Árboles AVL

Anexo

Seudocódigo de las operaciones de inserción y borrado

Árbol AVL: Definición

Un árbol AVL (Adelson–Velskii–Landis) es un árbol binario de búsqueda que cumple con la condición de estar balanceado

- La propiedad de balanceo que cumple dice:

Para cada nodo del árbol, la diferencia de altura entre el subárbol izquierdo y el subárbol derecho es a lo sumo 1

Características

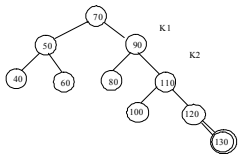
- *La propiedad de balanceo es fácil de mantener y garantiza que la altura del árbol es de O(log n)*
- *En cada nodo del árbol se guarda información de la altura*
- *La altura del árbol vacío es -1*
- *Se debe mantener el balanceo al realizar las operaciones sobre el árbol (inserción y borrado)*
- *La inserción se realiza igual que en un árbol binario de búsqueda, puede destruirse la propiedad de balanceo, entonces se debe* ***Rebalancear el árbol***

Problemas: Desbalanceo

- *Al insertar un elemento se actualiza la información de la altura de los nodos que están en el camino desde el nodo insertado a la raíz*
- *El desbalanceo sólo se produce en ese camino, ya que sólo esos nodos tienen sus subárboles modificados*

Rebalanceo del árbol

***Ejemplo** de rotación simple : Rotación Simple con Derecho al Insertar clave 130*

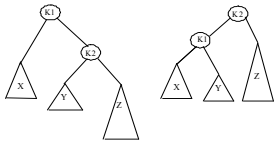



```
K1 : raiz del árbol desbalanceado

K2 := k1.der;
K1.der := k2.izq;
K2.izq := k1;

K1.altura:=max (altura (k1.izq),altura (k1.der ))+1;
K2.altura :=max (altura ( k2.der ) , k1.altura ) + 1;

K1 := k2 ;    {  nueva raiz  }
```



K1 < K2

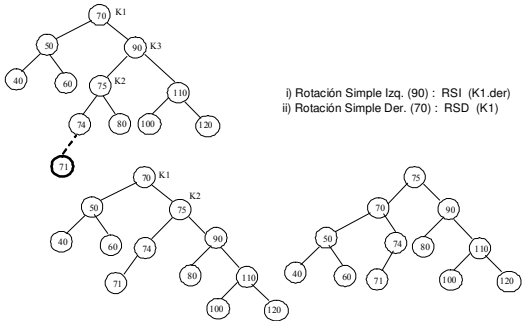
todos los datos en X < K1 (< K2)

todos los datos en Z > K2 (> K1)

K1 < todos los datos en Y < K2

Rebalanceo del árbol

***Ejemplo** de rotación doble : Rotación doble con Derecho al Insertar la clave 71*

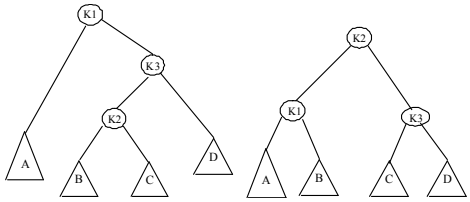


i) Rotación Simple Izq. (90) : RSI (K1.der)
ii) Rotación Simple Der. (70) : RSD (K1)

Rebalanceo del árbol

Ejemplo de rotación doble : Rotación doble con Derecho al Insertar la clave 71

Rotación Doble Derecho (R-S Izquierdo (hijo derecho(K1)) , R-S Derecho (K1))



Rebalanceo del árbol

- *Para restaurar el balanceo del árbol:*
  - *se recorre el camino de búsqueda en orden inverso*
  - *se controla el equilibrio de cada nodo*
  - *si está desbalanceado se realiza una modificación simple: rotación*
  - *después de rebalancear el nodo, la inserción termina*
  - *este proceso puede llegar a la raíz*

Árbol AVL - altura

```
private int altura (avl_nodo p ) {
/* retorna el valor de la altura de p */
        if  p == nil {
                        return -1;
                        } else {
                        return  p.getAltura();
        }
}
```


Árbol AVL – Balancear (c/alturas)

```
/*  Procedimiento para Balancear consid. "alturas"  de los subárb.
   */
Modulo   Balancear ( T : AVL )
if (  altura ( T .izq ) - altura ( T .der )  =  2 )        then
   if  (altura ( T.izq.izq ) >= altura ( T.izq .der  ) then
                        rotación_s_izq ( T ) {subárb. extremo izquierdo }
                         else
                           rotación_d_izq( T ) {subárb. medio }
                         else
if (  altura ( T .der ) - altura ( T .izq ) =  2)    then
   if  (altura ( T.der.der ) >= altura ( T.der.izq  ) then
                        rotación_s_der ( T ) {subárb. extremo derecho }
                        else
                        rotación_d_der ( T ) {subárb. medio }

 End  Balancear
```


Árbol AVL - Insert

```
PROCEDURE  INSERT ( x : element_type; var T: AVL )
 begin
   if  T = nil      then
                /*  Insertar  una hoja  como  en  el  ABB  más el campo de "altura =0"   */
                 else
                  begin
                  if      x <  T .dato    then
                                  INSERT ( x, T .izq ) ;
                                 else
                                 if      x >  T .dato  then
                                                 INSERT ( x, T .der ) ;
                                         else: { x  está en el árbol y no hay nada que hacer }
                end    { si   T <> nil }
        /*  Balancear si es necesario   */
        Balancear ( T );
        /*  Actualizar altura  del  nodo  raíz del sub árbol  T  - siempre   */
        T.altura := max (altura (T.der) , altura (T.izq) ) + 1;
end { INSERT  }
```


Árbol AVL – Delete

```
PROCEDURE  DELETE ( x :element_type; T: AVL )
    begin
    if  T <> nil  then
          if  T .dato = x    then
                    /* Borrar según los 3 casos posibles */
          if ( T.izq  = nil    .and. T.der  = nil )  then
                                  T = nil;   /*  nodo es una hoja  */
                          else  if ( T.izq  = nil )  then  /* nodo  c/ solo hijo derecho */
                                  T =  T.der ;
                           else  if ( T.der  = nil )  then  /* nodo  c/ solo hijo izquierdo */
                                  T =  T.izq ;
                            else
                                  T.dato =  delete_min (T.der) /* Suc. in orden */
                      else
                  /*    continúa ….  */
```

Árbol AVL – Delete (cont.)

```
/*  …..  continuación  */

            if x < T .dato  then
                        DELETE ( x, T.izq ) ;
                    else
                        DELETE ( x, T.der ) ;
          else :  { x  no está en el árbol y no hay nada que hacer }
          endif  { si  T <> nil }
/* Balancear si es necesario  */
        Balancear ( T );
/* Actualizar altura del nodo raíz del sub árbol T - siempre  */
        T.altura := max (altura (T.der), altura (T.izq) ) + 1;

end { DELETE }
```


Árbol AVL – Delete (cont.)

```
DELETE_MIN  ( T: AVL , min : elementType )  /* retorna el mínimo del árbol T y
 lo borra */

    if T <>  nil  then
            if  T .izq <> nil  then
                        DELETE_MIN ( T.izq , min) ;
                        Balancear ( T );
                        T.altura := max (altura (T.der), altura (T.izq) ) + 1;
                        else
                        min  =  T.dato
                        Balancear ( T );
                        T.altura := max (altura (T.der), altura (T.izq) ) + 1;

end { DELETE_MIN }
```