

# Árboles rojo negro

Clase 11

IIC 2133 - Sección 3

Prof. Eduardo Bustos

# Sumario

**Introducción**

Árboles rojo-negro

Inserciones

Cierre

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**



# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k$

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k$
  - las llaves  $k''$  del hijo derecho son  $k < k''$

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k$
  - las llaves  $k''$  del hijo derecho son  $k < k''$
- Si es 3-nodo con llaves  $k_1 < k_2$

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k$
  - las llaves  $k''$  del hijo derecho son  $k < k''$
- Si es 3-nodo con llaves  $k_1 < k_2$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k_1$

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k$
  - las llaves  $k''$  del hijo derecho son  $k < k''$
- Si es 3-nodo con llaves  $k_1 < k_2$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k_1$
  - las llaves  $k''$  del hijo central son  $k_1 < k'' < k_2$

# Árboles de búsqueda 2-3

## Definición

Un **árbol de búsqueda 2-3** es una EDD que almacena **(llave, valor)** según

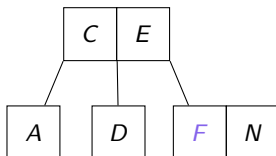
1. Un árbol 2-3 tiene una nodo que puede ser un **2-nodo** (con una llave) o un **3-nodo** (con 2 llaves distintas y ordenadas)
2. El nodo puede no tener hijos o tener exactamente
  - 2 hijos árboles 2-3 si es un 2-nodo
  - 3 hijos árboles 2-3 si es un 3-nodo

y que además, satisface la **propiedad de árboles 2-3**

- Si es 2-nodo con llave  $k$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k$
  - las llaves  $k''$  del hijo derecho son  $k < k''$
- Si es 3-nodo con llaves  $k_1 < k_2$ 
  - las llaves  $k'$  del hijo izquierdo son  $k' < k_1$
  - las llaves  $k''$  del hijo central son  $k_1 < k'' < k_2$
  - las llaves  $k'''$  del hijo derecho son  $k_2 < k'''$

# Inserciones en árboles 2-3

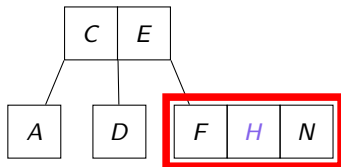
Insertamos la llave  $F$





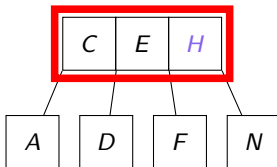
## Inserciones en árboles 2-3

Insertamos la llave *H* y se produce un nodo no válido



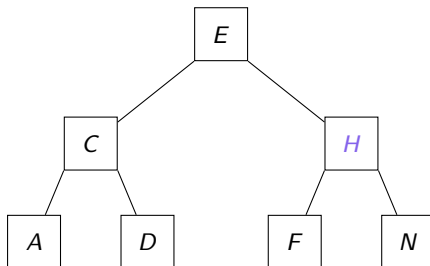
## Inserciones en árboles 2-3

Subimos la llave *H* y se produce un nuevo nodo no válido



## Inserciones en árboles 2-3

Hacemos **split** de la raíz actual, subiendo la llave *E* como nueva raíz



# Objetivos de la clase

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios
- ☐ Comprender el modelo de árbol rojo-negro

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios
- ☐ Comprender el modelo de árbol rojo-negro
- ☐ Comprender relación entre rojo-negro y árboles 2-4

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios
- ☐ Comprender el modelo de árbol rojo-negro
- ☐ Comprender relación entre rojo-negro y árboles 2-4
- ☐ Comprender inserción en rojo-negro con ayuda de árboles 2-4



# Sumario

Introducción

**Árboles rojo-negro**

Inserciones

Cierre

## Convirtiendo un 2-3 en binario

## Convirtiendo un 2-3 en binario

Para transformar un árbol 2-3 en un ABB nos centramos en los dos tipos de nodos

# Convirtiendo un 2-3 en binario

Para transformar un árbol 2-3 en un ABB nos centramos en los dos tipos de nodos

- Un **2-nodo** se puede representar como un nodo de un ABB sin problemas

# Convirtiendo un 2-3 en binario

Para transformar un árbol 2-3 en un ABB nos centramos en los dos tipos de nodos

- Un **2-nodo** se puede representar como un nodo de un ABB sin problemas
- Un **3-nodo** no se puede representar de esa forma... necesitamos separar las llaves y los hijos

# Convirtiendo un 2-3 en binario

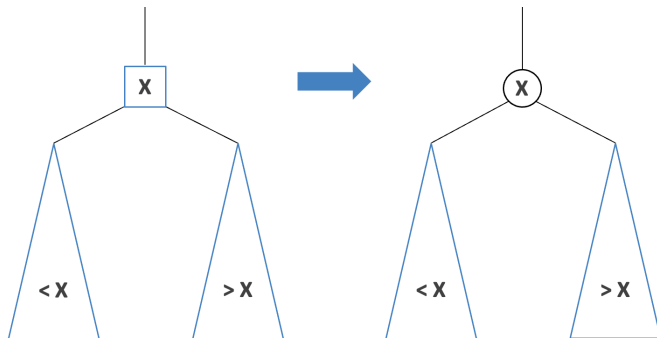
Para transformar un árbol 2-3 en un ABB nos centramos en los dos tipos de nodos

- Un **2-nodo** se puede representar como un nodo de un ABB sin problemas
- Un **3-nodo** no se puede representar de esa forma... necesitamos separar las llaves y los hijos

¿Cómo llevamos estos nodos a representación en ABB?

