BST

```
delete (root, node):
    node < root
        return search(root→left, node)
elif node > root
        return search(root→right, node)
elif node = root
    if node is leaf
        return root = Null
    else:
        if node has right child:
            recursivamente remplazar
            su valor con el nodo
            sucesor y borra el succesor
            de su posición inicial
        elif node has left child
            Lo mismo
    return root
```

Complejidad

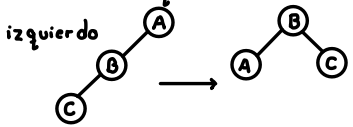
$O(\log n)$ → mejor caso

$O(n)$ → peor caso

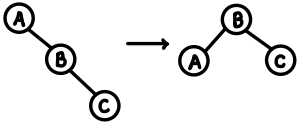
AVL

Complejidad siempre $O(\log n)$

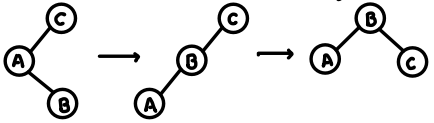
Right Rotation. nodo es insertado en el subárbol izquierdo de un subarbol izquierdo



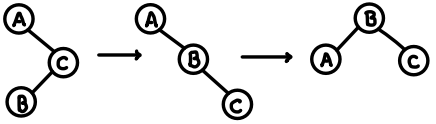
Left Rotation. nodo es insertado en el subarbol derecho de un subárbol derecho



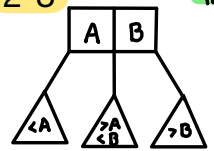
Left-Right Rotation. nodo es insertado en el subárbol derecho de un subárbol izquierdo



Right-Left Rotation. nodo es insertado en un subárbol izquierdo de un subárbol derecho



2-3



insert **Complejidad siempre es** $O(\log n)$

1. Si el árbol está vacio, asigna un nodo como raiz
2. Actualiza el número permitido de claves en el nodo
3. Busca la inserción en el nodo apropiado
4. Si el nodo esta lleno:
   a) inserta los nodos en orden creciente
   b) empuja el nodo mediano hacia arriba.
5. Si el nodo no esta lleno:
   a) inserta el nodo en orden creciente

Propiedades

1. Cada nodo x, se almacenan las llaves en orden
2. Cada nodo tiene un bool si es hoja o no
3. Si n es el orden del árbol, cada nodo interno puede contener como max n-1 llaves
4. Cada nodo, excepto la raíz puede tener como max n hijos y al menos n/2 hijos
5. La raíz tiene al menos 2 hijos y contiene un minimo de 1 llave.

delete (borramos d)

Si n tiene 3 hijos.

· elimine el elemento secundario con valor d, luego corrija el valor izquierdo, el valor medio de los antepasados de n (si es necesario)

Si n tiene 2 hijos:

· Si n es la raíz del árbol, elimine el nodo que contiene d. Reemplace el nodo raiz con el otro hijo.

Tip

las operaciones toman tiempo proporcional a la altura del árbol.

altura minima BST : $\log(n)$ o peor caso $O(n)$

para balancear : $O(\log n)$

Si n tiene un hermano izquierdo o derecho con 3 hijos:

- eliminar el nodo que contiene d.
- usar uno de los hijos del hermano
- corregir el valor izquierdo, el valor medio de n y los hermanos y antepasados de n.

Si los hermanos de n tienen solo 2 hijos:

- eliminar el nodo que tenga d
- hacer que el hijo restante de n sea hijo del hermano de n
- arreglar valores izquierdos y medios
- elimine n como hijo de su padre.

Red-Black

Propiedades

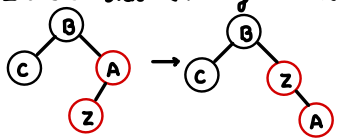
- nodo es rojo o negro
- nodo raiz o hoja (Nil) son negros
- Si un nodo es rojo, entonces sus hijos son negros
- todas las rutas de un nodo a sus descendientes Nil contienen el mismo número de nodos negro

insert

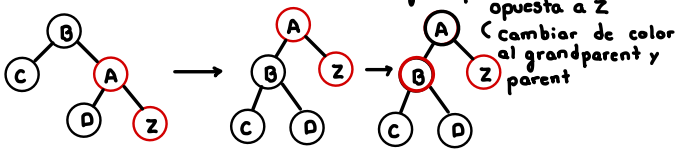
1. Insertar z como un nodo rojo
2. Vuelva a colocar y rotar nodos para corregir infracciones

Solucionar infracciones

1. Z = raiz → cambiar a negro
2. Z.uncle = red → cambiar color de grandparent, uncle and parent
3. Z uncle = black (triangle) → rotar Z.parent en dirección opuesta a Z



4. Z uncle = black (line) → rotar Z.grandparent en dirección opuesta a Z (cambiar de color al grandparent y parent



Tablas de Hash

Busqueda → O(1) y peor caso O(n)

función de hash = h mod m ← tamaño de la tabla

"111" string = $\#(S_0) + \#(S_1)\cdot R + \ldots + \#(S_n)\cdot R^{n-1}$ (R primo)

Backtracking

Backtracking(x):

```
if x is not solution
    return false
if x is a new solution:
    add to list of solutions
backtrack(expand x)
```

Fuerza bruta. probar todas las combinaciones y ver si sirven

X → conjunto de variables

D → dominio de variables

R → restricciones

En el peor caso, tiene complejidad $O(k^n)$

Heurísticas: conocimiento experto, asignar con menor espacio de búsqueda. Estrategias para elegir la proxima variable.

Poda: restricciones adicionales que se deducen de las otras. Descartar caminos.

Propagación. asignar un valor acota los otros dominios. Luego, si algo no resulta se puede contar desde que se asigno el valor que propago.

issolvable(X, D, R):

```
if X = ∅, return true
x ← la mejor variable de X
for v ∈ Dx, de mejor a peor
    if x = v no es válida
        continue
    x ← v propagar
    if issolvable(X-{x}, D, R)
        return true
    x ← ∅ propagar
return false
```