Balance

La clase pasada definimos el **balance AVL**

- Las alturas de sus hijos no difieren en más que 1 entre ellas
- Cada hijo a su vez está **AVL-balanceado**

¿Será posible tener otra noción de balance?

Árboles balanceados de otra manera

Queremos un árbol de búsqueda en que el balance esté dado porque todas las hojas están a la misma profundidad

¿Es esto posible con árboles binarios? ¿Y ternarios?

¿Será posible combinarlos?

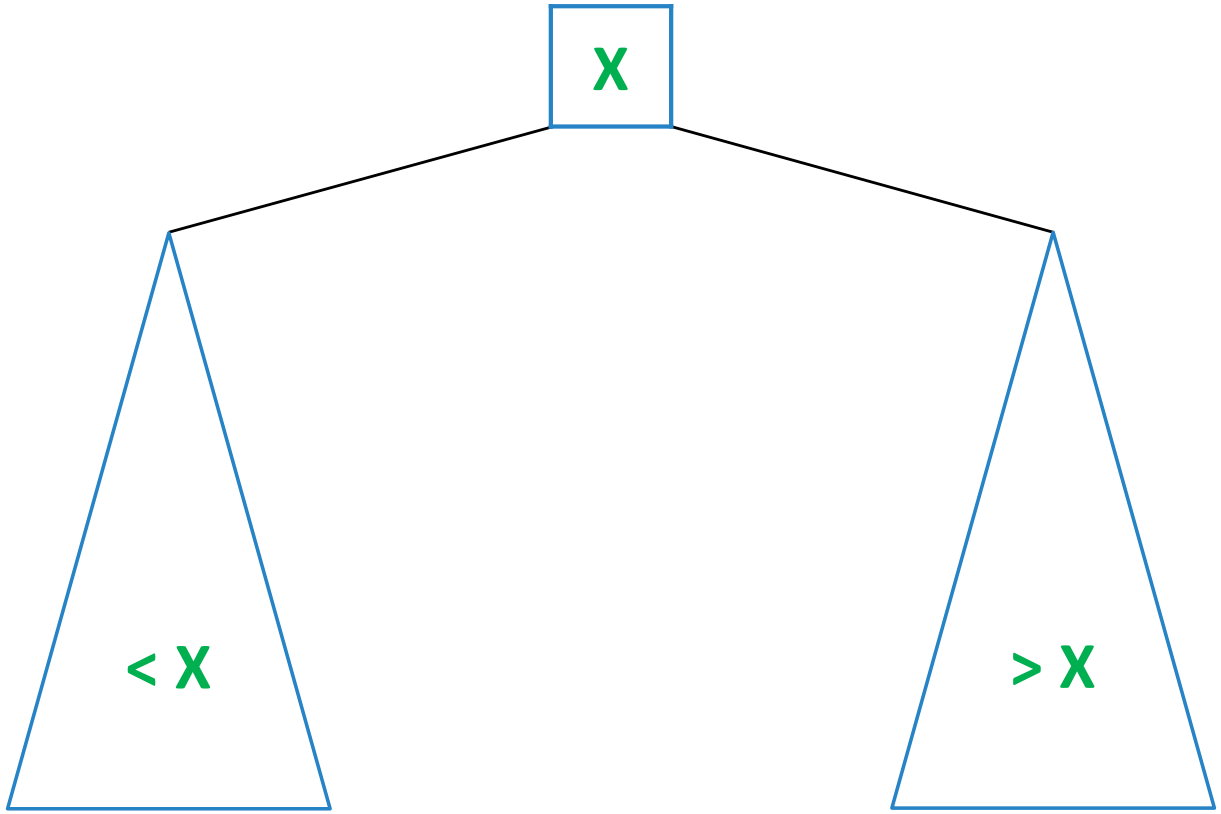
Árboles 2-3

En un árbol 2-3, hay dos tipos de nodos:

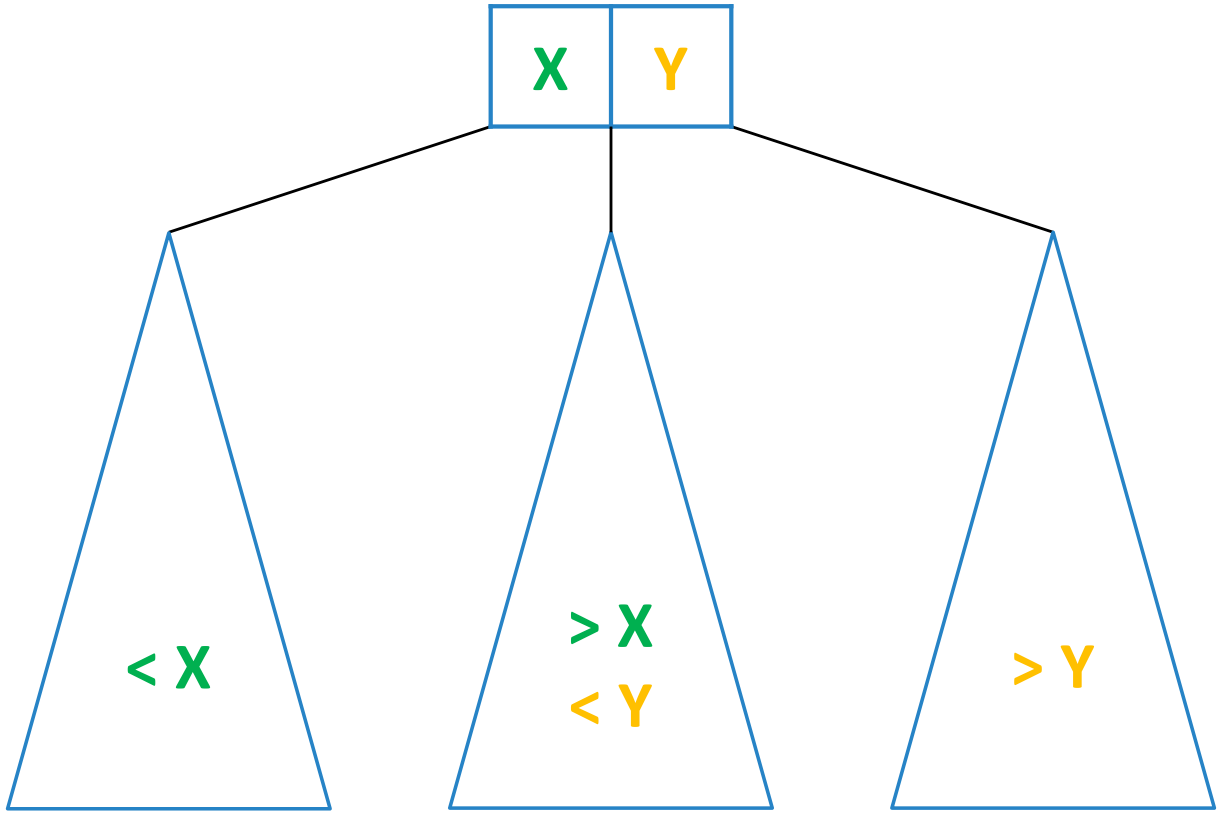
- *Nodo 2*, con 2 hijos y una clave
- *Nodo 3*, con 3 hijos y dos claves distintas y ordenadas

