UNIVERSIDAD ICESI

FACULTAD DE INGENIERÍA

METODO DE LA INGENIERIA

Aplicación para la solución del Laboratorio 3

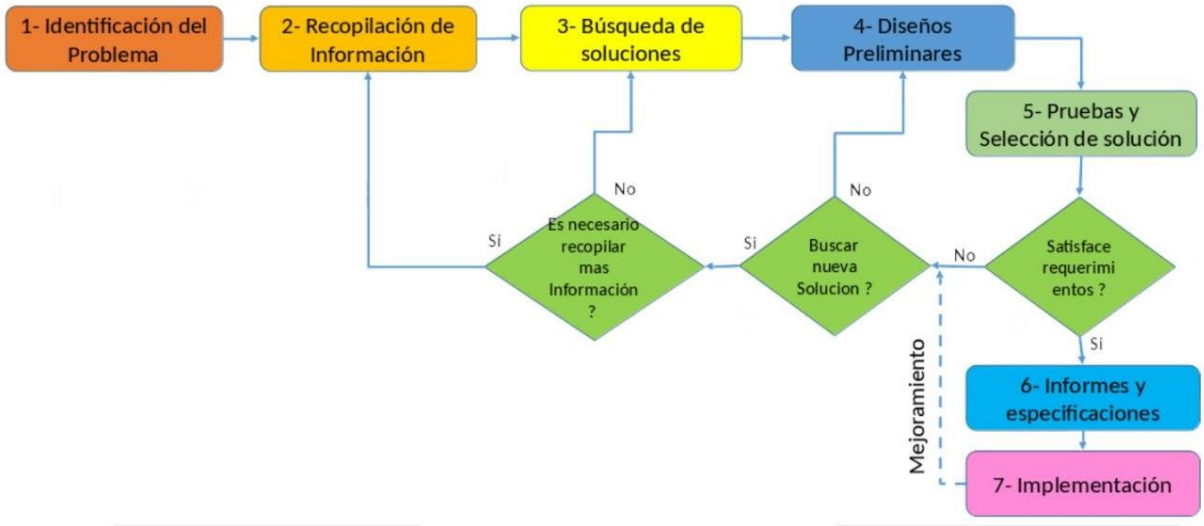
Jeferson Lerma Girón – A00301456
jeferson.lerma@correo.icesi.edu.co

Daniel Alejandro Gómez Paramo – A00305232
danielalejo1998@gmail.com

Desarrollo de la solución:

Para resolver la situación anterior se eligió el Método de la Ingeniería para desarrollar la solución siguiendo un enfoque sistemático y acorde con la situación problemática planteada.

Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.



Fase 1. Identificación del problema

La Federación Internacional de Baloncesto, mejor conocida como FIBA, es el ente regulador del basquetbol a nivel mundial, aquél que define las reglas de este deporte a nivel internacional y el organismo no sólo encargado de organizar y coordinar las más importantes competiciones orbitales sino de reunir a todos los practicantes de este deporte a nivel profesional.

Ante la reciente avalancha de números en donde cualquier cifra proveniente del juego(parido de baloncesto) es susceptible de deparar un mensaje mínimamente útil, la FIBA ha decidido aprovechar esta coyuntura y consolidar en una aplicación, los datos de mayor relevancia de cada uno de los profesionales del baloncesto en el planeta, de manera que se puedan efectuar diferentes consultas que permitan realizar análisis sobre estos datos, se conozcan patrones acerca del desarrollo del deporte, los criterios que toman más fuerza o, en general, hacia dónde se dirige el deporte en la actualidad.

Identificación y definición concreta y sin ambigüedad del problema:

Definición del Problema

la implementación de una herramienta para el manejo de información de gran tamaño que permita ingresar datos ya sea de manera masiva (archivos csv, por ejemplo) o a través de una interfaz; eliminar o modificar datos; realizar consultas de jugadores utilizando como criterios de búsqueda las categorías estadísticas incluídas (por ejemplo, encontrar aquellos jugadores que han anotado 10 puntos por partido, o más de 20 rebotes por partido). Como primera versión del software, la FIBA solicita, como mínimo el incluir los datos por jugador, de los siguientes ítems: nombre, edad, equipo, y 5 rubros estadísticos (e.g. puntos por partido, rebotes por partido, asistencias por partido, robos por partido, bloqueos por partido). Asimismo, debe tener en cuenta que esta herramienta puede llegar a almacenar millones de datos y la rapidez para la obtención de estos resulta fundamental para el desempeño de la aplicación.

