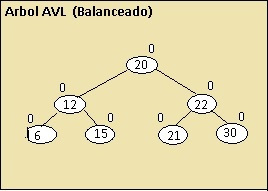
**1- *TAD ARBOL AVL:***

El árbol AVL va a ser utilizado para ordenar los Jugadores por algunos de los criterios escogidos.

**Nombre:** ÁRBOL AVL



**Invariantes:**

-    El peso en cada punto del Árbol se calcula como, el número de nodos del camino derecho más largo menos el número de nodos del camino izquierdo más largo, y el resultado de esta resta debe estar entre -1 y 1.

**Operaciones:**

-    Crear Árbol AVL: Árbol (Creadora)

**Descripción:**

Construye un árbol vacío, se solicita memoria para almacenar elementos en el Árbol.

**Postcondicion:** Árbol A ≠ Null

-    Agregar (Elemento E): Árbol. (Modificadora)

**Precondicion:** Árbol A ≠ Null; Elemento E a agregar   ≠ Null.

**Descripción:** Agrega a la Árbol A un nuevo elemento.

-    Eliminar (Elemento E): Árbol A (Modificadora)

**Precondicion:** Árbol A ≠ Null, E (elemento a Eliminar) ≠ Null.

**Descripción:** Elimina el elemento E, pasa por parámetro del Árbol AVL.

-    Buscar (Elemento E): Árbol A (Analizadora)

**Precondiciones:** Árbol A ≠ Null, E (elemento a Buscar) ≠ Null.

**Descripción**: Busca un elemento E en el árbol A y lo retorna.

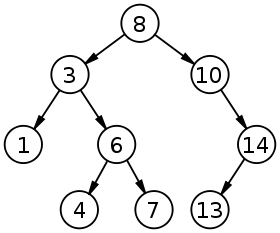
**Postcondicion:**

Si el elemento E existe devuelve el Objeto que representa a E, En caso contrario retorna Null.

**2- TAD Árbol binario de búsqueda:**

El árbol Binario de Búsqueda va a ser utilizado para ordenar los Jugadores por algunos de los criterios escogidos.

**Nombre:** TAD ÁBB



**Invariantes:**

-Dado un nodo x en el árbol, en una posición p, siempre los hijos derechos son mayores que su padre, y los hijos izquierdos menores que el.

**Operaciones:**

* Crear Árbol Binario De Búsqueda: Árbol (Creadora)

**Descripción:**

Construye un árbol vacío, se solicita memoria para almacenar elementos en el Árbol.

**Postcondicion:** Árbol A ≠ Null.

-    Agregar (Elemento E): Árbol (Modificadora)

**Precondicion:** Árbol A ≠ Null; Elemento E a agregar ≠ Null.

**Descripción**: Agrega a la Árbol A un nuevo elemento.

-    Eliminar (Elemento E): Árbol A (Modificadora)

**Precondicion:** Árbol A ≠ Null, E (elemento a Eliminar) ≠ Null.

**Descripción:** Elimina el elemento E, pasa por parámetro del Árbol binario de busqueda.

-    Buscar (Elemento E): Árbol A (Analizadora)

**Precondiciones:** Árbol A ≠ Null, E (elemento a Buscar) ≠ Null.

Descripción: Busca un elemento E en el árbol A y lo retorna.

**Postcondicion:**

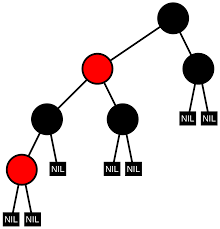
Si el elemento E existe devuelve el Objeto que representa a E, En caso contrario retorna Null.

**3- *TAD ARBOL ROJO Y NEGRO:***

El árbol rojinegro va a ser utilizado para ordenar los mercados de divisas por algunos de los criterios escogidos.

**Nombre:** Árbol Roji-Negro.

Un árbol rojinegro se puede definir gráficamente como:



**Invariantes:**

-    La raíz siempre es negra.

-    Un hijo Rojo no puede tener un padre Rojo

-    Un nodo puede ser de cualquier color (Rojo o Negro)

-    Todas las ramas del árbol tienen el mismo número de nodos negros.

**Operaciones:**

-       Crear Árbol Rojo y Negro: Árbol (Creadora)

**Descripción:** Construye un árbol vacío, se solicita memoria para almacenar elementos en el Árbol.

**Postcondicion**: Árbol A ≠ Null.

-       Agregar (Elemento E): Árbol (Modificadora)

**Precondición:** Árbol A ≠ Null; Elemento E a agregar   ≠ Null.

**Descripción:** Agrega al Árbol A un nuevo elemento

**Postcondicion:**

-       Eliminar (Elemento E): Árbol A (Modificadora)

**Precondicion:** Árbol A ≠ Null, E (elemento a Eliminar) ≠ Null.

**Descripción:** Elimina el elemento E, pasa por parámetro del árbol Rojo y Negro.

- Buscar (Elemento E): Árbol A.

**Precondición:** Árbol A ≠ Null, E (elemento a Buscar) ≠ Null.

**Descripción**: Busca un elemento E en el árbol A y lo retorna.

**Postcondicion:**

Si el elemento E existe devuelve el Objeto que representa a E, En caso contrario retorna Null.