**INFORME DE INGENIERÍA**

**LABORATORIO UNIDAD 3**

**MARIA CAMILA LENIS RESTREPO**

**JUAN SEBASTIAN PALMA GARCÍA**

**JAVIER ANDRÉS TORRES REYES**

**ALGORITMOS Y ESTRUCTURAS DE DATOS**

**2018-2**

**INFORME DE INGENIERÍA**

**Paso 1: Identificación del problema**

**Definición del problema**

Manejar información de gran tamaño de manera eficiente a partir de árboles binarios balanceados para un programa en cuestión.

**Justificación**

Es importante realizar esta aplicación ya que en el mundo han existido y existen grandes cantidades de jugadores con sus respectivas estadísticas que los diferencian de cada uno. Si se toma en cuenta que la información de estos jugadores no se elimina, sino que solo incrementa al aparecer nuevos jugadores en el día a día. Así que, si los datos existentes siguen aumentando, el programa actual se va a convertir en obsoleto ya que no esta diseñado para manejar esa gran cantidad de ítems o en este caso de jugadores se torna lento al buscar los datos de ciertos jugadores en base a las estadísticas que se prefieren aplicar. Así que sea tomado la decisión de diseñar un programa que no solo permita una mayor facilidad de búsqueda, sino que permita evaluar la eficiencia de diferentes tipos de estructura y su funcionalidad. También al aplicar este tipo de programa a la FIBA, se puede modificar el mismo para aplicarlo a otros deportes y mejorar la eficiencia de búsqueda para los mismos.

**Requerimientos funcionales**

1. Ingresar datos de los jugadores ya sea de manera masiva (archivos cvs) o por medio de la interfaz. Estos datos deben incluir el nombre del jugador, edad, equipo y 5 rubros estadísticos sean: puntos por partido, rebotes por partido, asistencias por partido, robos por partido y bloqueos por partido.
2. Modificar los datos de un jugador en específico dado su nombre. Si no existe, el programa mostrará un mensaje de advertencia. Los datos por modificar pueden ser nombre, edad, equipo y los 5 rubros estadísticos. Puede modificar uno o varios datos en una sola consulta.
3. Eliminar un jugador dando su nombre. Si este no existe, en el programa se mostrará un mensaje de advertencia.
4. Realizar consultas de jugadores utilizando una categoría. La búsqueda debe realizarse sobre los atributos del jugador, y puede o no ser una igualdad (igual a, mayor o menor que). Se debe retornar el jugador o grupo de jugadores que cumplan con las condiciones dadas, o un mensaje de advertencia si no existe ninguno.
5. Mostrar el tiempo que toma la consulta, ya sea que se haya realizado de dos maneras (árboles no balanceado y balanceados) o solo de una (árboles binarios balanceados) para todas las consultas.

**Requerimientos no funcionales**

* Búsqueda eficiente para categorías de 4 rubros estadísticos, por medio de índices.
  + Dos tipos de consulta deben ser con árboles AVL y los otros dos con árboles rojinegros. Donde cada árbol guardará el valor del índice (atributo) y la posición de este dato en el disco.
  + Deben existir dos consultas para árboles binarios no balanceados. Esto quiere decir que una consulta debe ser realizada por árboles AVL y por ABB y la otra por árboles rojinegros y ABB.
  + La complejidad no puede ser lineal.
* Manejo de software de gran tamaño: El programa deberá contener por lo menos 200000 datos válidos sobre jugadores.

**Paso 2: Recopilación de la información**

**Baloncesto:**

**Sobre rubros estadísticos del baloncesto**

Los rubros estadísticos más usados para definir las habilidades de un jugador de baloncesto son los siguientes:

|  |  |
| --- | --- |
| Abreviación | Significado |
| #G | Número de partidos jugados |
| MPG: Minutes per Game | Minutos por partido |
| PTS: Points | Puntos anotados |
| FGM-FGA: Field Goals Made- Attempted | Tiros logrados/tiros intentados |
| FG%: Field Goal Percentage | Porcentaje de anotaciones |
| 3PM-3PA: 3pointer made-Attempted | Triples logrados/intentados |
| 3P%: 3 Pointer Percentage | Porcentaje de triples |
| FTM-FMA: Free Throws Made-Attempted | Tiros libres anotados/intentados |
| FT%: Free Throws Percentage | Porcentaje de tiros libres |
| PPG: Points per game | Puntos por partido |
| RPG: Rebounds per game | Rebotes por partido |
| APG: Assists per game | Asistencias por partido |
| SPG: Steals per game | Robos por partido |
| BPG: Blocks per game | Bloqueos por partido |

Cabe resaltar que dependiendo de la posición del equipo en la que se encuentre el jugador así mismo pueden variar sus estadísticos. Por ejemplo, un poste no tendrá la misma cantidad de triples que un alero o armador, y un armador no recuperará más rebotes que un poste.