Cabe resaltar la importancia de garantizar un rápido acceso a los datos, es decir, eficiencia en las consultas. Definitivamente la complejidad temporal no puede ser lineal, en lo que a la búsqueda de jugadores se refiere, ya que sería muy lento, teniendo en cuenta los datos de millones de basquetbolistas guardados.

La aplicación que se desarrolle debe estar en la capacidad de recuperar jugadores de acuerdo con la categoría de búsqueda seleccionada y el valor dado para ella (recuerde que no necesariamente se piden consultas en donde el atributo haga parte de una igualdad). El criterio de búsqueda puede ser cualquiera de los atributos estadísticos, pero para 4 de ellos la búsqueda debe ser muy rápida. Estas características para los cuales la búsqueda de datos de jugadores es muy eficiente, se denominan índices.

En conclusión, se resumieron en 5 problemas principales, los cuales consisten en:

1) Realizar(implementar) una interfaz que muestre los datos de los jugadores.

2) Realizar(implementar) las opciones o funcionalidades de agregar, eliminar o buscar un jugador, usando diferentes criterios de búsqueda.

3) Realizar las anteriores opciones de manera eficiente, es decir que los tiempos de ejecución sean los mínimos posibles.

4) Controlar el manejo de los datos (en este caso sería los datos de los jugadores) utilizando grandes cantidades de datos utilizando archivos de tipo CSV, además de manejar estos datos en memoria secundaria.

Especificación de los requerimientos funcionales asociados con las necesidades planteadas en el enunciado:

Nombre	R1. Implementar una interfaz de usuario.
Resumen	Se realizará una interfaz que permita manejar los datos de los jugadores.
Entradas	
N/A.	
Resultados	
Permite visualizar la información de los jugadores que se seleccionen.	

Nombre	R2. Manejar los datos del programa utilizando archivos CSV.
Resumen	El programa funcionará utilizando archivos tipo CSV, los cuales se podrán ubicar y seleccionar desde el programa.
Entradas	
Resultados	
Permite almacenar muchos datos de forma organizada en un archivo CSV, para ser usados en el programa.	

Nombre	R3. Realizar los métodos de: Agregar, eliminar o buscar a un jugador en específico, utilizando diferentes criterios de búsqueda.
Resumen	El usuario a través de la interfaz podrá buscar, eliminar o agregar un jugador según el criterio que este elija, además de poder usar diferentes criterios de búsqueda que podrá utilizar.
Entradas	
Nombre, equipos, puntos por partido, bloqueos, entre otros datos del jugador, que serían los criterios que se escojan para que el usuario pueda utilizarlos.	
Resultados	
Se realiza correctamente la acción correspondiente que el usuario haya escogido: eliminar, agregar o buscar un jugador.	

Requerimientos no funcionales:

Nombre	RNF 1. Realizar las operaciones de búsqueda, inserción y eliminación de manera eficiente con una complejidad temporal menor que O(n).
Resumen	Se realizarán las operaciones de búsqueda, inserción y eliminación, utilizando algoritmos y estructuras que sean eficientes de la manera más rápida y que tenga complejidad temporal menor que O(n).
Entradas	
N/A.	
Resultados	
Programa eficiente.	

Nombre	RNF 2. Presentar el tiempo de búsqueda de los jugadores.
Resumen	Permite saber el tiempo en que se realizan las búsquedas dentro del programa.
Entradas	
N/A.	
Resultados	
Se muestra en pantalla el tiempo que tardan en realizarse las búsquedas dentro del programa.	

Fase 2. Recopilación de la información necesaria.

Para poder entender y contextualizarnos con las definiciones y conceptos que se tratarán se hace una búsqueda de las definiciones de los términos más estrechamente relacionados con el problema planteado. Es importante realizar esta búsqueda en fuentes reconocidas y confiables para conocer cuáles elementos hacen parte del problema y cuáles no, esto para resolver el problema de la manera más efectiva posible.

