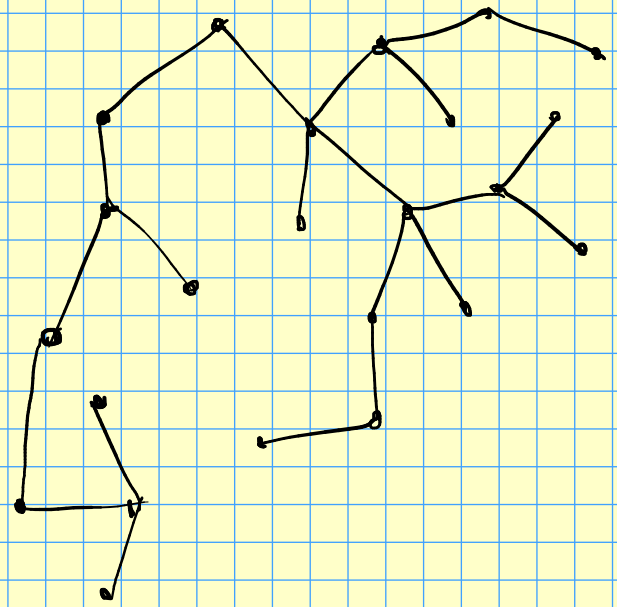
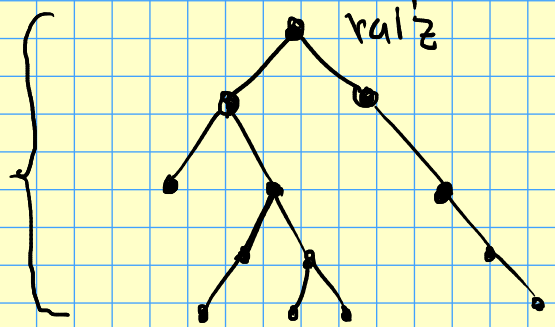
Árbol binario de búsqueda auto-balanceable

Árbol.



Árbol binario.

- Raíz: Un vértice.
- Cada vértice tendrá dos posibles hijos: uno izquierdo y uno derecho
- Ejemplo.



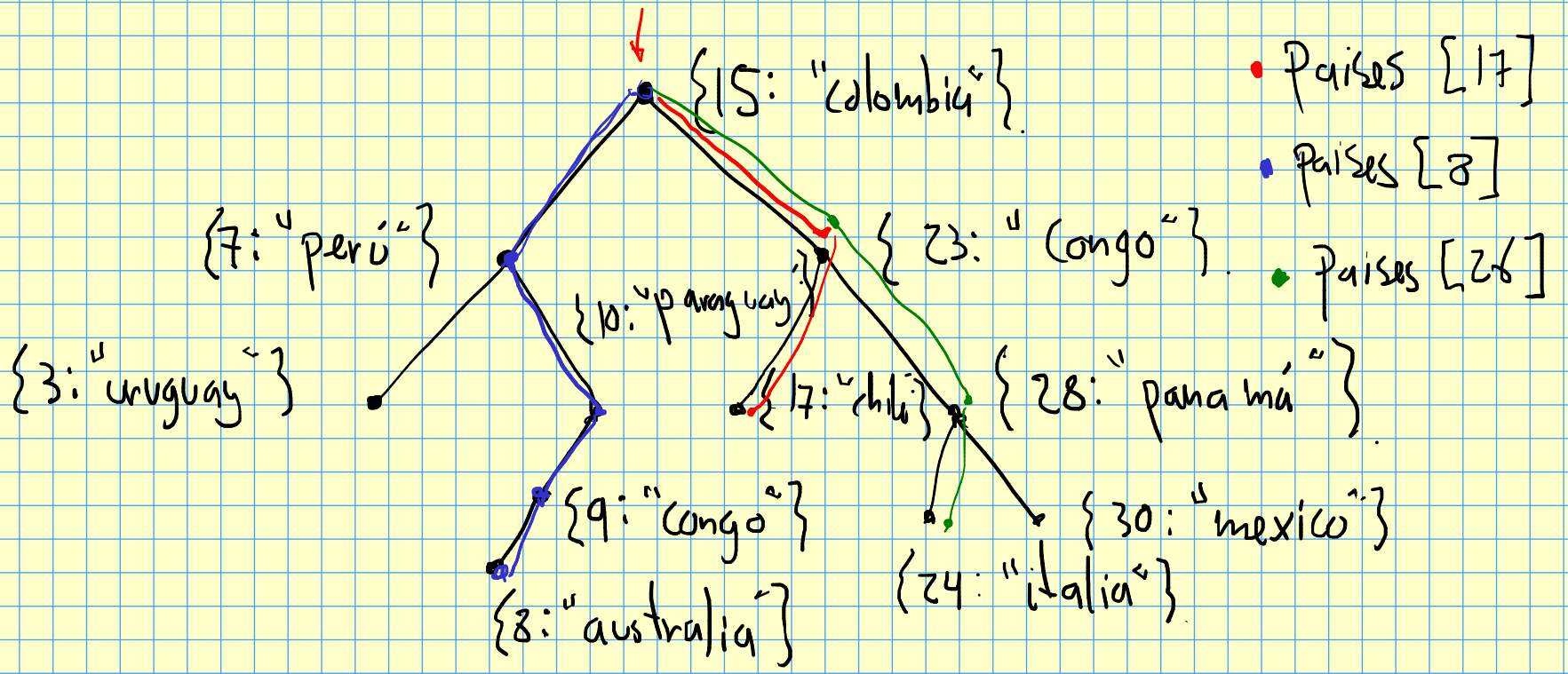
- hijo
- padre
- descendiente.
- ancestro.
- sub-árbol izquierdo
- sub-árbol derecho.
- altura.

