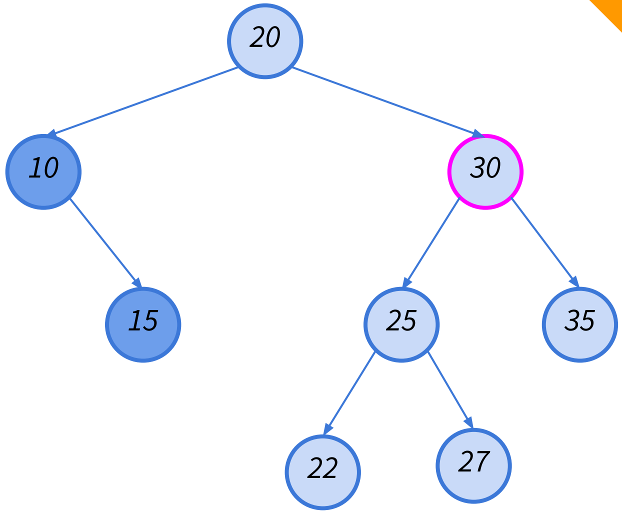
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

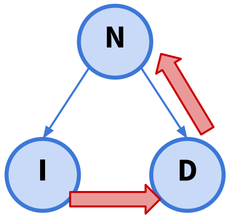




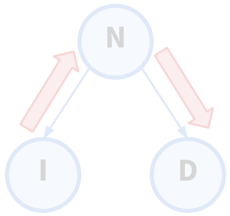
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10



In-Order

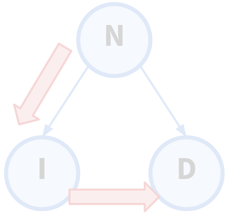


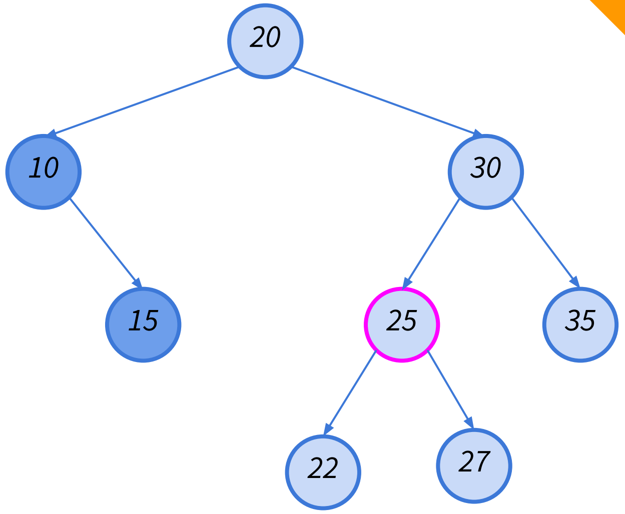
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

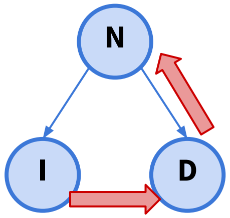




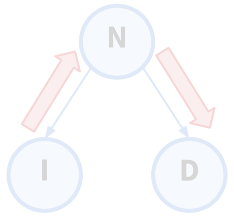
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10



In-Order

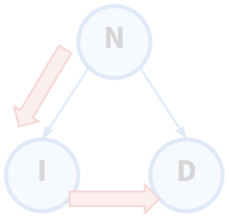


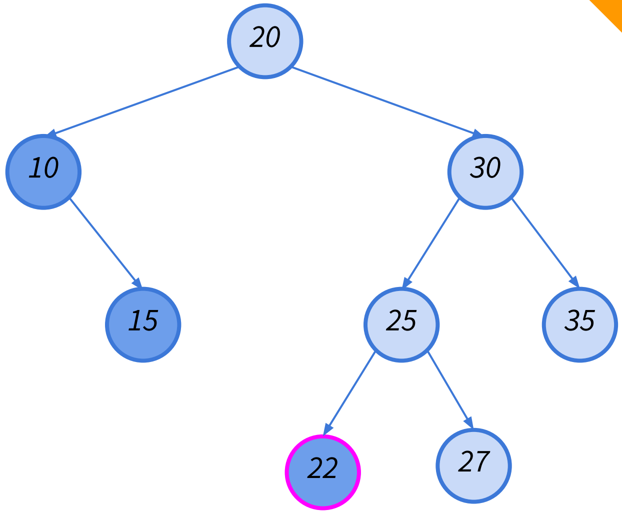
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

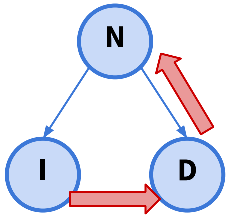




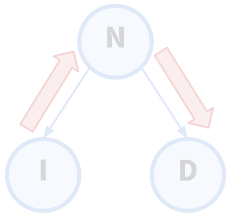
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22



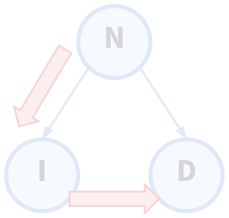
In-Order



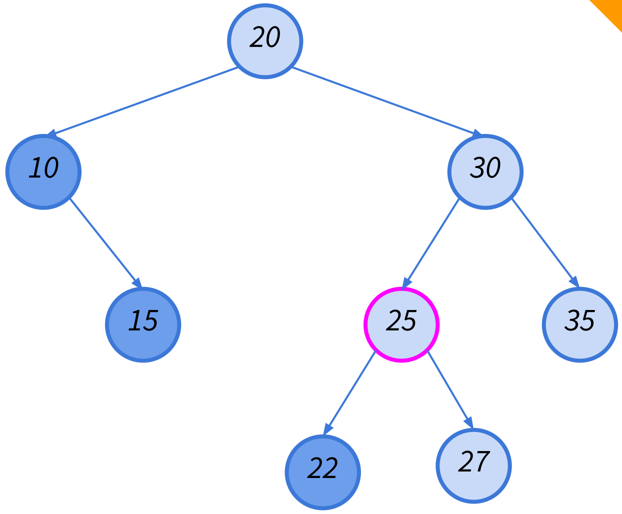
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

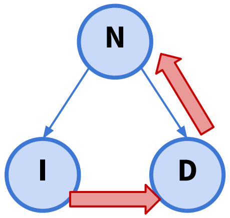


→ 20 10 15 30 25 22 27 35



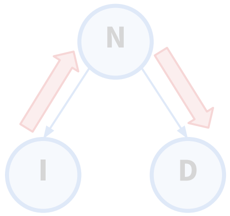
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```



→ 15 10 22

In-Order

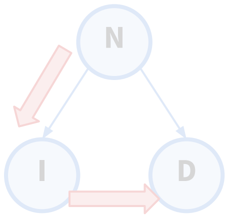


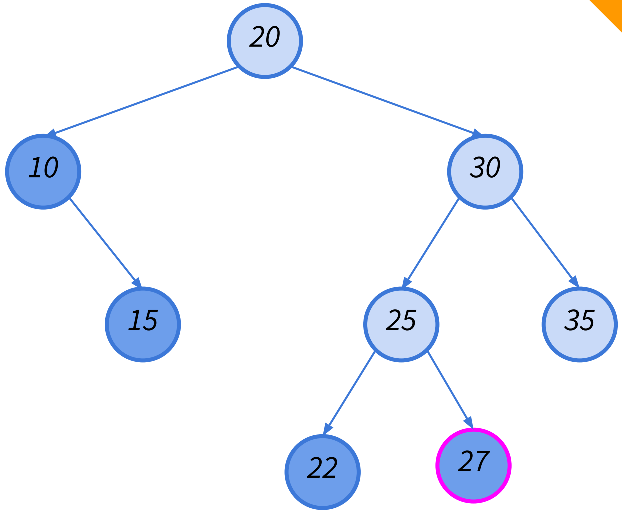
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

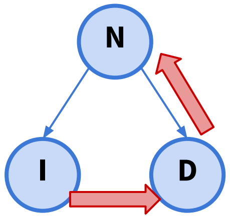




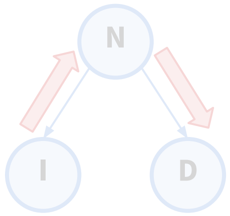
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22 27



In-Order

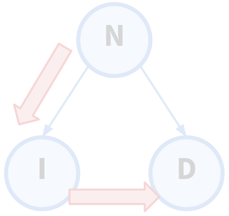


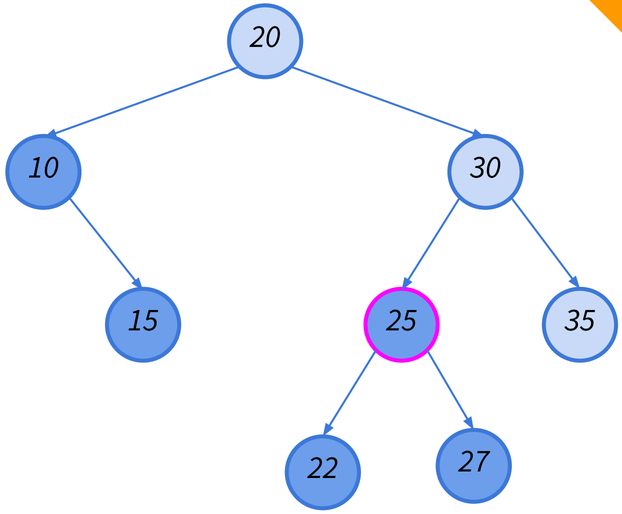
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

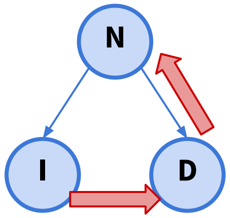




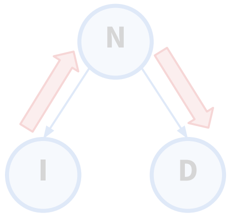
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22 27 25



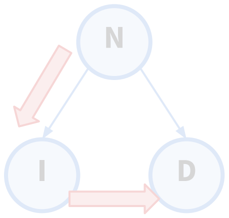
In-Order



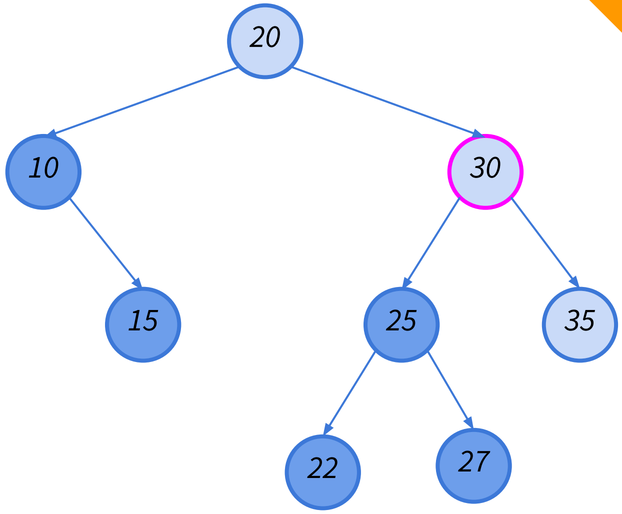
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

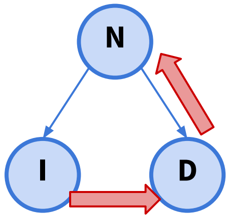


→ 20 10 15 30 25 22 27 35



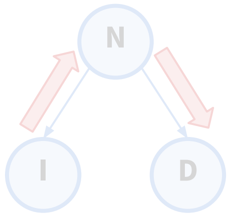
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```



→ 15 10 22 27 25

In-Order

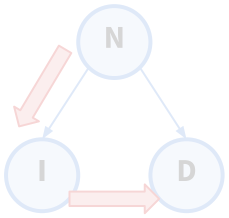


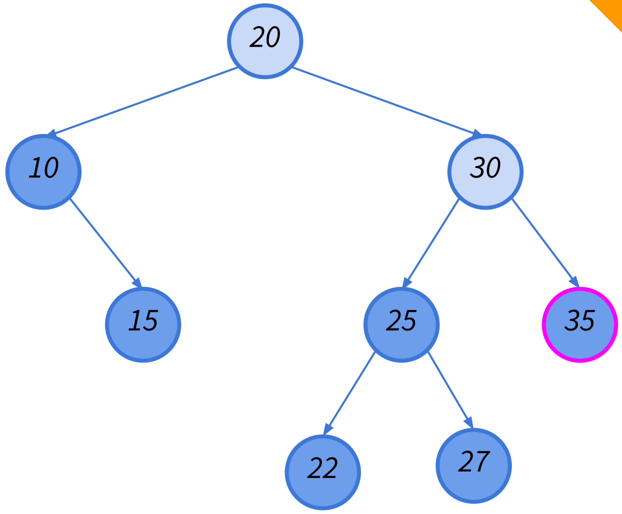
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

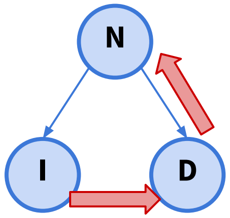




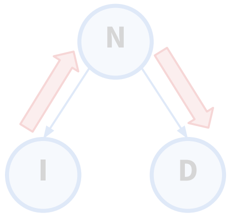
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22 27 25 35