Enlaces:

https://jarroba.com/lectura-escritura-ficheros-java-ejemplos/

https://www.salleurl.edu/es/programacion-avanzada-y-estructura-de-datos

http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_BB.htm

https://es.wikipedia.org/wiki/Estructura_de_datos

https://es.wikipedia.org/wiki/%C3%81rbol_binario

https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda

https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro

https://es.wikipedia.org/wiki/%C3%81rbol_AVL

https://www.ics.uci.edu/~goodrich/teach/cs260P/notes/AVLTrees.pdf

https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda_auto-balanceable

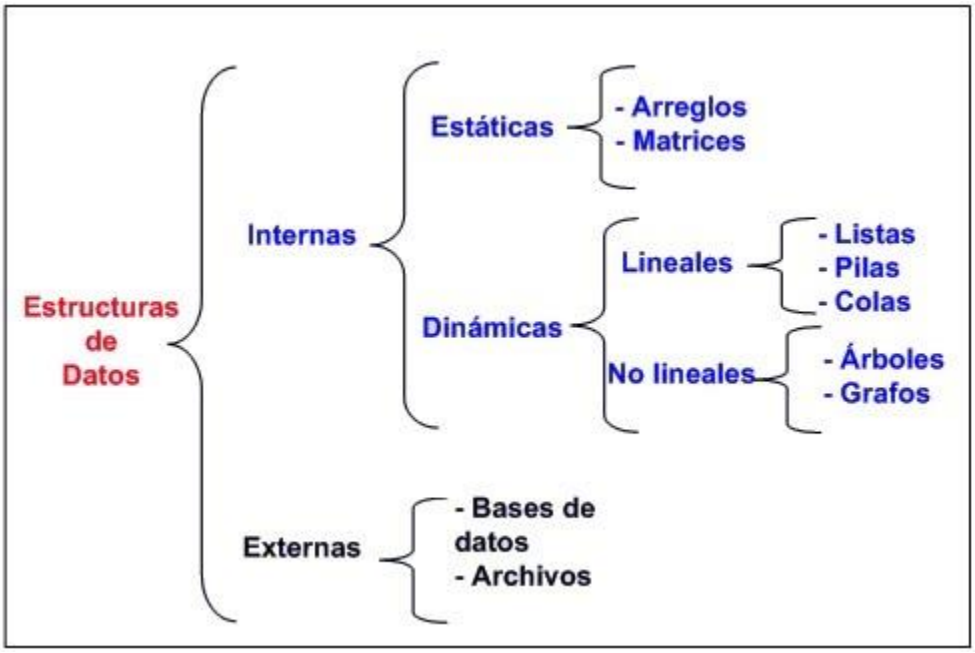
https://stackoverflow.com/questions/29836015/pseudocode-for-binary-search-tree
https://darkbyteblog.wordpress.com/2011/03/08/java-flujos-de-datos-entrada-y-salida-estandar/

https://www.thecodingdelight.com/avl-tree-implementation-java/
https://www.google.com.co/search?q=lectura+y+escritura+de+archivos+en+java&source=lnms&tbm=isch&sa=X&ved=0ahUKEwj76Nnr_qHeAhXrzVkKHVuuADsQ_AUIDigB&biw=1366&bih=626#imgrc=NCPra5AY1ueYtM:

http://ocw.udl.cat/enginyeria-i-arquitectura/programacio-2/continguts-1/4-manejo-bai81sico-dearchivos-en-java.pdf

Estructuras de datos eficientes:

En las estructuras de datos eficientes se trabaja el análisis algorítmico (costes, casos, notación asintótica, resolución de recurrencias simples), la especificación y la verificación formal de algoritmos recursivos simples. Se introduce el análisis de la eficiencia de los programas como un criterio más de calidad. Se estudia también el diseño de algoritmos recursivos simples y múltiples, y varias técnicas de resolución de problemas combinatorios. - En la segunda parte, se introduce el concepto y especificación de tipos abstractos de datos como estructuras de datos lineales, árboles, tablas y grafos. Las técnicas básicas mencionadas incluyen conocimientos teóricos y prácticos, habilidades, experiencias y sentido crítico, todas ellas fundamentadas en teorías y técnicas sólidas, comprobadas y bien establecidas.



Lectura y escritura de archivos:

Podemos utilizar diferentes formas de guardar los datos dentro de un programa, la más común es la lectura y escritura de archivos, estos pueden ser de texto(.txt) o archivos un poco más complejos como hojas de excel, archivos en bases de datos online, archivos .CSV, entre muchos otros. Se utilizan algoritmos especiales para leer estos archivos, y otros algoritmos para poder escribir sobre estos, y que la información sea persistente. Ejemplo:

Para escribir archivos podemos usar este tipo de algoritmo:

Sistema.java *EscritorArchivos.java

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;


public class EscritorArchivos {

    void escribir(String nombre){

        try {
            PrintWriter escritor = new PrintWriter(new FileWriter("archivo.loquesea"));
            escritor.write(nombre);

        } catch (IOException e) {

            e.printStackTrace();
        }

    }


}
```

Y para leer los archivos podemos utilizar el siguiente:

```java
 */
public class leertexto {
    public static void main(String[] args) {
        try{
            FileReader fr = new FileReader("C:\\Users\\javasolution\\Desktop\\numeros.txt");
            BufferedReader br =new BufferedReder(fr);
            String cadena;

            while((cadena=))

        }catch(Exception ex ){

        }
    }
```

Fase 3. Búsqueda de soluciones creativas

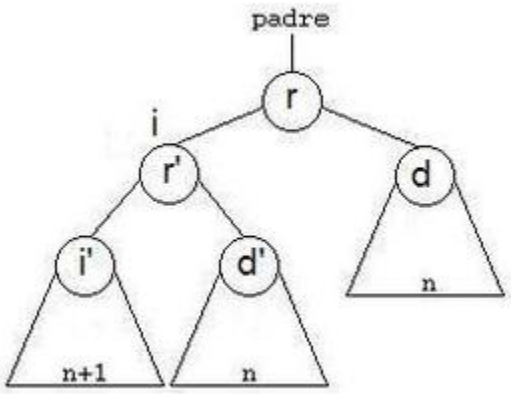
Por todo lo anterior planteado se hace indispensable buscar estructuras de datos con posible potencial para modelar cada uno de los problemas plateados. Para efectos del curso, describiremos las estructuras de datos que conocemos hasta el momento: Árboles Binarios de Búsqueda, Arboles AVL, Arboles Roji- Negros.

Árboles Binarios de búsqueda autobalanceables:

Son estructuras que implementan árboles binarios de búsqueda(ABB), pero con propiedades que le permiten auto balancearse según sea el caso que los haga estar no balanceados, se tiene que mantener su altura, o el número de niveles de nodos de una raíz o padre, según un criterio que se le asigne, normalmente es que sea lo menor posible siempre.

Dentro de ellas encontramos los arboles roji-negros y los arboles AVL(Adelson-Velskii y Landis).

Ejemplo:



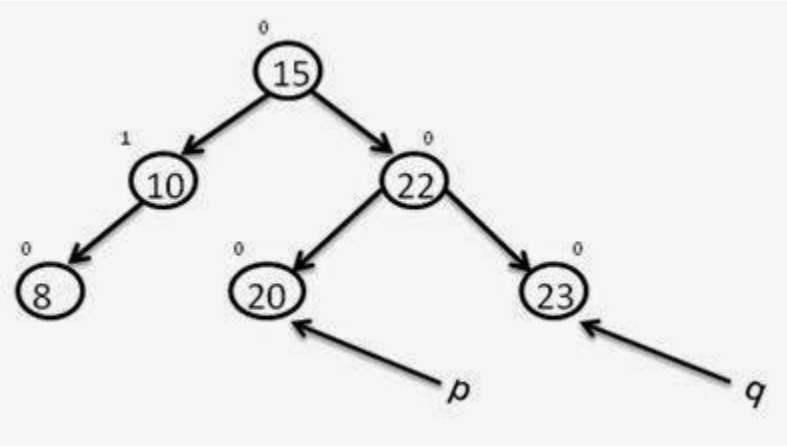
Árboles AVL:

Un árbol AVL es un tipo especial de árbol binario ideado por los matemáticos rusos AdelsonVelskii y Landis. Fue el primer árbol de búsqueda binario auto-balanceable que se ideó en el mundo.

Los árboles AVL están siempre equilibrados de manera que, para todos los nodos(Nodes), la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n). El factor de balanceo(utilizado en el curso de AED) puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

Para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o borrado se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

Ejemplo:



Árbol Roji-Negro:

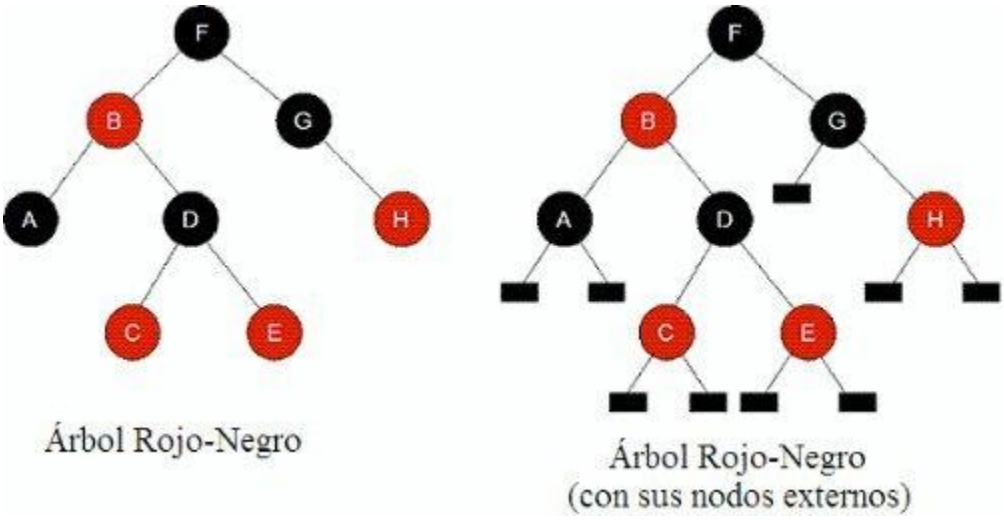