Prop. Un árbol de altura h tiene menos de 2^h vértices.

Árbol binario de búsqueda

Árbol binario + las siguientes cosas:

- Cada vértice tiene una llave y un valor.
- Para cada vértice las llaves de sus descendientes de la izquierda son menores y las de los de la derecha son mayores.



En un árbol binario de búsqueda se puede hacer búsqueda binaria. El algoritmo correctamente detecta que la llave no está por un argumento del mínimo ancestro común.

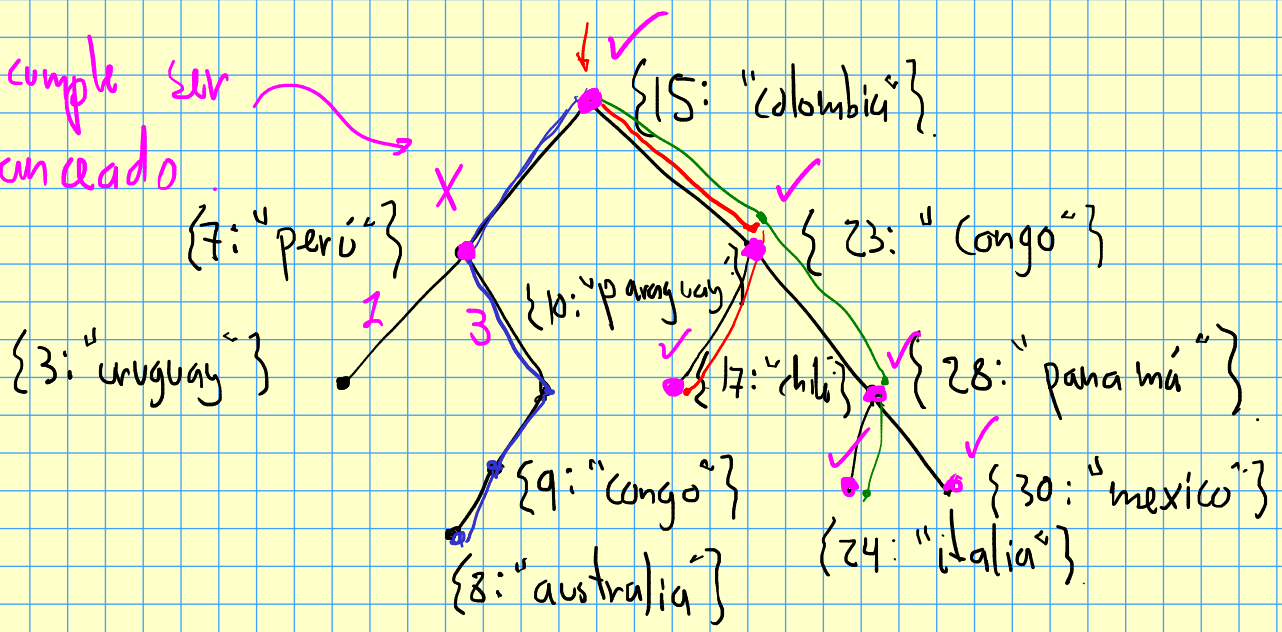
Obs. El algoritmo tarda tiempo $O(h)$ donde h es la altura del árbol.

