Algoritmos y estructuras de datos

Estructuras Jerárquicas

CEIS

Escuela Colombiana de Ingeniería

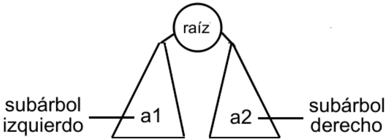
2021-2

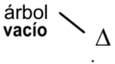
Agenda

Agenda

Árbol binario

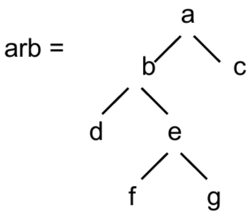
Un **árbol binario** es una estructura recursiva, compuesta por un elemento, denominado la raíz, y por dos árboles binarios asociados, denominados subárbol derecho y subárbol izquierdo.
El hecho de definir la estructura de datos en términos de sí misma es lo que hace que se denomine recursiva.

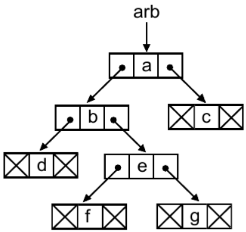




Árbol binario

Un árbol binario se puede representar como una estructura de datos enlazadas en donde cada nodo es un objeto.
Generalmente cada nodo tiene tres atributos el valor asociado (v), el apuntador al hijo izquierdo (*left*) y el apuntador al hijo derecho (*right*).



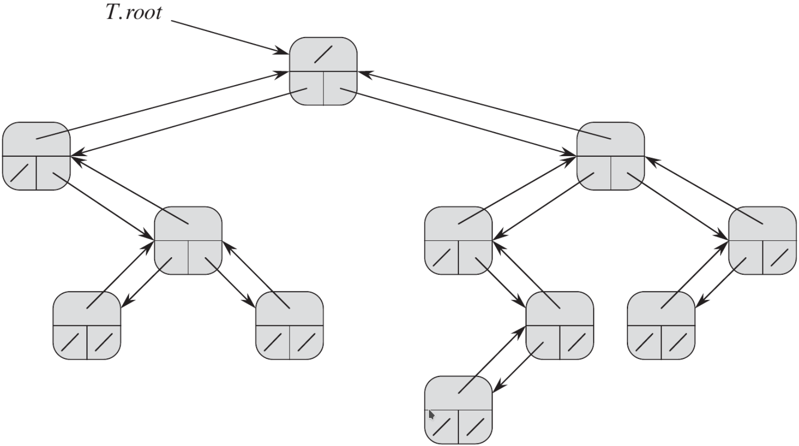


Árbol binario

Un árbol binario se puede representar como una estructura de datos enlazadas en donde cada nodo es un objeto.

Generalmente cada nodo tiene tres atributos el valor asociado (v), el apuntador al hijo izquierdo (*left*) y el apuntador al hijo derecho (*right*).

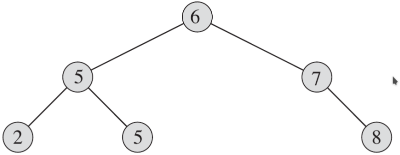
Algunas veces, se adiciona un tercer atributo para apuntar al padre (p). La raíz del árbol es el único nodo cuyo padre es NIL.



Árbol binario

Se puede recorrer todos los los nodos de un árbol de tres formas diferentes:

- *Inorden:*
  1. imprime en inorden el subarbol izquierdo
  2. imprime la raiz
  3. imprime en inorden el subarbol derecho
- *Preorden:*
  1. imprime la raiz
  2. imprime en preorden el subarbol izquierdo
  3. imprime en preorden el subarbol derecho
- *Postorden:*
  1. imprime en postorden el subarbol izquierdo
  2. imprime en postorden el subarbol derecho
  3. imprime la raiz



Árbol binario

Se puede recorrer todos los los nodos de un árbol de tres formas diferentes:

- *Inorden:*
  1. imprime en inorden el subarbol izquierdo
  2. imprime la raiz
  3. imprime en inorden el subarbol derecho
- *Preorden:*
  1. imprime la raiz
  2. imprime en preorden el subarbol izquierdo
  3. imprime en preorden el subarbol derecho
- *Postorden:*
  1. imprime en postorden el subarbol izquierdo
  2. imprime en postorden el subarbol derecho
  3. imprime la raiz

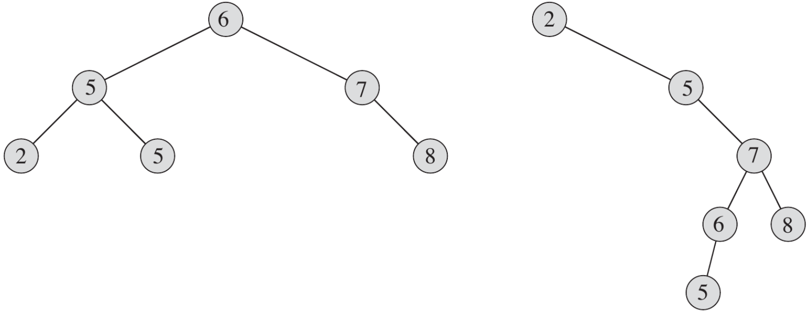
```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print x.v
    INORDER-TREE-WALK(x.right)
```

INORDER-TREE-WALK($T.root$)

Agenda

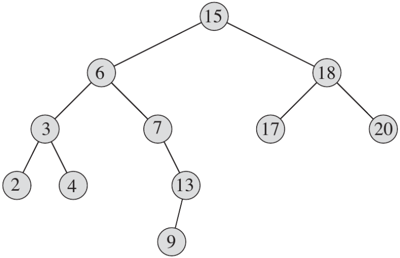
Arbol binario de búsqueda

Los arboles binarios de búsqueda **BST** tienen la llave como atributo adicional *key* y deben cumplir la siguiente condición: Sea x un nodo en un árbol binario de búsqueda. Si y es un nodo en el subárbol izquierdo de x, entonces $y.key <= x.key$.
Si y es un nodo en el subárbol derecho de x, entonces $y.key >= x.key$



Arbol binario de búsqueda

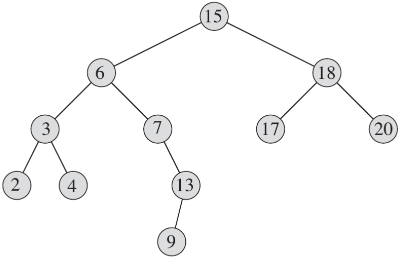
Búsqueda



Para buscar 13 seguimos la ruta,
$15 \to 6 \to 7 \to 13$

Arbol binario de búsqueda

Búsqueda



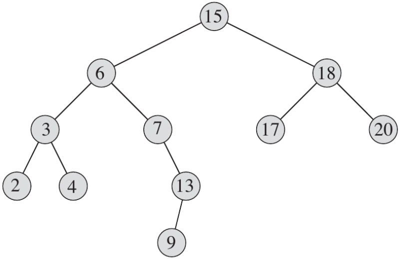

```
TREE-SEARCH(x, k)
if x == NIL or k == x.key
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)
```

$$O(h)$$

h es la altura del árbol

Arbol binario de búsqueda

Búsqueda



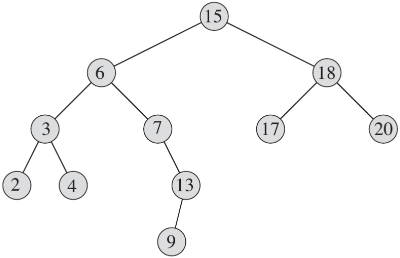

```
ITERATIVE-TREE-SEARCH(x, k)
    while x ≠ NIL and k ≠ x.key
        if k < x.key
            x = x.left
        else x = x.right
    return x
```

$$O(h)$$

h es la altura del árbol

Arbol binario de búsqueda

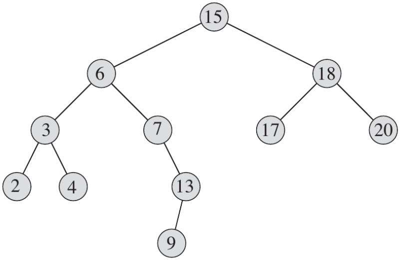
Mínimo y Máximo



- El mínimo, 2, se encuentra siguiendo los apuntadores izquierdos
- El máximo, 20, se encuentra siguiendo los aputadores derechos

