

## **Tarea Integradora 2**

Santiago Trochez Velasco - A00369326  
Gianni Stiven Benavides - A00362358  
Luis Miguel Ossa Arias - A00369982  
Johnatan Garzón Montesdeoca

Algoritmos y Estructura de Datos, Ingeniería de Sistemas  
Santiago de Cali, Colombia  
2021

### Etapa 1. Contexto Problemático.

La federación internacional de baloncesto está en busca de una aplicación que permita tener a la mano los datos relevantes de cada uno de los profesionales de este deporte, y de esta forma, poder realizar un análisis detallado de estos datos, identificar patrones y datos que tienen mayor relevancia en la actualidad. Por lo anterior, se requiere una aplicación que permita ingresar, buscar, modificar y encontrar los datos estadísticos relevantes de cada uno de los basquetbolistas, y así, resolver el problema de manejar una cantidad masiva de datos de cada uno de los jugadores registrados para realizar búsquedas eficientes que permitan analizar dichos datos.

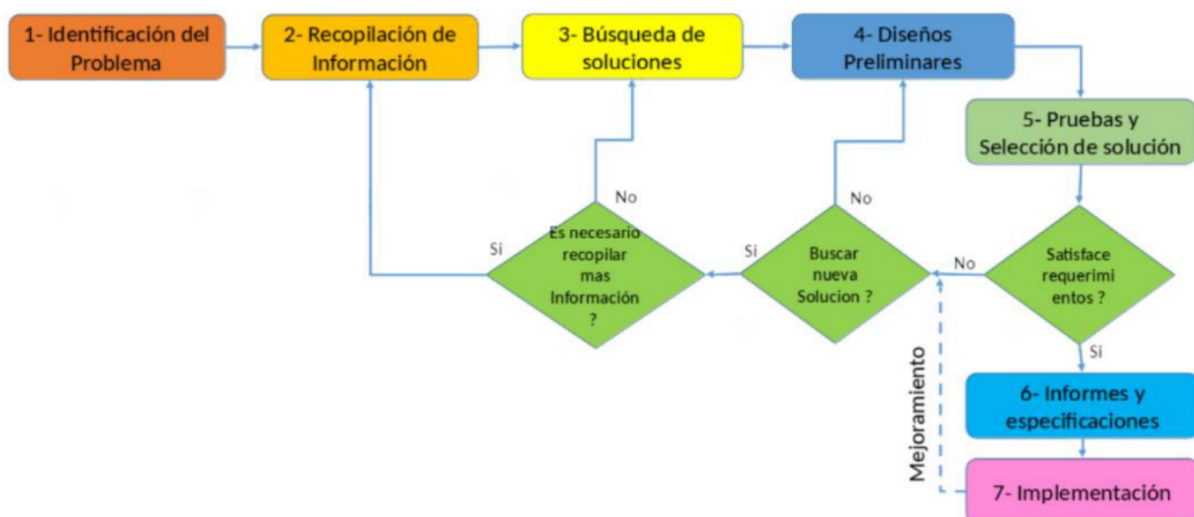
### Definición del Problema:

La FIBA requiere la creación de un software que esté en la capacidad manejar datos estadísticos masivos de cada uno de los basquetbolistas que se encuentren registrados en la base de datos.

### Desarrollo de la solución

Para sobrellevar el problema anteriormente mencionado se seleccionó el Método de la Ingeniería para desarrollar una solución, mediante un enfoque sistemático y acorde a la situación problemática planteada.

Para esto, se implementó la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, se define el siguiente diagrama de flujo, para seguir los pasos en el desarrollo de la solución.



### **Especificación de Requerimientos.**

El software debe estar en la capacidad de:

**Req 1. Gestionar** un jugador que se encuentra en la base de datos.

Req 1.1. Agregar un jugador en la base de datos de la aplicación, donde cada uno tendrá un nombre, apellido, edad, equipo, puntos, rebotes, asistencias, robos y bloqueos por partido.

Req 1.2. Eliminar un jugador de la base de datos de la aplicación.

Req 1.3 Modificar un atributo de un jugador que se encuentra registrado en la base de datos de la aplicación.

**Req 2. Permitir** que se ingresen datos de manera masiva mediante archivos de texto.

**Req 3. Buscar** a un jugador por medio de un dato estadístico.

Req 3.1 Buscar a un jugador por medio de los puntos anotados por partido.

Req 3.2 Buscar a un jugador por medio de los rebotes realizados por partidos.

Req 3.3 Buscar a un jugador por asistencias por partido.

Req 3.4 Buscar a un jugador por robos por partido.

Req 3.5 Buscar a un jugador por medio de bloqueos por partido.