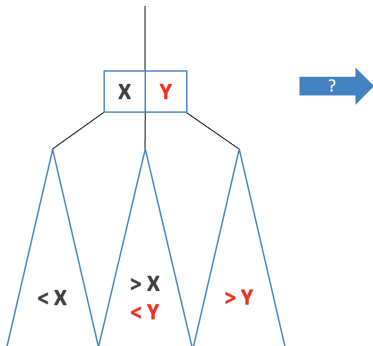
# Convirtiendo un 2-3 en binario

Los 2-nodos se representan igual



# Convirtiendo un 2-3 en binario

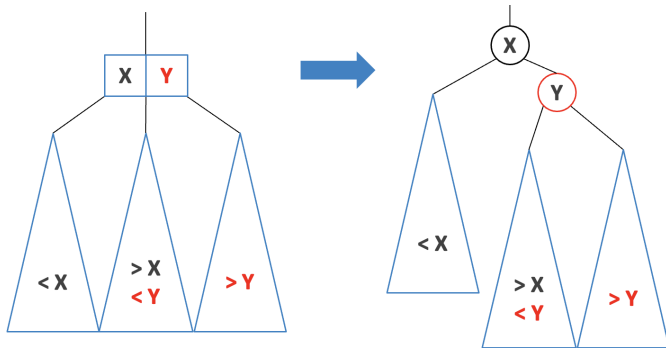
¿Cómo separamos las llaves de un 3-nodo?





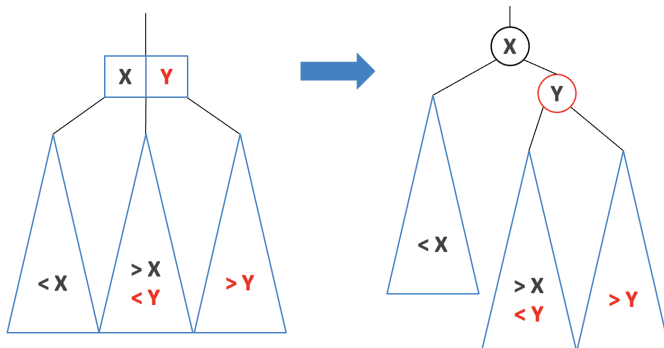
# Convirtiendo un 2-3 en binario

Reasignamos dos de los hijos a un nuevo nodo



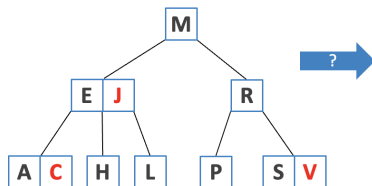
# Convirtiendo un 2-3 en binario

Reasignamos dos de los hijos a un nuevo nodo



Notemos que la diferencia de alturas entre los subárboles es 1

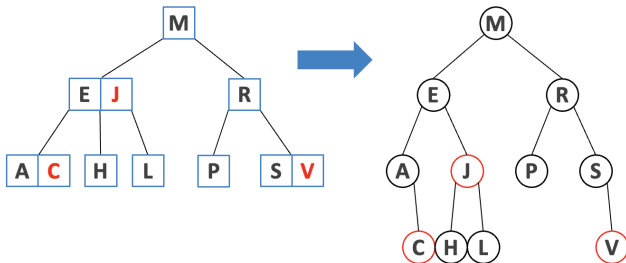
## Convirtiendo un 2-3 en binario



### Ejercicio

Convierta el árbol 2-3 anterior en un ABB

## Convirtiendo un 2-3 en binario



Esta coloración motiva una nueva idea de balance en ABBs

# Árboles rojo-negro

# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro

# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro
2. La raíz del árbol es negra



# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro
2. La raíz del árbol es negra
3. Si un nodo es rojo, sus hijos deben ser negros

# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro
2. La raíz del árbol es negra
3. Si un nodo es rojo, sus hijos deben ser negros
4. La cantidad de nodos negros camino a cada hoja desde un nodo cualquiera debe ser la misma

# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro
2. La raíz del árbol es negra
3. Si un nodo es rojo, sus hijos deben ser negros
4. La cantidad de nodos negros camino a cada hoja desde un nodo cualquiera debe ser la misma

Adicionalmente, las hojas vacías se consideran nodos negros

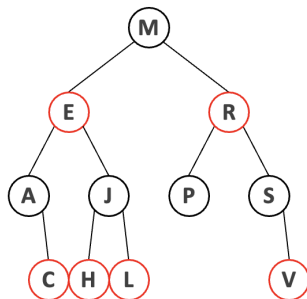
# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro
2. La raíz del árbol es negra
3. Si un nodo es rojo, sus hijos deben ser negros
4. La cantidad de nodos negros camino a cada hoja desde un nodo cualquiera debe ser la misma

Adicionalmente, las hojas vacías se consideran nodos negros



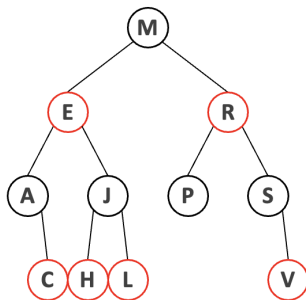
# Árboles rojo-negro

## Definición

Un **árbol rojo-negro** es un ABB que cumple

1. Cada nodo es rojo o negro
2. La raíz del árbol es negra
3. Si un nodo es rojo, sus hijos deben ser negros
4. La cantidad de nodos negros camino a cada hoja desde un nodo cualquiera debe ser la misma

Adicionalmente, las hojas vacías se consideran nodos negros



Esta es una nueva noción de balance en ABBs

# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

- Tendremos una estrategia de restauración (rotaciones)

# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

- Tendremos una estrategia de restauración (rotaciones)
- En lugar de usar *x.balance* usaremos *x.color*



# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

- Tendremos una estrategia de restauración (rotaciones)
- En lugar de usar *x.balance* usaremos *x.color*