Arbol binario de búsqueda

Búsqueda




```
TREE-MINIMUM(x)
    while x.left ≠ NIL
        x = x.left
    return x
```

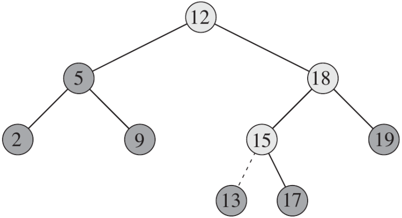
```
TREE-MAXIMUM(x)
    while x.right ≠ NIL
        x = x.right
    return x
```

$O(h)$

h es la altura del árbol

Arbol binario de búsqueda

Insertar

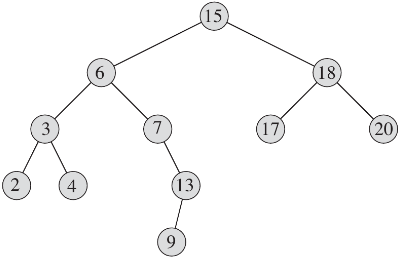


Para insertar 13,

debemos encontrar primero la posición en el arbol

Arbol binario de búsqueda

Búsqueda



TREE-INSERT(T, z)

```
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z          // tree T was empty
elseif z.key < y.key
    y.left = z
else y.right = z
```

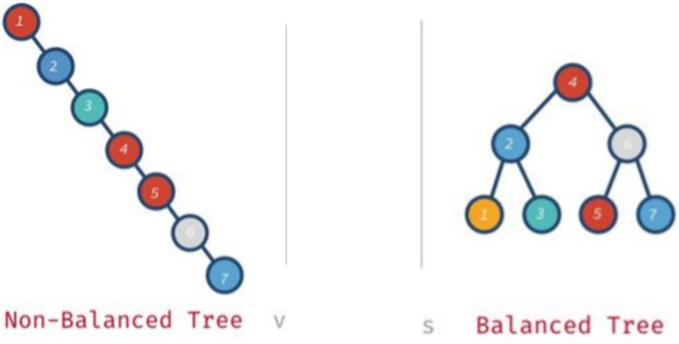
$$O(h)$$

h es la altura del árbol

Agenda

Arbol binario de búsqueda balanceados

Un **BST** está balanceado si dos subárboles hermanos no difieren en su altura por más de un nivel, o en otras palabras, no existen dos hojas cuya diferencia en profundidad sea mayor a un nivel.



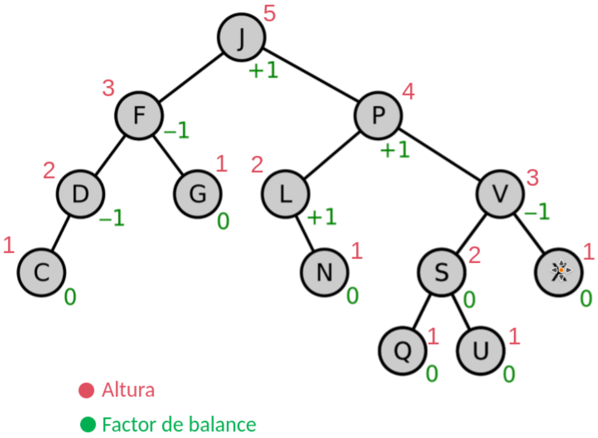
Altura h=n
O(n)

Altura h=log(n)
O(log(n))

Árbol AVL

Un árbol **AVL** es un tipo especial de árbol binario ordenado balanceado ideado por los matemáticos rusos **A**delson-**V**elskii y **L**andis.