In-Order

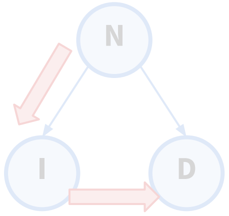


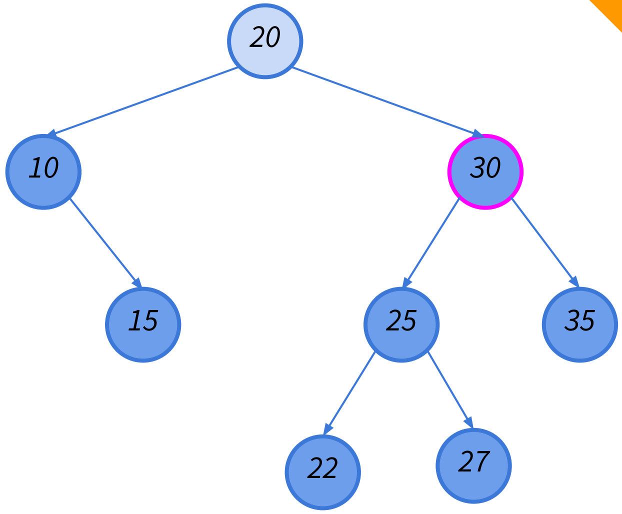
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

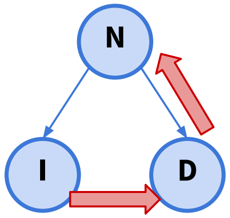




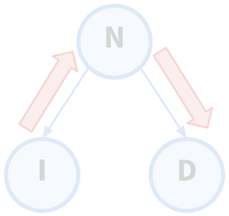
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22 27 25 35 30



In-Order

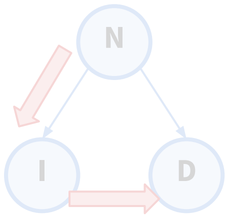


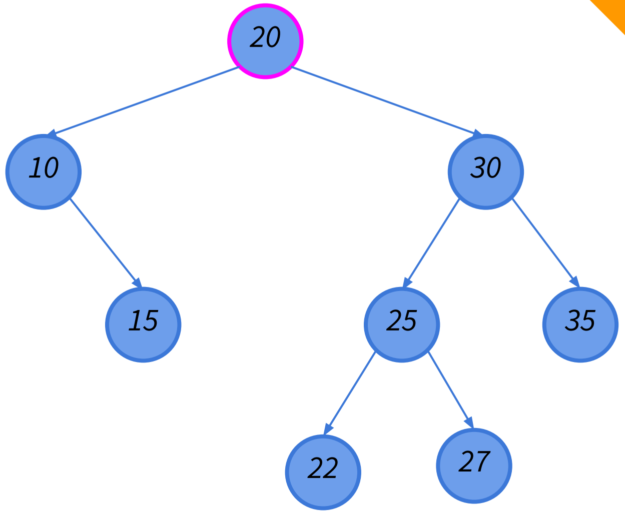
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

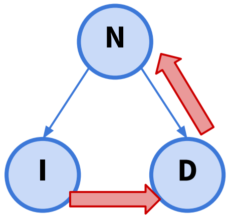




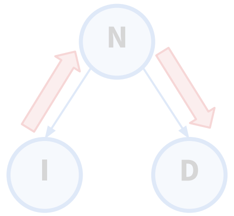
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22 27 25 35 30 20



In-Order

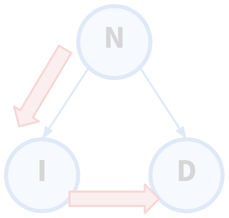


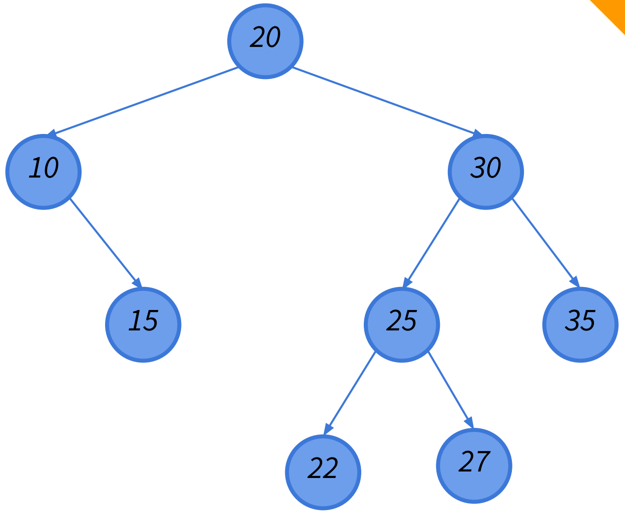
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
      if (notEmpty(node)){
            printNode(node)
            preOrder(node.left)
            preOrder(node.right)
      }
}
```

→ 20 10 15 30 25 22 27 35

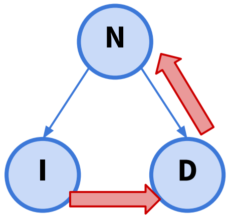




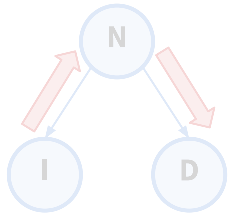
Post-Order

```
void postOrder(btn node) {
      if (notEmpty(node)){
            postOrder(node.left);
            postOrder(node.right);
            printNode(node);
      }
}
```

→ 15 10 22 27 25 35 30 20



In-Order

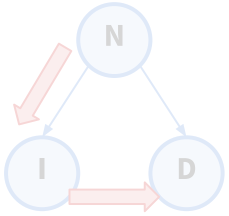


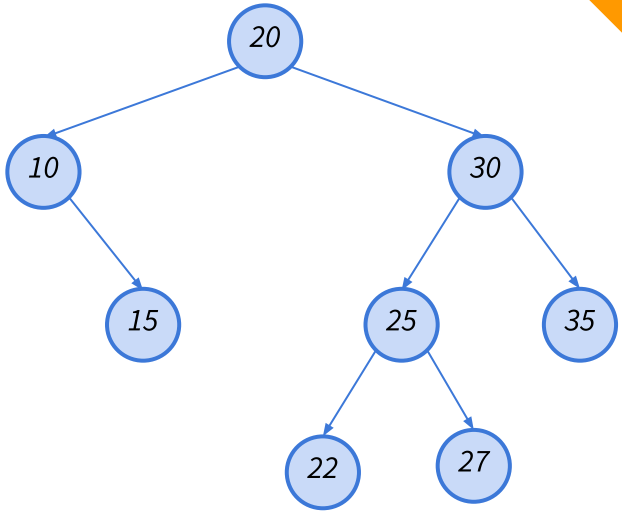
Árboles binarios - Recorrido

Pre-Order

```
void preOrder(btn node) {
        if (notEmpty(node)){
                printNode(node)
                preOrder(node.left)
                preOrder(node.right)
        }
}
```

→ 20 10 15 30 25 22 27 35

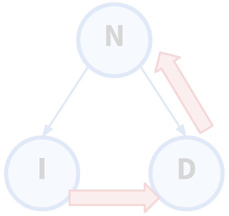




Post-Order

```
void postOrder(btn node) {
        if (notEmpty(node)){
                postOrder(node.left);
                postOrder(node.right);
                printNode(node);
        }
}
```

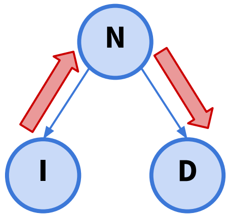
→ 15 10 22 27 25 35 30 20



In-Order

```
void inOrder(btn node) {
        if (notEmpty(node)){
                inOrder(node.left);
                printNode(node);
                inOrder(node.right);
        }
}
```

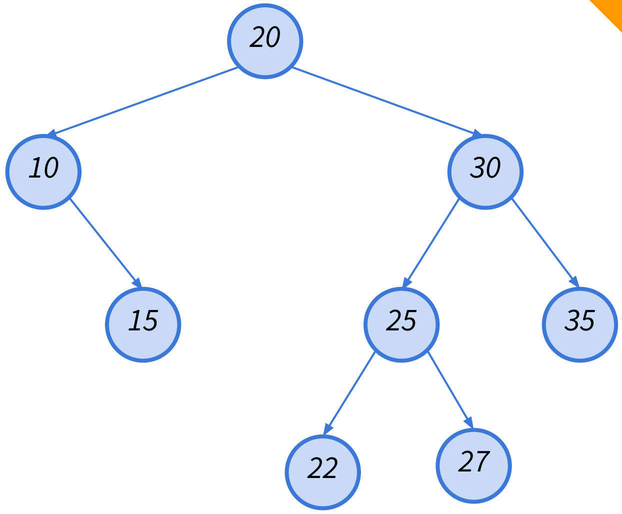
→ 10 15 20 22 25 27 30 35



Árboles Binarios Búsqueda (ABB)

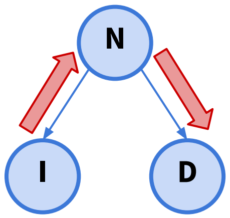
Binary Search Tree (BST)

- Para todo nodo
  - Todos los descendientes de la rama izquierda tienen un valor menor, y
  - Todos los descendientes de la rama derecha tienen un valor mayor



In-Order

```
void inOrder(btn node) {
      if (notEmpty(node)){
            inOrder(node.left);
            printNode(node);
            inOrder(node.right);
      }
}
```



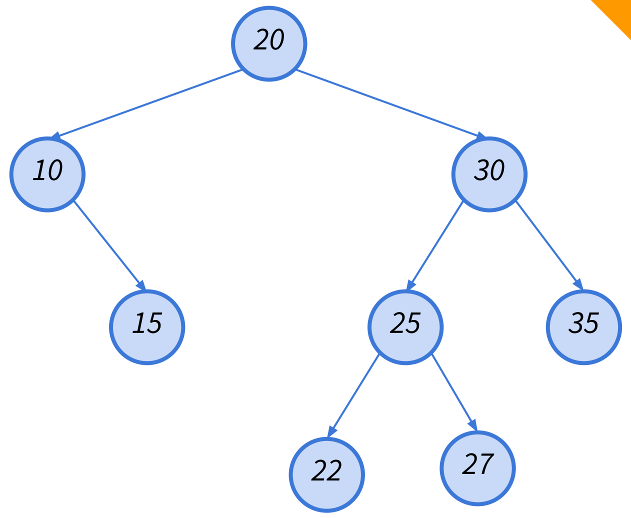
→ 10 15 20 22 25 27 30 35

Árboles Binarios Búsqueda (ABB)

Buscar un nodo

- Para todo nodo
  - Todos los descendientes de la rama izquierda tienen un valor menor, y
  - Todos los descendientes de la rama derecha tienen un valor mayor

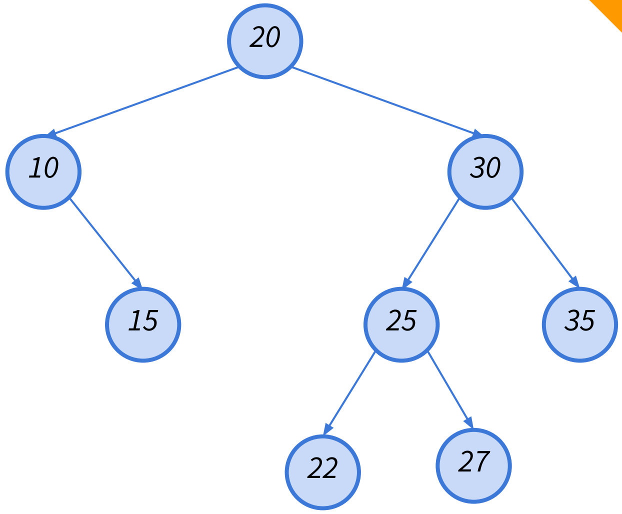
- Buscar
  - Iterativo
  - Recursivo



Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

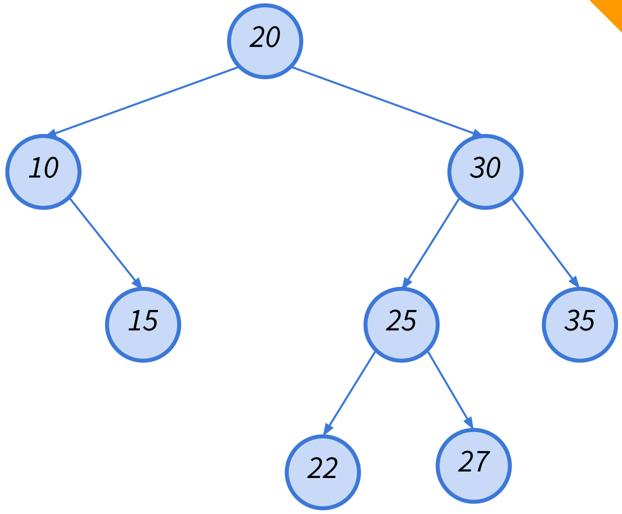


```
btn find(btn node, int value){




}
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

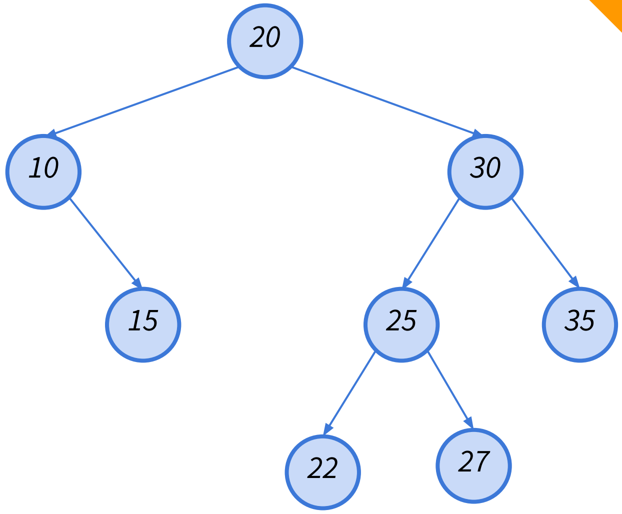
```
btn {
     datatype value
     btn  left
     btn  right
}



btn find(btn node, int value){

          if (node.value > value) {
               node = node.left
          }




}
```



Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



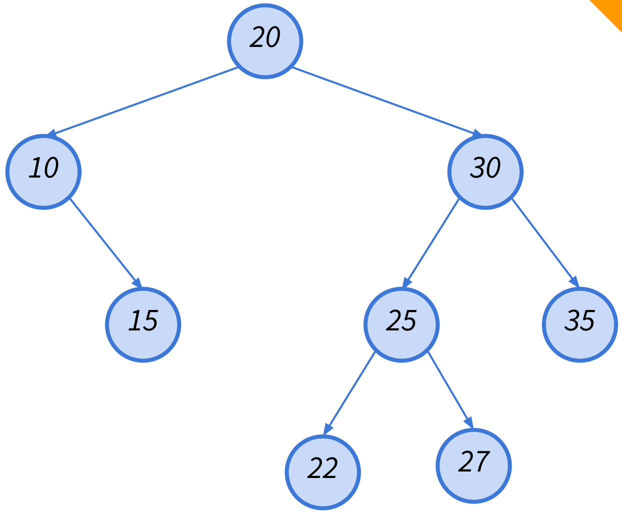
```
btn find(btn node, int value){
     while                    (node.value != value) {
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }

}
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while                    (node.value != value) {
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```

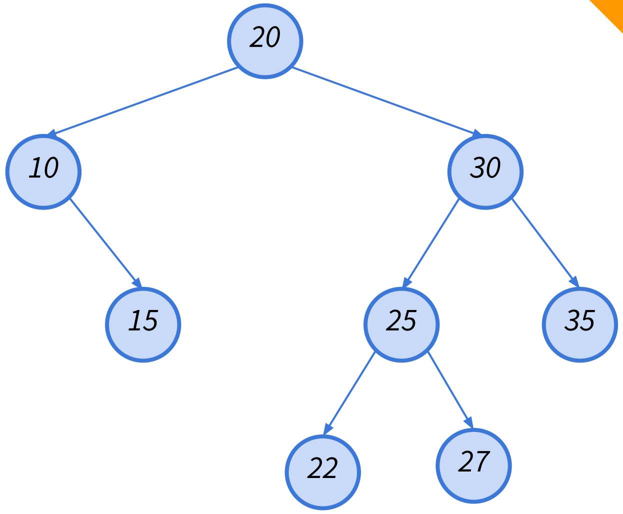


Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```

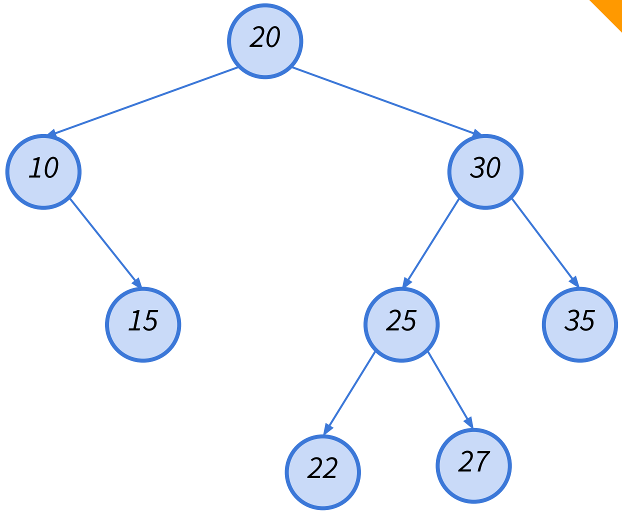


Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



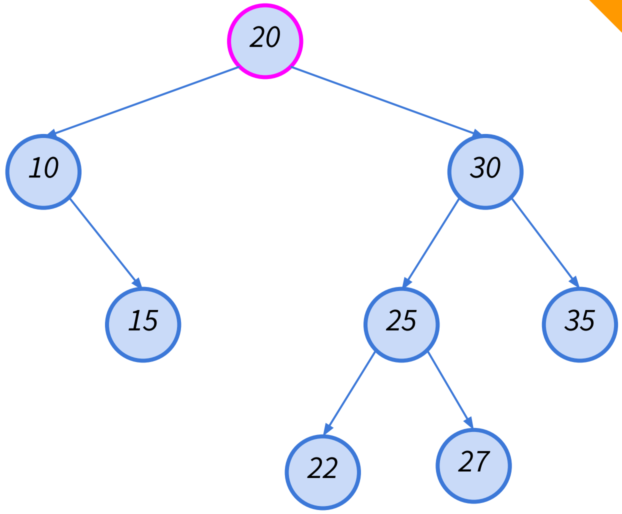
Invocación

```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```

Invocación

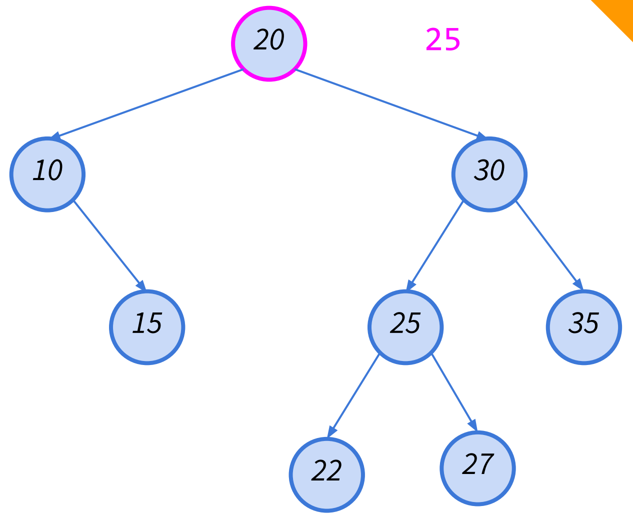
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

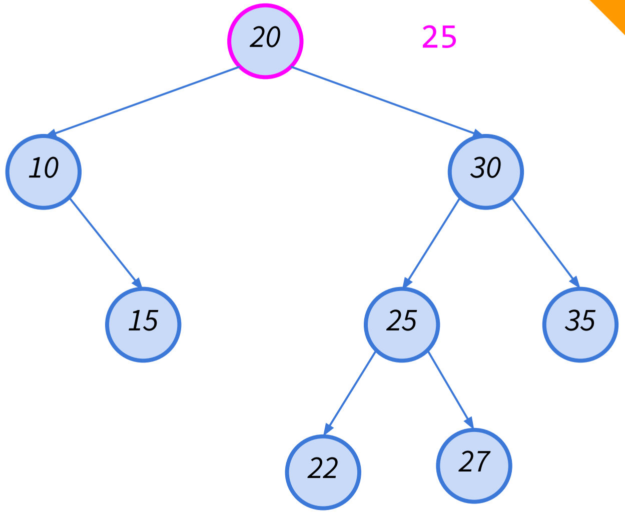
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

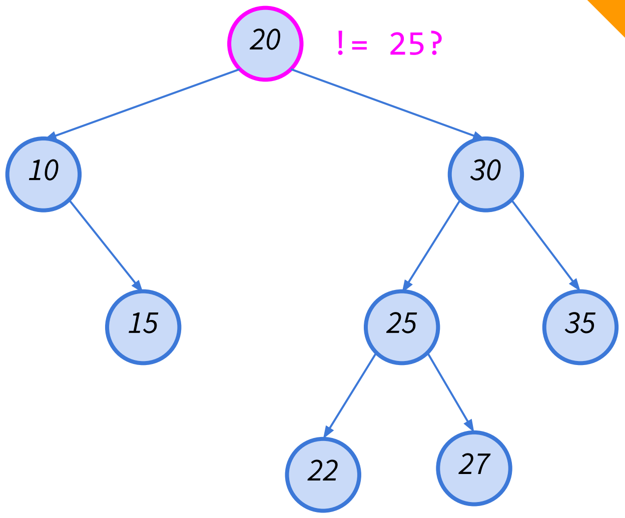
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

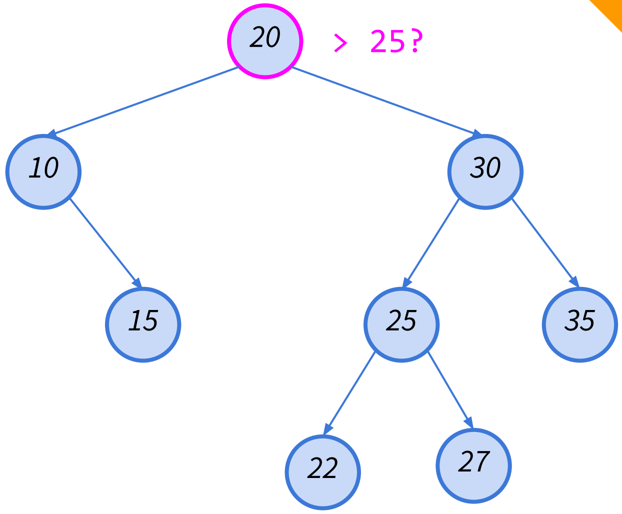
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

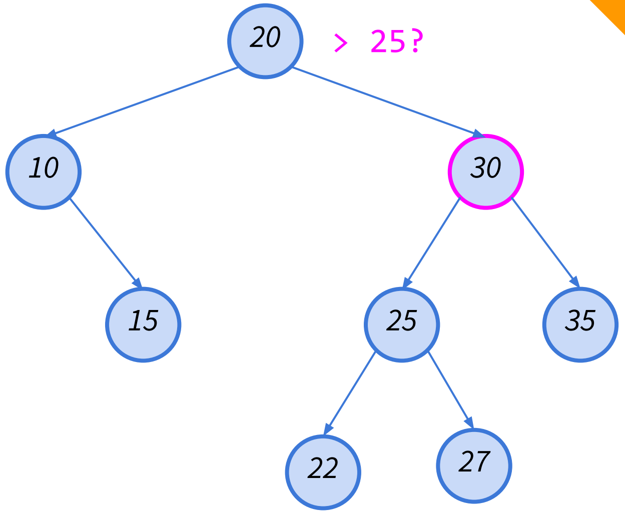
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



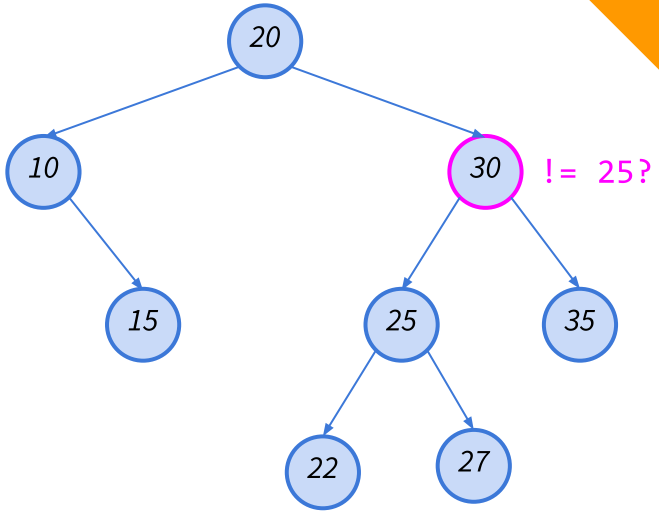
Invocación

```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```

Invocación

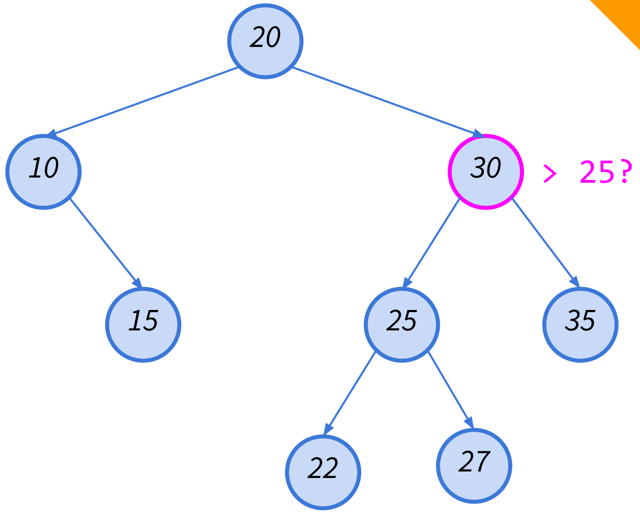
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



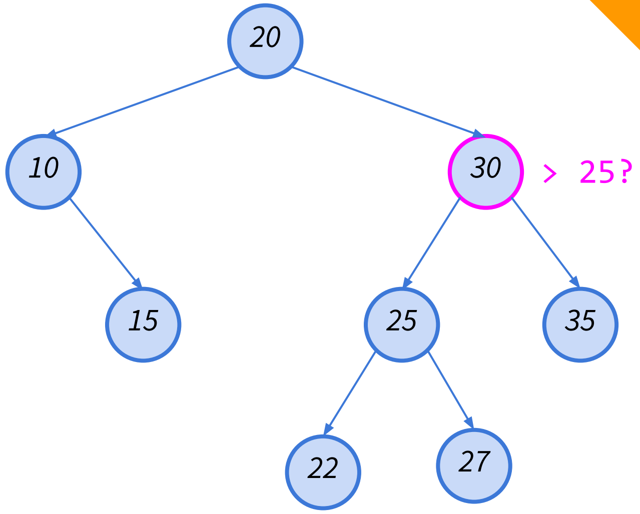
Invocación

