***UNIVERSIDAD ABIERTA PARA ADULTOS***

***(UAPA)***

**Escuela de Ingeniería y Tecnología**

***Estructura de datos & Algoritmo***

***NOMBRE***

*GARIBALDIS*

***APELLIDO***

*PEREZ ORTIZ*

***MATRICULA***

*100029560*

***FACILITADOR***

*Maikel Aparicio*

**Ciudad: Santo Domingo**

**País: Rep. Dominicana**

**Fecha: 6 septiembre de 2021**

**Inserción**

La inserción de elementos en un árbol AVL de forma análoga a cómo lo haríamos para árboles binarios de búsqueda salvo que en cada recursión del algoritmo verificaremos y corregiremos el equilibrio del árbol. También es importante ir actualizando las alturas de cada nodo en cada recursión dado que las rotaciones, inserciones y eliminaciones pueden modificarlas.

Dado que ya vimos funciones tanto para balancear un árbol y para actualizar la altura de un nodo (ambas de tiempo de ejecución constante), estamos listos para implementar el algoritmo de inserción. Esperamos que sea intuitivo.

void insertar (AVLTree \*\* t, int x);

/\* inserta x en el árbol en un tiempo O(log(n)) peor caso. \*/

void

insertar (AVLTree \*\* t, int x)

{

if (es\_vacio (\*t))

\*t = hacer (x, vacio (), vacio ()); /\* altura actualizada

automáticamente \*/

else

{

if (x < raiz (\*t))

insertar (&(\*t)->izq, x);

else

insertar (&(\*t)->der, x);

balancear (t);

actualizar\_altura (\*t);

}

}

**Eliminación**

La estrategia para diseñar el algoritmo de eliminación sobre árboles AVL es la misma que para la inserción: Se utiliza el mismo algoritmo que sobre árboles binarios de búsqueda, pero en cada recursión se detectan y corrigen errores por medio de balancear () y se actualiza la altura del nodo actual.

Recordamos un poco la idea del algoritmo de eliminación sobre árboles binarios de búsqueda. Primero se recorre el árbol para detectar el nodo a eliminar. Una vez hecho esto hay tres casos a diferenciar por su complejidad:

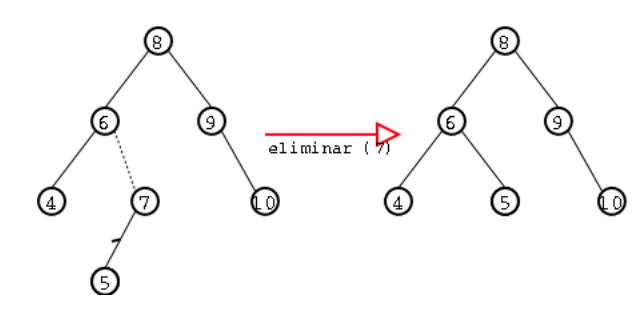
• Si dicho nodo es una hoja procedemos a eliminarlos de inmediato, sin más.

Si dicho nodo tiene un sólo hijo, el nodo puede eliminarse después de ajustar un

apuntador del padre para saltar el nodo.

**Esto se muestra en Figure 13.**

**Figure 13. Eliminación de un nodo (7) con un sólo hijo.**



• Si dicho nodo tiene dos hijos el caso es un poco más complicado. Lo que se estila hacer (y que de hecho se hace en el algoritmo gracias a la función auxiliar eliminar\_min()) reemplazar el nodo actual por el menor nodo de su subárbol derecho (y luego eliminar éste).

void eliminar (AVLTree \*\* t, int x);

/\* elimina x del árbol en un tiempo O(log(n)) peor caso.

Precondición: existe un nodo con valor x en el árbol

t. \*/int eliminar\_min (AVLTree \*\* t);

/\* Función auxiliar a eliminar(). Elimina el menor nodo del árbol

\*t devolviendo su contenido (el cual no se libera de

memoria). Se actualizan las alturas de los nodos.

Precondición: !es\_vacio(\*t) \*/

void

eliminar (AVLTree \*\* t, int x)

{

AVLTree \*aux;

if (x < raiz (\*t))

eliminar (&(\*t)->izq, x);

else if (x > raiz (\*t))

eliminar (&(\*t)->der, x);

else /\* coincidencia! \*/

{

if (es\_vacio (izquierdo (\*t)) && es\_vacio (derecho (\*t)))

{/\* es una hoja \*/

free (\*t);

(\*t) = vacio();

}

else if (es\_vacio (izquierdo (\*t)))

{/\* subárbol izquierdo vacio \*/

aux = (\*t);

(\*t) = (\*t)->der;

free (aux);

}

else if (es\_vacio (derecho (\*t)))

{/\* subárbol derecho vacio \*/

aux = (\*t);

(\*t) = (\*t)->izq;

free (aux);

}

else /\* caso más complicado \*/

{

(\*t)->dato = eliminar\_min (&(\*t)->der);

}

}

balancear (t);

actualizar\_altura (\*t);

}

int

eliminar\_min (AVLTree \*\* t)

{

if (es\_vacio (\*t))

{

fprintf (stderr,

"No se respeta precondición de eliminar\_min()\n");

exit(0);

}

else

{

if (!es\_vacio (izquierdo (\*t)))

{

int x = eliminar\_min (&(\*t)->izq);

balancear (t);

actualizar\_altura (\*t);

return x;

}

else

{

AVLTree \*aux = (\*t);

int x = raiz (aux);

\*t = derecho (\*t);

free (aux);

balancear (t);

actualizar\_altura (\*t);

return x;

}

}

}