Es un árbol binario de búsqueda equilibrado, una estructura de datos utilizada en informática y ciencias de la computación, en donde cada nodo tiene también un atributo de color, cuyo valor puede ser o rojo o negro. Las hojas de un árbol rojo negro son irrelevantes y no tienen datos. También se hace alusión a un concepto de puntero null, al cual los nodos de las hojas apuntan hacia el.

Para que un árbol binario ordenado sea considerado como un árbol rojo negro debe cumplir con las siguientes propiedades:

1. La raíz del árbol es negra.
2. Los hijos de un nodo rojo son negros.
3. Las hojas del árbol son negras.
4. Todas las ramas del árbol (caminos desde la raíz hasta una hoja) tienen el mismo número de nodos negros.

Este tipo de estructuras sirven para el almacenamiento de elementos entre los que exista una relación de orden. La complejidad de la búsqueda en los árboles rojo negro es O (logn).

Ejemplo:



Árbol Binario de Búsqueda(ABB):

La búsqueda en árboles binarios es un método de búsqueda simple, dinámico y eficiente considerado como uno de los fundamentales en Ciencia de la Computación. De toda la terminología sobre árboles, tan sólo recordar que la propiedad que define un árbol binario es que cada nodo tiene a lo más un hijo a la izquierda y uno a la derecha. Para construir los algoritmos consideraremos que cada nodo contiene un registro con un valor clave a través del cual efectuaremos las búsquedas. En las implementaciones que presentaremos sólo se considerará en cada nodo del árbol un valor del tipo tElemento aunque en un caso general ese tipo estará compuesto por dos: una clave indicando el campo por el cual se realiza la ordenación y una información asociada a dicha clave o visto de otra forma, una información que puede ser compuesta en la cual existe definido un orden.

Un árbol binario de búsqueda(ABB) es un árbol binario con la propiedad de que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo x son menores que el elemento almacenado en x ,y todos los elementos almacenados en el subárbol derecho de x son mayores que el elemento almacenado en x.

Fase 4: Transición de la formulación de ideas, a los diseños preliminares.