**Req 4. Guardar** información en una memoria secundaria de manera persistente.

**Req 5. Buscar** un jugador por medio de dos datos estadísticos que se almacenarán en árboles binarios.

Req 5.1 Guardar a un jugador por medio de las asistencias en un árbol binario de búsqueda.

Req 5.2 Guardar a un jugador por medio de los robos por partido en un árbol binario de búsqueda.

**Req 6. Mostrar** el tiempo que tarda cada búsqueda en realizarse.

### **Etapa 2. Marco Teórico.**

Con el objetivo de abarcar el problema con una mayor claridad, se buscarán las definiciones relacionadas con el problema previamente definido.

#### Definiciones:

Fuente:

<https://www.significados.com>  
<https://es.wikipedia.org>

<https://gestion.pe/economia/management-empleo/>

[Árboles binarios Balanceados](#)

<https://www.rae.es>

*Base de datos:* Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

*Índice:* El índice de una base de datos es una estructura de datos que mejora la velocidad de las operaciones, por medio de un identificador único de cada fila de una tabla, permitiendo un rápido acceso a los registros de una tabla en una base de datos.

*Interfaz:* Dispositivo capaz de transformar las señales generadas por un aparato en señales comprensibles por otro.

*Masivo:* Se aplica en gran cantidad.

*Criterio:* Como criterio se denomina el principio o norma según el cual se puede conocer la verdad, tomar una determinación, u opinar o juzgar sobre determinado asunto.

*Memoria secundaria:* La memoria secundaria, memoria auxiliar, memoria periférica o memoria externa, también conocida como almacenamiento primario, es el conjunto de dispositivos y soportes de almacenamiento de datos que conforman el subsistema de memoria de la computadora, junto con la memoria primaria o principal.

*Eficiencia:* se define como la relación entre los recursos utilizados en un proyecto y los logros conseguidos con el mismo. Se da cuando se utilizan menos recursos para lograr un mismo objetivo o cuando se logran más objetivos con los mismos o menos recursos.

*Árbol binario balanceado:* Un árbol binario balanceado es un árbol binario en el cual las alturas de los dos subárboles de todo nodo difiere a lo sumo en 1. El balance de un nodo en un árbol binario se define como la altura de su subárbol izquierdo menos la altura de su subárbol derecho.

*Estructura de datos:* En ciencias de la computación, una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Por lo general, las estructuras de datos eficientes son clave para diseñar algoritmos eficientes.

**Estado de la práctica respecto al problema.**

### **1. Implementación de hashtables para el manejo masivo de datos.**

Para el manejo masivos de datos una solución práctica puede ser la utilización de hash para acelerar los procesos relacionados con la búsqueda, registro y eliminación de conjuntos de datos. Imaginando que se buscan apellidos en una base de datos incluyendo a todos los trabajadores, la tarea en este punto puede suponer una inversión de tiempo enorme, ya que se busca una coincidencia en cada uno de los campos de la base de datos, esto, de manera secuencial. Es en este momento que, si se convierte el término a buscar en un elemento hash para así encontrarlo en una tabla hash, el proceso normalmente tomará mucho menos tiempo.

**Fuente:** <https://bit.ly/2Z9W9su>

## **2. Implementación de árboles binarios en el manejo masivo de datos.**

Cuando a manejo masivo de datos se refiere, los árboles binarios son de suma utilidad, ya que estos nos permiten guardar en cada una de sus hojas un valor de cualquier tipo que puede ser clasificado como menor o mayor, para así implementar una búsqueda de complejidad temporal logarítmica. En la vida, los árboles binarios son muy usados para la búsqueda de una gran cantidad de datos o en la codificación de Huffman, el cual es el método usado para comprimir imágenes o datos sin riesgo a pérdida, con la construcción de codificación de redundancia mínima. Sin la utilización de estos árboles, los datos tendrían que ser almacenados de forma lineal y el manejo de datos masivos sería casi imposible teniendo en cuenta la cantidad de memoria RAM que esto tomaría.

**Fuente:** <https://bit.ly/3m4j6WW>

## **Etapa 3. Búsqueda de soluciones creativas.**

Se realiza la búsqueda de diferentes soluciones que se pueden dar para resolver el problema de manejar una cantidad masiva de datos de forma rápida y eficiente. Para desarrollar este punto de la actividad se optó por realizar una **lluvia de ideas**, para poder generar una cantidad aceptable de opciones.