**Sobre bases de datos de jugadores**

Las bases de datos utilizadas como ejemplos en este trabajo fueron tomadas de:

* <https://data.world/jgrosz99/nba-player-data-1978-2016>

**Sobre árboles binarios de búsqueda (no balanceados)**

Es una estructura de datos que se compone de una raíz, nodo inicial del árbol, y dos hijos que se convierten en subárboles a su izquierda y derecha. Para que un árbol binario exista, es necesario implementar el nodo raíz.

Diferencias de un árbol binario lleno y completo

Lleno: hace referencia a cuando los nodos de cada nivel se encuentran con sus dos hijos o con ninguno.

Completo: hace referencia a todas las hojas en el nivel n y n-1 tienen un hijo izquierdo y a su vez uno derecho o ninguno.

El árbol lleno es completo, mientras que el completo no es lleno.

Propiedades del Árbol binario

Sea K un nivel cualquiera del árbol binario, el máximo de nodos en el árbol es de 2k

Sea K la altura del árbol binario, el máximo número de nodos disponibles es de 2k+1-1

Recorridos

* PreOrden: este recorrido consiste en recopilar la información des de el nodo raíz, luego los nodos de la izquierda y por último, los nodos de la derecha.
  + Toma la raíz, luego izquierda y luego derecha.
* InOrden: este recorrido se encarga de recorrer los nodos de izquierda, centro(raíz), y por ultimo el de la derecha.
* PostOrden: Se recorren los datos de derecha a izquierda.

Tipos de Arboles

* Arbol de Expresion
* Arbol Binario de Búsqueda

<https://medium.com/@matematicasdiscretaslibro/capítulo-12-teoria-de-arboles-binarios-f731baf470c0>

**Sobre árboles binarios de búsqueda balanceados**

**Árboles AVL**

Un árbol AVL es un árbol binario de búsqueda que tiene la condición de que la altura de los subárboles presentes en el tienen una diferencia de altura menor o igual a 1,. El árbol AVL tienen la característica de que todas sus alturas son equivalentes a O(log n) lo que significa que mantiene una consistencia gracias a su característica de equilibrio, o en otras palabras, su capacidad de auto balancearse.

Tipos de Rotaciones

* Simples
* Dobles

El Auto Balanceo del árbol consiste en realizar una serie de acciones finitas que tienen la función de modificar la forma de un árbol siempre y cuando se agreguen datos o se eliminen al mismo. Esta acción de auto balanceo consiste una serie de rotaciones simple o dobles que se aplican dentro de los métodos de inserción y eliminación mediante un llamado.

<http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf>

**Árboles Rojinegros**

Un árbol rojo y negro consiste en cuatro características especiales, que definen por obligación si es o no un árbol rojo y negro.

* Es un árbol estricto, todo nodo nulo se toma en las operaciones; todas las hojas del árbol tienen que ser nulas.
* Cada nodo tiene que ser rojo o negro, no otro color
* Todos los nodos nulos tienen que ser negros
* La raíz del árbol obligatoriamente es negra, esto ayuda a simplificar las operaciones

Condiciones para la funcionalidad del árbol rojo y negro

* Todo nodo rojo tiene por obligación dos hijos de color negro
* Todo camino de la raíz hasta una hoja debe de tener la misma cantidad de nodos negros en total

La inserción de nodos en este árbol consiste en la creación de un nodo aparte de color rojo con la información que deseamos que contenga, y se inserta de igual manera a un árbol binario. Luego el método al insertar el nuevo nodo llama al método de rebalanceo de inserción y este se encarga de modificar los colores de la rama para que estas sean consistentes en la altura de nodos de color negro. La eliminación sirve de la misma manera que la de los arboles binarios, solo que se le agrega el método de rebalanceo de eliminación.