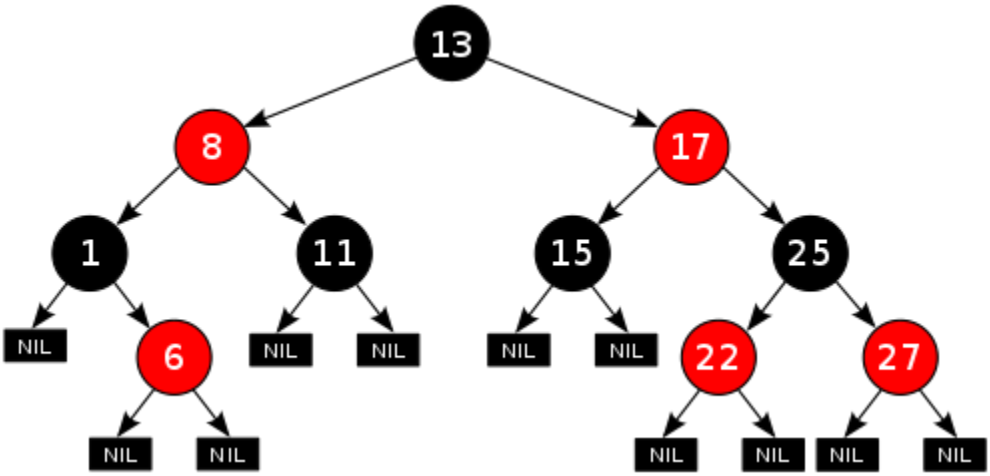
Se han escogido 3 estructuras de datos, debido que son las que más se adaptan a las necesidades del enunciado. La primera es Arboles Rojo y Negros, la segunda es Arboles AVL, y la tercera son Arboles de Búsqueda Binaria (ABB). Todas las estructuras se definen completamente a través de sus respectivos TADS, para que posteriormente se puedan implementar en el lenguaje de programación correspondiente.

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

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

Nombre: TAD Árbol Rojo Y Negro.

Un árbol roji negro se puede definir gráficamente como:



Invariantes:

- La raíz siempre es negra.
- Un nodo puede ser de cualquier color (Rojo o Negro)
- Un hijo Rojo no puede tener un padre Rojo
- Todas las ramas del árbol tienen el mismo número de nodos negros.

Operaciones:

Creadoras:

- **Crear Árbol Rojo y Negro: Árbol**

Descripción:

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

Postcondiciones: Árbol A ≠ Nill.

MODIFICADORAS:

- **Agregar (**Element E**): Árbol.**

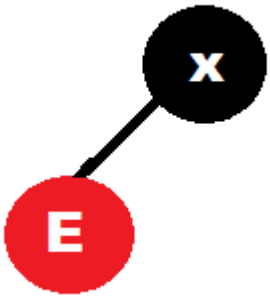
Precondiciones: Árbol A ≠ Nill; Elemento E a agregar ≠ Nill.

Descripción: Agrega a la Árbol A un nuevo elemento.

Postcondiciones:

Sea E el elemento a agregar en el Árbol A dado por:

Entonces:



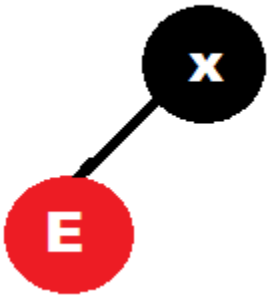
- **Eliminar (**Element E**): Árbol A.**

Precondiciones: Árbol A ≠ Nill, E (elemento a Eliminar) ≠ Nill.

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

Postcondiciones:

Sea E el elemento a eliminar:



Entonces:

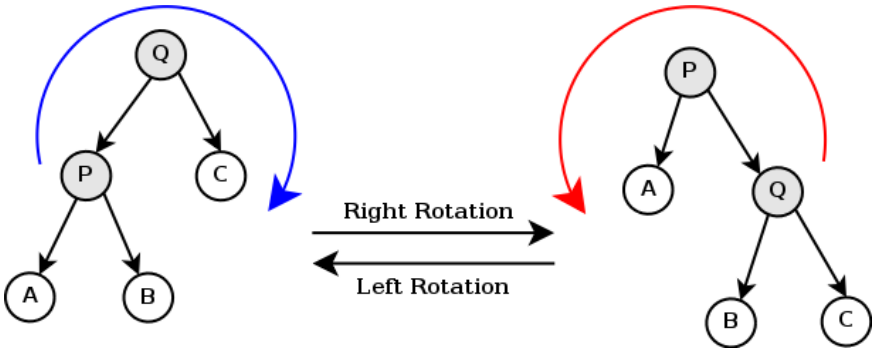
Rotar A la Izquierda o´ A la derecha (): void.

Precondiciones: Árbol A ≠ Nill

Descripción: Dado un puntero x, que representa la posición dentro del Árbol A, la rotación cambia el hijo con el padre dejando a este último como “hijo” y conservando el orden dentro del árbol.

Postcondiciones:

Dado Árbol A entonces:



Analizadoras:

- **Buscar (Element E): Árbol A.**

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

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