Para facilitar la comprensión del rebalanceo, notamos que

# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

- Tendremos una estrategia de restauración (rotaciones)
- En lugar de usar *x.balance* usaremos *x.color*

Para facilitar la comprensión del rebalanceo, notamos que

- Los árboles 2-3 son fáciles de visualizar

# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

- Tendremos una estrategia de restauración (rotaciones)
- En lugar de usar *x.balance* usaremos *x.color*

Para facilitar la comprensión del rebalanceo, notamos que

- Los árboles 2-3 son fáciles de visualizar
- No todo árbol rojo-negro tiene un árbol 2-3 equivalente

# Árboles rojo-negro

Al igual que en los árboles AVL, los cambios en el árbol pueden romper la propiedad de balance

- Tendremos una estrategia de restauración (rotaciones)
- En lugar de usar *x.balance* usaremos *x.color*

Para facilitar la comprensión del rebalanceo, notamos que

- Los árboles 2-3 son fáciles de visualizar
- No todo árbol rojo-negro tiene un árbol 2-3 equivalente
- Pero sí tiene un **árbol 2-4 equivalente**

# Árboles rojo-negro y árboles 2-4

# Árboles rojo-negro y árboles 2-4

Un **árbol 2-4** es un árbol 2-3 que además puede tener **4-nodos**

# Árboles rojo-negro y árboles 2-4

Un **árbol 2-4** es un árbol 2-3 que además puede tener **4-nodos**

- tiene 3 llaves ordenadas distintas

# Árboles rojo-negro y árboles 2-4

Un **árbol 2-4** es un árbol 2-3 que además puede tener **4-nodos**

- tiene 3 llaves ordenadas distintas
- si no es hoja, tiene 4 hijos



# Árboles rojo-negro y árboles 2-4

Un **árbol 2-4** es un árbol 2-3 que además puede tener **4-nodos**

- tiene 3 llaves ordenadas distintas
- si no es hoja, tiene 4 hijos

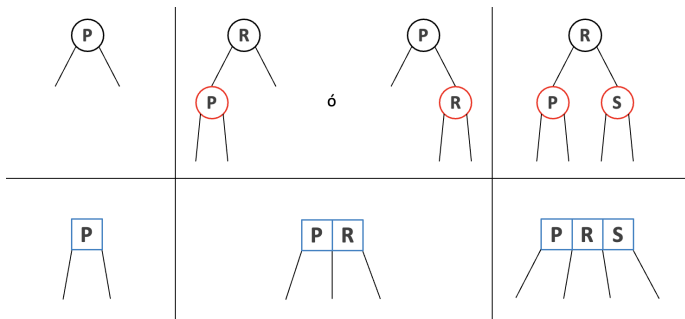
Con este nuevo modelo, podemos tener árboles equivalentes según

# Árboles rojo-negro y árboles 2-4

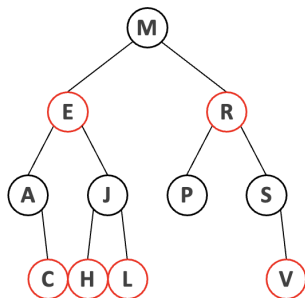
Un **árbol 2-4** es un árbol 2-3 que además puede tener **4-nodos**

- tiene 3 llaves ordenadas distintas
- si no es hoja, tiene 4 hijos

Con este nuevo modelo, podemos tener árboles equivalentes según



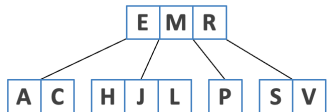
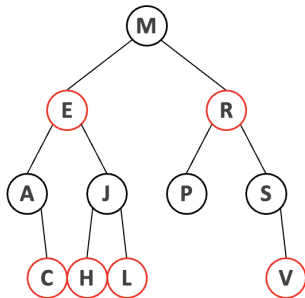
## Árboles rojo-negro y árboles 2-4



### Ejercicio

Obtenga un árbol 2-4 equivalente al árbol rojo-negro anterior

## Árboles rojo-negro y árboles 2-4



# Sumario

Introducción

Árboles rojo-negro

**Inserciones**

Cierre

# Inserción en árboles rojo-negro

# Inserción en árboles rojo-negro

Estudiaremos cómo insertar nodos en árboles rojo negro

# Inserción en árboles rojo-negro

Estudiaremos cómo insertar nodos en árboles rojo negro

- Usaremos un árbol 2-4 equivalente como ayuda



# Inserción en árboles rojo-negro

Estudiaremos cómo insertar nodos en árboles rojo negro

- Usaremos un árbol 2-4 equivalente como ayuda
- Nos permitirá comprender mejor cómo efectuar **rotaciones** y **cambios de color**

# Inserción en árboles rojo-negro

Estudiaremos cómo insertar nodos en árboles rojo negro

- Usaremos un árbol 2-4 equivalente como ayuda
- Nos permitirá comprender mejor cómo efectuar **rotaciones** y **cambios de color**

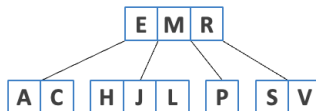
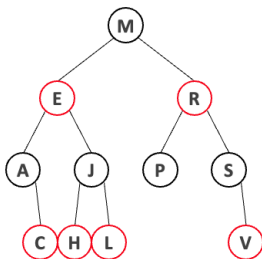
Trabajaremos con el siguiente árbol como punto de partida

# Inserción en árboles rojo-negro

Estudiaremos cómo insertar nodos en árboles rojo negro

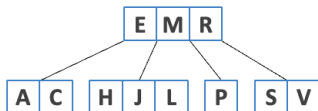
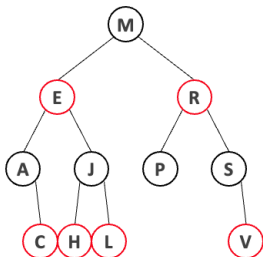
- Usaremos un árbol 2-4 equivalente como ayuda
- Nos permitirá comprender mejor cómo efectuar **rotaciones** y **cambios de color**

Trabajaremos con el siguiente árbol como punto de partida



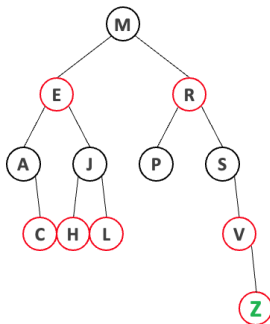
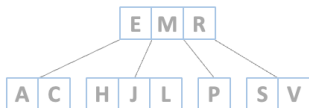
# Inserción en árboles rojo-negro

Insertamos la llave Z. ¿Dónde debiera ubicarse?