### **Alternativa 1. Búsqueda de atributos por medio de búsqueda Binaria con método de ordenamiento Bubble sort**

Es el método de búsqueda más eficaz de utilizar, ya que nos permite realizar búsquedas de un atributo en un tiempo  $O(\log n)$ . Esto es de gran ventaja al intentar buscar un elemento en una cantidad masiva de datos. Este método se basa en dividir a la mitad un arreglo para así identificar en qué parte del arreglo se encuentra el atributo que se quiere encontrar. Este método actúa de forma recursiva hasta llegar a un caso base o encontrar un atributo que se busca. En este caso el bubble sort consta de comparar cada objeto de la lista con el siguiente, intercambiándolos de lugar si están en un orden erróneo.

### **Alternativa 2. Búsqueda de atributos por medio de búsqueda Binaria con método de ordenamiento Insertion sort**

Es el método de búsqueda más eficaz de utilizar, ya que nos permite realizar búsquedas de un atributo en un tiempo  $O(\log n)$ . Esto es de gran ventaja al intentar buscar un elemento en una cantidad masiva de datos. Este método se basa en dividir a la mitad un arreglo para así identificar en qué parte del arreglo se encuentra el atributo que se quiere encontrar. Este método actúa de forma recursiva hasta llegar a un caso base o encontrar un atributo que se busca. Consta de tomar uno por uno los elementos de un arreglo y recorrerlo hacia su posición con respecto a los anteriormente ordenados. Así empieza con el segundo elemento y lo ordena con respecto al primero. Luego sigue con el tercero y lo coloca en su posición ordenada con respecto a los dos anteriores, así sucesivamente hasta recorrer todas las posiciones del arreglo.

**Alternativa 3. Búsqueda de atributos por medio de búsqueda Binaria con método de ordenamiento Selection sort**

Es el método de búsqueda más eficaz de utilizar, ya que nos permite realizar búsquedas de un atributo en un tiempo  $O(\log n)$ . Esto es de gran ventaja al intentar buscar un elemento en una cantidad masiva de datos. Este método se basa en dividir a la mitad un arreglo para así identificar en qué parte del arreglo se encuentra el atributo que se quiere encontrar. Este método actúa de forma recursiva hasta llegar a un caso base o encontrar un atributo que se busca. El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo.

**Alternativa 4. Búsqueda de atributos por medio de Búsqueda lineal**

Este método de búsqueda es el más simple de utilizar, ya que solo se necesita un parámetro que se compare con toda una lista de atributos para encontrar cuál es el atributo que es igual a dicho parámetro. Esto significa que se tiene que pasar de forma secuencial por cada uno de los atributos que se encuentran en un arreglo para así encontrar el atributo semejante.

**Alternativa 5. Búsqueda Binaria de árbol balanceado**

Este método es uno de los más eficientes para realizar búsqueda masiva de datos, ya que con la implementación de los árboles AVL se obtiene un árbol binario balanceado que impide que la búsqueda que se realice a dicho árbol sea lineal, osea, que los hijos del árbol no se concentren en una parte de este, lo que hace a la búsqueda muy eficiente, con una complejidad temporal de  $O(\log n)$ .

**Alternativa 6. Búsqueda Binaria de árbol balanceado Rojo-Negro**

Este método consiste en tener un árbol balanceado con nodos que se diferencian por dos características, que sean negro o rojos, además, este tipo de árbol binario tiene que cumplir con algunos criterios, como lo es que la raíz es de color negro, que todo nodo rojo tiene un padre negro, que un nodo rojo tiene dos hijos negros, entre otros. De lo anterior se dice que al

cumplir con esas condiciones, el árbol se encuentra en equilibrio y se tiene que el camino desde la raíz hasta una hoja no es más largo que dos veces el camino más corto.

#### **Etapa 4. Transición de las Ideas a los Diseños Preliminares.**

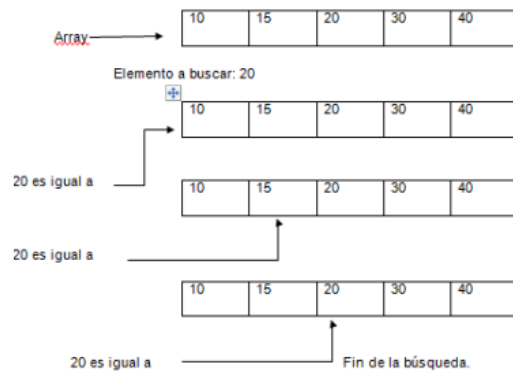
Las alternativas 1, 2 y 3 son descartadas, porque aunque se utilice la búsqueda binaria, los métodos de ordenamiento previos que se utilizan para organizar los datos con que se realizará la búsqueda binaria, no son nada eficientes cuando se trata de organizar datos en una cantidad masiva, por ende, no cumplen con el criterio del problema, el cual es encontrar un atributo en una cantidad de datos masiva de forma eficiente.

