**UNIVERSIDAD ICESI**

*Facultad de Ingeniería*

*Algoritmos y Estructuras de Datos*

*Periodo II de 2018*

**Laboratorio 3 FIBA**

C. Gironza Mamian1,C. Olano Rosas2, N. Salazar1,

*1. Ingeniería telemática, Facultad de ingeniería, Universidad Icesi, Lina Marcela Quintero Villareal, Cali Colombia*

2*Ingeniería de Sistemas, Facultad de ingeniería, Universidad Icesi,* *Norha Milena Villegas Machado, Cali Colombia*

**1. identificación del problema**

El problema consiste en el manejo de datos de los jugadores de Basketball de la FIBA, se requiere crear un programa en el cual se permita acceder a la información de los jugadores a través de diferentes criterios de búsqueda, además se busca poder hacer que esta información sea persistente, el programa debe manejar con facilidad una gran cantidad de información, en total una cantidad superior a 200.000 datos

**1.1. Requerimientos funcionales**

|  |
| --- |
| **R1. Buscar jugador** |
| **Descripción:** se encarga de buscar al jugador en el árbol escogido |
| **Entradas:** el criterio de búsqueda sobre el cual se ha creado el árbol |
| **Salidas:** Una lista de jugadores con ese mismo criterio |



|  |
| --- |
| **R2. Generar memoria secundaria** |
| **Descripción:** se encarga de leer la base de datos, generar los árboles y mantener la información en memoria secundaria como archivos |
| **Entradas:** Una base de datos |
| **Salidas:** La iniciación de los árboles y objetos en memoria secundaria |

|  |
| --- |
| **R3. Eliminar jugador** |
| **Descripción:** Saca de la base de datos tanto del árbol como en memoria secundaria el jugador que se desea eliminar |
| **Entradas:** Jugador que se quiere eliminar |
| **Salidas:** Se eliminó de ambas memorias al jugador |

|  |
| --- |
| **R4. Modificar jugador** |
| **Descripción:** Permite cambiar los atributos de un jugador tanto en memoria principal como secundaria |
| **Entradas:** Jugador que se quiere modificar |
| **Salidas:** El jugador tiene actualizados sus datos |

|  |
| --- |
| **R5. Agregar jugador** |
| **Descripción:** crea un nuevo jugador y lo agrega a memoria secundaria y principal |
| **Entradas:** Todos los atributos necesarios para crear un jugador: el nombre del jugador, la altura del jugador, el equipo del jugador, índice de eficiencia del jugador, porcentaje de tiros del jugador, porcentaje de rebotes ofensivos del jugador, porcentaje de rebotes defensivos, número de juegos del jugador. |
| **Salidas:** Se agregó un jugador al programa |

**2. Recopilación de la información**

**2.1. preguntas de investigación :**

Para la recopilación de la información se han planteado las siguientes preguntas de interés:

a. ¿Qué es FIBA?

b. ¿Cuales son las características más relevantes del los jugadores de Basketball?

c. ¿Existen otros programas, plataformas, paginas, etc , que manejan las estadísticas de los jugadores de este deporte ?

d. ¿Cómo organiza FIBA a sus jugadores (¿qué categorías existen dentro de este juego?)?

**2.2.Resultados de la Búsqueda:**

a. ¿Qué es FIBA?

Federación Internacional de Baloncesto(FIBA), es el organismo internacional que se dedica a la vigilancia y regulación de las normas de baloncesto mundial (wikipedia, 2018).

b. ¿Cuales son las características más relevantes del los jugadores de Basketball?

el basketball es un deporte de equipo en el cual hay dos bandos cada uno con cinco jugadores, los cuales juegan en periodos llamados cuartos de diez (es decir 40 minutos de juego divididos en 4 partes, sin contar los 6 minutos que se dan entre cada periodo), el objetivo del juego es ganar anotando la mayor cantidad de canastas posibles (esto consiste en hacer pasar la pelota por un aro que se encuentra a 3,05 m sobre el nivel del suelo)