Si los subárboles de un nodo tienen altura h_1 y h_2, entonces $\underbrace{|h_1 - h_2|}_{\text{Factor de balance}} \leq 1$

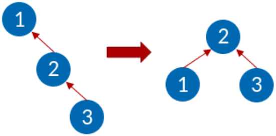


Árbol AVL

Rotaciones simples

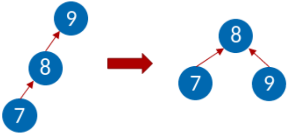
Rotaciones a la izquierda

Desbalance al insertar un nodo en el subárbol derecho de un subárbol derecho



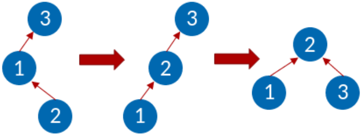
Rotaciones a la derecha

Desbalance al insertar un nodo en el subárbol izquierdo de un subárbol izquierdo

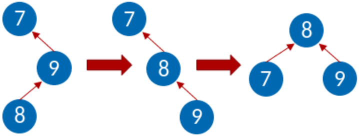


Rotaciones dobles

Rotaciones izquierda-derecha



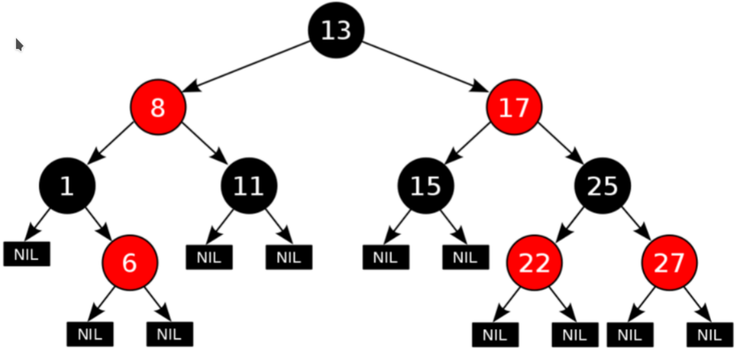
Rotaciones derecha-izquierda



Árbol Rojo Negro

Propiedades

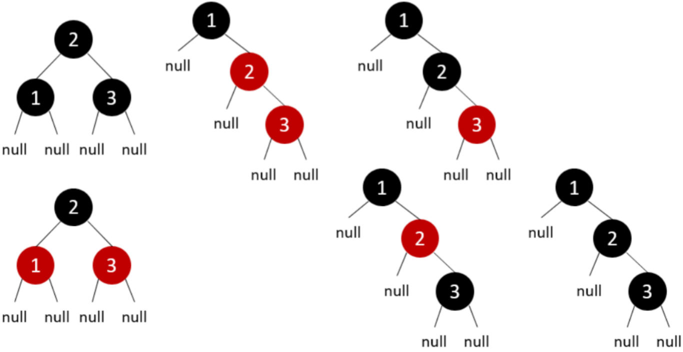
1. Cada nodo es **rojo** o **negro**
2. La raíz es de color **negro**
3. Dos nodos **rojos** no pueden aparecer consecutivamente. Si un nodo es **rojo**, sus dos hijos son de color **negro**
4. Todas las rutas desde la raíz a las hojas vacías (none) debe pasar por el mismo número de nodos **negros**



¿Son Rojo Negro?