Luego de descartar las opciones no factibles para desarrollar el problema, se realizará una amplia revisión de las alternativas restantes, para así definir una de ellas como la solución pertinente al problema.

#### **Alternativa 4. Búsqueda de atributos por medio de Búsqueda lineal**

- La complejidad temporal de este método es lineal:  $O(n)$ .
- La búsqueda tiene dificultades de eficiencia y almacenamiento de memoria RAM si se enfrenta a una cantidad masiva de datos.

Búsqueda Lineal:

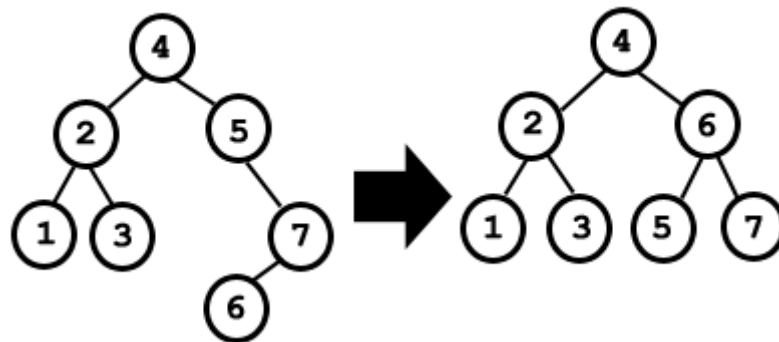


Aquí está representada la búsqueda lineal, la cual consiste en buscar en cada posición de un arreglo un elemento determinado.

#### **Alternativa 5. Búsqueda Binaria de árbol balanceado**

- Como el objetivo de esta estructura de datos es mantener un alto equilibrio, se tiene que los costos de insertar y eliminar de manera dinámica aumentan.
- El costo de la búsqueda está relacionado directamente con el tamaño del árbol binario.

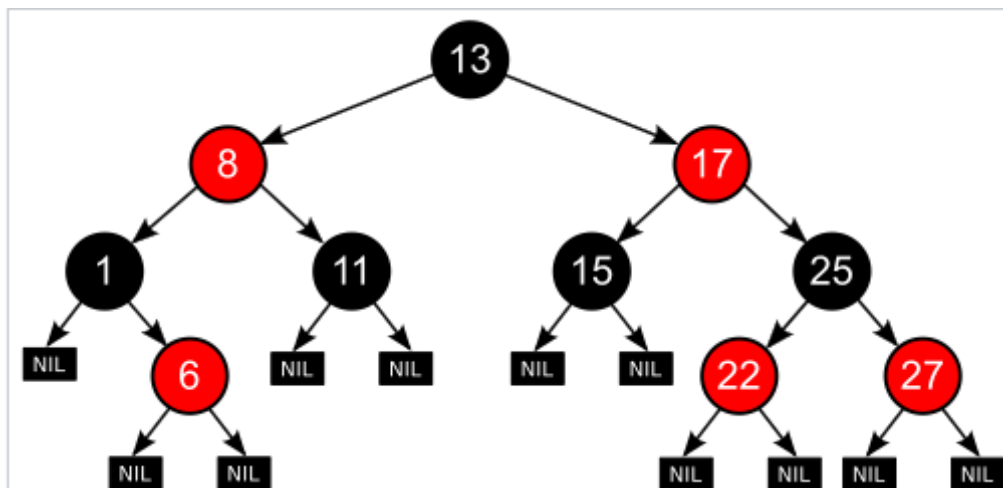
Árbol binario Balanceado (AVL):



Aquí se puede ver cómo es el funcionamiento del árbol AVL, cuyo objetivo es balancear los árboles, lo que significa que el factor de balanceo debe ser 1, 0 o -1.

#### Alternativa 6. Búsqueda Binaria de árbol balanceado Rojo-Negro

- Este método está limitado a ciertas propiedades que hacen tediosa su implementación.
- Su complejidad espacial es mayor que la implementación de otros árboles binarios por la utilización de nodos centinelas.



Aquí se puede ver el funcionamiento de los árboles binarios balanceados negro - rojo, cuyo objetivo es balancear el árbol binario dependiendo de las propiedades que trae consigo este tipo de árbol.

#### **Etapas 5. Evaluación y Selección de la Mejor Solución.**

##### Criterios:

Se van a definir criterios para así evaluar las alternativas de la solución al problema. Cada uno de los criterios nos permitirá definir cuáles son las características fundamentales que debe tener la solución. Cada característica representada en los criterios tendrá un valor que significa qué tanto peso tiene esta solución del problema.