**3. Búsqueda de Ideas Creativas**

1. Pensar simple teniendo como soluciòn de problema un solo árbol de jugadores ejecutándose en memoria principal, para poder hacer la búsqueda, de cualquier jugador, se recorrerán todas las características del jugador.
2. Tener un solo árbol pero que use el hash visto en la anterior unidad, y asì mantener el árbol con la menor cantidad de nodos posibles, este árbol no sería de jugadores sino de colecciones.
3. Tener un árbol de índices generados por el hash de cada jugador, pedirle al usuario todas las características del jugador que quiere buscar, hacerle hash a esa entrada e ir directamente al índice para poder decirle si el usuario existe o no.
4. Tener 4 árboles, uno por cada rubro estadístico, cada nodo de los árboles tendrá una relación con una colección de jugadores que tengan el mismo rubro que se busca.
5. Una solución creativa pero que no aplica los temas de esta unidad sería usar una tabla hash, pues esta tiene una complejidad de O(logn), esta puede tener dos variantes, o bien podría tener collecion de jugadores o una colección de índices que abran un archivo con las otra información del jugador.
6. Usar cuatro árboles que funcionen por medio de índices para acceder a jugadores almacenados en memoria secundaria y en el momento de buscarlos, agregarlos a memoria principal.

**4. Diseños Preliminares**

para el orden de los jugadores se han fijado cinco atributos los cuales son:

a. Name : el nombre del Jugador

b. Age: la altura del jugador

c. Team: el equipo del jugador

d. Per: indice de eficiencia del jugador

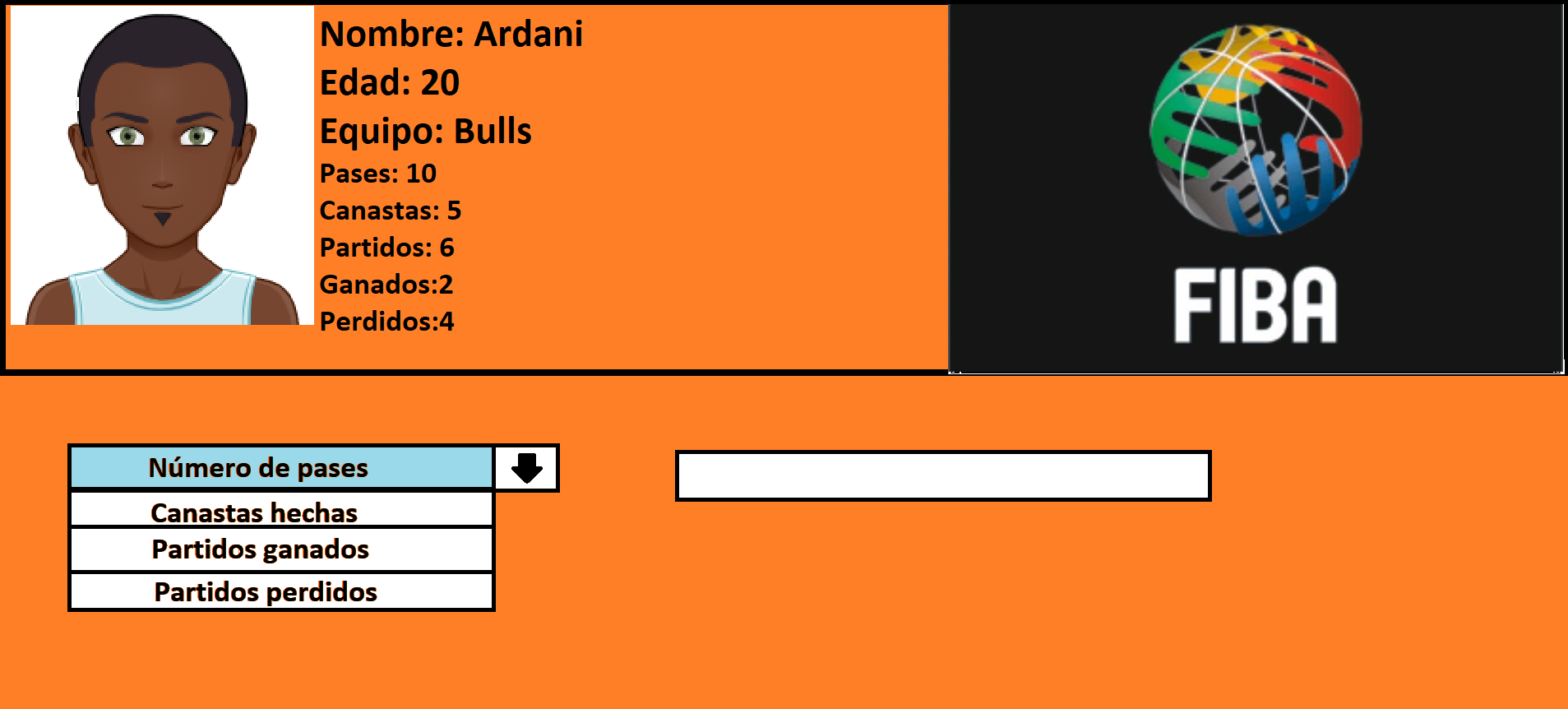
e. Ts%: porcentaje de tiros del jugador

f. Orb%: porcentaje de rebotes ofensivos del jugador

g. Derb%: porcentaje de rebotes defensivos

h. G: número de juegos del jugador

**4.1. Diseño preliminar de la Interfaz de Usuario**



*Imagen.1. Diseño preliminar de la interfaz de usuario*

Para la interfaz de usuario se pensó en el diseño de la Imagen.1 la cual muestra un panel de con las opciones de búsqueda y otro con los resultados de la misma

**4.2 Filtro de los soluciones creativas**

1. Pasa por su simplicidad en el manejo del tema.

5. Por el uso de una estructura de datos conocida que tiene una complejidad muy parecida a un arbol binario.

6. Por el manejo del tema de árboles y usar la estrategia de divide y vencerás.

**5. Evaluación de las Soluciones**

evaluación de 1 a 5 siendo uno muy malo y cinco muy bueno.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Relevancia en el tema | Complejidad temporal | Complejidad espacial |
| Soluciòn 1 | 4 | 1 | 1 |
| Soluciòn 5 | 1 | 5 | 5 |
| Soluciòn 6 | 5 | 5 | 3 |

**Solución 1:** 6 pts: usa el tema de árboles pero para buscar, pero la búsqueda es aproximadamente O(nlogn) y espacial serían n objetos de tipo jugador.

**Solución 5:** 11 pts: no es del tema de árboles binarios, su complejidad es de O(logn) y su complejidad espacial sería una tabla hash.

**Solución 6:** 13 pts: usa el tema de árboles y usa el método del experto su complejidad temporal es O(logn) y su complejidad espacial son 4 árboles binarios de índices

Por la evaluación anterior tenemos que la solución más viable para la implementación del problema es la Solución 6.

**6. especificaciones**

**6.1.TAD :**

|  |
| --- |
| TAD NodeABB |
| NodeAbb <next, key, father, left, right> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * search (String key): --> NodeABB * getMax(): --> NodeABB * getSucessor(): --> NodeABB * compareTo(NodeABB): --> NodeABB |

|  |
| --- |
| Search (String key) |
| “busca un objeto en elarbol”  {Pre: key != null & key != ’ ’}  {post: el odjeto u objetos con las keys iguales sera retornado (en caso de ser mas de uno se retornara la lista )} |

|  |
| --- |
| getMax() |
| “retorna el maximo valor”  {Pre: root != null (el arbol ha sido inicializado)}  {post: el mayor de todos los nodos ha sido retornado} |

|  |
| --- |
| getSucessor() |
| “retorna el menor de los hijos mayores ”  {Pre: root != null (el arbol ha sido inicializado)}  {post: el sucesor del nodo de refencia es devuelto} |

|  |
| --- |
| comparateTo(NodeABB o) |
| “compara dos nodos del arbol deacuerdo a su llave”  {Pre: root != null (el arbol ha sido inicializado) o.key != null y o.key != ’ ’ }  {post: determina cual de los dos nodos comparados es mayor o menor } |

|  |
| --- |
| TAD NodeAVL |
| NodeAVL <next, key, father, left, right, balanceFactor> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * calculateBalanceFactor(): --> NodeAVL |

|  |
| --- |
| calculateBalancefactor () |
| “calcula el factor de balanseo del nodo segun la altura de sus sudarboles”  {Pre: no aplica}  {post: el factor de balance ha sido calculado} |

|  |
| --- |
| TAD NodeBR |
| NodeBR <next, key, father, left, right, color, uncle, grandFather> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * repaint |