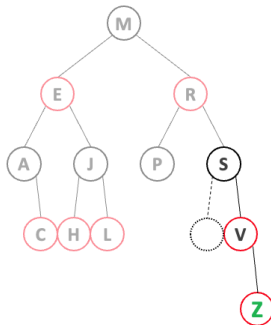
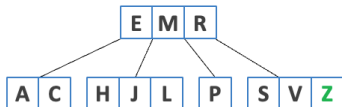
# Inserción en árboles rojo-negro

Para no romper la propiedad 4 de los rojo-negro, insertamos siempre como **nodo rojo**



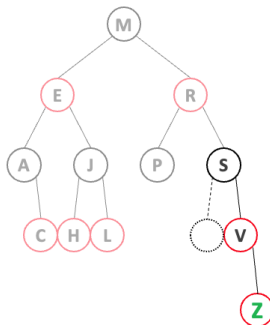
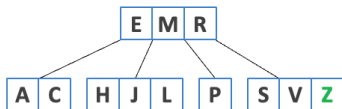
# Inserción en árboles rojo-negro

Actualizamos el árbol 2-4 equivalente, el cual nos sugiere una posible rotación de los nodos  $S - V$



# Inserción en árboles rojo-negro

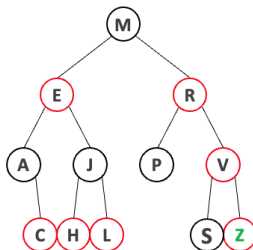
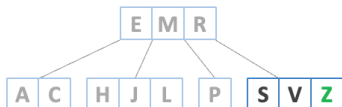
Además, notamos que el tío del nodo nuevo  $Z$  es negro (pues es un nodo vacío)



¿Qué pasa si el tío es rojo?  
Lo veremos más adelante

# Inserción en árboles rojo-negro

Efectuamos la rotación de acuerdo a lo que nos sugiere el nodo SVZ del 2-4

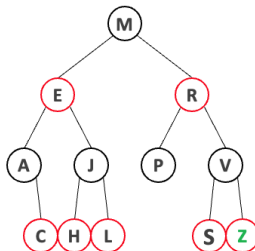
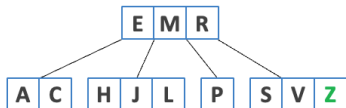


Ojo que ahora se rompe la propiedad 4 (los colores)



# Inserción en árboles rojo-negro

Cambiamos color de *S* y *V*, los nodos que se rotaron



Esta rotación/coloración fue suficiente  
para entregar un nuevo árbol rojo-negro

# Inserción en árboles rojo-negro

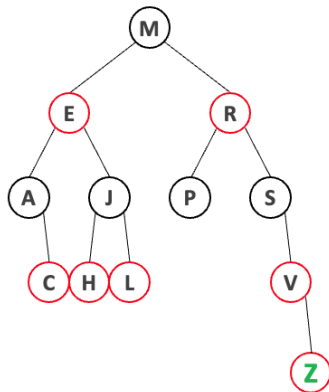
A este tipo de inserción le llamaremos **exterior**

**input** : Nodo  $x$

**output**:  $\emptyset$

FixBalance ( $x$ ):

**if**  $x$  fue inserción exterior :



# Inserción en árboles rojo-negro

A este tipo de inserción le llamaremos **exterior**

**input** : Nodo  $x$

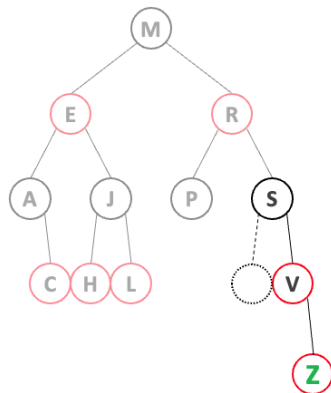
**output**:  $\emptyset$

**FixBalance** ( $x$ ):

**if**  $x$  fue inserción exterior :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = negro$  :



# Inserción en árboles rojo-negro

A este tipo de inserción le llamaremos **exterior**

**input** : Nodo  $x$

**output**:  $\emptyset$

**FixBalance** ( $x$ ):

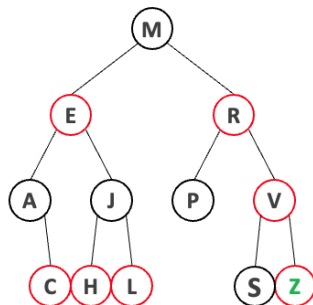
**if**  $x$  fue inserción exterior :

$t \leftarrow x.\text{uncle} \triangleright$  tío de  $x$

**if**  $t.\text{color} = \text{negro}$  :

$g \leftarrow x.p.p \triangleright$  abuelo de  $x$

    Rotation( $g, x.p$ )



# Inserción en árboles rojo-negro

A este tipo de inserción le llamaremos **exterior**

**input** : Nodo  $x$

**output**:  $\emptyset$

FixBalance ( $x$ ):

if  $x$  fue inserción exterior :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