Criterio A. Eficiencia. Se prefiere una solución con mejor eficiencia que las demás. La eficiencia puede ser:



- [4] Constante.
- [3] Logarítmica.
- [2] Lineal.
- [1] Cuadrática.

Criterio B. Complejidad. Se refiere a una solución que sea fácil y sencilla de implementar:

- [3] Fácil de implementar.
- [2] Complejidad media de implementación.
- [1] Complejo de implementar.

Criterio C. Óptimo. Se refiere al buen uso de los recursos para el funcionamiento de la solución:

- [3] Compatible para cualquier equipo de cómputo de bajos recursos.
- [2] Compatible para cualquier equipo de cómputo de recursos medios.
- [1] Compatible para cualquier equipo de cómputo de altos recursos.

#### Evaluación:

Evaluando los criterios anterior de las tres alternativas definidas, se obtiene la siguiente tabla:

	Criterio A	Criterio B	Criterio C	Total
Alternativa 4.	2	3	2	7
Alternativa 5.	3	2	3	8
Alternativa 6.	3	1	3	7

#### Selección:

A partir de los resultados obtenidos en la evaluación, podemos concluir que la alternativa 5 (Búsqueda Binaria de árbol balanceado) es la mejor solución, ya que obtuvo un puntaje casi perfecto teniendo en cuenta los criterios de evaluación para una solución pertinente.

## TADS DE ESTRUCTURAS DE DATOS

TAD Árbol Binario de Búsqueda			
<b>Árbol</b> = {Raíz = $\langle \text{raíz} \rangle$ }			
{inv: $\text{nodo.hijoIzquierdo} \leq \text{nodo.padre} \wedge \text{nodo.hijoDerecho} > \text{nodo.padre}$ }			
Operaciones Primitivas			
CrearÁrbol:			Constructora
AgregarNodo:	Valor	→ Booleano	Modificadora
EliminarNodo:	Valor	→ Booleano	Modificadora
BuscarNodo:	Valor	→ Nodo	Analizadora
ObtenerPredecesor:		→ Nodo	Analizadora
ObtenerSucesor:		→ Nodo	Analizadora

### CrearÁrbol()

“Crear un árbol binario con una raíz nula”

{pre: TRUE}

{post: raíz = nulo}

### AgregarNodo(e)

“Agregar un nodo al árbol binario”

{pre: árbol =  $\langle \rangle$  ∨ árbol =  $\langle p, q, r \rangle$ }

{post: árbol =  $\langle e \rangle$  ∨ árbol =  $\langle p, q, r, e \rangle$ }

### EliminarNodo(e)

“Eliminar un nodo del árbol binario”

{pre: árbol =  $\langle e \rangle$  ∨ árbol =  $\langle p, q, r, e \rangle$ }

{post: árbol =  $\langle \rangle$  ∨ árbol =  $\langle p, q, r \rangle$ }

### BuscarNodo(e)

“Buscar un nodo dentro del árbol binario”

{pre: árbol =  $\langle e \rangle$  ∨ árbol =  $\langle p, q, r, e \rangle$  ∨ árbol =  $\langle \rangle$ }

{post: e ∨ nulo}

### ObtenerPredecesor()

“Retornar el nodo predecesor del nodo”

{pre: nodo a analizar ≠ nulo}

{post: e ∨ nulo}

**ObtenerSucesor()**

“Retornar el nodo sucesor del nodo”

{pre: nodo a analizar  $\neq$  nulo}

{post: e  $\vee$  nulo}

**TAD Árbol Binario de Búsqueda Balanceado (AVL)**

**Árbol** = {Raíz =  $\langle$  raíz  $\rangle$ }

{inv: nodo.hijoIzquierdo  $\leq$  nodo  $\wedge$  nodo.hijoDerecho  $\geq$  nodo  $\wedge$   
| (nodo.hijoDerecho.Altura - nodo.hijoIzquierdo.Altura) |  $\leq$  1}

**Operaciones primitivas**

CrearÁrbol			Constructora
AgregarNodo:	Valor	$\rightarrow$ Booleano	Modificadora
EliminarNodo:	Valor	$\rightarrow$ Booleano	Modificadora
BuscarNodo:	Valor	$\rightarrow$ Nodo	Analizadora
ObtenerPredecesor:		$\rightarrow$ Nodo	Analizadora
ObtenerSucesor:		$\rightarrow$ Nodo	Analizadora
ObtenerBalanceo:	Nodo	$\rightarrow$ Entero	Analizadora
ObtenerAlturaMáxima:	Nodo	$\rightarrow$ Entero	Analizadora
ObtenerAltura:	Nodo	$\rightarrow$ Entero	Analizadora
RotaciónSimpleDerecha:	Nodo	$\rightarrow$ Nodo	Modificadora
RotaciónSimpleIzquierda:	Nodo	$\rightarrow$ Nodo	Modificadora
RotaciónDobleDerecha:	Nodo	$\rightarrow$ Nodo	Modificadora
RotaciónDobleIzquierda:	Nodo	$\rightarrow$ Nodo	Modificadora