|  |
| --- |
| repaint() |
| “repinta o cambia el color de los nodos por el color apuesto ”  {Pre: no aplica}  {post: los nodos repiendados ivierten su color } |

|  |
| --- |
| TAD ABBTree |
| ABBTree <root, size> |
| Invariante  -     ∀ |
| Operaciones Primitivas:   * addNode(NodeABB n): --> ABBTree() * leftRotate(NodeABB actual): --> ABBTree() * rightRotate(NodeABB actual): --> ABBTree() * search(NodeABB actual): --> ABBTree() |

|  |
| --- |
| addNode() |
| “añade un nodo nuevo al arbol”  {Pre: no aplica}  {post: el tamaño del arbol aumenta en una unidad } |

|  |
| --- |
| leftRotate(NodeABB actual) |
| “rota hacia la izquierda el nodo ’actual’ ”  {Pre: no aplica}  {post: el arbol ha sido rotado } |

|  |
| --- |
| rightRotate(NodeABB actual) |
| “rota hacia la derecha el nodo ’actual’ ”  {Pre: no aplica}  {post: el arbol ha sido rotado } |

|  |
| --- |
| search(NodeABB actual) |
| “busca al nodo ’actual’ dentro del arbol ”  {Pre: no aplica}  {post: el arbol ha sido rotado } |

|  |
| --- |
| TAD AVLTree |
| ABBTree <root, size> |
| Invariante  -     ∀ AVLTree es un ABB completo |
| Operaciones Primitivas:   * balancedTree(NodeAVL newAVL, NodeAVL father): --> AVLTree |

|  |
| --- |
| balancedTree(NodeAVL newAVL, NodeAVL father) |
| “ balancea el arbol segun el factor de carga de cada nodoAVL”  {Pre: no aplica}  {post: el arbol ha sido balanceado } |

**6.2.Diseño de Casos de Prueba:**

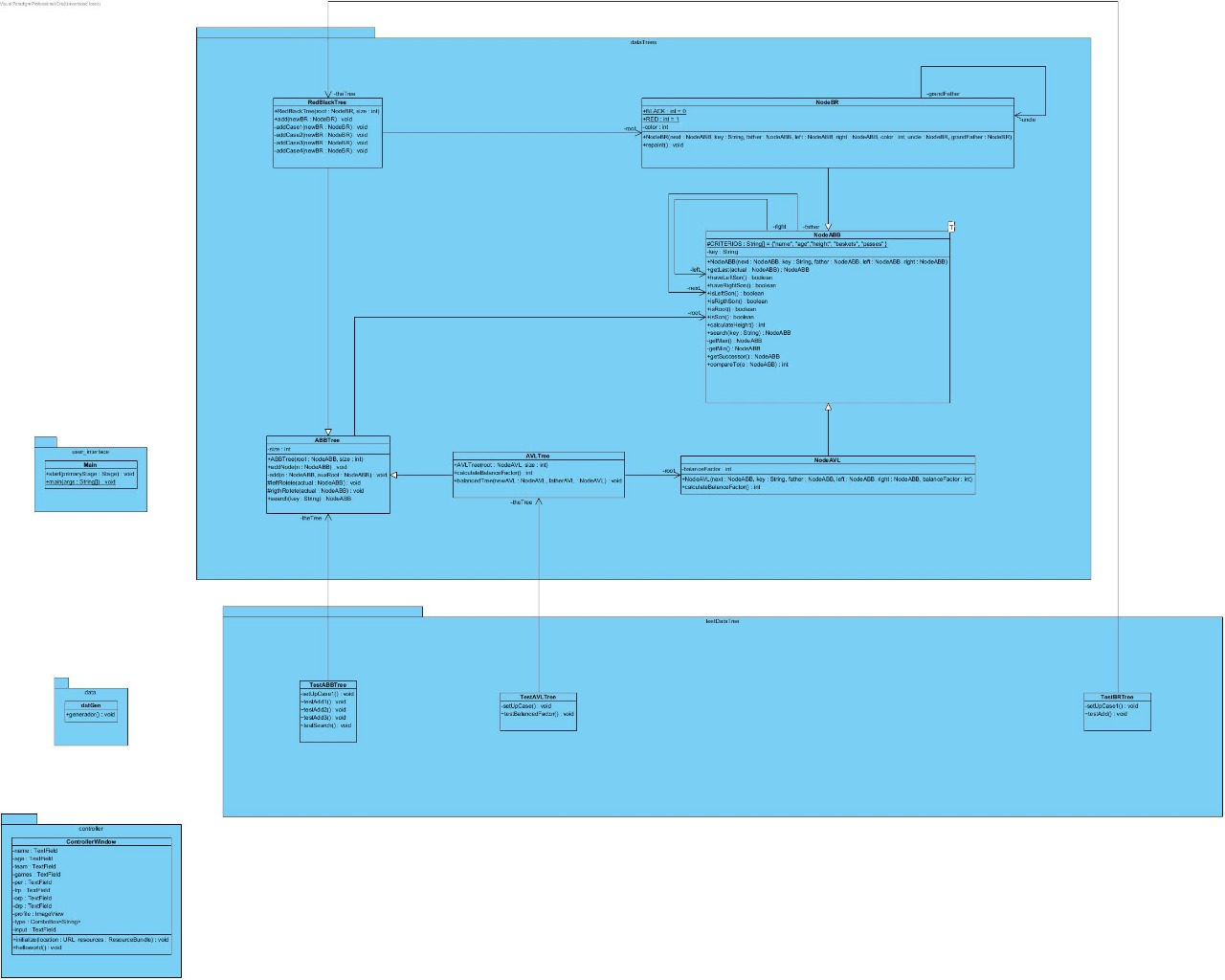
|  |  |  |  |
| --- | --- | --- | --- |
| clase : ABBTree | | método : addNode | |
|
| caso # | Descripción | estado inicial | resultado |
| 1 | se agrega un nuevo nodo al árbol vacío este nodo pasa a ser la raíz | el árbol está vacío | el árbol tiene un nuevo nodo y solamente tiene uno |
| 2 | se agrega un conjunto de nodos con diferentes llaves cuyos valores son: 5,3,7,2,4 | el árbol está vacío | el árbol ahora tiene los nodos indicados |
| 3 | se agrega un conjunto de nodos y se comprueba que aquellos que tengan el mismo valor se guardan en una lista | el árbol está vacío | se agregan los nuevos nodo y se crea una lista con tres elementos cuya llave es igual a cuatro |