```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```

Invocación

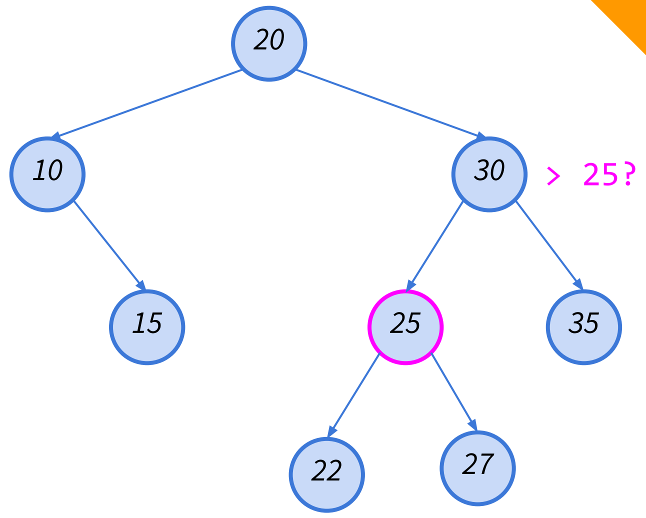
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

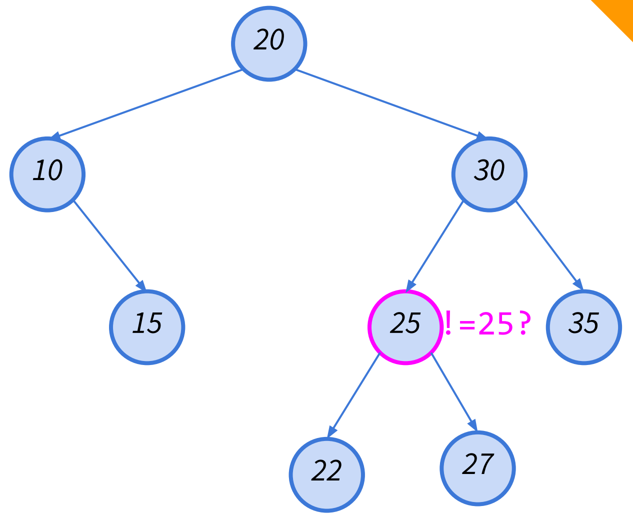
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

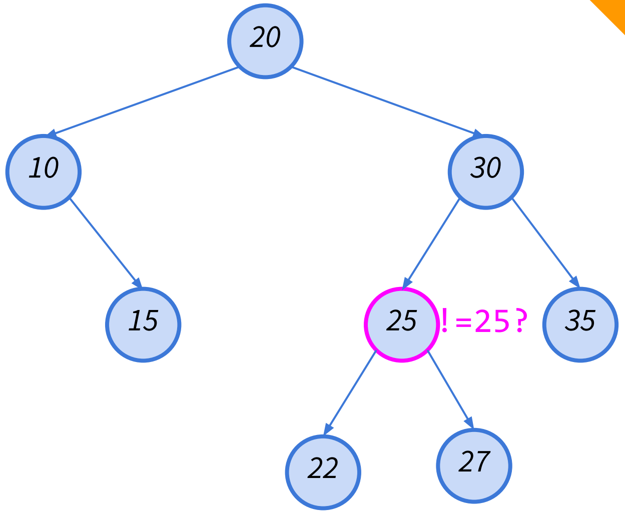
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

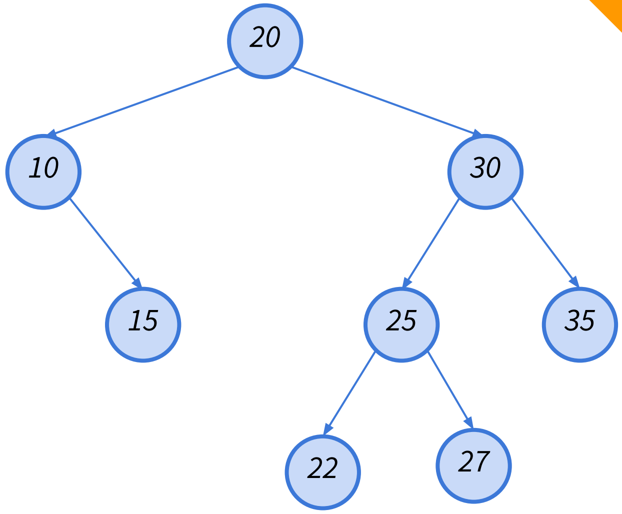
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn   left
     btn   right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

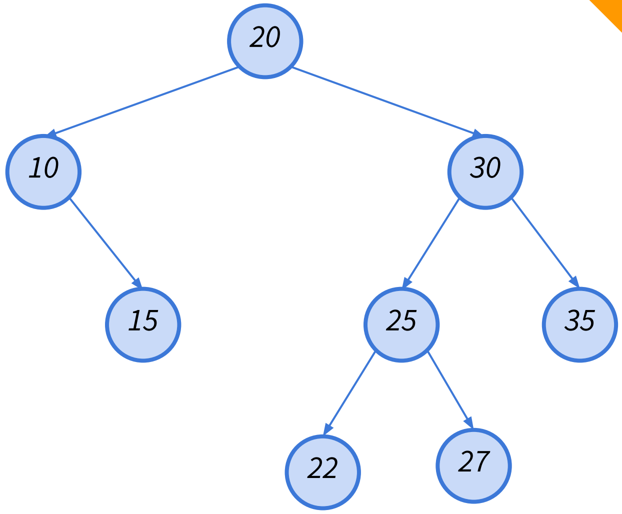
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Iterativo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){
     while (notEmpty(node) && (node.value != value)){
          if (node.value > value) {
               node = node.left
          } else {
               node = node.right
          }
     }
     return node
}
```



Invocación

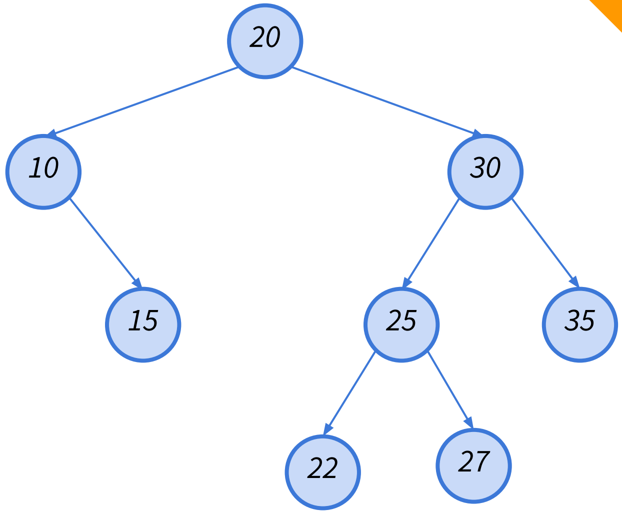
```
...
btn b = find(root, 25)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Recursivo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

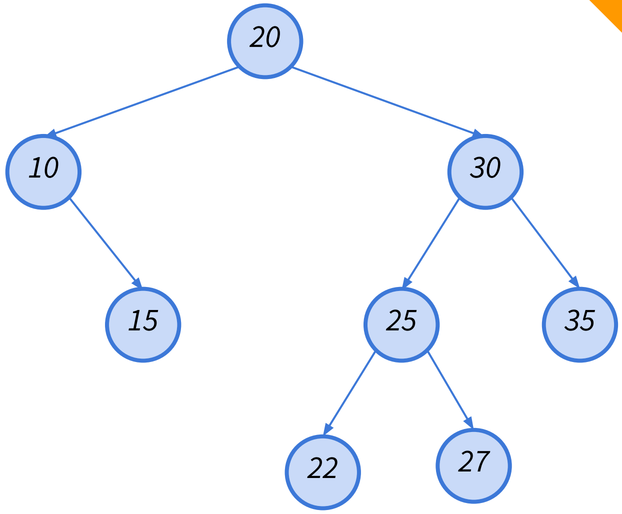
```
btn find(btn node, int value){







}
```



Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Recursivo

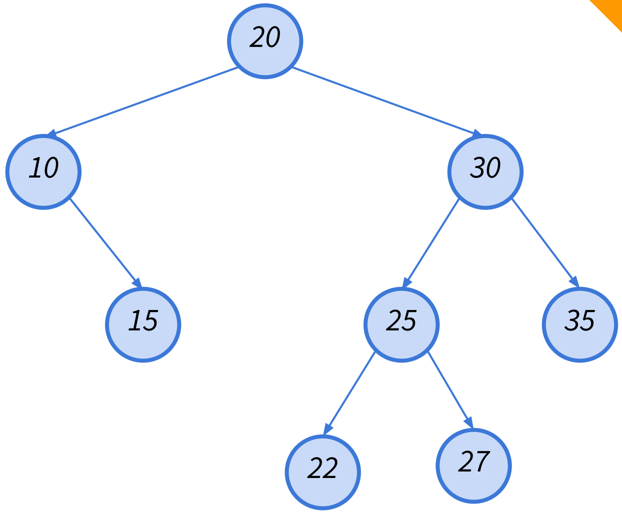
```
btn {
     datatype value
     btn  left
     btn  right
}



btn find(btn node, int value){

          if (node.value = value) {
               return node
          }





}
```



Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Recursivo

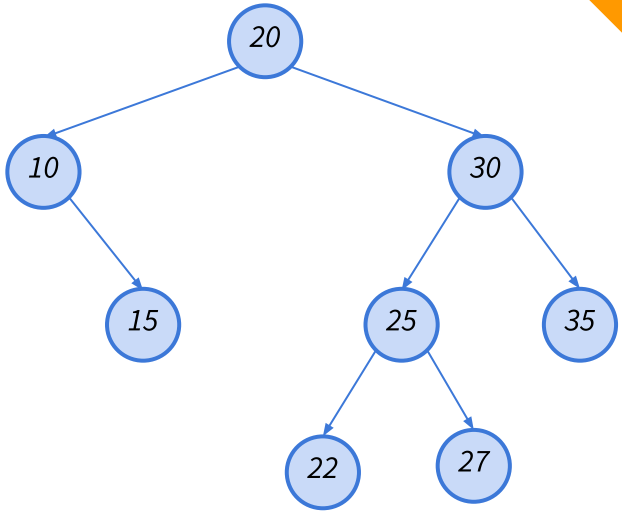
```
btn {
     datatype value
     btn  left
     btn  right
}



btn find(btn node, int value){

          if (node.value = value) {
               return node
          } else if (node.value > value) {
               return find (node.left, value)
          }




}
```



Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Recursivo

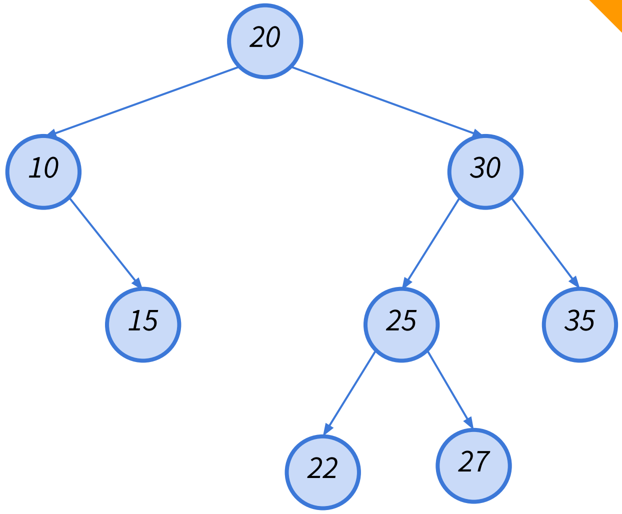
```
btn {
     datatype value
     btn  left
     btn  right
}



btn find(btn node, int value){

          if (node.value = value) {
               return node
          } else if (node.value > value) {
               return find (node.left, value)
          } else {
               return find (node.right, value)
          }



}
```



Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Recursivo

```
btn {
     datatype value
     btn  left
     btn  right
}



btn find(btn node, int value){
     if (notEmpty(node)){
          if (node.value = value) {
               return node
          } else if (node.value > value) {
               return find (node.left, value)
          } else {
               return find (node.right, value)
          }
     } else {
          return NULL
     }
}
```

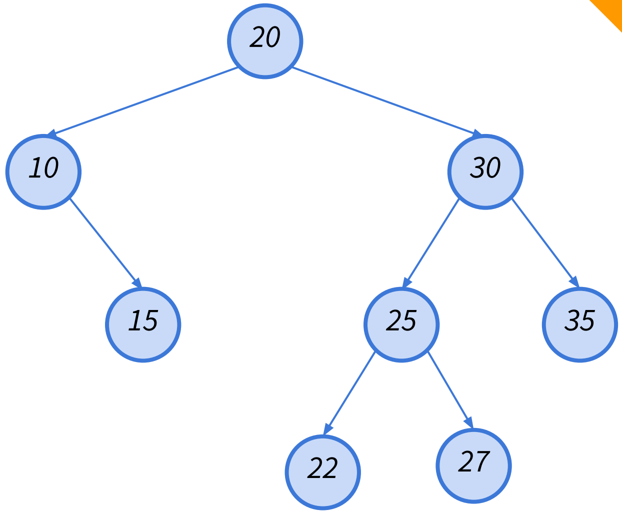


Árboles Binarios Búsqueda (ABB)

Buscar un nodo - Recursivo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn find(btn node, int value){

          if (isEmpty(node) || (node.value = value)) {
               return node
          } else if (node.value > value) {
               return find (node.left, value)
          } else {
               return find (node.right, value)
          }