Esto permite que todas las hojas estén a la misma profundidad

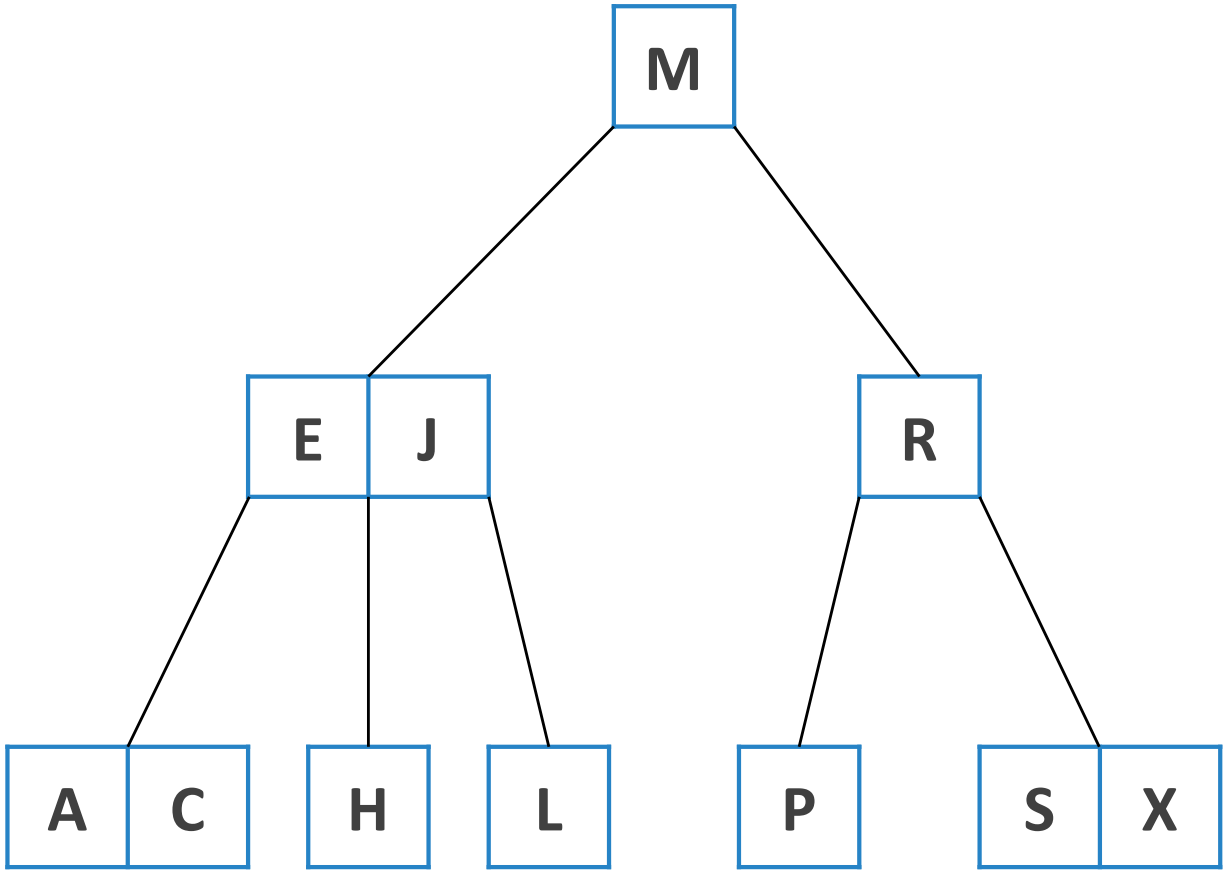
Nodo 2
(los árboles 2-3 son árboles de búsqueda)



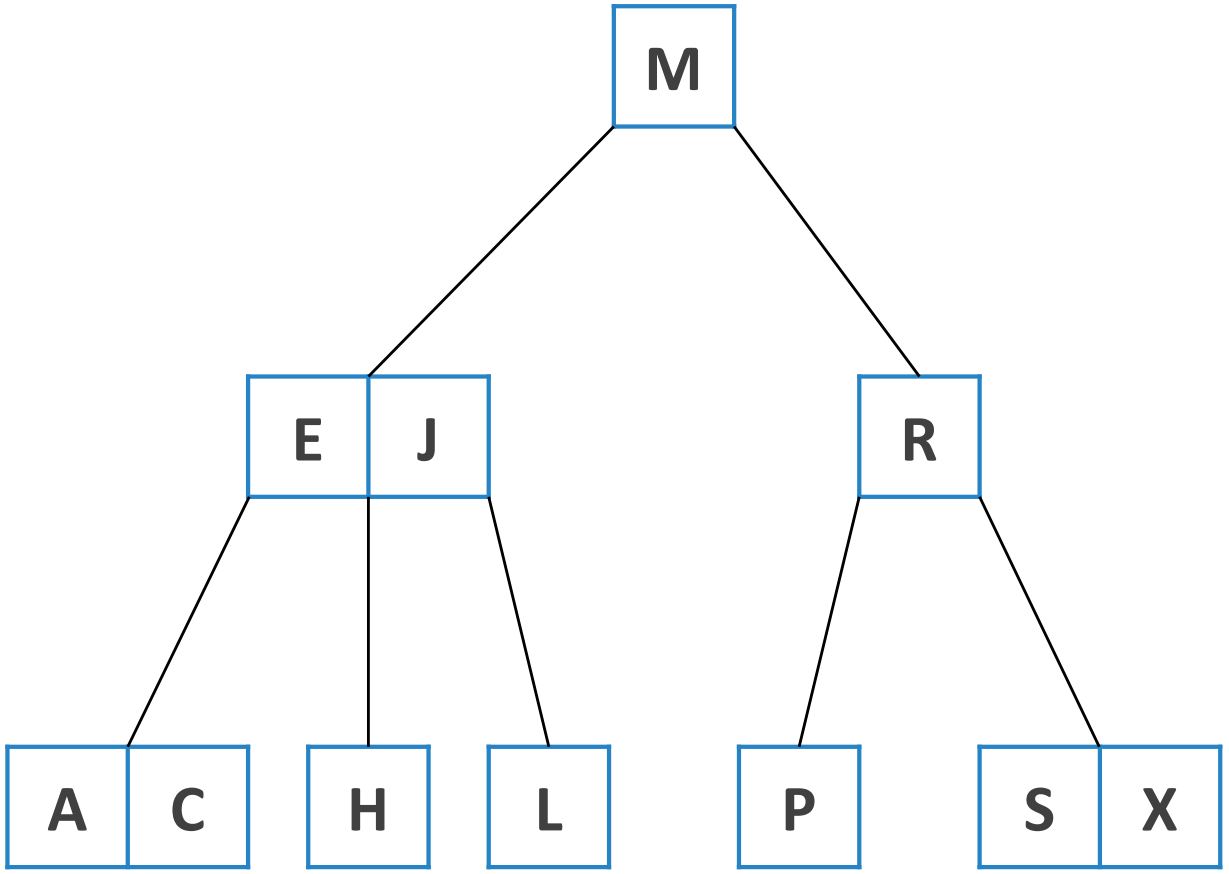
Nodo 3
(los árboles 2-3 son árboles de búsqueda)



Ejemplo de árbol 2-3
(notar que las claves están ordenadas)