**CrearÁrbol()**

“Crear un árbol binario balanceado con una raíz nula”

{pre: TRUE}

{post: raíz = nulo}

**AgregarNodo(e)**

“Agregar un nodo al árbol binario balanceado”

{pre: árbol  $\langle$   $\rangle$   $\vee$  árbol  $\langle$  p, q, r  $\rangle$ }

{post: True si árbol  $\langle$  e  $\rangle$   $\vee$  árbol  $\langle$  p, q, r, e  $\rangle$ }

**EliminarNodo(e)**

“Eliminar un nodo del árbol binario”

{pre: árbol =  $\langle$  e  $\rangle$   $\vee$  árbol =  $\langle$  p, q, r, e  $\rangle$ }

{post: True si árbol =  $\langle$   $\rangle$   $\vee$  árbol =  $\langle$  p, q, r  $\rangle$ }

**BuscarNodo(e)**

“Buscar un nodo dentro del árbol binario”

{pre: árbol =  $\langle e \rangle \vee$  árbol =  $\langle p, q, r, e \rangle \vee$  árbol =  $\langle \rangle$ }  
{post: e  $\vee$  nulo}

**ObtenerAltura(Nodo)**

“Obtiene la altura de un nodo”

{pre: nodo a obtener altura  $\neq$  nulo}  
{post: altura}

**ObtenerBalanceo(Nodo)**

“Cambiar el factor de balanceo de un nodo”

{pre: nodo a calcular balanceo  $\neq$  nulo}  
{post: valor del factor de balanceo}

**ObtenerAlturaMáxima(Nodo)**

“Obtiene la altura máxima de un nodo”

{pre: nodo a obtener la altura máxima  $\neq$  nulo}  
{post: altura máxima del nodo}

**ObtenerPredecesor()**

“Retornar el nodo predecesor del nodo”

{pre: nodo a analizar  $\neq$  nulo}  
{post: e  $\vee$  nulo}

**ObtenerSucesor()**

“Retornar el nodo sucesor del nodo”

{pre: nodo a analizar  $\neq$  nulo}  
{post: e  $\vee$  nulo}

**RotaciónSimpleDerecha(Nodo)**

“Rotar un nodo de forma simple a la derecha”

{pre: nodo a rotar  $\neq$  nulo}  
{post: nodo rotado}

**RotaciónSimpleIzquierda (Nodo):**

“Rotar un nodo de forma simple a la izquierda”

{*pre*: nodo a rotar  $\neq$  nulo}

{*post*: nodo rotado}

**RotaciónDobleDerecha**(Nodo):

“Rotar un nodo de forma doble a la derecha”

{*pre*: nodo a rotar  $\neq$  nulo}

{*post*: nodo después de ser rotado dos veces}

**RotaciónDobleIzquierda**(Nodo):

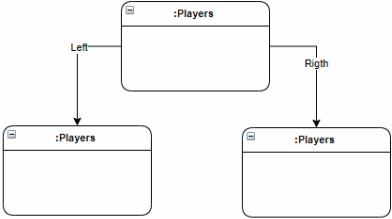
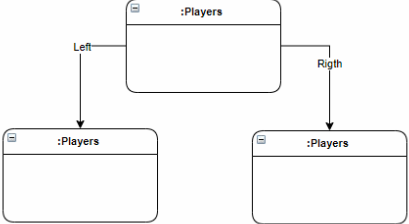
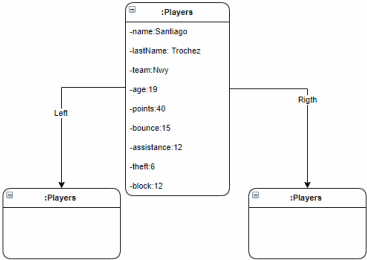
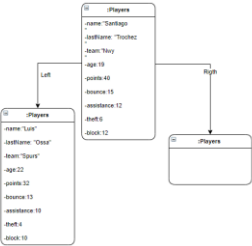
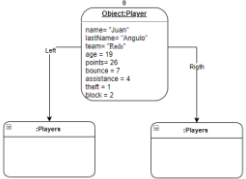
“Rotar un nodo de forma doble a la izquierda”