}
```



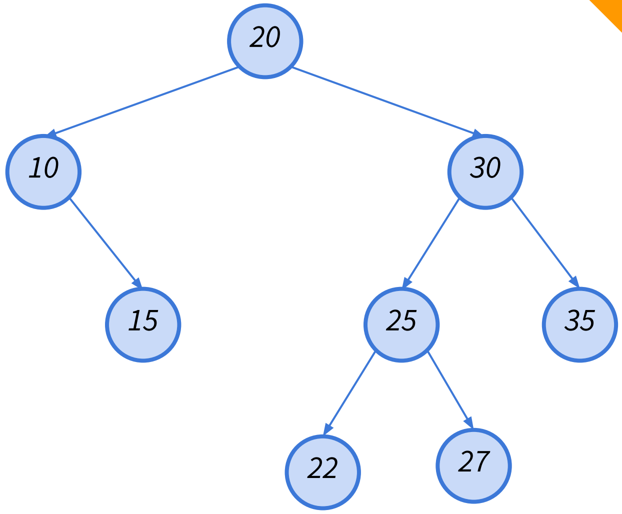
Invocación

```
...
btn b = find(root, 15)
if (notEmpty(b)) {
...
```

Árboles Binarios Búsqueda (ABB)

Buscar un nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn find(btn node, int value){

          if (isEmpty(node) || (node.value = value)) {
               return node
          } else if (node.value > value) {
               return find (node.left, value)
          } else {
               return find (node.right, value)
          }

}
```

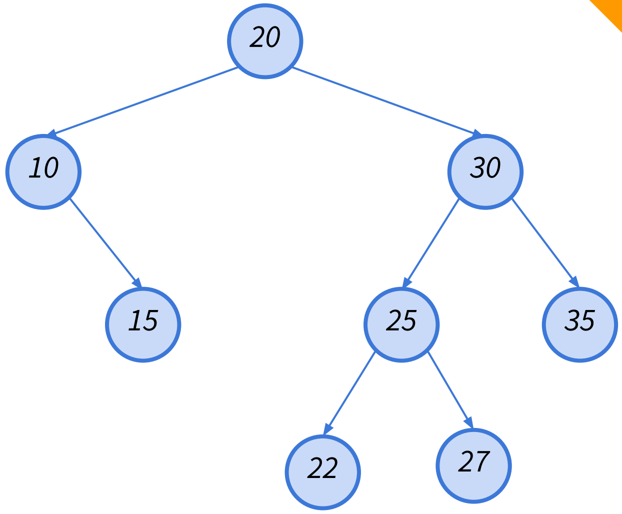
C

```c
struct btn *find(struct btn *node, int value) {
    if ((node == NULL) || (node->value == value)) {
        return node;
    } else if (node->value > value) {
        return find(node->left, value);
    } else {
        return find(node->right, value);
    }
}
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}



btn insert(btn node, int value){









}
```



Invocación

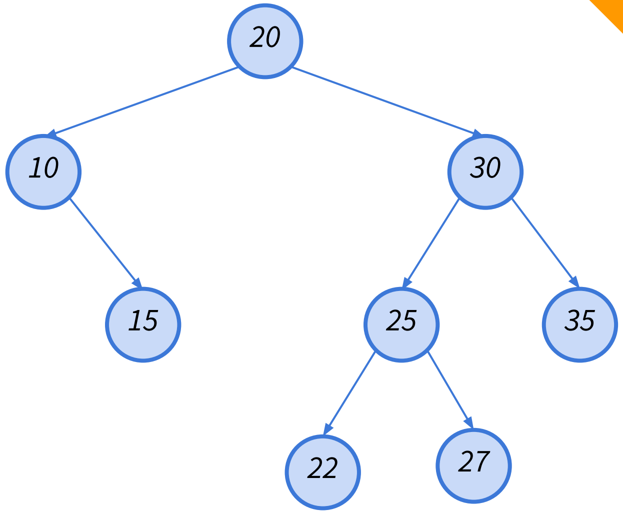
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     }



}
```



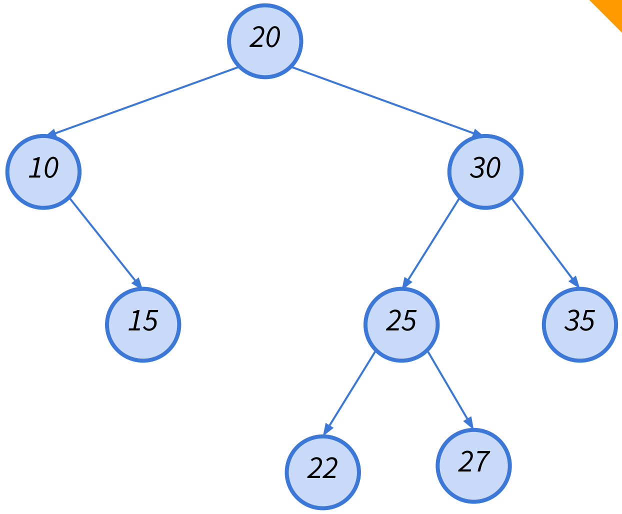
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}



btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          }



     }
}
```



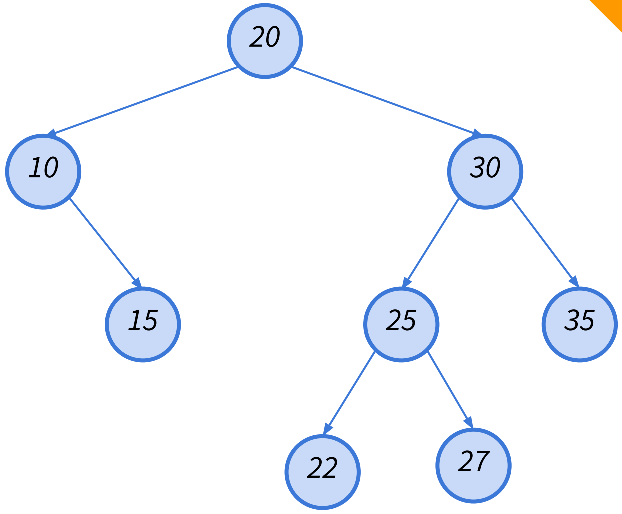
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}



btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }

     }
}
```



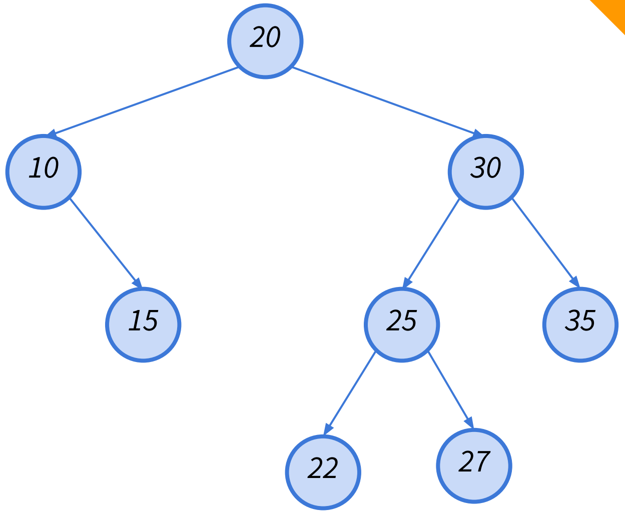
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

Invocación

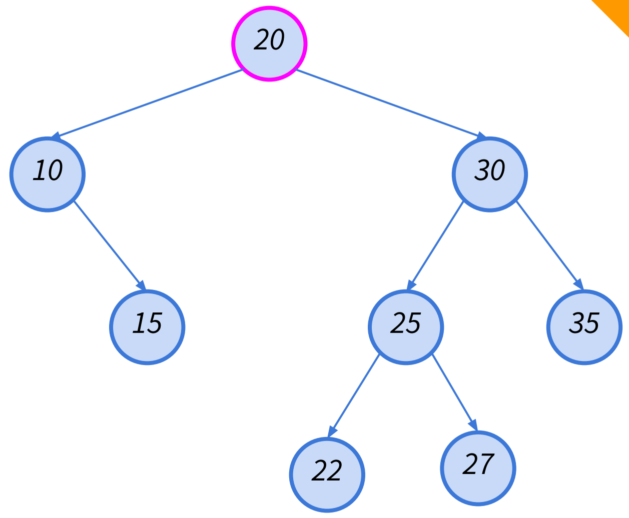
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



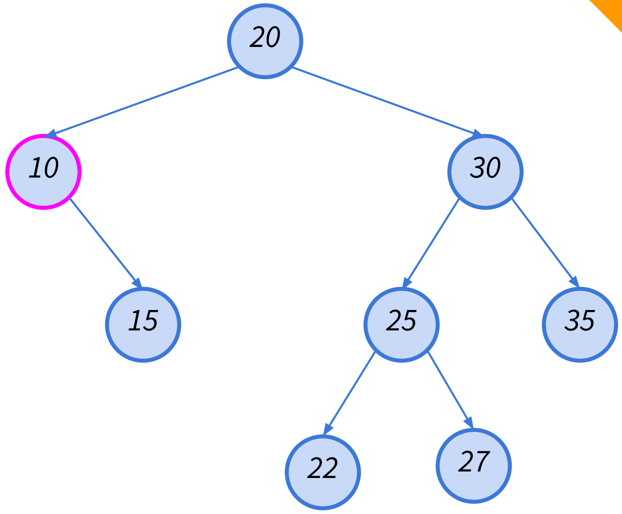
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

Invocación

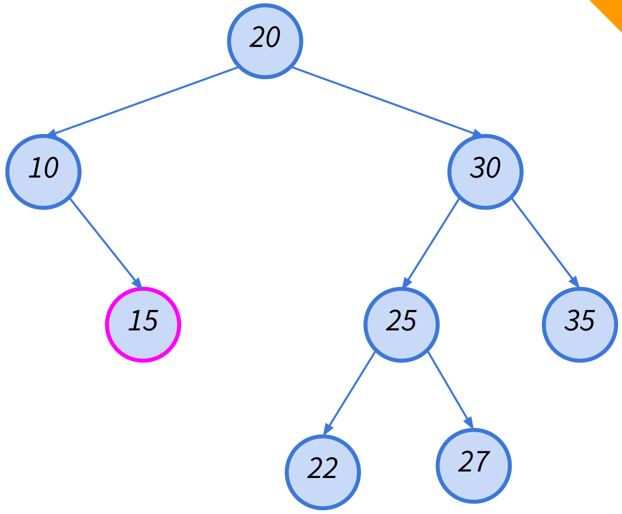
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



Invocación

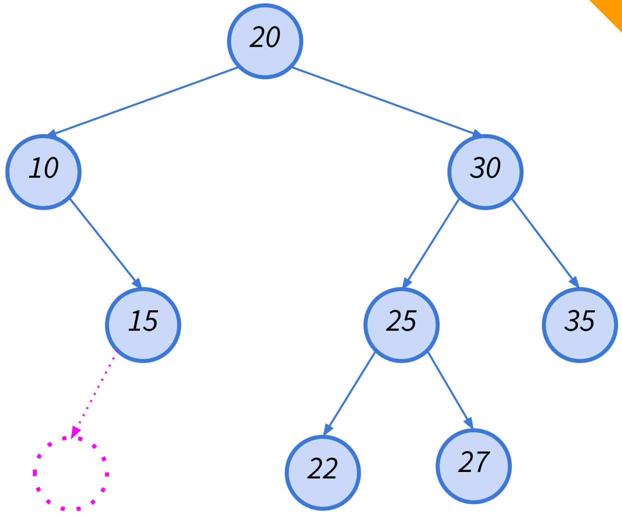
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



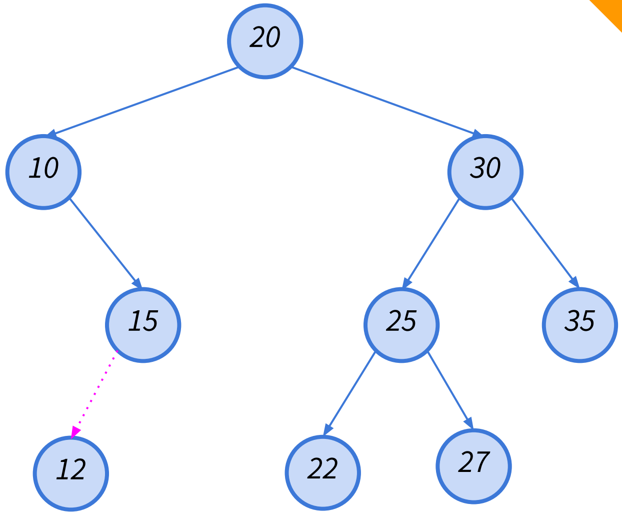
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}



btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



Invocación

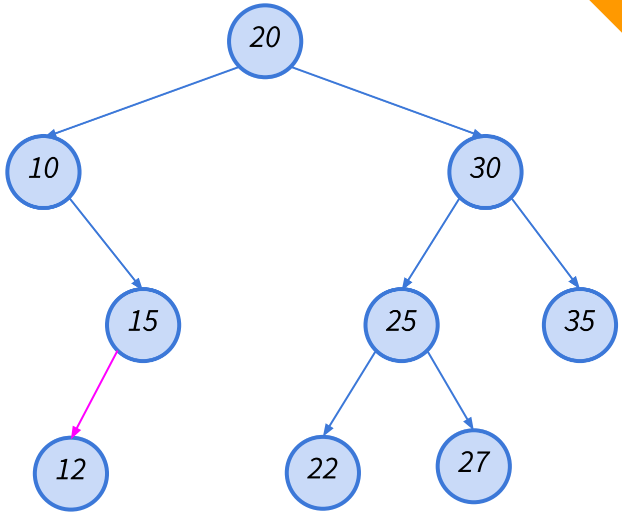
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



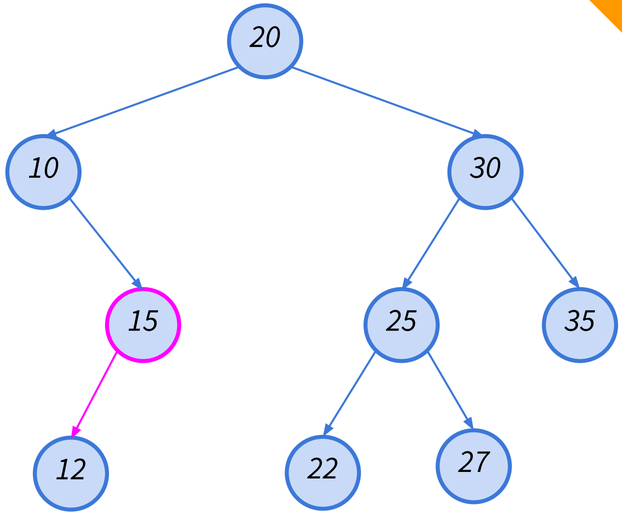
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

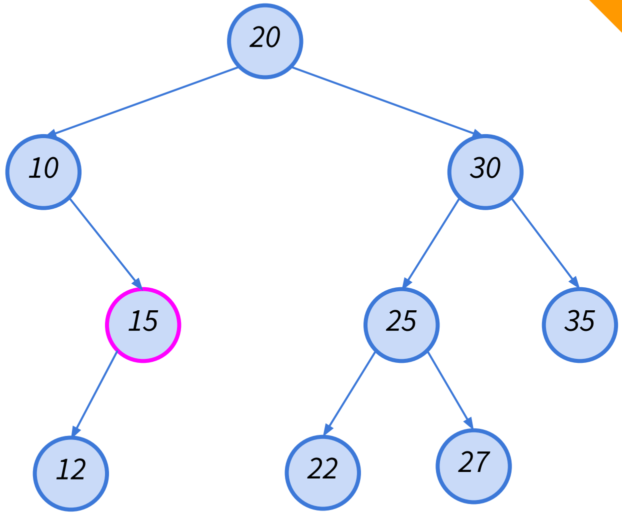
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

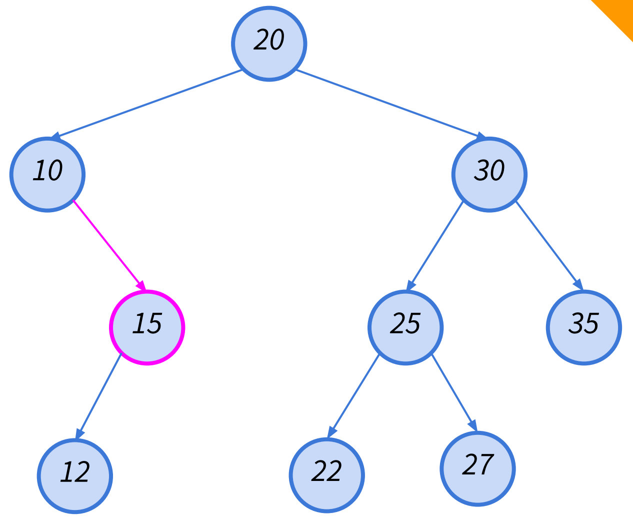
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

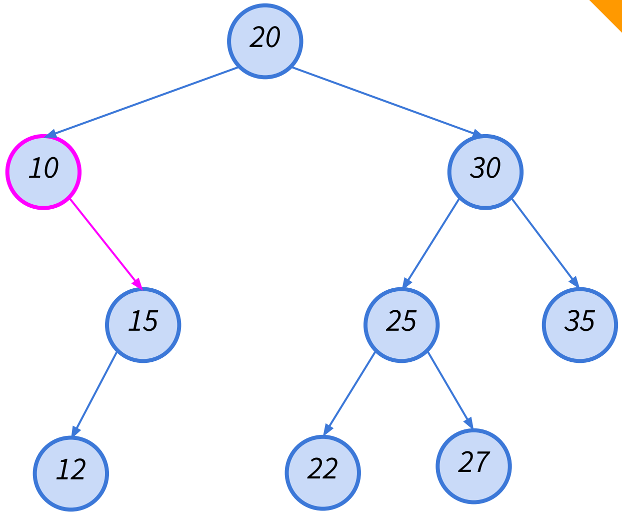
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

Invocación

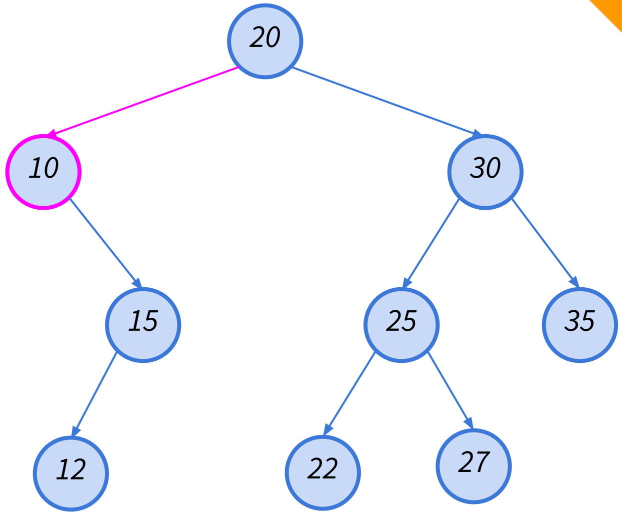
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



Invocación

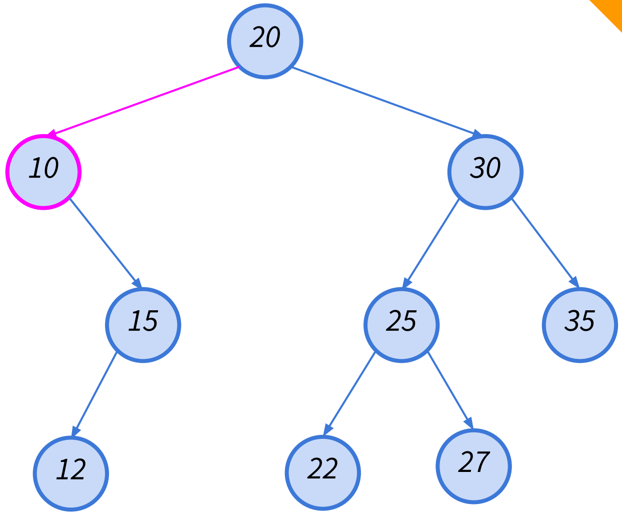
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



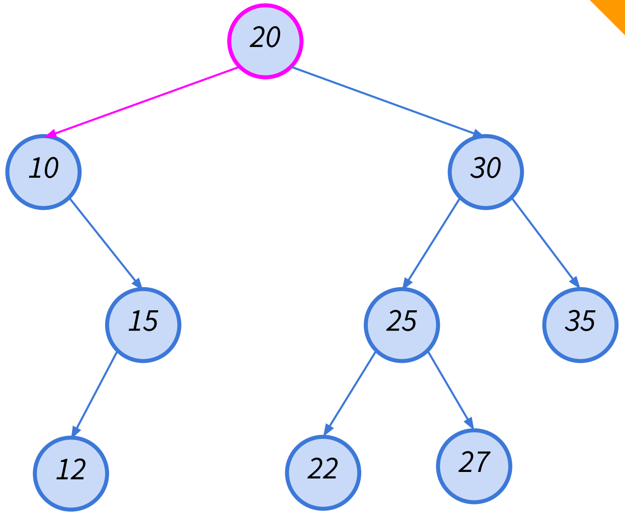
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```