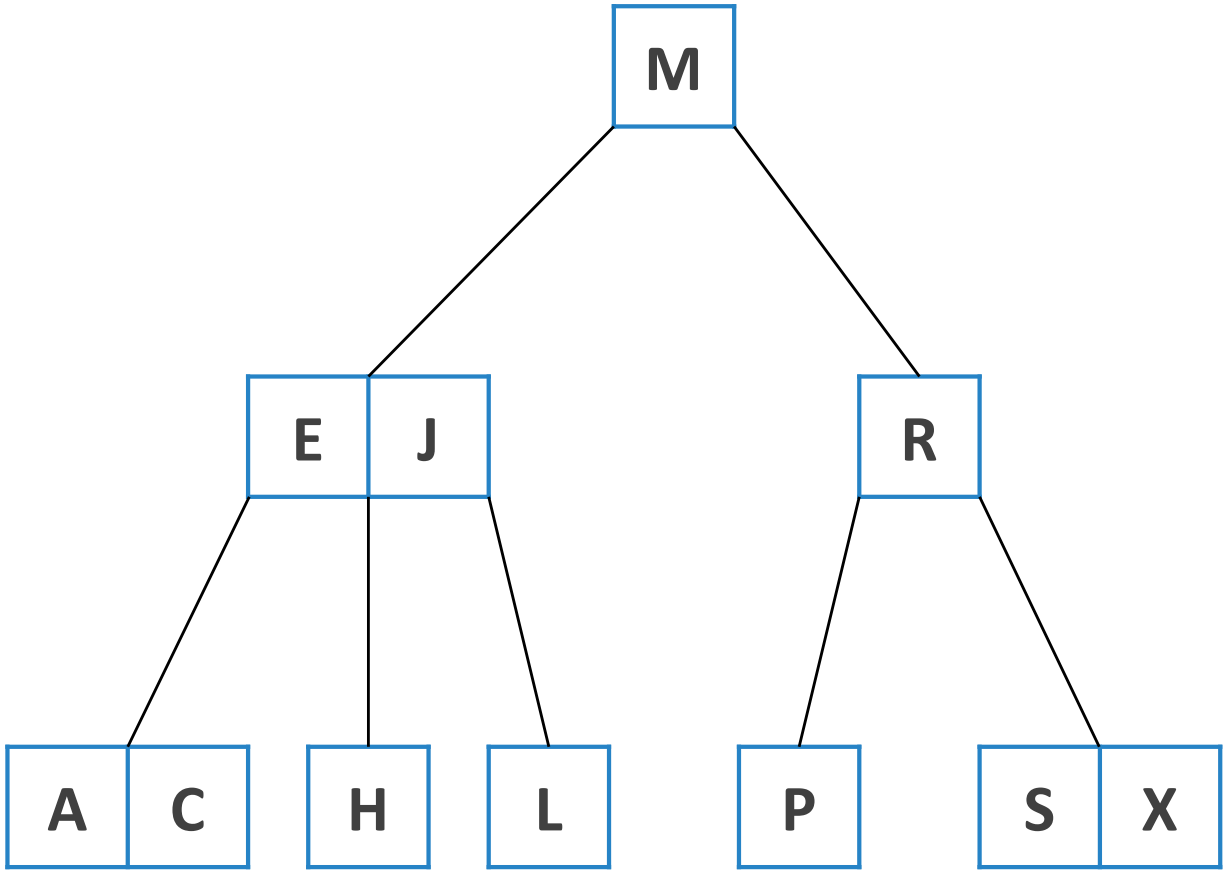


¿Cómo buscamos una clave en un árbol 2-3

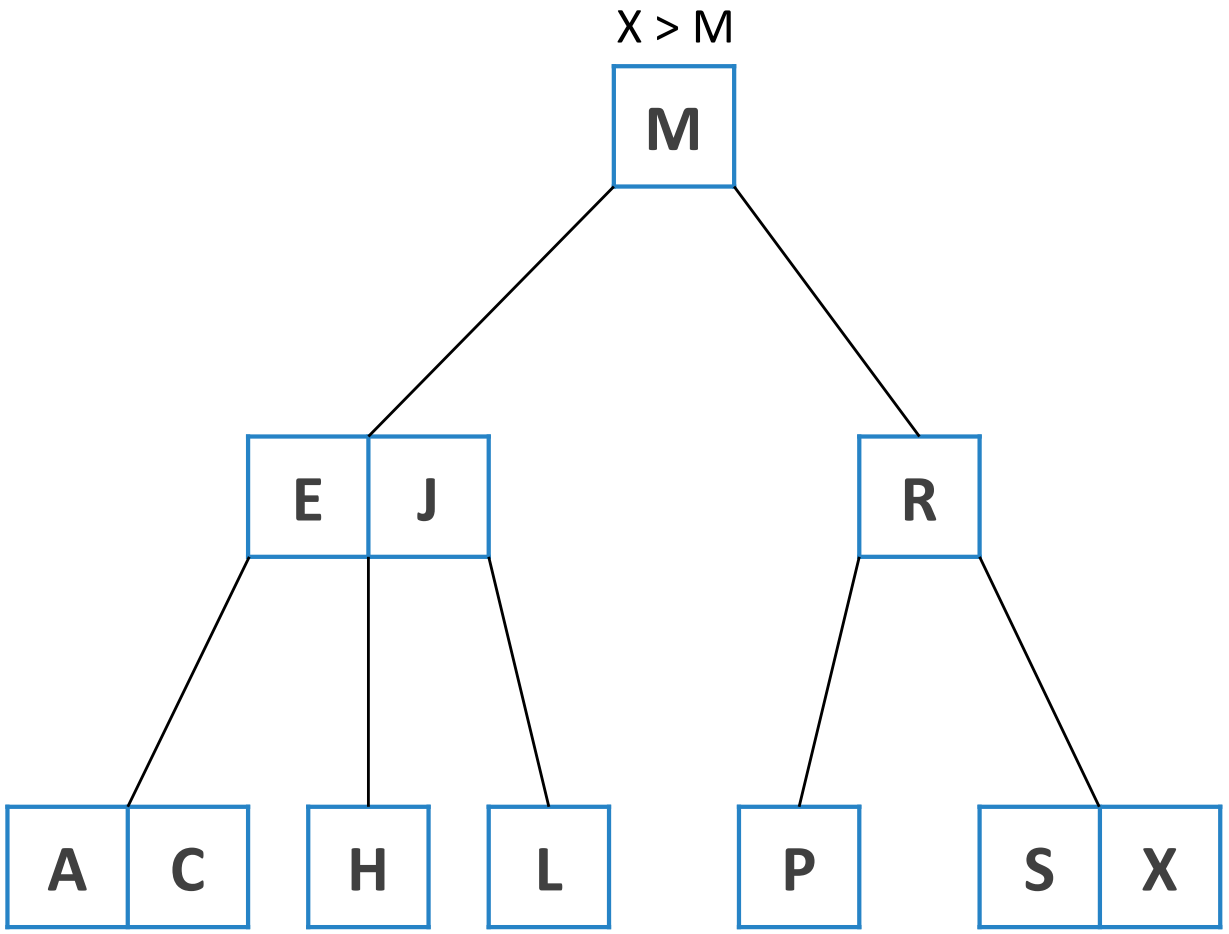


Aprovechamos el hecho de que el árbol está ordenado

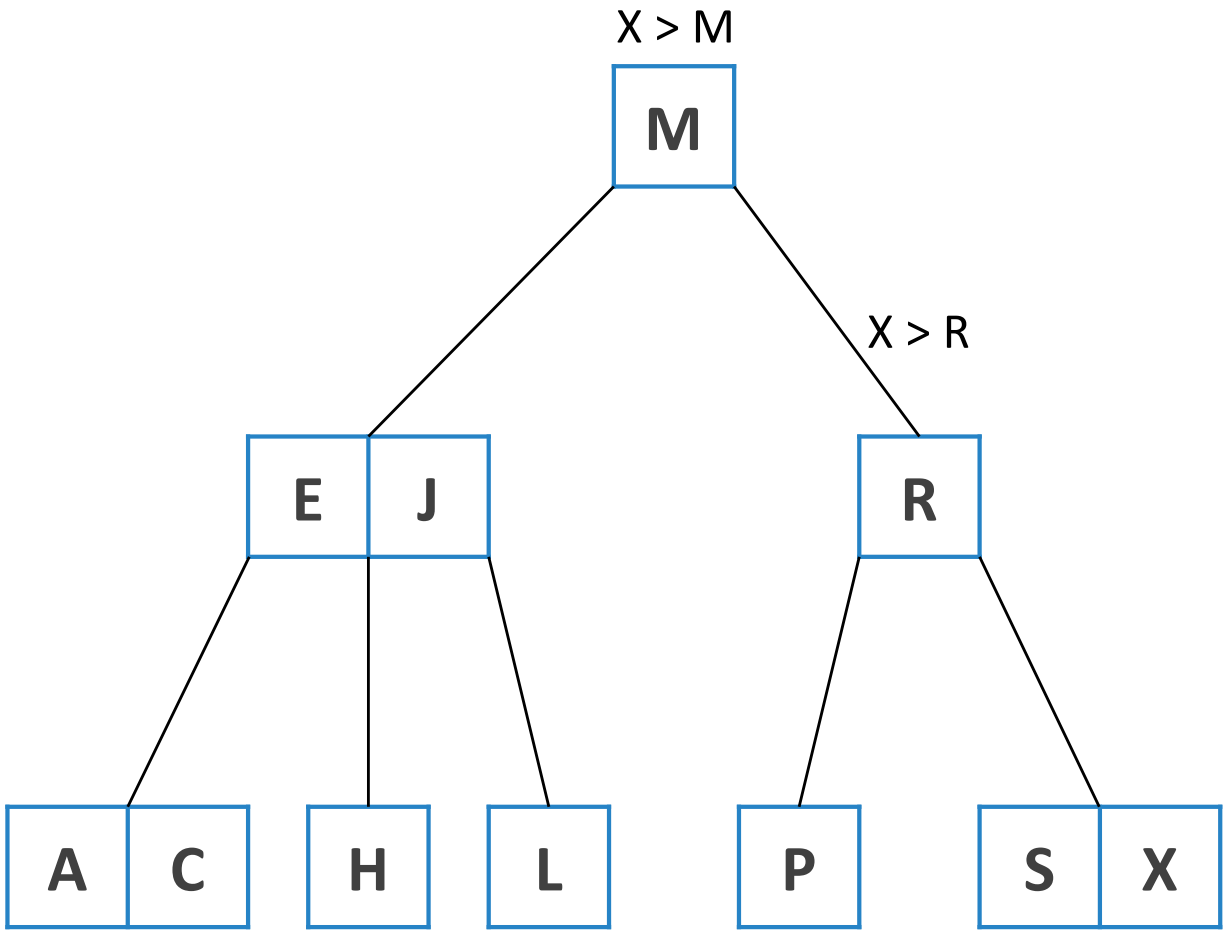
Busquemos la *X*



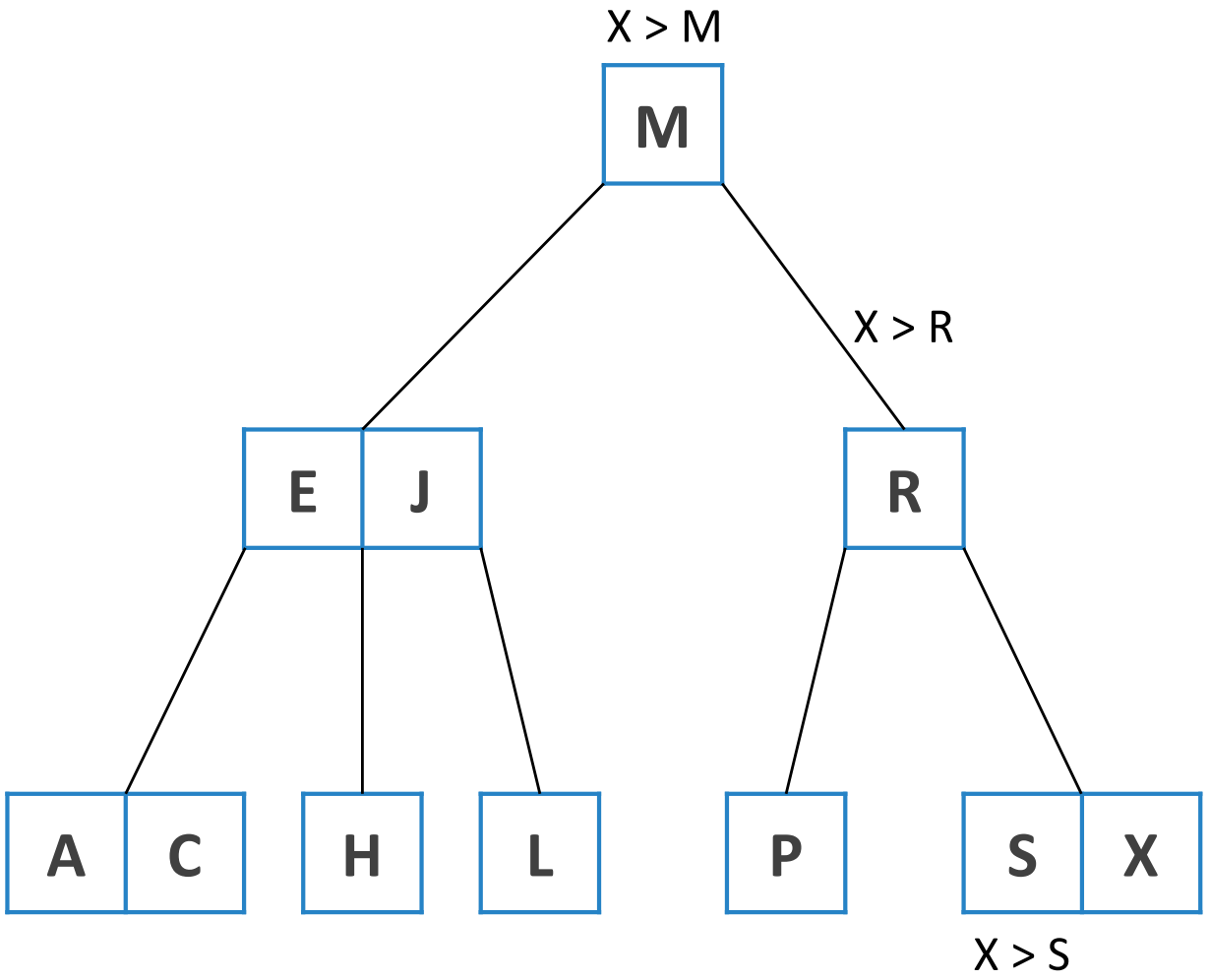
Busquemos la *X*