Propiedades

1. Cada nodo es **rojo** o **negro**
2. La raíz es de color **negro**
3. Dos nodos **rojos** no pueden aparecer consecutivamente. Si un nodo es **rojo**, sus dos hijos son de color **negro**
4. Todas las rutas desde la raíz a las hojas vacías (None) debe pasar por el mismo número de nodos **negros**

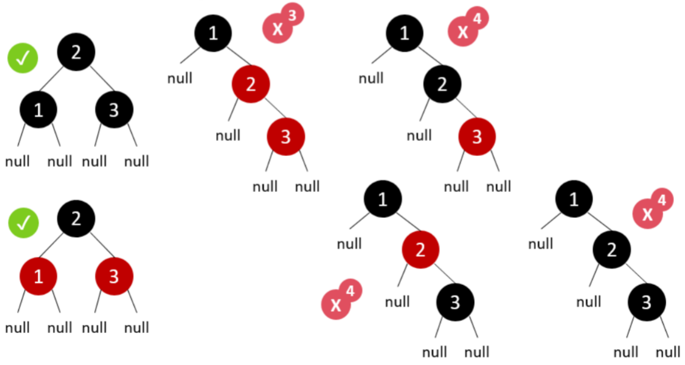


Árbol Rojo Negro

¿Son Rojo Negro?

Propiedades

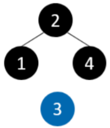
1. Cada nodo es **rojo** o **negro**
2. La raíz es de color **negro**
3. Dos nodos **rojos** no pueden aparecer consecutivamente. Si un nodo es **rojo**, sus dos hijos son de color **negro**
4. Todas las rutas desde la raíz a las hojas vacías (None) debe pasar por el mismo número de nodos **negros**



