**BALONCESTO ( Análisis)**

**ETAPA 1: Aplicación del Método de la Ingeniería para solución de problemas**

**Contexto problemático.**

El baloncesto es uno de los deportes más importantes y populares a nivel orbital, ya sea por la simplicidad de su juego, por su práctica masiva a lo largo y ancho del planeta, por el espectáculo que genera a su alrededor o por la revolución estadística que ha traído consigo. El baloncesto nació en EEUU en el año 1891, lo inventó un Sacerdote Canadiense llamado James Naismith, este sacerdote viajó a Massachussets (EEUU) como profesor de “cultura física”, una vez allí observó que las bajas temperaturas no permitían practicar deportes al aire libre, así surgió el baloncesto como deporte que se podía practica en recintos cerrados.

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 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

Desarrollo de la solución.

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 incluidas (por ejemplo, encontrar aquellos jugadores que han anotado 10 puntos por partido, o más de 20 rebotes por partido).

Para el manejo de datos se debe utilizar una estructura de datos recursiva, en este caso arboles binarios balanceados o no balanceados para lograr una mayor eficiencia en la búsqueda de grandes cantidades de atributos de cada jugador que se encuentre en la estructura.

Identificación del problema.

* Existe una gran cantidad de jugadores en la FIBA.
* No se pude realizar un análisis de los datos de cada jugador.
* No se tiene claro hacia dónde se dirige este deporte a futuro.
* Es difícil manejar esta gran cantidad de datos de una forma rápida y efectiva.

Recopilación de información.

* Baloncesto: Deporte que se practica, en una cancha rectangular, entre dos equipos de cinco jugadores que tratan de introducir el balón en la canasta contraria, que se encuentra a una altura de 3,05 m, valiéndose solo de las manos; los encestes valen uno, dos o tres puntos y gana el equipo que logra más puntos en los 40 minutos que dura el encuentro.
* FIBA: La Federación Internacional de Baloncesto es el organismo que se dedica a regular las normas del baloncesto mundialmente, así como de celebrar periódicamente competiciones y eventos en sus dos disciplinas.
* Punto: Un punto es una cantidad que muestra el logro de un jugador o un equipo junto.
* Rebote: acto de conseguir la posesión del balón después de un lanzamiento de campo o de un [tiro libre](https://es.wikipedia.org/wiki/Tiro_libre_(baloncesto)) fallado.
* Asistencia: pase a un jugador que se encuentra en una posición de ventaja o que le ayuda a conseguir una canasta.
* Bloqueo:  acción de juego en la que un jugador defensivo desvía el tiro de un jugador ofensivo del equipo contrario legalmente, sin tocar al jugador ofensivo.

Búsqueda de soluciones creativas.

Para el anejo de los datos de los jugadores se manejarán arboles binarios balanceados o arboles binarios no balanceados.

1.Árboles binarios balanceados: 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.

2. Árboles binarios no balanceados: Árbol binario en el que no se tinen en cuenta el numero de nodos de tanto el lado de recho como el del lado izquierdo de dicho árbol.

Transición de las ideas a los diseños preliminares.

Alternativa 1: Árboles binarios balanceados.

* En el peor caso, la búsqueda necesita O(log n).
* La inserción puede necesitar reorganizar todo el árbol, O(n).
* Para todo nodo, la cantidad de nodos de su subárbol izquierdo difiere como máximo en 1 de la cantidad de nodos del subárbol derecho.
* Es un árbol binario de búsqueda, con una condición de balanceo más débil que hace que no sea tan costoso el proceso de balancear un árbol.
* Para todo nodo, la altura de sus subárboles difiere como máximo en 1.

Alternativa 2: Árboles binarios no balanceados.

* si el árbol crece o decrece descontroladme, el rendimiento puede disminuir considerablemente.
* El caso más desfavorable se produce cuando se inserta un conjunto de claves ordenadas en forma ascendente o descendente.
* el número promedio de comparaciones que se deben realizar para localizar una determinada clave en un árbol binario de búsqueda con crecimiento descontrolado es N/2, cifra que muestra un rendimiento muy pobre en la estructura.

Evaluación y selección de la mejor solución:

Criterio A: Complejidad.

* [8] complejidad constante.
* [7] complejidad logarítmica.
* [6] complejidad raíz.
* [5] complejidad lineal.
* [4] complejidad nlogn.
* [3] complejidad polinomica.
* [2] complejidad exponencial.
* [1] complejidad factorial.

Criterio B Eficiencia.

* [3] muy eficiente.
* [2] eficiente.
* [1] nada eficiente.

Criterio C: Facilidad de implementación.

* [2] fácil.
* [1] difícil.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Criterio A | Criterio B | Criterio C | Total |
| Alternativa 1 | Peor caso. O(Logn). 7 | Eficiente.  2 | Fácil implementación. 2 | 11 |
| Alternativa 2 | Peor caso. O(n). 5 | Eficiente.  2 | Fácil implementación.  2 | 9 |

Por la evaluación anterior se escoge la alternativa 2, árboles binarios balanceados para aceder y almacenar los datos de los jugadores de baloncesto.