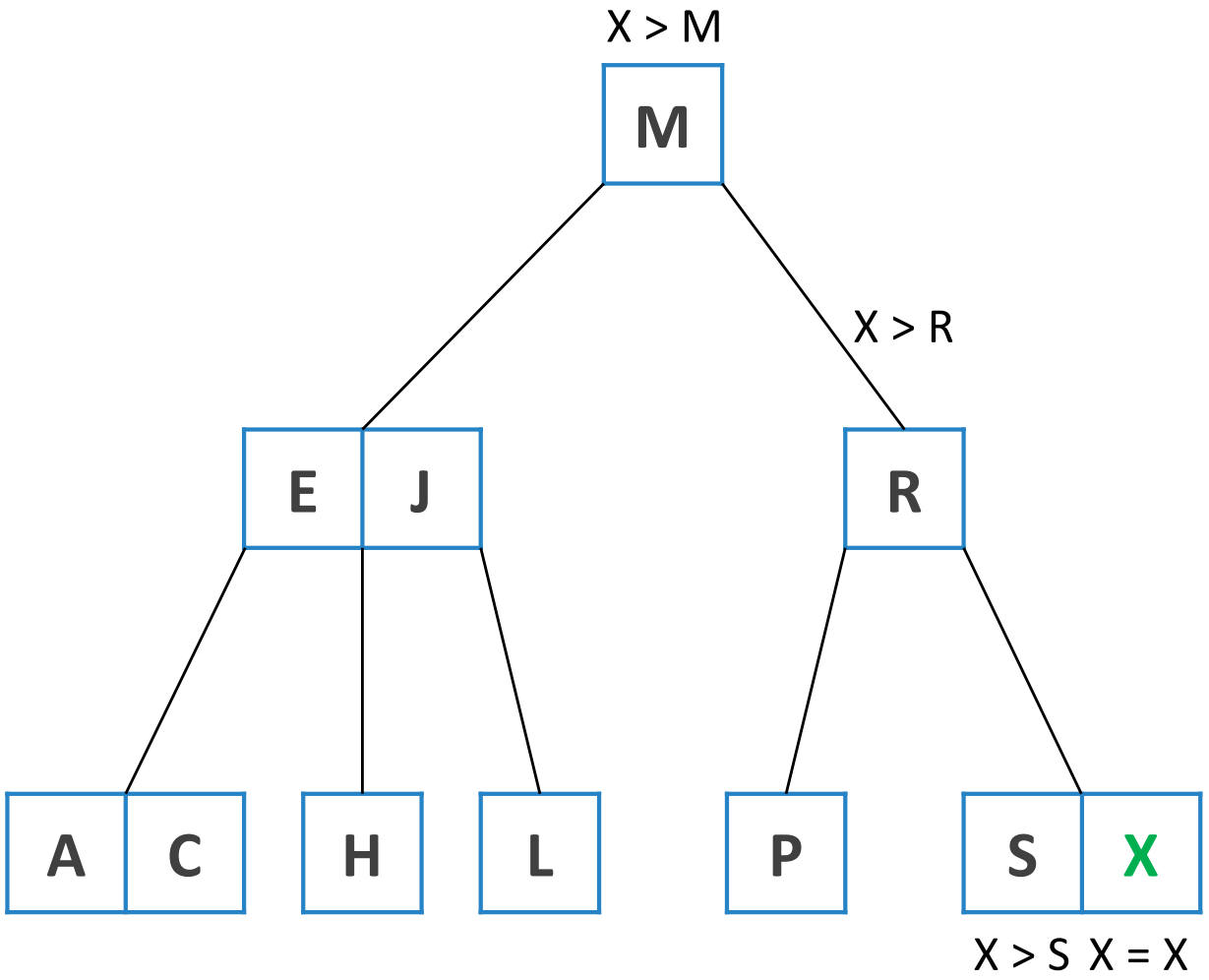
Busquemos la *X*



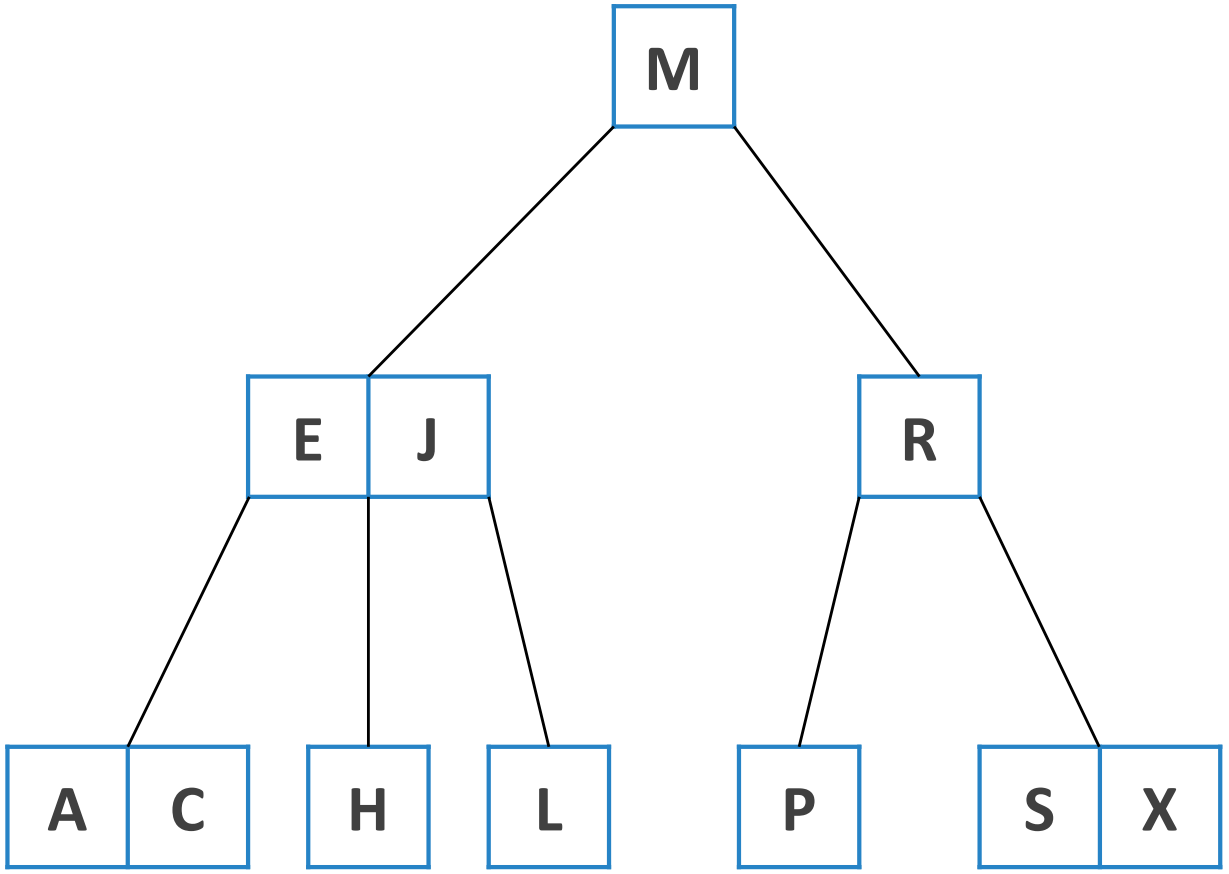
Busquemos la *X*



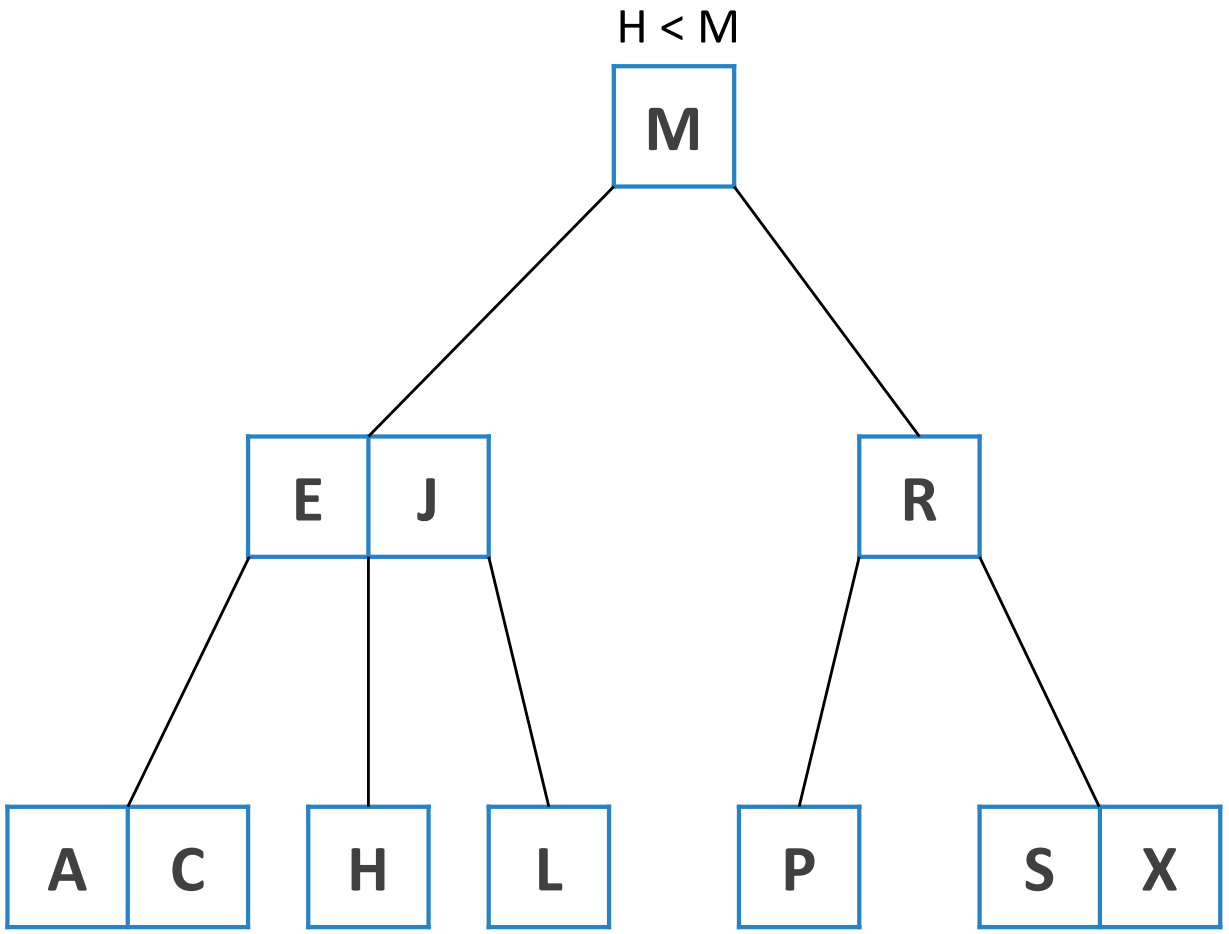
Busquemos la *X*



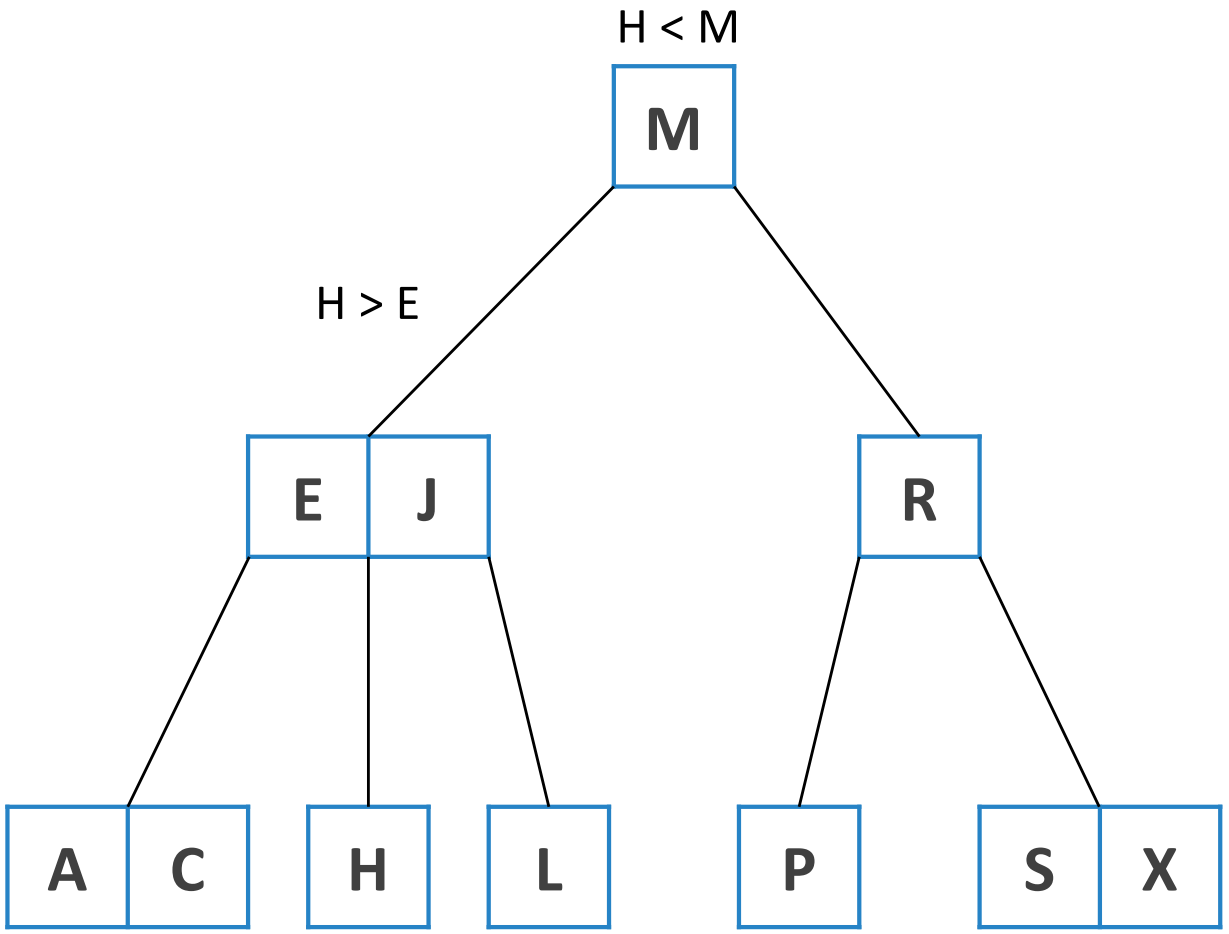
Busquemos la *H*



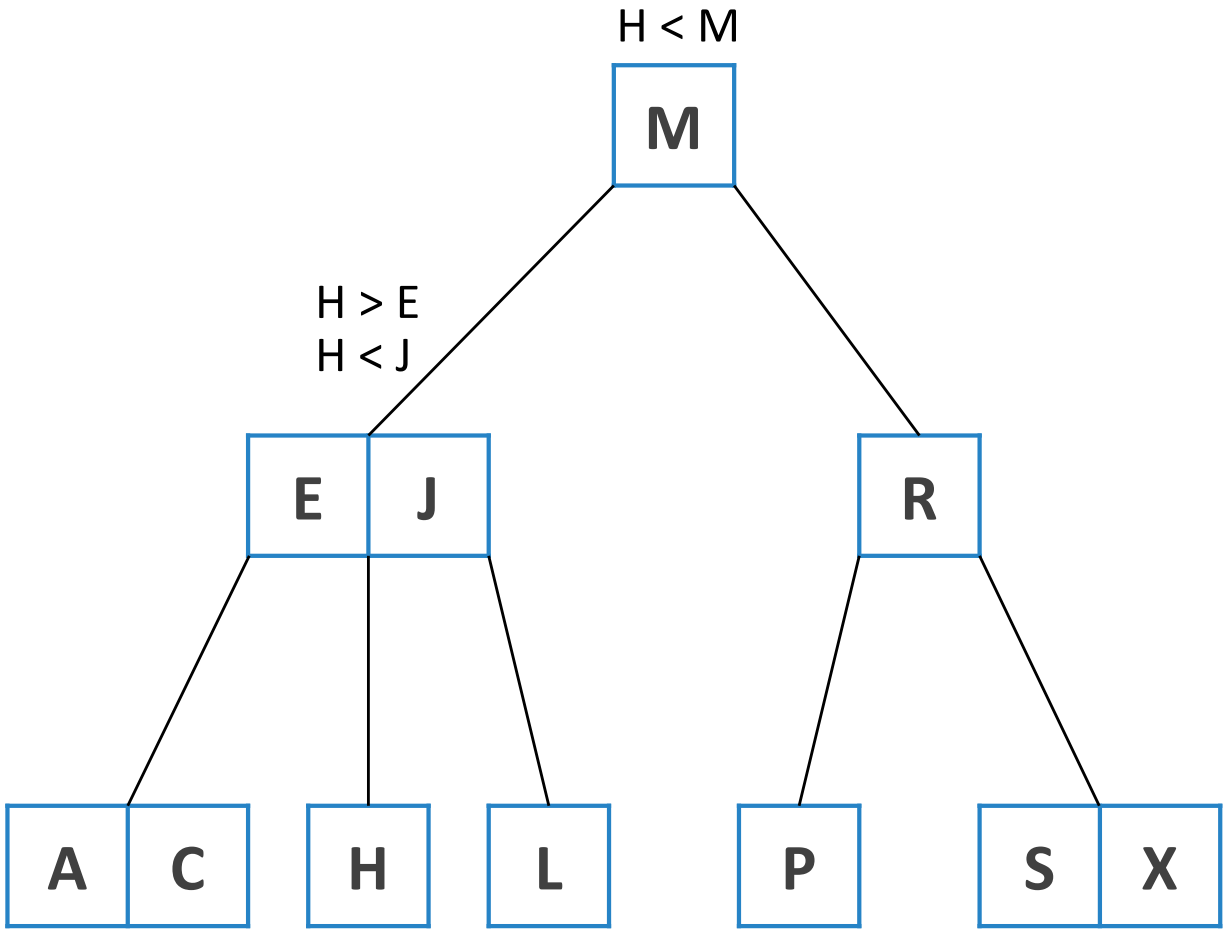
Busquemos la *H*



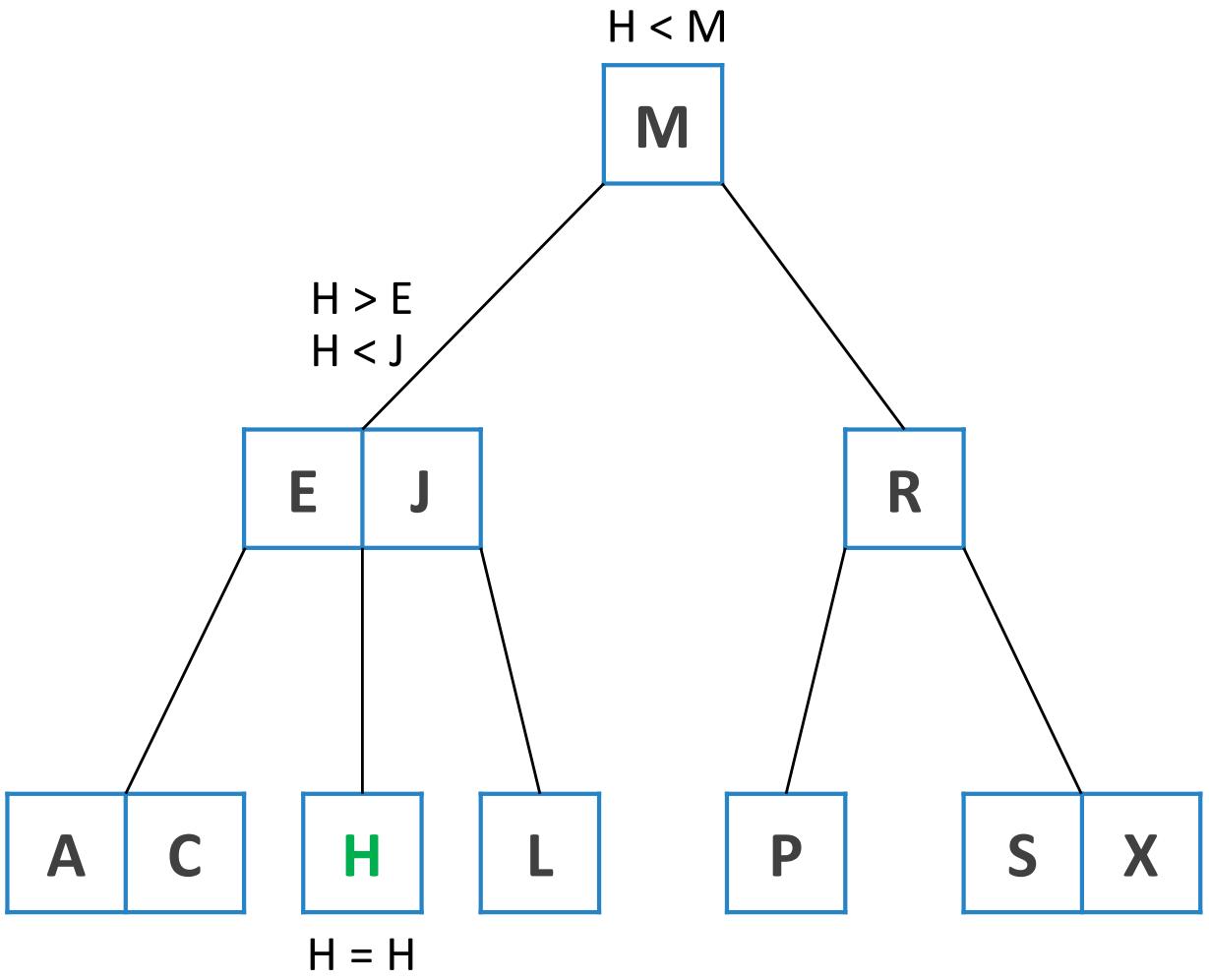