Árbol Rojo Negro

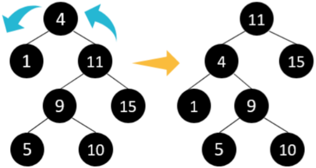
Insertando

Cambiar de color

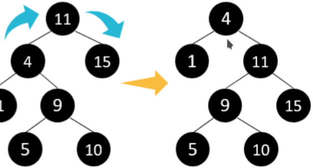


Rotaciones

Rotaciones a la izquierda



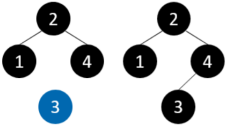
Rotaciones a la derecha



Árbol Rojo Negro

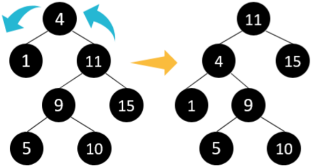
Insertando

Cambiar de color

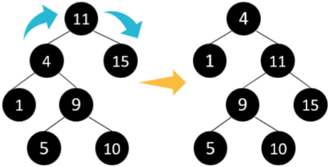


Rotaciones

Rotaciones a la izquierda

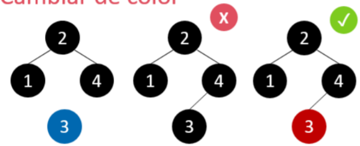


Rotaciones a la derecha



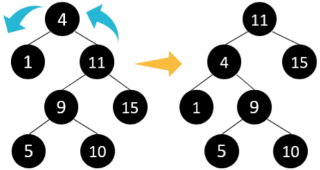
Insertando

Cambiar de color

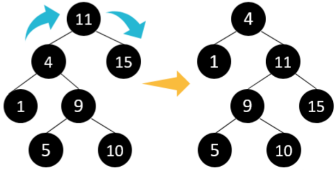


Rotaciones

Rotaciones a la izquierda

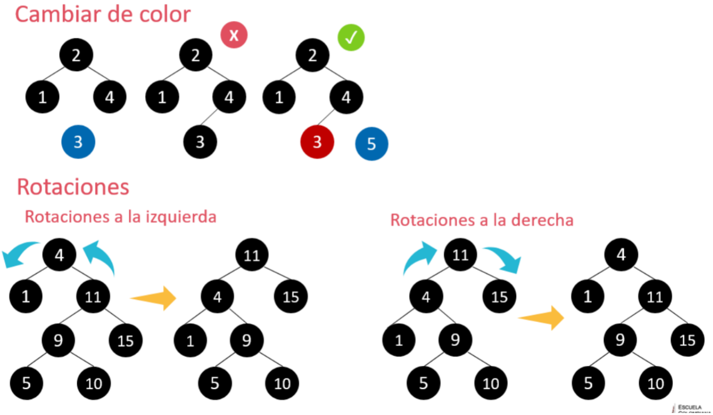


Rotaciones a la derecha



Árbol Rojo Negro

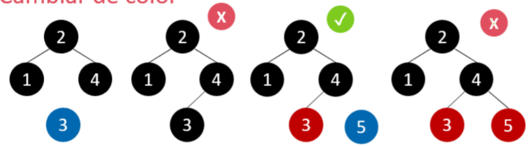
Insertando



Árbol Rojo Negro

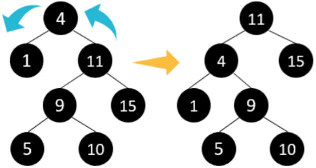
Insertando

Cambiar de color

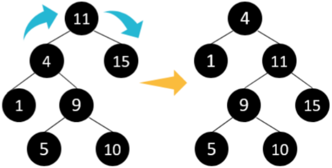


Rotaciones

Rotaciones a la izquierda

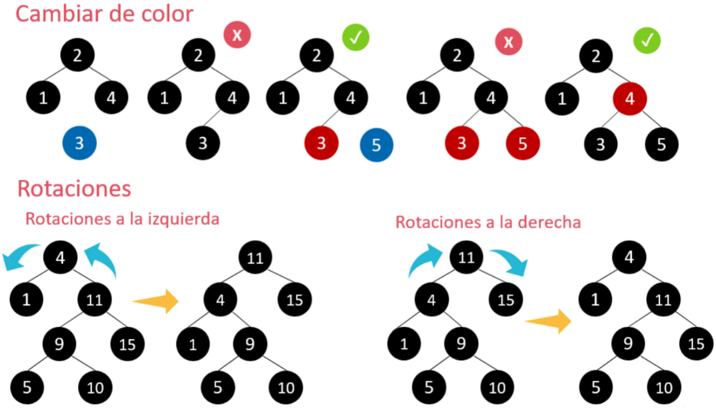


Rotaciones a la derecha



Árbol Rojo Negro

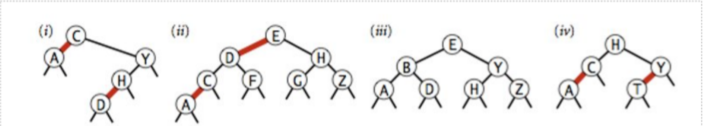
Insertando



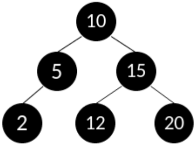
Agenda

Ejercicios

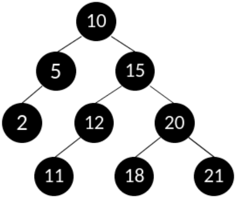
1. ¿Cuál de los siguientes árboles son red-black BSTs?



2. Dibuje el paso a paso de insertar las letras desde la A hasta la K, de manera ascendente y descendente, a un red-black BST que está vacío inicialmente.
3. Dibuje el árbol AVL resultante al aplicar las siguientes operaciones:



$insert(4)$



$insert(25)$

Ejercicios

12.2-1
Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could *not* be the sequence of nodes examined?

a. 2, 252, 401, 398, 330, 344, 397, 363.

b. 924, 220, 911, 244, 898, 258, 362, 363.

c. 925, 202, 911, 240, 912, 245, 363.

d. 2, 399, 387, 219, 266, 382, 381, 278, 363.

e. 935, 278, 347, 621, 299, 392, 358, 363.

12.2-2
Write recursive versions of TREE-MINIMUM and TREE-MAXIMUM.

12.2-3
Write the TREE-PREDECESSOR procedure.

Ejercicios

12.2-4

Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professor's claim.

12.2-5

Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

12.2-6

Consider a binary search tree T whose keys are distinct. Show that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x. (Recall that every node is its own ancestor.)