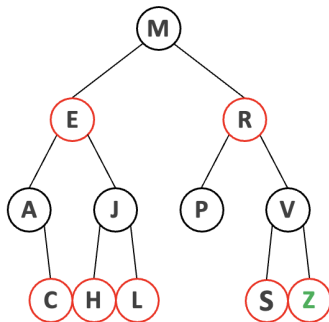
if  $t.color = negro$  :

$g \leftarrow x.p.p \triangleright$  abuelo de  $x$

Rotation( $g, x.p$ )

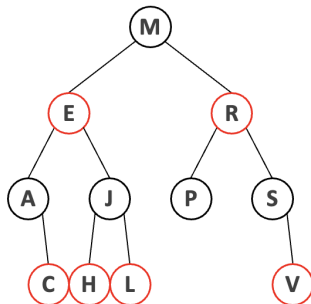
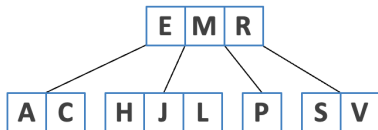
$x.p.color \leftarrow negro$

$g.color \leftarrow rojo$



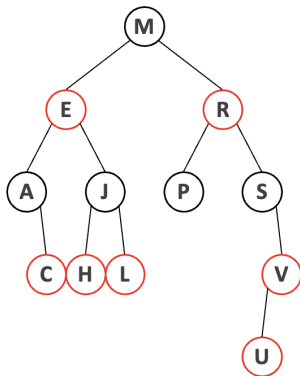
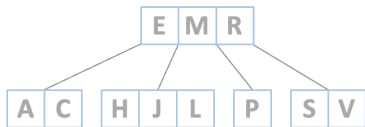
# Insertión en árboles rojo-negro

Nueva inserción: insertamos la llave  $U$ . ¿Dónde debiera ubicarse?



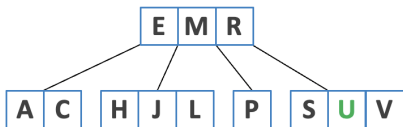
# Inserción en árboles rojo-negro

Tal como antes, la insertamos como nodo rojo... a este tipo de inserción le llamamos **interior**



# Inserción en árboles rojo-negro

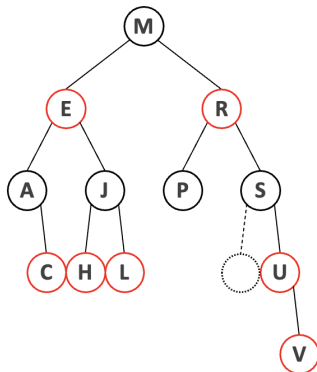
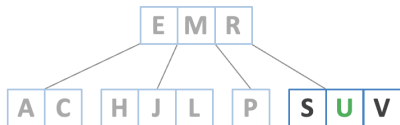
Vemos que tiene tío negro y al actualizar el 2-4 se nos sugiere una rotación





# Inserción en árboles rojo-negro

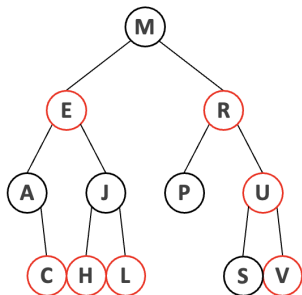
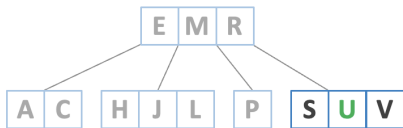
Efectuamos primera rotación  $U - V$



En este punto ya estamos con un escenario como el caso de inserción exterior

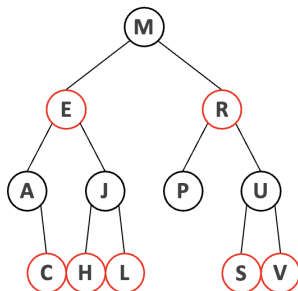
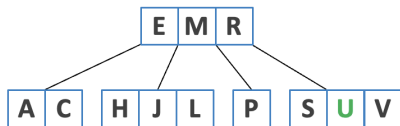
# Inserción en árboles rojo-negro

Luego, una segunda rotación  $S - U$  que deja a  $U$  como padre de  $S, V$



# Inserción en árboles rojo-negro

Finalmente, cambiamos el color de los últimos nodos rotados



# Inserción en árboles rojo-negro

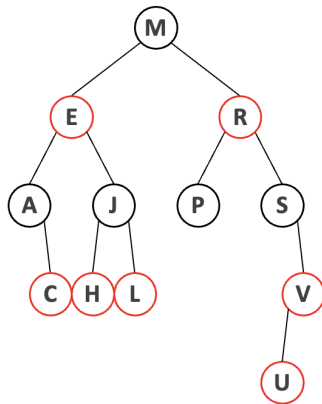
Resumimos la estrategia para una inserción **interior**

**input** : Nodo  $x$

**output**:  $\emptyset$

**FixBalance** ( $x$ ):

**if**  $x$  fue inserción interior :



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción **interior**

**input** : Nodo  $x$

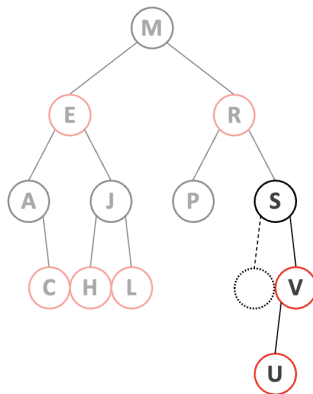
**output**:  $\emptyset$

**FixBalance** ( $x$ ):

**if**  $x$  fue inserción interior :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = negro$  :



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción **interior**

**input** : Nodo  $x$

**output**:  $\emptyset$

FixBalance ( $x$ ):