Busquemos la *H*



Busquemos la *H*



Busquemos la *H*



Inserción en un árbol 2-3

Al insertar nuevos datos al árbol, podría cambiar su altura

Queremos mantener todas las hojas a igual profundidad

¿Cómo podemos insertar los datos para que se cumpla esto?

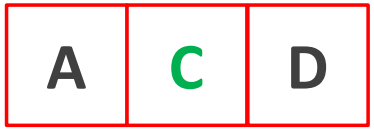
Insertemos los datos *D*, *A*, *C*, *E*, *N*, *F*, *H* en un árbol 2-3 inicialmente vacío

Se inserta el dato normalmente

Insertemos los datos *A*, *C*, *E*, *N*, *F*, *H*

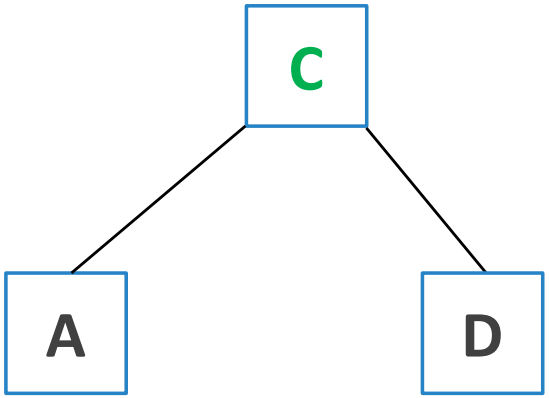
Este dato se inserta ordenadamente (y el nodo cambia de 2 a 3)

Insertemos los datos *C*, *E*, *N*, *F*, *H*