```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```

Invocación

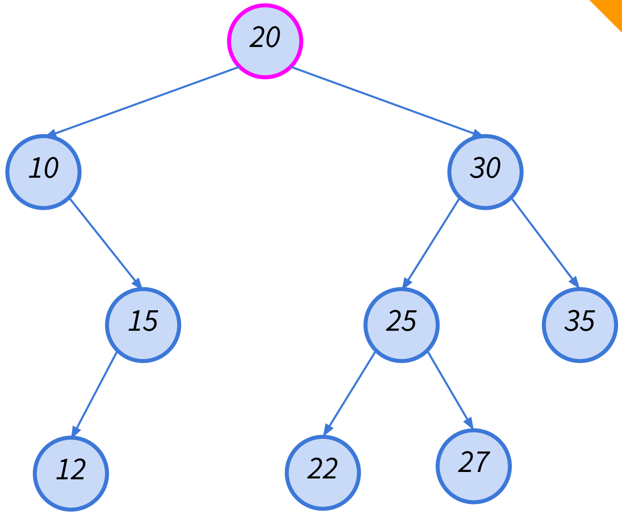
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



Invocación

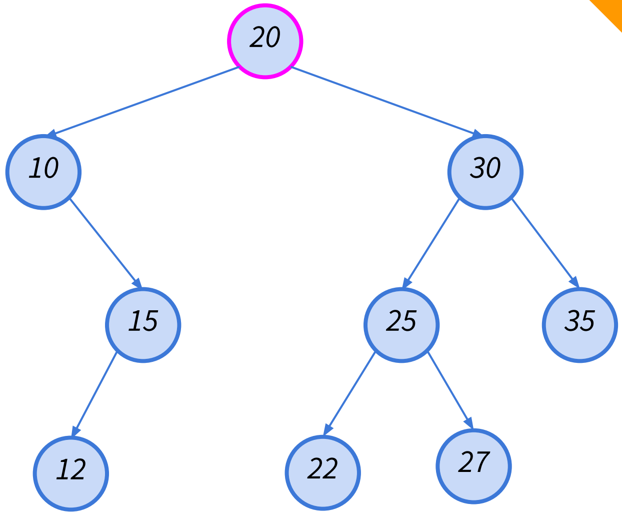
```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}
```

```
btn insert(btn node, int value){
     if (isEmpty(node)){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.right = insert(node.right, value)
          }
          return node
     }
}
```



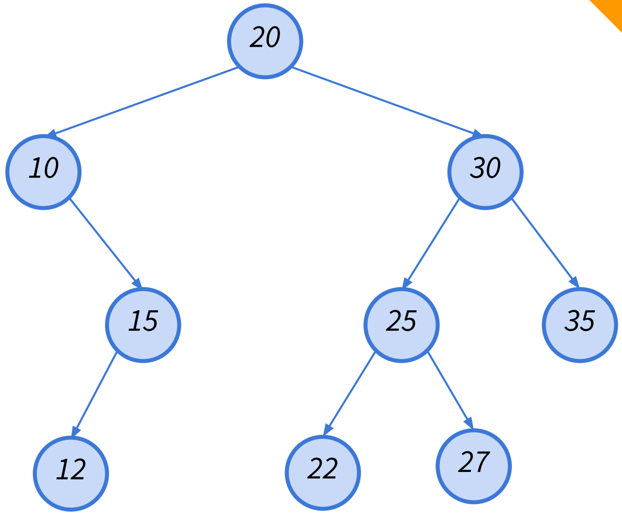
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}

btn create(btn node, int value){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
}

btn insert(btn node, int value){
     if (isEmpty(node)){
          return create(node, value)
     } else {
          if (node.value > value) {
               node.left = insert(node.left, value)
          } else if (node.value < value) {
               node.left = insert(node.right, value)
          }
          return node
     }
}
```



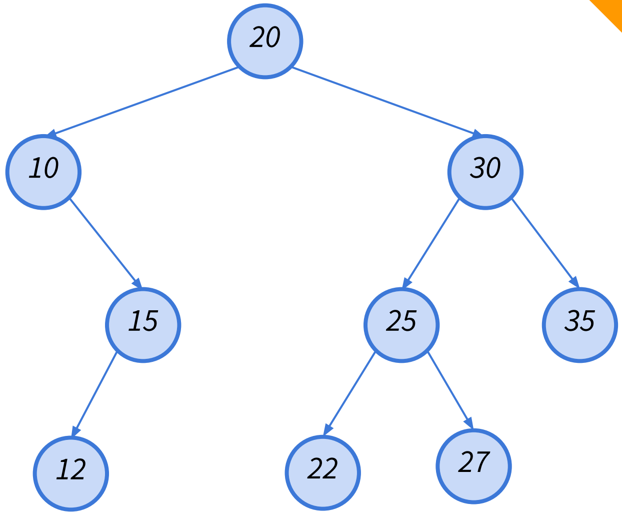
Invocación

```
...
root = insert(root, 12)
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}

btn create(btn node, int value){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
}

btn insert(btn node, btn new){
     if (isEmpty(node)){
          return new
     } else {
          if (node.value > new.value) {
               node.left = insert(node.left, new)
          } else if (node.value < value) {
               node.left = insert(node.right, new)
          }
          return node
     }
}
```



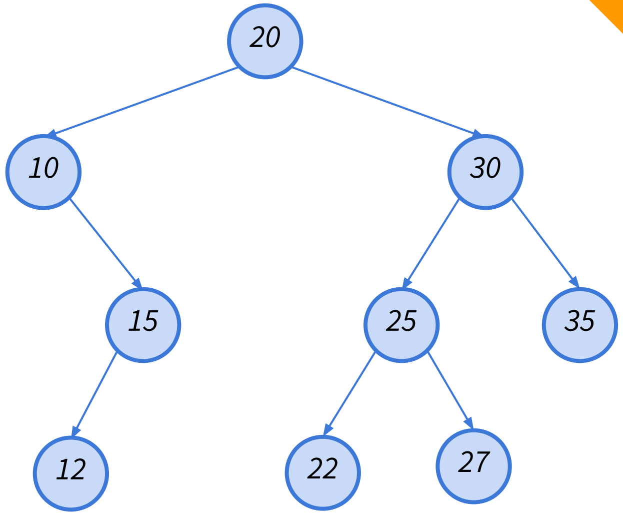
Invocación