Preparación de informes y especificaciones:

**1.** **Especificación de Requerimientos Funcionales**

|  |  |
| --- | --- |
| **Nombre:** | R1. Ingresar un nuevo jugador. |
| **Resumen:** | Se agrega un único jugador a la aplicación. |
| **Entradas:** | Nuevo jugador a agregar. |
| **Resultado:** | Jugador agregado con éxito en la estructura de datos. |

|  |  |
| --- | --- |
| **Nombre:** | R2. Ingresar múltiples jugadores. |
| **Resumen:** | Se agregan varios jugadores en la estructura de datos. |
| **Entradas:** | Archivo que contiene a los jugadores. |
| **Resultado:** | Jugadores agregados con éxito. |

|  |  |
| --- | --- |
| **Nombre:** | R3. Eliminar Jugador. |
| **Resumen:** | Se elimina un jugador de la estructura de datos. |
| **Entradas:** | Atributo del jugador a eliminar. |
| **Resultado:** | Jugador eliminado de la estructura. |

|  |  |
| --- | --- |
| **Nombre:** | R4. Buscar jugador. |
| **Resumen:** | Se busca un jugador por medio de uno de sus atributos. |
| **Entradas:** | Atributo del jugador que se desea buscar. |
| **Resultado:** | Todos los datos del jugador. |

**Etapa 2: Diseño**

**Definición de tipos abstractos de datos.**

|  |  |
| --- | --- |
| **TAD** | **AVL** |
|  | |
| **Invariantes**  Cada subárbol es un árbol AVL.  Los subárboles de cada nodo difieren en altura como máximo en uno. | |
| **Operaciones**  leftRotate(T,x) 🡪 InterfazMain 🡪 Player  insertFixup(T,x) 🡪 InterfazMain 🡪 Player  transplant(T,u,v) 🡪 InterfazMain 🡪 Player  deleteFixup(T,z) 🡪 InterfazMain🡪 Player | |
| * **Pre:** El árbol(T) existe. el nodo(x) existe y pertenece al árbol(T). * **Post:** El nodo (x) ha sido rotado hacia el lado izquierdo del árbol (T). * **Def:** El método se encarga de rotar un nodo hacia el lado izquierdo del árbol.   **public T leftRotate(T,x)**   * **Pre:** El árbol(T) existe. * **Post:** El nodo (x) ha sido insertado en el árbol(T). * **Def:** El método se encarga de insertar un nuevo nodo(x) en el árbol(T) y de reordenar(balancear) el árbol(T).   **public T insertFixup(T,x)**   * **Pre:** El árbol(T) existe. el nodo(z) existe y pertenece al árbol(T). * **Post:** El nodo (z) ha eliminado del árbol(T). * **Def:** El método se encarga de eliminar un nodo del árbol(T) y reordenarlo(balancearlo).   **public T deleteFixup(T,z)** | |

|  |  |
| --- | --- |
| **TAD** | **ARN** |
|  | |
| **Invariantes**  Todo nodo es rojo o negro  La raíz es negra  Toda hoja (nil) es negra.  Si un nodo es rojo, entonces sus hijos son negros  Cada camino de un nodo a sus hojas descendientes contiene el mismo número de nodos negros. | |
| **Operaciones**  leftRotate(T,x) 🡪 InterfazMain 🡪 Player  insert(T, x) 🡪 InterfazMain 🡪 Player  insertFixup(T,z) 🡪 InterfazMain 🡪 Player  transplant(T,u,v) 🡪 InterfazMain 🡪 Player  delete(T,z) 🡪 InterfazMain🡪 Player | |
| * **Pre:** El árbol(T) existe. el nodo(x) existe y pertenece al árbol(T). * **Post:** El nodo (x) ha sido rotado hacia el lado izquierdo del árbol (T). * **Def:** El método se encarga de rotar un nodo hacia el lado izquierdo del árbol.   **public T leftRotate(T,x)**   * **Pre:** El árbol(T) existe * **Post:** El nodo (x) ha sido insertado en el árbol(T). * **Def:** El método se encarga de insertar un nuevo nodo(x) en el árbol(T).   **public T insert(T,x)**   * **Pre:** El árbol(T) existe. * **Post:** El nodo (x) ha sido insertado en el árbol(T). * **Def:** El método se encarga de insertar un nuevo nodo(x) en el árbol(T) y de reordenar(balancear) el árbol(T).   **public T insertFixup(T,z)**   * **Pre:** El árbol(T) existe. el nodo(z) existe y pertenece al árbol(T). * **Post:** El nodo (z) ha eliminado del árbol(T). * **Def:** El método se encarga de eliminar un nodo del árbol(T).   **public T delete(T,z)** | |

**Diseño de Casos para pruebas unitarias**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entrada | Salida | Descripción de la prueba | Método | Clase |
| 5 | True | El método retorna true si hay un jugador con los puntos buscados en el archivo, retorna false en caso contrario | Escenario1(); | InterfazMain |
| 200 | False |
| ("Arturo",24,"NOB",1,34,54,65,34,30); | True | El método retorna true si el jugador a eliminar se encuentra y ha sido eliminado, retorna false si no se encuentra. | Escenario2(): |
| ("Platón",700,"COL",1,34,54,65,34,30); | False |
| ("Arturo",24,"NOB",1,34,54,65,34,30) | True | Retorna true si el jugador se encuentra registrado y posteriormente se | Escenario3(); |
| ("Sócrates",24,"NOB",1,34,54,65,34,30) | False |