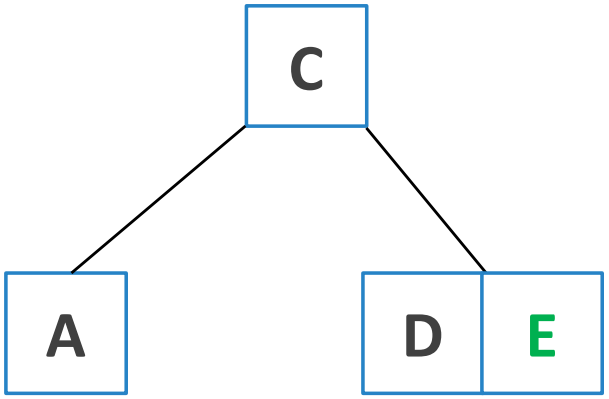
Este nodo ya no es un nodo 2 ni 3, por lo que debemos separarlo

Insertemos los datos *C*, *E*, *N*, *F*, *H*



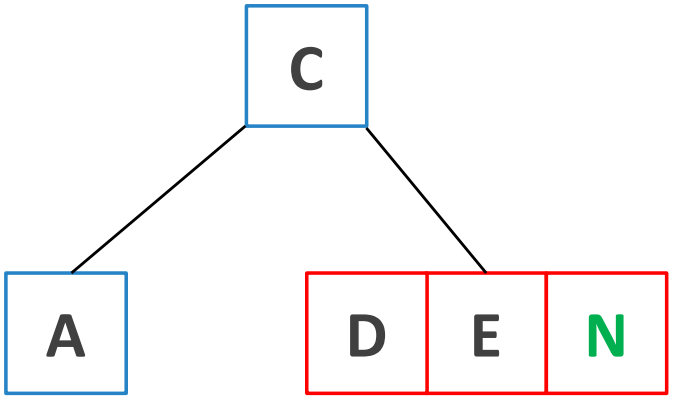
Llamaremos ***split*** a esta operación, donde sube el elemento del medio (y ahora solo tenemos nodos 2)