Postcondiciones:

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

2- ***TAD ARBOL AVL:***

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

Nombre: TAD 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:

Creadoras:

- **Crear Árbol AVL: Árbol**

Descripción:

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

Postcondiciones: Árbol A ≠ Nill.

Modificadoras:

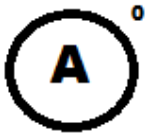
- **Agregar (**Element E**): Árbol.**

Precondiciones: Árbol A ≠ Nill; Elemento E a agregar ≠ Nill.

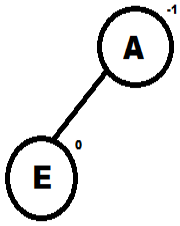
Descripción: Agrega a la Árbol A un nuevo elemento.

Postcondiciones:

Sea E el elemento a agregar en el Árbol A dado por:



Entonces:



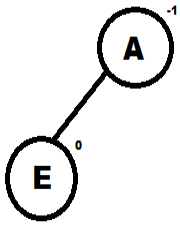
- **Eliminar (**Element E**): Árbol A.**

Precondiciones: Árbol A ≠ Nill, E (elemento a Eliminar) ≠ Nill.

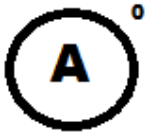
Descripción: Elimina el elemento E, pasa por parámetro del Árbol AVL.

Postcondiciones:

Sea E el elemento a eliminar:



Entonces:



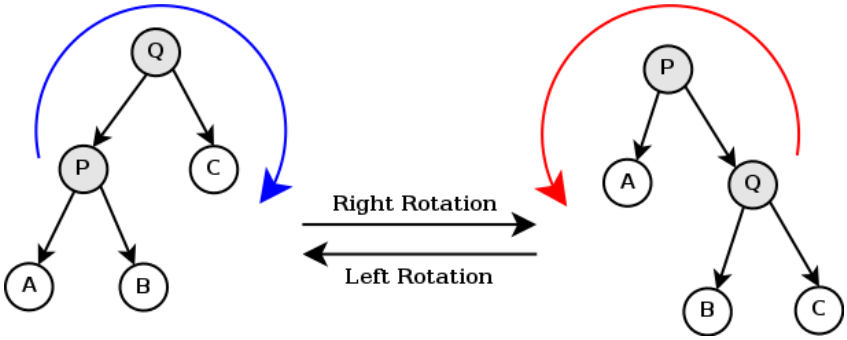
Rotar A la Izquierda o´ A la derecha (): void.

Precondiciones: Árbol A ≠ Nill

Descripción: Dado un puntero x, que representa la posición dentro del Árbol A, la rotación cambia el hijo con el padre dejando a este último como “hijo” y conservando el orden dentro del árbol.

Postcondiciones:

Dado Árbol A entonces:



- **Analizadoras:**

Analizadoras:

- **Buscar (Element E): Árbol A.**

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

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

Postcondiciones:

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

3- **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 ÁRBOL BINARIO DE BUSQUEDA

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:

Creadoras:

- **Crear Árbol Binario De Búsqueda: Árbol**

Descripción:

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

Postcondiciones: Árbol A ≠ Nill.

Modificadoras:

- **Agregar (**Element E**): Árbol.**

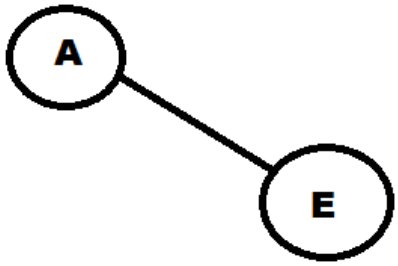