```
...
root = insert(root, create(12))
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}

btn create(btn node, int value){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
}

bool insert(btn ↑node, btn new){
     if (isEmpty(new)) return TRUE;
     if (isEmpty(node)){
          node = new
          return TRUE
     } else {
          if (node.value > new.value) {
               return insert(node.left, new)
          } else if (node.value < value) {
               return = insert(node.right, new)
          } else {
               return FALSE
          }
     }
}
```



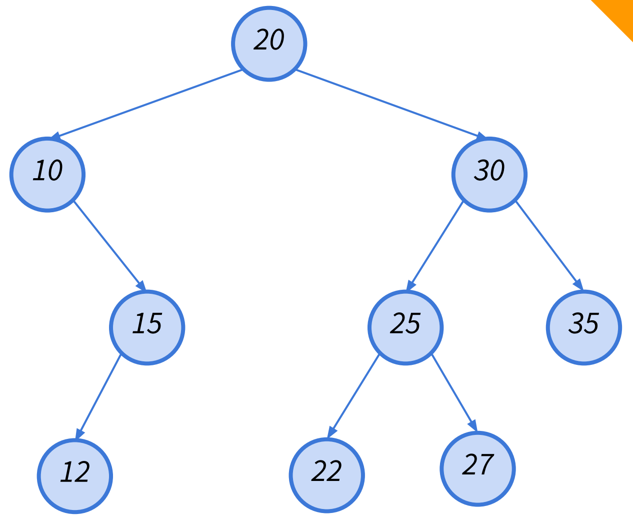
Invocación

```
...
btn new = create(12)
if (!insert(root, new)){
     free(new)
}
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}

btn create(btn node, int value){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
}

bool insert(btn ↑node, btn new){
     if (isEmpty(new)) return TRUE;
     if (isEmpty(node)){
          node = new
          return TRUE
     } else {
          if (node.value > new.value) {
               return insert(node.left, new)
          } else if (node.value < value) {
               return = insert(node.right, new)
          } else {
               return FALSE
          }
     }
}
```



<u>*Invocación*</u>

```
...
btn new = create(12)
if (!insert(root, new)){
     free(new)
}
...
```

Árboles Binarios Búsqueda (ABB)

Agregar un Nodo

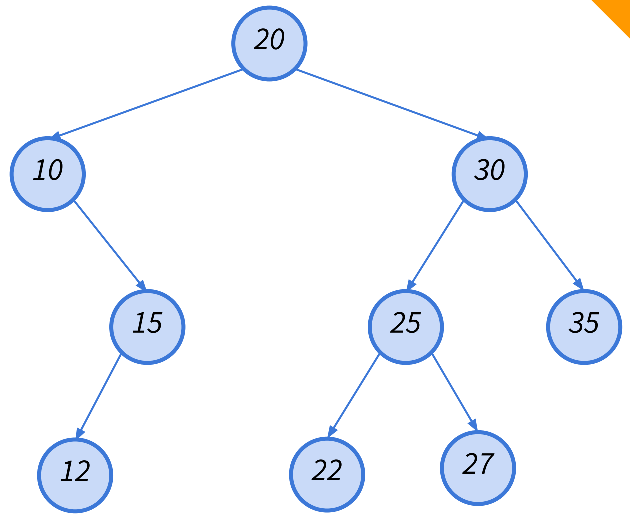
```
btn {
     datatype value
     btn  left
     btn  right
}

btn create(btn node, int value){
          btn aux = NEW()
          aux.value = value
          aux.left = NULL
          aux.right = NULL
          return aux
}

bool insert(btn ↑node, btn new){
     if (isEmpty(new)) return TRUE;
     if (isEmpty(node)){
          node = new
          return TRUE
     } else {
          if (node.value > value) {
               return insert(node.left, new)
          } else if (node.value < value) {
               return = insert(node.right, new)
          } else {
               return FALSE
          }
     }
}
```

C

```c
struct btn *create(int value) {
    struct btn *n =
             (struct btn *)malloc(sizeof(struct btn));
    n->value = value;
    n->left = NULL;
    n->right = NULL;
    return n;
}

int insertOver(struct btn **node, struct btn *new) {
    if (node == NULL) return 0;
    if (new == NULL) return 1;
    if ((*node) == NULL) {
        *node = new;
        return 1;
    } else {
        if ( (*node)->value > new->value) {
            // si el nuevo es menor inserta a izquierda
            return insertOver(&((*node)->left), new);
        } else if ((*node)->value < new->value) {
            // si el nuevo es mayor a la derecha
            return insertOver(&((*node)->right), new);
        } else {
            return 0;
        }
    }
}

... // invocación
        new = create(12);
        if (!insertOver(&root, new)){
            printf("\n Duplicado: %d", new->value);
            free(new);
        };
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

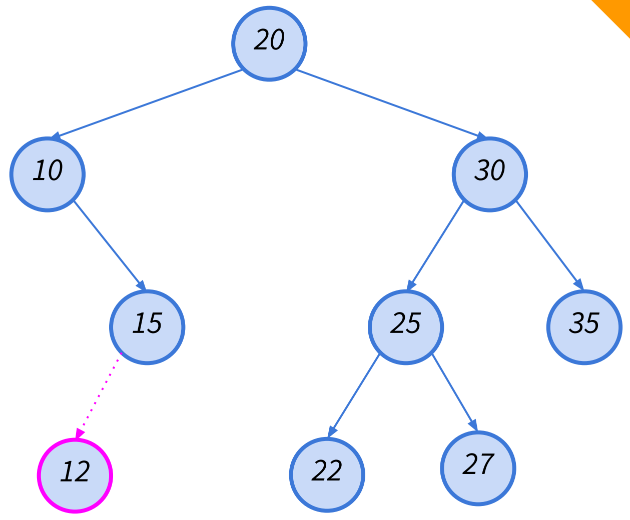


Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

Casos:

1. Cuando el nodo es una hoja

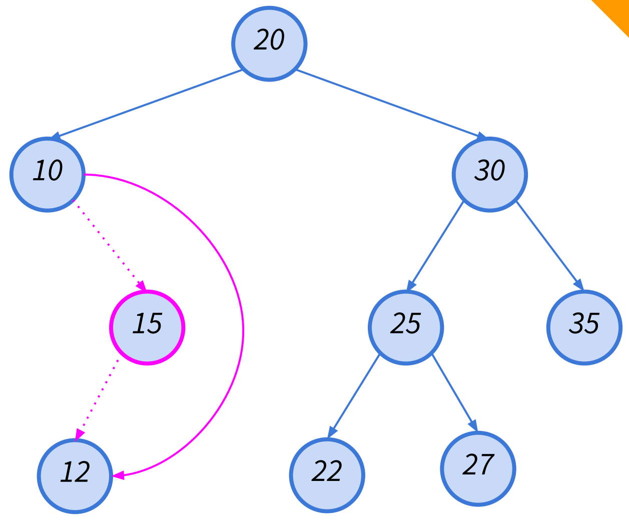


Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

Casos:

1. Cuando el nodo es una hoja
2. Cuando el nodo tiene un solo hijo

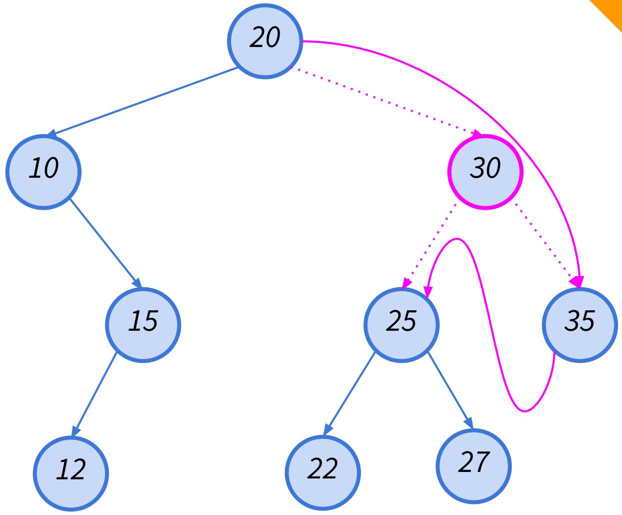


Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

Casos:

1. Cuando el nodo es una hoja
2. Cuando el nodo tiene un solo hijo
3. Cuando el nodo tiene ambos hijos

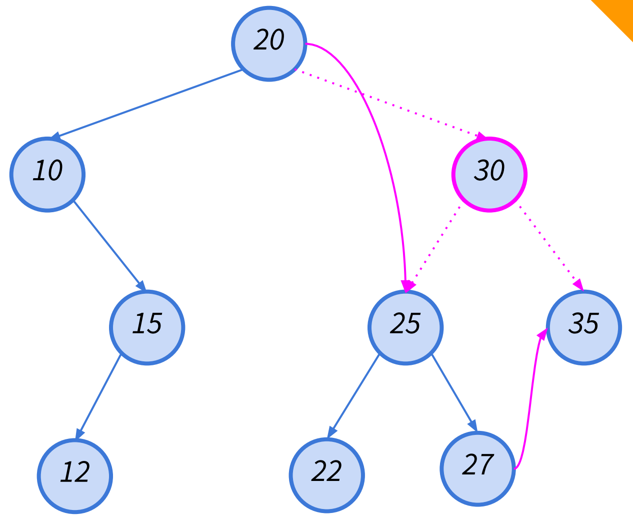


Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

Casos:

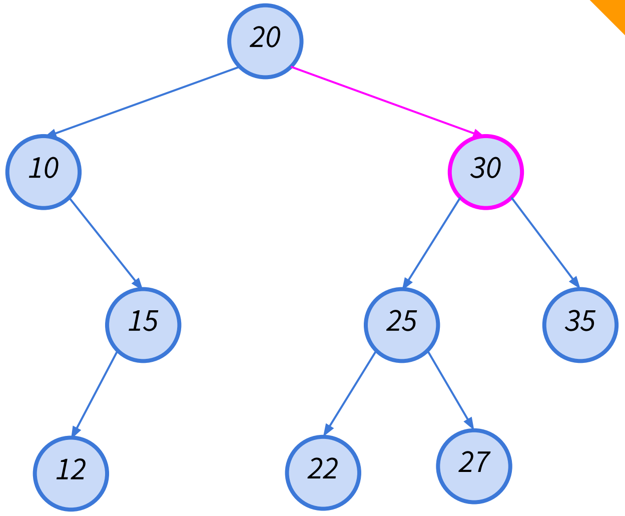
1. Cuando el nodo es una hoja
2. Cuando el nodo tiene un solo hijo
3. Cuando el nodo tiene ambos hijos



Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}


btn delete(btn ↑node){





}
```



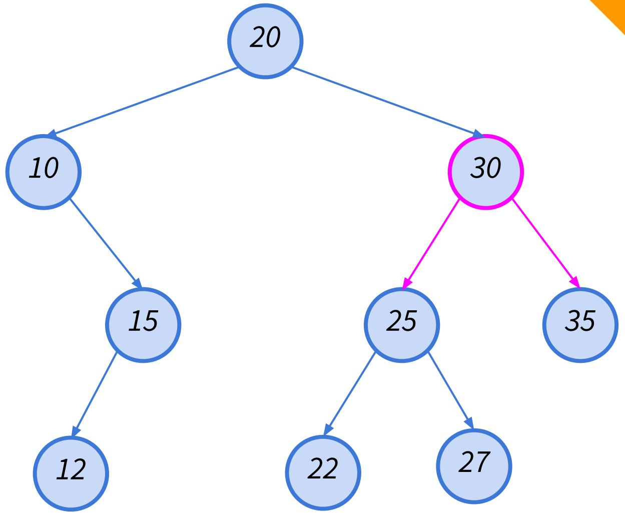
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}


btn delete(btn ↑node){


          btn aux = node
          insert (aux.right, aux.left)



}
```



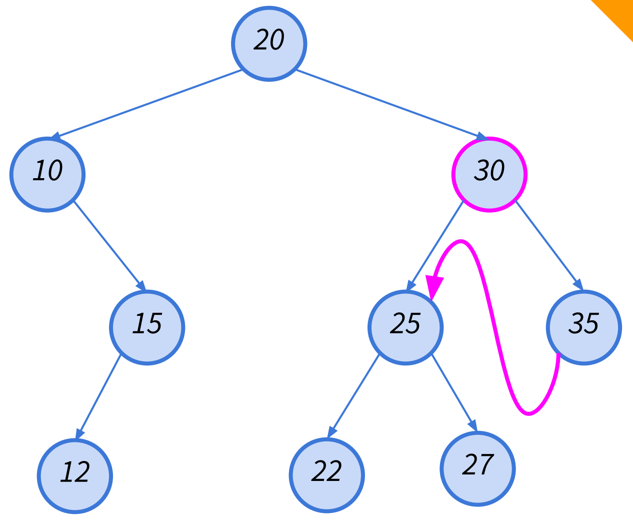
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}


btn delete(btn ↑node){


          btn aux = node
          insert (aux.right, aux.left)



}
```



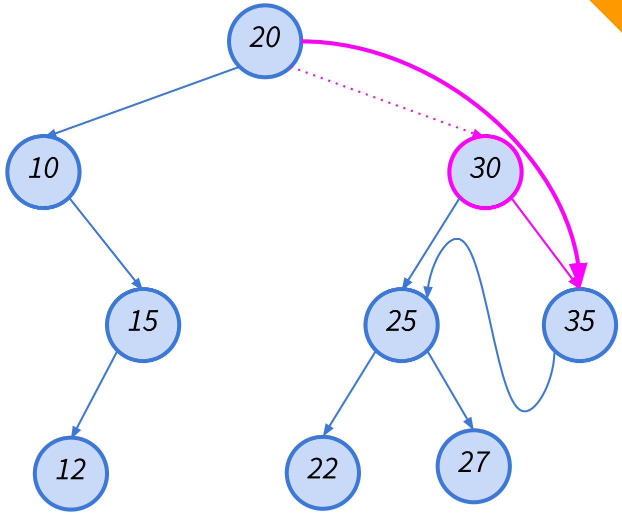
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}


btn delete(btn ↑node){


          btn aux = node
          insert (aux.right, aux.left)
          node = aux.right



}
```