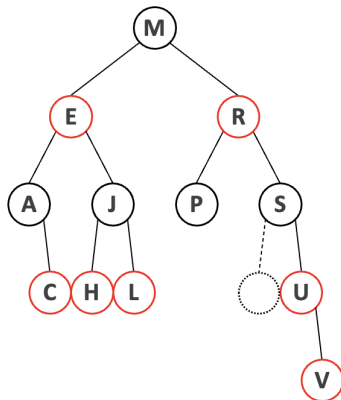
if  $x$  fue inserción interior :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

if  $t.color = negro$  :

$g \leftarrow x.p.p \triangleright$  abuelo de  $x$

Rotation( $x.p, x$ )



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción **interior**

**input** : Nodo  $x$

**output**:  $\emptyset$

FixBalance ( $x$ ):

**if**  $x$  fue inserción interior :

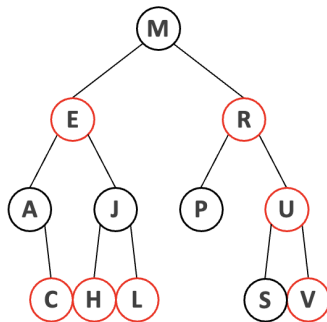
$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = negro$  :

$g \leftarrow x.p.p \triangleright$  abuelo de  $x$

      Rotation( $x.p, x$ )

      Rotation( $g, x$ )



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción **interior**

**input** : Nodo  $x$

**output**:  $\emptyset$

**FixBalance** ( $x$ ):

**if**  $x$  fue inserción interior :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = negro$  :

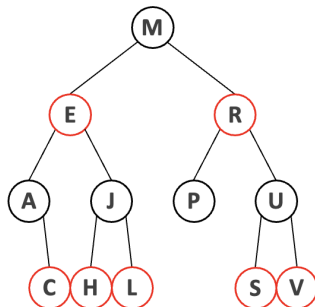
$g \leftarrow x.p.p \triangleright$  abuelo de  $x$

      Rotation( $x.p, x$ )

      Rotation( $g, x$ )

$x.color \leftarrow negro$

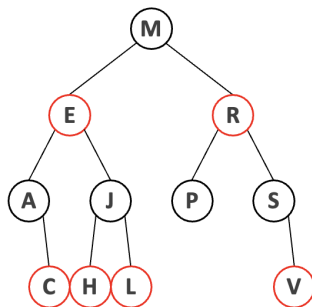
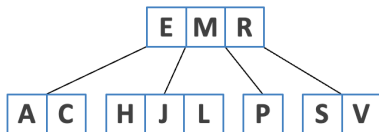
$g.color \leftarrow rojo$





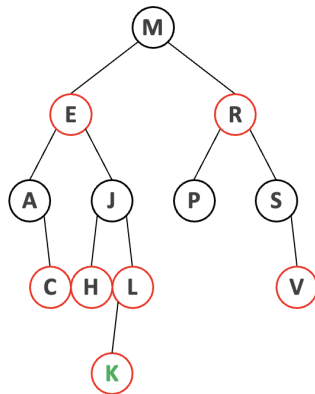
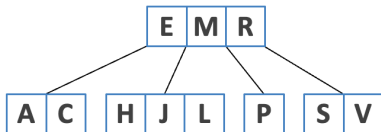
# Inserción en árboles rojo-negro

Nueva inserción: insertamos la llave  $K$ . ¿Dónde debiera ubicarse?



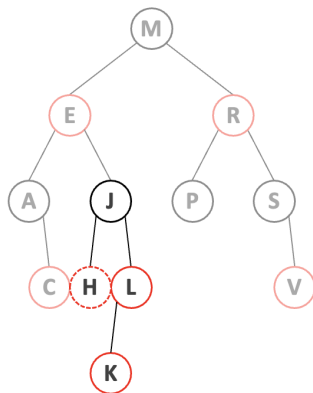
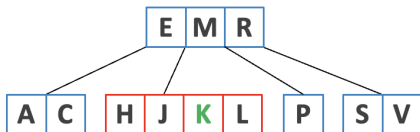
# Inserción en árboles rojo-negro

Lo agregamos como hoja roja



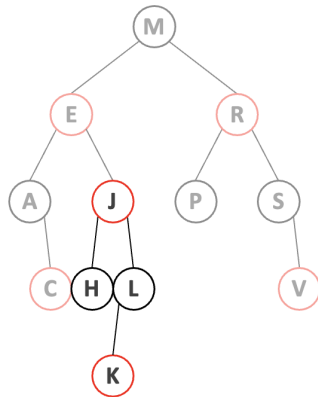
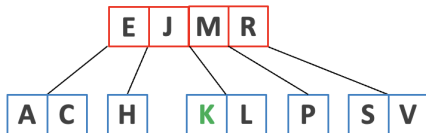
# Inserción en árboles rojo-negro

Actualizamos el árbol 2-4 y notamos un conflicto: notemos que el tío de  $K$  es rojo



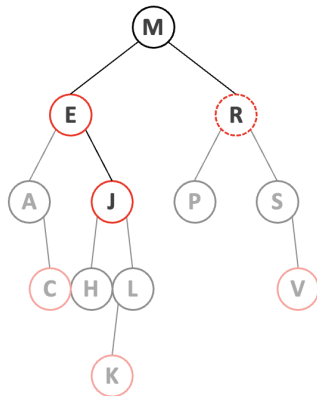
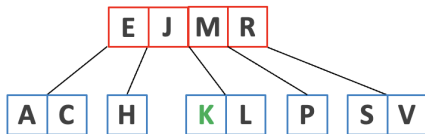
# Inserción en árboles rojo-negro

Al modificar el 5-nodo ilegal, se nos sugiere el cambio de colores en el árbol rojo-negro