|  |  |  |  |
| --- | --- | --- | --- |
| clase : ABBTree | | método : Search | |
|
| caso # | Descripción | estado inicial | resultado |
| 1 | buscar la cantidad de elementos existentes en un árbol | el árbol está lleno | todos los elementos del árbol han sido buscados y encontrados |

|  |  |  |  |
| --- | --- | --- | --- |
| clase : AVLTree | | metodo : testBalancedFactor | |
|
| caso # | Descripción | estado inicial | resultado |
| 1 | determina el balance y rota los nodos de la forma adecuada | el arbol esta desbalanceado a medida que se agregan nuevos nodos y poco a poco se va balanceando | el árbol está lleno y completo por lo cual cumple con las normas para ser un AVL |

|  |  |  |  |
| --- | --- | --- | --- |
| clase : RedBlackTree | | método : testAdd | |
|
| caso # | Descripción | estado inicial | resultado |
| 1 | balancear el árbol rojo y negro | el árbol no se encuentra balanceado | el árbol ha sido balanceado a medida que vayan agregando los nodos |

**6.3. Diagrama de clases**

****

**7..Implementación:**

**Buscar Jugador**

**Construcción**

public NodeABB search( String key ) {

if ( root == null )

return null ;

else

return root.search(key);

}

**Generar Información de jugador**

**Construcción**

public void generador()throws IOException

{

File file = new File("Files/Data/data.csv");

FileReader filereader = new FileReader(file);

file.mkdirs();

BufferedReader reader = new BufferedReader(new FileReader(new File("doc/data.csv")));

String temp="";

String alv="";

while((temp=reader.readLine())!=null)

{

alv+=temp+="separatorPro2017Lol";

}

String players[] = alv.split("separatorPro2017Lol");

for (int i = 0; i < players.length; i++) {

file = new File("doc1/player "+i);

BufferedWriter writer = new BufferedWriter(new FileWriter(file));

String[] criterio = players[i].split(",");

String player = "";

for (int j = 0; j < criterio.length; j++) {

player+=criterio[j]+"\n";

}

writer.write(player);

writer.close();

}

reader.close();

}