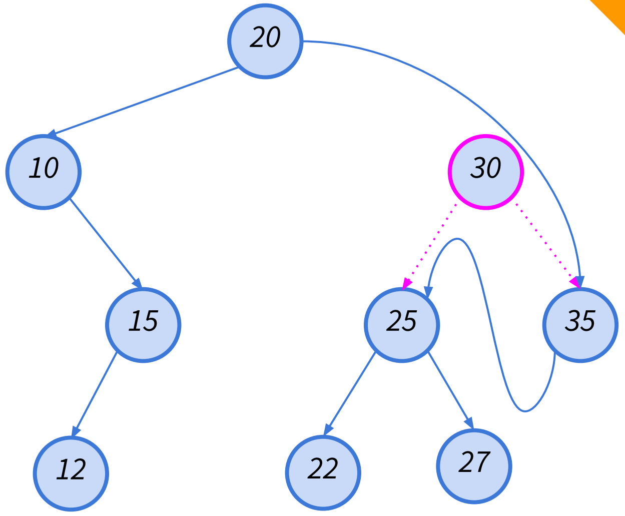
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn   left
     btn   right
}


btn delete(btn ↑node){


          btn aux = node
          insert (aux.right, aux.left)
          node = aux.right
          aux.right = NULL
          aux.left = NULL


}
```



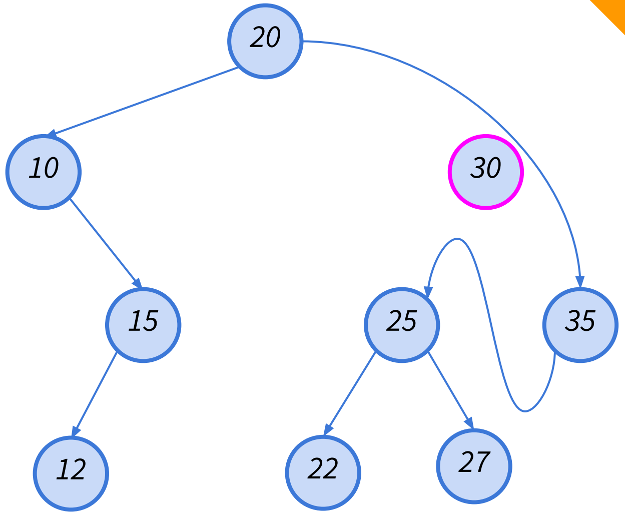
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}


btn delete(btn ↑node){


          btn aux = node
          insert (aux.right, aux.left)
          node = aux.right
          aux.right = NULL
          aux.left = NULL
          return aux

}
```



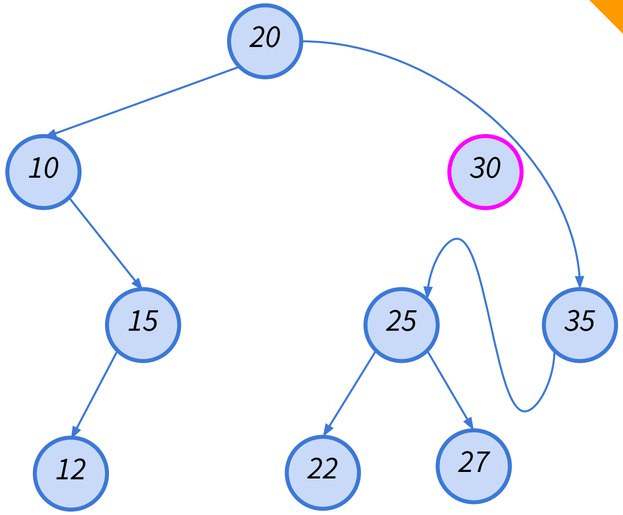
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

```
btn {
     datatype value
     btn  left
     btn  right
}


btn delete(btn ↑node){
     if (isEmpty(node)){
          return NULL
     } else {
          btn aux = node
          insert (aux.right, aux.left)
          node = aux.right
          aux.right = NULL
          aux.left = NULL
          return aux
     }
}
```



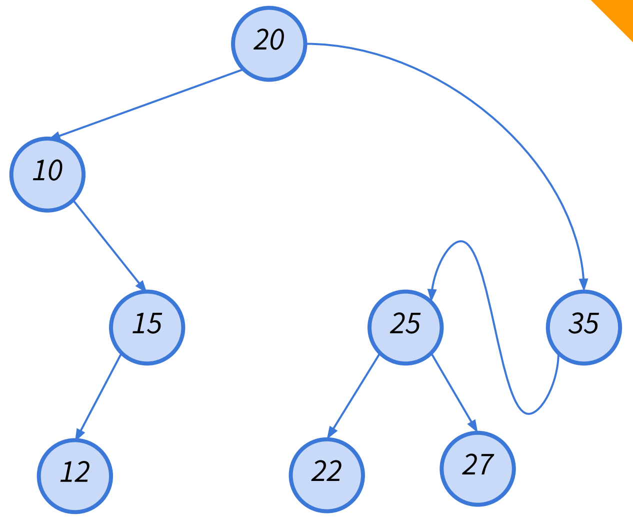
Invocación

```
...
btn deleted
deleted = delete(root.right)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

¿Y si eliminamos la raíz?



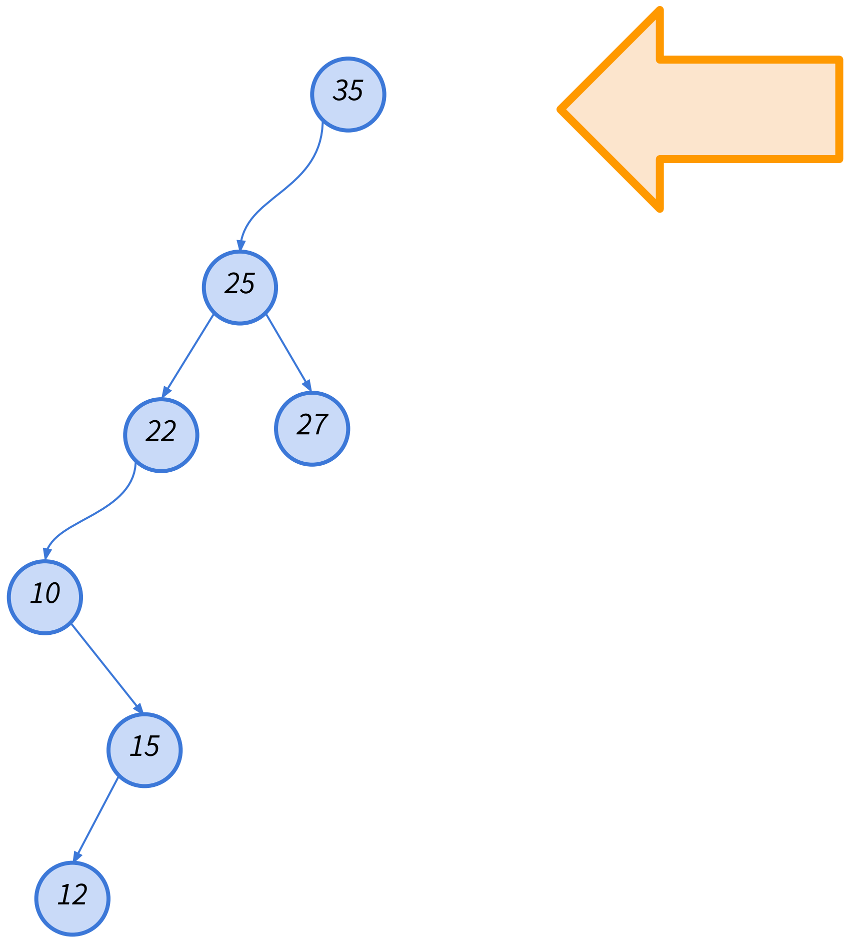
Invocación

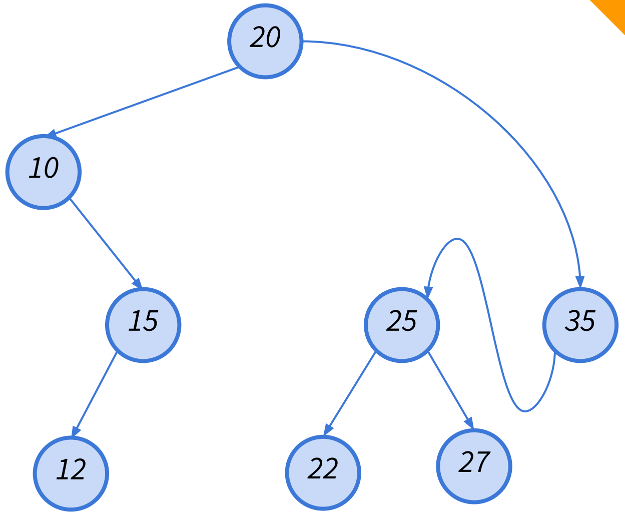
```
...
btn deleted
deleted = delete(root)
...
```

Árboles Binarios Búsqueda (ABB)

Eliminar un Nodo

¿Y si eliminamos la raíz?





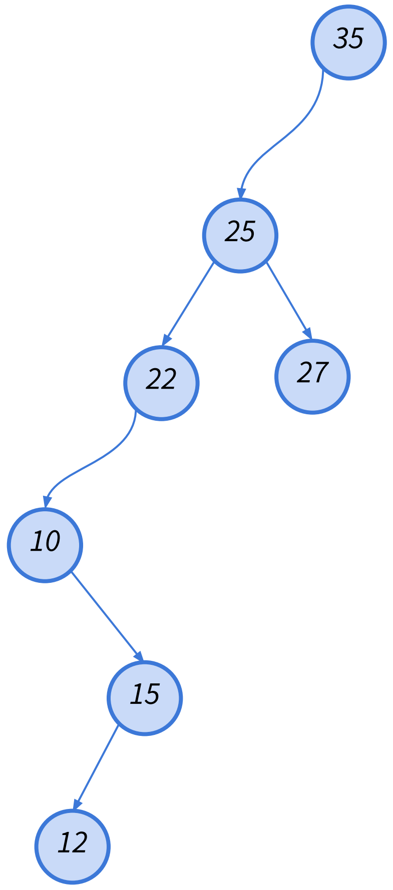
Invocación

```
...
btn deleted
deleted = delete(root)
...
```

Árboles Binarios Búsqueda (ABB)

Concepto de equilibrio

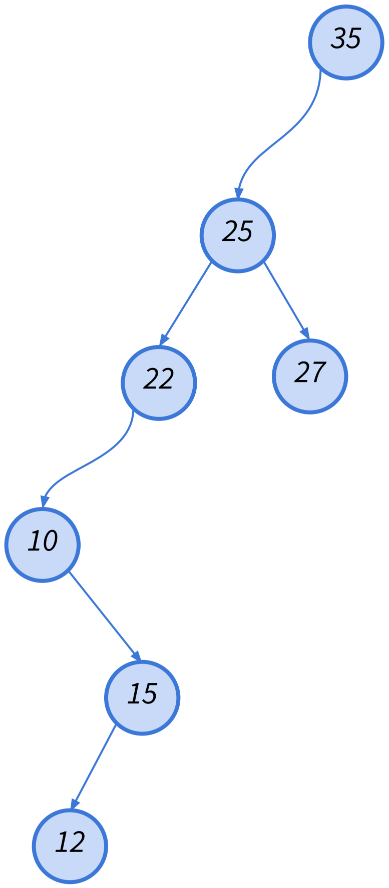
¿Sigue siendo un ABB?



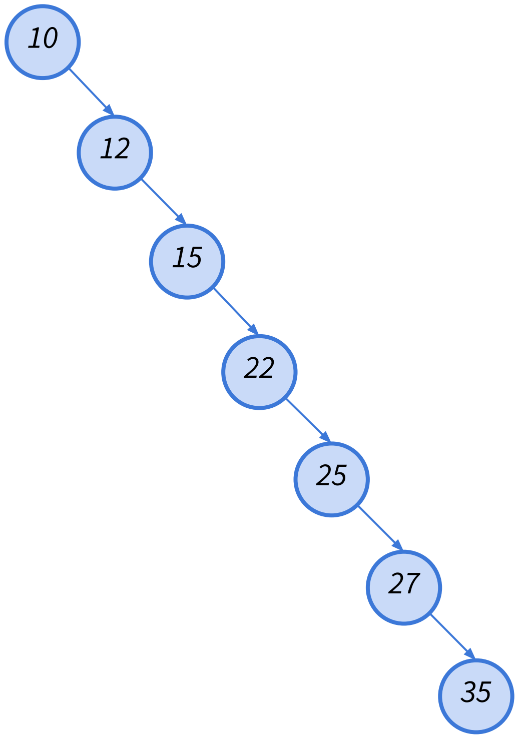
Árboles Binarios Búsqueda (ABB)

Concepto de equilibrio

¿Sigue siendo un ABB?



¿Y si el el orden de inserción fuera 10, 12, 15, 22, 25, 27, 35?



Unidad 5: Árboles

Algoritmos y Estructuras de Datos

Concepto y propiedades
Implementaciones
Árboles Binarios
Árboles Binarios de búsqueda