Insertemos los datos *E*, *N*, *F*, *H*



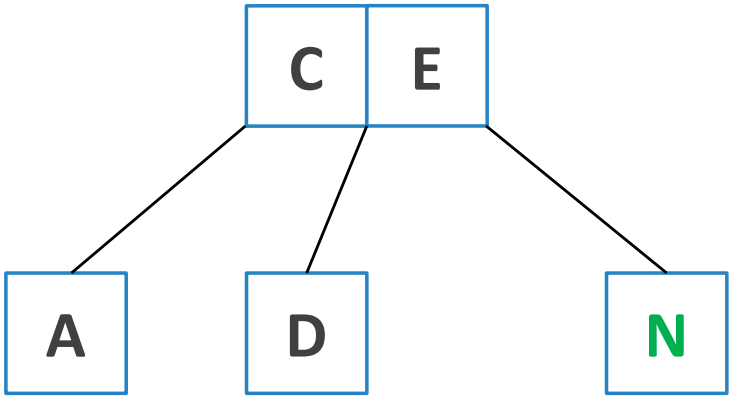
La inserción siempre se hace en las hojas (que pueden tener que cambiar de nodo 2 a nodo 3)

Insertemos los datos *N*, *F*, *H*



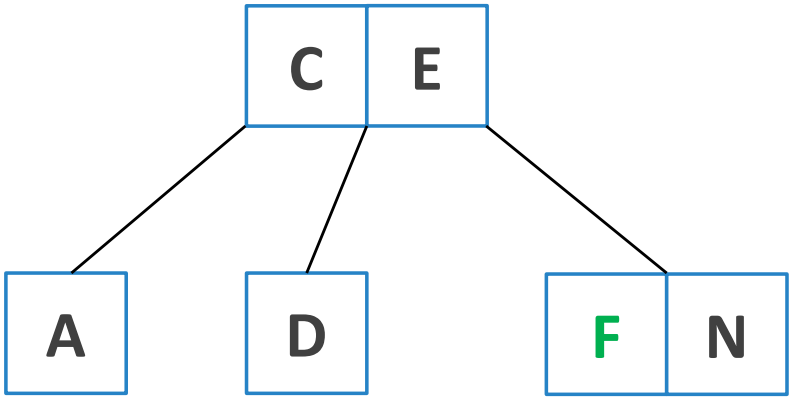
Volvemos a violar la propiedad, volvemos a hacer ***split***

Insertemos los datos *N*, *F*, *H*

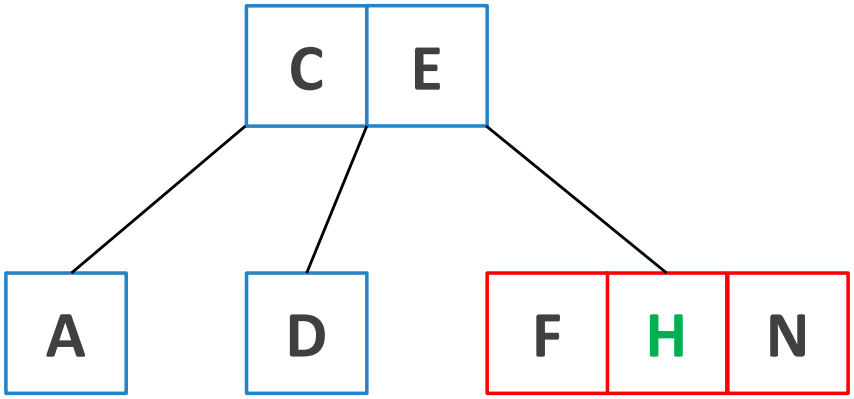


El dato que sube se posiciona ordenadamente en el nodo superior (que cambió de 2 a 3)

Insertemos los datos *F*, *H*

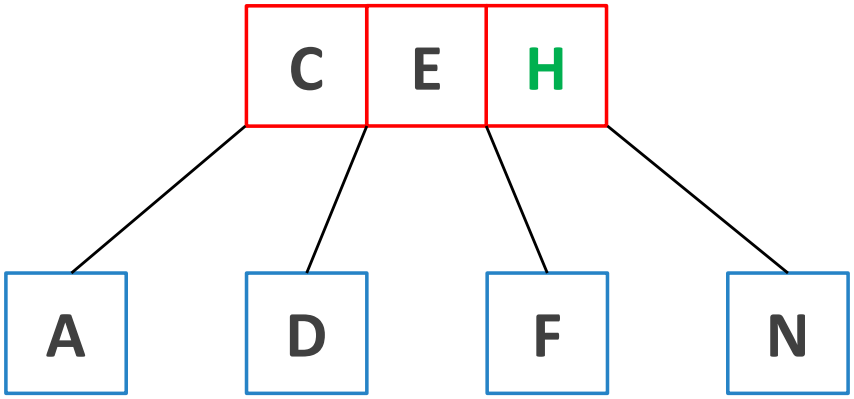


Insertemos el dato *H*



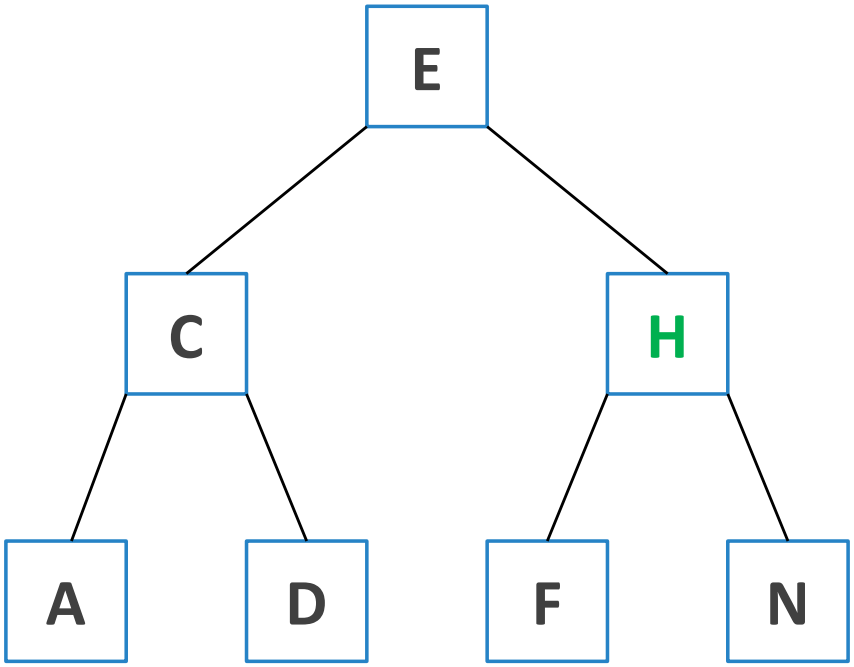
Volvemos a violar la propiedad, volvemos a hacer ***split***

Insertemos el dato *H*



Y esto puede causar una reacción en cadena

Insertemos el dato *H*



Sólo cuando se hace ***split*** de la raíz, la altura del árbol aumenta en 1

Inserción en árboles 2-3: resumen

La inserción siempre se hace —inicialmente— en una hoja

Si un nodo está lleno, se hace subir el dato que habría quedado al medio —la clave mediana— al nodo padre

¡El árbol sólo aumenta de altura cuando la raíz está llena y debe recibir un dato desde un hijo!

Altura de árbol 2-3

¿Es esta noción de balance mejor que la de AVL?

¿Cuál es la altura de un árbol 2-3 de *n* nodos?

¿Cuál es el costo de una búsqueda en el árbol 2-3?

¿Cuál es el costo de una inserción en el árbol 2-3?

Altura de un árbol 2-3

El mejor caso es que todos los nodos sean 3:

$$h = \log_3 n$$

El peor caso es que todos los nodos sean 2:

$$h = \log_2 n$$

Por lo tanto,

$$h \in \Theta(\log n)$$

Costo de las operaciones

A diferencia de los árboles binarios, ahora podríamos tener que comparar más de una vez por nivel

Por lo tanto, el costo de buscar o insertar es $O(2 \cdot h) = O(h)$