Precondiciones: Árbol A ≠ Nill; Elemento E a agregar ≠ Nill.

Descripción: Agrega a la Árbol A un nuevo elemento.

Postcondiciones:

Sea E el elemento a agregar al Árbol a en el siguiente estado.

Entonces si E es mayor que A:



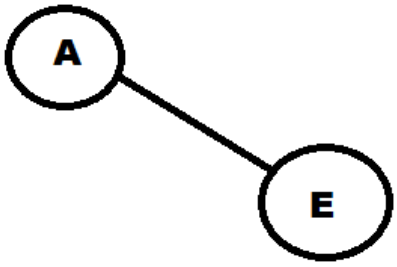
- **Eliminar (**Element E**): Árbol A.**

Precondiciones: Árbol A ≠ Nill, E (elemento a Eliminar) ≠ Nill.

Descripción: Elimina el elemento E, pasa por parámetro del Árbol Binario de Busqueda.

Postcondiciones:

Sea E el elemento a eliminar:



Entonces:

Analizadoras:

- **Buscar (Element E): Árbol A.**

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

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

Postcondiciones:

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

Fase 5: Evaluación y selección de la mejor solución.

Se escogieron unos criterios para seleccionar que estructuras se iban a implementar para modelar la solución del problema.

Criterios de selección de las estructuras de datos		Si/no	
A	La estructura tiene complejidad lineal	3	1
B	La estructura utiliza una complejidad temporal menor que O(n) para sus búsquedas	5	1
C	La estructura es autobalanceable	5	1
D	La estructura permite utilizar n cantidad de datos	5	1
E	La estructura es estable	5	1

Fase 6: Preparación de informes y especificaciones.

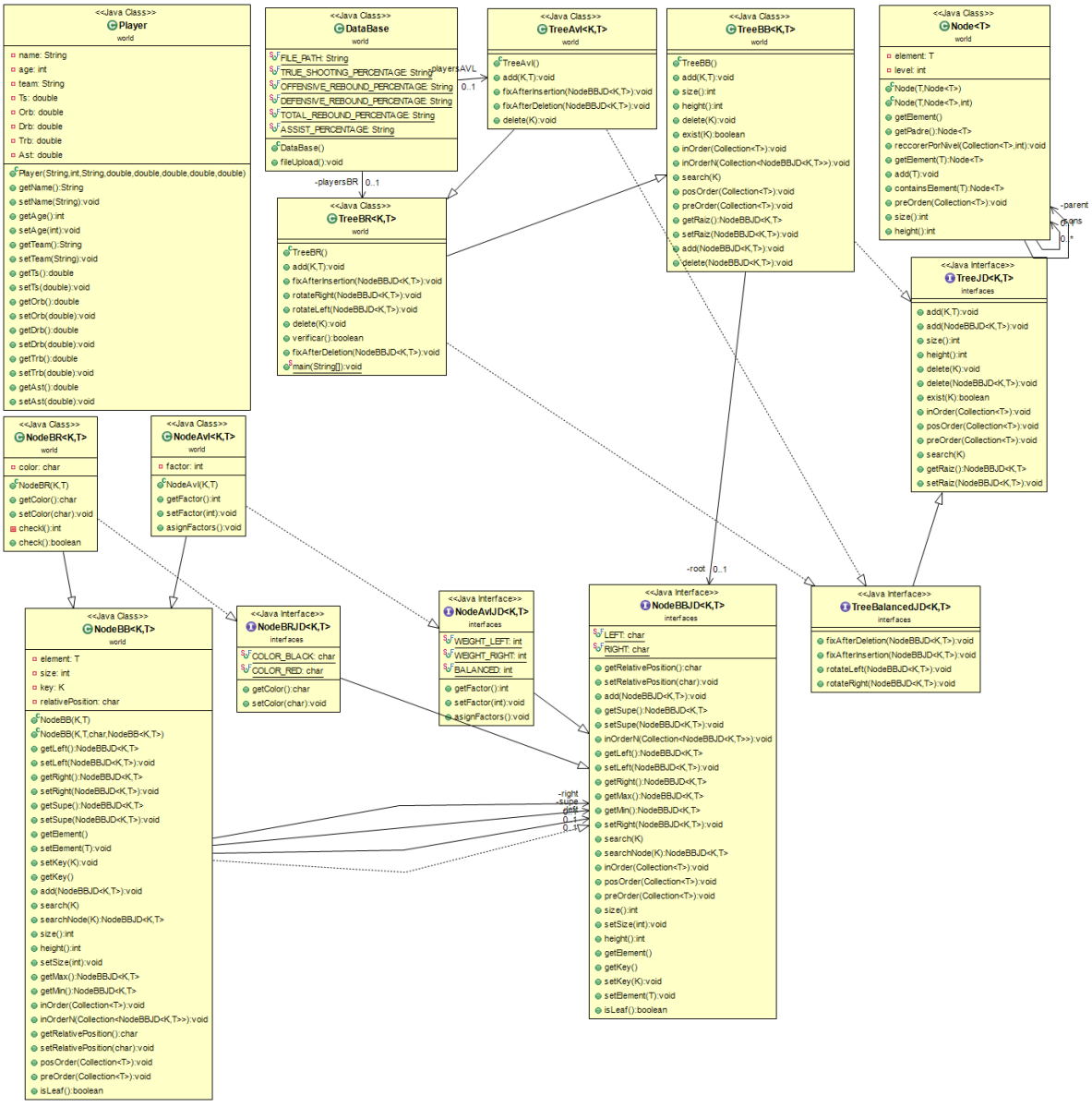
Calificación de las estructuras escogidas						TOTAL
Criterios	A	B	C	D	E	
Estructura						
ABB	3	5	1	5	5	19
A-AVL	3	5	5	5	5	23
A-Roji-negro	3	5	5	5	5	23
Arbol N-Ario	1	1	1	5	5	13

Para concluir, después de realizar un amplio estudio y análisis de cada una de las estructuras presentadas anteriormente, y escogidas bajo unos criterios especificados en el punto anterior; seleccionamos las 3(tres) siguientes estructuras: Árbol binario de búsqueda, Árbol AVL y Árbol Roji-Negro. Con estas se realizará la implementación de la solución, para así satisfacer los requerimientos del problema a tratar.

Fase 7: Implementación de la solución.

UNIVERSIDAD ICESI

Se adjunta el diagrama de clases del mundo:



UNIVERSIDAD ICESI