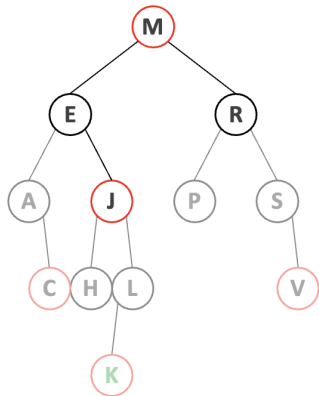
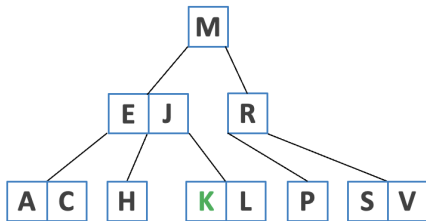
# Inserción en árboles rojo-negro

Revisamos recursivamente hacia arriba, revisando el tío de *J*, que nuevamente es rojo



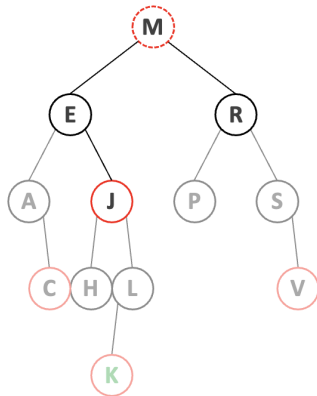
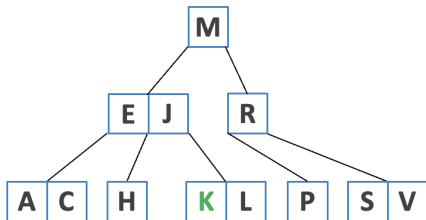
# Inserción en árboles rojo-negro

Nuevo cambio de colores que involucra solo a los tres nodos superiores



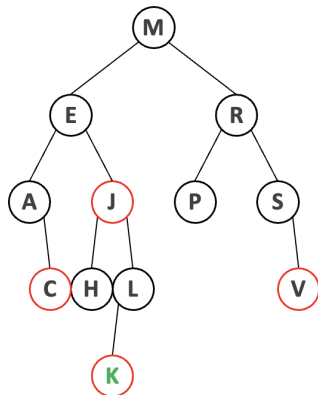
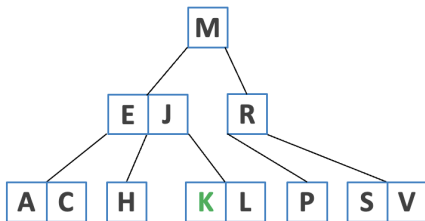
# Inserción en árboles rojo-negro

No hay más tíos que revisar, pero ahora falla la condición de que la raíz sea negra



# Inserción en árboles rojo-negro

Le cambiamos su color y terminamos





# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción con tío rojo

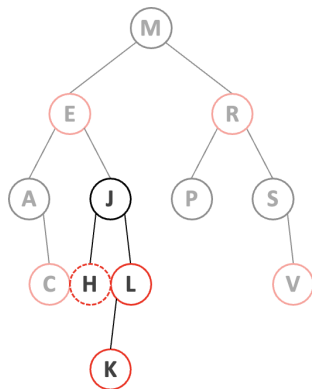
**input** : Nodo  $x$ , árbol rojo-negro  $A$

**output:**  $\emptyset$

**FixBalance** ( $x$ ):

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = rojo$  :



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción con tío rojo

**input** : Nodo  $x$ , árbol rojo-negro  $A$

**output**:  $\emptyset$

FixBalance ( $x$ ):

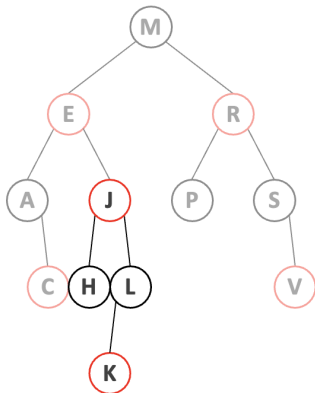
$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = rojo$  :

$x.p.color \leftarrow negro$

$t.color \leftarrow negro$

$x.p.p.color \leftarrow rojo$



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción con tío rojo

**input** : Nodo  $x$ , árbol rojo-negro  $A$

**output:**  $\emptyset$

FixBalance ( $x$ ):

**while**  $x.p \neq \emptyset \wedge x.p.color = rojo$  :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = rojo$  :

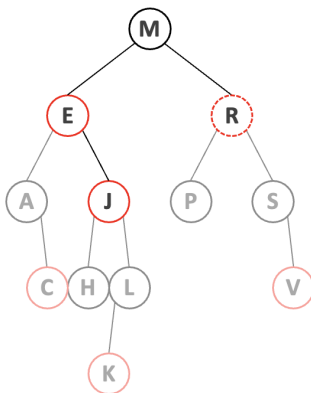
$x.p.color \leftarrow negro$

$t.color \leftarrow negro$

$x.p.p.color \leftarrow rojo$

$x \leftarrow x.p.p$

$A.root.color \leftarrow negro$



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción con tío rojo

**input** : Nodo  $x$ , árbol rojo-negro  $A$

**output:**  $\emptyset$

**FixBalance** ( $x$ ):

**while**  $x.p \neq \emptyset \wedge x.p.color = rojo$  :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = rojo$  :