{*pre*: nodo a rotar  $\neq$  nulo}

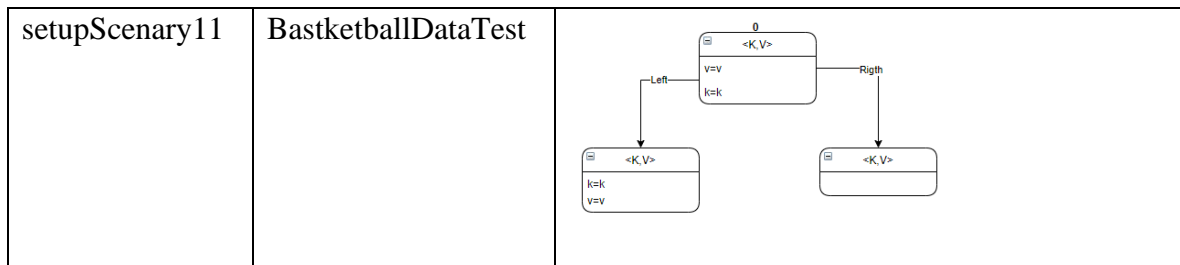
{*post*: nodo después de ser rotado dos veces}

## CASOS DE PRUEBA

### Configuración:

Nombre	Clase	Escenario
setupScenary1	BasketballDataTest	<p>Árbol binario de búsqueda.</p>  <pre> graph TD     Root[":Players"] -- Left --&gt; LeftChild[":Players"]     Root -- Right --&gt; RightChild[":Players"] </pre>
setupScenary2	BasketballDataTest	<p>Arbol Avl</p>  <pre> graph TD     Root[":Players"] -- Left --&gt; LeftChild[":Players"]     Root -- Right --&gt; RightChild[":Players"] </pre>
setupScenary3	BasketballDataTest	 <pre> graph TD     Root[":Players&lt;br/&gt;-name: Santiago&lt;br/&gt;-lastName: Trochez&lt;br/&gt;-team: Nuy&lt;br/&gt;-age: 19&lt;br/&gt;-points: 40&lt;br/&gt;-bounce: 15&lt;br/&gt;-assistance: 12&lt;br/&gt;-theft: 6&lt;br/&gt;-block: 12"] -- Left --&gt; LeftChild[":Players"]     Root -- Right --&gt; RightChild[":Players"] </pre>
setupScenary4	BasketballDataTest	 <pre> graph TD     Root[":Players&lt;br/&gt;-name: Santiago&lt;br/&gt;-lastName: Trochez&lt;br/&gt;-team: Nuy&lt;br/&gt;-age: 19&lt;br/&gt;-points: 40&lt;br/&gt;-bounce: 15&lt;br/&gt;-assistance: 12&lt;br/&gt;-theft: 6&lt;br/&gt;-block: 12"] -- Left --&gt; LeftChild[":Players&lt;br/&gt;-name: Luis&lt;br/&gt;-lastName: Cruz&lt;br/&gt;-team: Quere&lt;br/&gt;-age: 22&lt;br/&gt;-points: 32&lt;br/&gt;-bounce: 13&lt;br/&gt;-assistance: 10&lt;br/&gt;-theft: 4&lt;br/&gt;-block: 10"]     Root -- Right --&gt; RightChild[":Players"] </pre>
SetupScenary5	BasketballDataTest	 <pre> graph TD     Root["ObjectPlayer&lt;br/&gt;-name= 'Juan'&lt;br/&gt;-lastName= 'Angulo'&lt;br/&gt;-team= 'Real'&lt;br/&gt;-age = 19&lt;br/&gt;-points = 25&lt;br/&gt;-bounce = 7&lt;br/&gt;-assistance = 4&lt;br/&gt;-theft = 1&lt;br/&gt;-block = 2"] -- Left --&gt; LeftChild[":Players"]     Root -- Right --&gt; RightChild[":Players"] </pre>

setupScenary6	BastketballDataTest	
SetupScenary7	BastketballDataTest	
setupScenary8	BastketballDataTest	
setupScenary9	BastketballDataTest	
setupScenary10	BastketballDataTest	



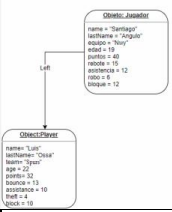
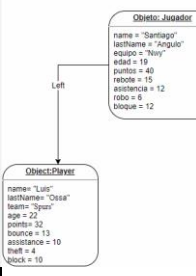
### Diseño de casos de prueba:

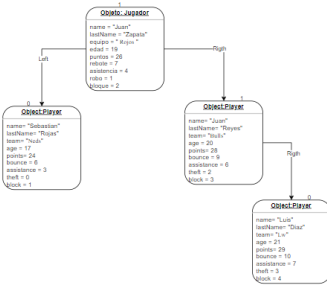
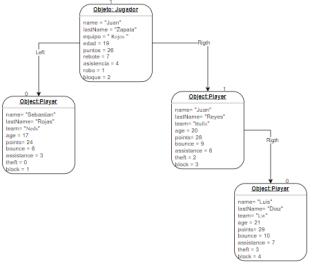
Objetivo de la prueba: Verificar que el árbol binario de búsqueda fue creado				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	create	setupScenary1		Se creó correctamente un árbol binario

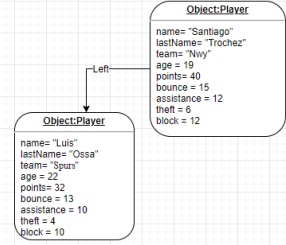
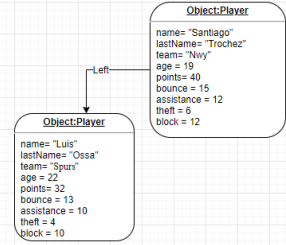
Objetivo de la prueba: Verificar que se creó el árbol Avl				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	create	setupScenary2		Se creó correctamente un árbol Avl

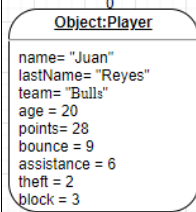
Objetivo de la prueba: Verificar que creo un nodo y se añadió a la raíz del árbol binario				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	addNode	setupScenary3	Object="Santiago","Trochez","Nwy",19,40,15,12,6,12	Se creó el jugador



Objetivo de la prueba: Validar que se modificó un objeto al árbol binario de búsqueda				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	addNode y deleteNode	setupScenary4	Object="Santiago","Angulo","Nwy",19,40,15,12,6,12	Se modificó <div>  </div>
BasketballData	addNode	setupScenary4	Object="Luis"." Ossa"," Nwy",19,40,15,12,6,12	No se pudo modificar por que ya hay un objeto con el mismo nombre <div>  </div>

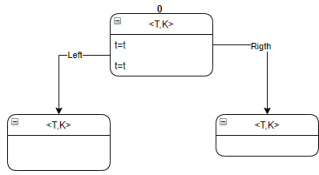
Objetivo de la prueba: Validar que se modificó un objeto al árbol Avl				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	addNode y deleteNode	setupScenary6	Objeto="Juan","Zapata","Reds",19,26,7,4,1,2	<div>  </div>
BasketballData	addNode	setupScenary5	Objeto="Juan","Diaz","Reds",21,29,10,7,3,4	No se pudo modificar por que ya hay un objeto con el mismo nombre <div>  </div>

Objetivo de la prueba: Validar que se agregó un objeto al árbol binario de búsqueda				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	addNode	setupScenariy4	Object="Luis"."Ossa","Spurs",22,32,13,10,4,10	
BasketballData	addNode	setupScenariy4	Object="Luis"."Ossa","Spurs",22,32,13,10,4,10	<p>No se creo,porque ya existe un jugador con esos datos</p> 

Objetivo de la prueba: Validar que se agregó un objeto al árbol Avl				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	addNode	setupScenariy5	Objeto="Juan"."Reyes","Bulls",20,28,9,6,2,3	<p>0</p> 

Objetivo de la prueba: Validar que se agregan más objetos al árbol Avl				
Clase	Método	Escenario	Valores de entrada	Resultado
BasketballData	addNode	setupScenary6	Objeto="Juan","Angulo","Reds",19,26,7,4,1,2	
BasketballData	addNode	setupScenary6	Objeto="Sebastian","Rojas","Neds",17,24,6,3,0,1	
BasketballData	addNode	setupScenary6	Objeto="Luis","Diaz","Lw",21,29,10,7,3,4	

Objetivo de la prueba: Validar que se eliminó un objeto del árbol Avl				
Clase	Método	Escenario	Valores de entrada	Resultado
AVLTree	deleteNode()		treeAvl.getRoot()	

Objetivo de la prueba: Validar que elimino un objeto de un árbol binario				
Clase	Método	Escenario	Valores de entrada	Resultado
BinaryTree	deleteNode		k	 <pre> graph TD     Node["&lt;T,K&gt; t=t t=t"] -- Left --&gt; LeftNode["&lt;T,K&gt;"]     Node -- Right --&gt; RightNode["&lt;T,K&gt;"] </pre>