Árbol binario de búsqueda balanceado

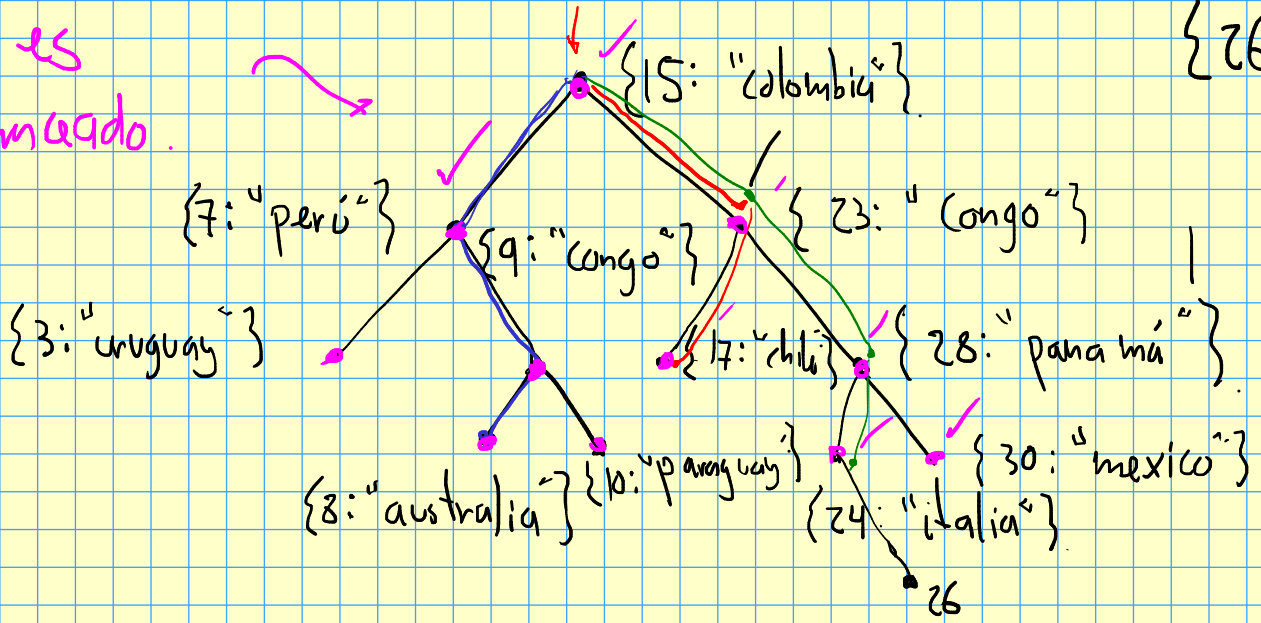
Árbol binario de búsqueda +

- Para cada vértice, la altura del subárbol izquierdo y el derecho debe diferir como mucho en 1 unidad.

No cumple ser balanceado.



Sí es balanceado.



{26: "francia"}

Eliminar [17]

Árbol binario de búsqueda autobalanceable.

Una estructura de datos que guarda información en un árbol binario de búsqueda balanceado y además tiene operaciones que preservan tener un árbol binario de búsqueda balanceado y que pueden hacer lo siguiente:

- Insertar nodos en tiempo $O(\log n)$
- Eliminar nodos en tiempo $O(\log n)$
- Buscar nodos en tiempo $O(\log n)$.
- Modificar nodos en tiempo $O(\log n)$
- Min $O(\log n)$
- Max $O(\log n)$.

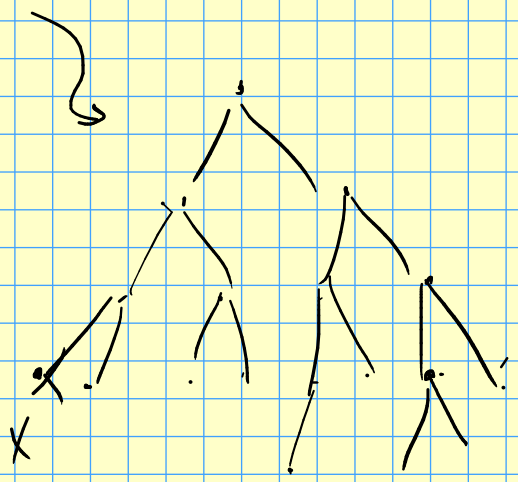
Uso de Árboles binarios de búsqueda autobalanceables para ordenar

2, 10, 11, 21, 14, 3, 8, 12, 27, 91, 4

F = [0]

Crear abba vacío.

Para j en los números dados] $n \log(n)$.
    abba.insert(j).



Para j de 1 a n] $n \log(n)$.
    m = abba.min
    abba.delete(m).
    F.agregar(m).

Al final, F tiene a los números en orden.

Es casi Selection Sort.