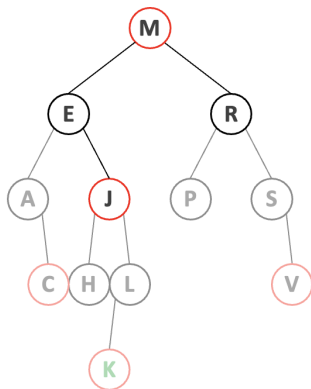
$x.p.color \leftarrow negro$

$t.color \leftarrow negro$

$x.p.p.color \leftarrow rojo$

$x \leftarrow x.p.p$

$A.root.color \leftarrow negro$



# Inserción en árboles rojo-negro

Resumimos la estrategia para una inserción con tío rojo

**input** : Nodo  $x$ , árbol rojo-negro  $A$

**output:**  $\emptyset$

**FixBalance** ( $x$ ):

**while**  $x.p \neq \emptyset \wedge x.p.color = rojo$  :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = rojo$  :

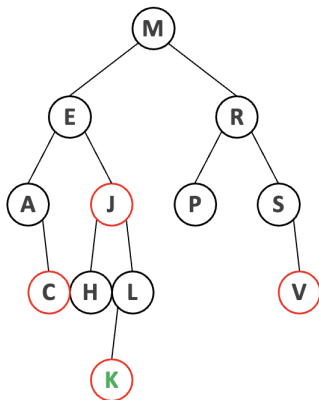
$x.p.color \leftarrow negro$

$t.color \leftarrow negro$

$x.p.p.color \leftarrow rojo$

$x \leftarrow x.p.p$

$A.root.color \leftarrow negro$



# Inserción en árboles rojo-negro

Al insertar, siempre lo hacemos como nodo rojo

# Inserción en árboles rojo-negro

Al insertar, siempre lo hacemos como nodo rojo

- Si el padre es negro, no hacemos nada

# Inserción en árboles rojo-negro

Al insertar, siempre lo hacemos como nodo rojo

- Si el padre es negro, no hacemos nada
- Si el padre es rojo, hay dos casos según el tío



# Inserción en árboles rojo-negro

Al insertar, siempre lo hacemos como nodo rojo

- Si el padre es negro, no hacemos nada
- Si el padre es rojo, hay dos casos según el tío
  - Tío negro: el nodo del árbol 2-4 crece, pero no colapsa

rotaciones y cambios de color

# Inserción en árboles rojo-negro

Al insertar, siempre lo hacemos como nodo rojo

- Si el padre es negro, no hacemos nada
- Si el padre es rojo, hay dos casos según el tío
  - Tío negro: el nodo del árbol 2-4 crece, pero no colapsa  
rotaciones y cambios de color
  - Tío rojo: el nodo del árbol 2-4 colapsa y hay que subir  
cambios de color hacia la raíz

# Inserción en árboles rojo-negro

**FixBalance** ( $x$ ):

**while**  $x.p \neq \emptyset \wedge x.p.color = rojo$  :

$t \leftarrow x.uncle \triangleright$  tío de  $x$

**if**  $t.color = rojo$  :

$x.p.color \leftarrow negro$

$t.color \leftarrow negro$

$x.p.p.color \leftarrow rojo$

$x \leftarrow x.p.p$

**else:**

**if**  $x$  es hijo interior de  $x.p$  :

$Rotation(x.p, x)$

$x \leftarrow x.p$

$x.p.color \leftarrow negro$

$x.p.p.color \leftarrow rojo$

$Rotation(x.p.p, x)$

$A.root.color \leftarrow negro$

# Inserción en árboles rojo-negro

## Ejercicio

Inserte en un árbol rojo-negro vacío las siguientes llaves consecutivas

41, 38, 31, 12, 19, 8

# Inserción en árboles rojo-negro

Insertamos el 41 como raíz

Insert 41

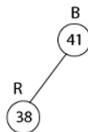
B



# Inserción en árboles rojo-negro

Insertamos el 38 y terminamos, pues su padre es negro

Insert 38



# Inserción en árboles rojo-negro

Insertamos el 31 y es hijo exterior de un nodo rojo: rotación+cambio

Insert 31



# Inserción en árboles rojo-negro

Insertamos el 12, hijo exterior de nodo rojo con tío rojo: cambios de color

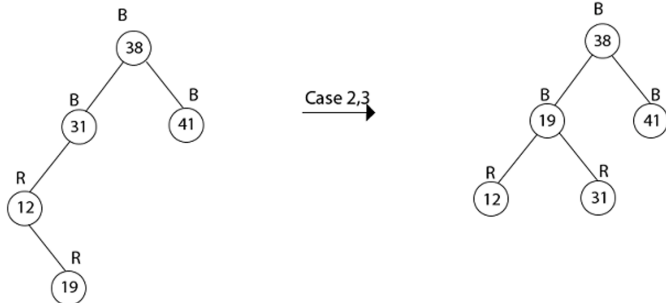
Insert 12





# Inserción en árboles rojo-negro

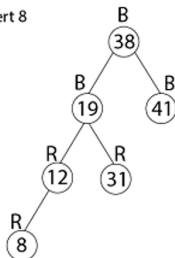
Insertamos el 19, hijo interior de nodo rojo con tío negro: rotación doble + cambio



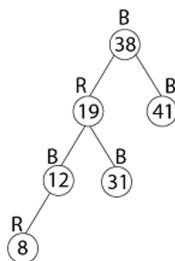
# Inserción en árboles rojo-negro

Insertamos el 8, hijo de nodo rojo y tío rojo: cambios de color

◀ Insert 8



Case-1 →



# Sumario

Introducción

Árboles rojo-negro

Inserciones

**Cierre**

# Objetivos de la clase

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios
- ☐ Comprender el modelo de árbol rojo-negro

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios
- ☐ Comprender el modelo de árbol rojo-negro
- ☐ Comprender relación entre rojo-negro y árboles 2-4

# Objetivos de la clase

- ☐ Representar árboles 2-3 como binarios
- ☐ Comprender el modelo de árbol rojo-negro
- ☐ Comprender relación entre rojo-negro y árboles 2-4
- ☐ Comprender inserción en rojo-negro con ayuda de árboles 2-4