El método de rebalanceo de inserción tiene tres casos:

* Caso 1: Nodo hermano del padre (Tio) rojo
* Caso 2: Nodo hermano del padre (Tio) negro y el nodo agregado esta a la derecha del padre
* Caso 3: Nodo hermano del padre (Tio) negro y el nodo agregado esta a la izquierda del padre

El método de rebalanceo de eliminación tiene cinco casos y dos casos triviales:

* Caso 1: Nodo borrado tiene hermano rojo y padre negro
* Caso 2: Nodo borrado tiene hermano negro no nulo, sobrinos negros, padre negro
* Caso 3: Nodo borrado tiene hermano negro no nulo, sobrino negro y padre rojo
* Caso 4: Nodo borrado tiene hermano negro no nulo, sobrino rojo/negro, padre de cualquier color
* Caso 5: Nodo borrado tiene hermano negro no nulo, sobrinos cualquiera/rojo, padre cualquier color
* Casos triviales
  + Nodo borrado es rojo: No requiere ningún ajuste
  + El hijo del nodo borrado es rojo, se le cambia el color a negro

<https://www.infor.uva.es/~cvaca/asigs/doceda/rojonegro.pdf>

**Paso 3: Búsqueda de soluciones creativas**

Para este problema en cuestión se identificaron dos subproblemas: el manejo de la información de los jugadores y la eficiencia de las consultas.

Para el manejo de información de los jugadores hay que tener en cuenta que se debe contar con las de 200000 jugadores y debe accederse a ellos de manera eficiente. Por lo tanto, se tienen las siguientes alternativas:

* Conectar el aplicativo a una base de datos en SQL para acceder a la información de los jugadores.
* Tomar los datos de los jugadores de un dataset y generar aleatoriamente los restantes para cumplir con el objetivo de más de 200000 jugadores.
* La misma que la anterior, pero guardando la información de cada jugador en un archivo de texto.
* Guardar la información de cada jugador en un árbol binario.

Para el manejo de la eficiencia de las consultas se deben cumplir con las restricciones de los requerimientos que son: 4 consultas sobre rubros estadísticos, 2 con árboles AVL, 2 con árboles rojinegros y adicional 2 con ABB para realizar la comparación con los árboles binarios balanceados. En síntesis, las ideas planteadas no pueden salirse del marco de que las consultas deben ser tratadas con árboles, por lo tanto, las ideas propuestas van en torno a la elección del tipo de árbol a usar para cada criterio.

Cada jugador debe contener 5 rubros estadísticos: puntos por partido, rebotes por partido, asistencias por partido, robos por partido, bloqueos por partido. Los 4 atributos para los cuales la búsqueda debe ser eficiente serán los 4 últimos, debido a que los puntos con enteros y los otros cuatro pueden ser decimales y resulta de mayor pertinencia eficiencia en su búsqueda.

**Paso 4: Transición de ideas a los diseños preliminares**

**Manejo de información de gran tamaño:**

Para este problema se ha decidido descartar la primera opción debido a

Entonces quedan las otras tres opciones que serían:

* Tomar los datos de los jugadores de un dataset y generar aleatoriamente los restantes para cumplir con el objetivo de más de 200000 jugadores y manejarlos en un archivo cvs, donde cada línea corresponde a un jugador.
* La misma que la anterior, pero guardando la información de cada jugador en un archivo de texto. Es decir, al inicio del programa se recorre todo el archivo cvs para ir generando los archivos de texto de cada jugador (jugadorx.txt) y guardarlos en un directorio llamado data, y así cada nodo de los árboles tendría el índice a buscar y el valor del jugador para buscarlo directamente en la carpeta data.
* Guardar la información de cada jugador en un árbol binario. Es decir, cada nodo tiene un objeto Jugador que tiene como atributos el nombre, equipo y los 5 rubros estadísticos de jugador.

**Eficiencia de las consultas:**

Los cuatro datos elegidos son de la misma naturaleza y no interfieren de manera significativa en el tipo de estructura que los almacena. Es decir, no hay restricciones para que la consulta con cierto tipo de dato debe realizarse con un árbol es específico. Por lo tanto, la elección de cuáles son los 2 rubros por consultar con árboles AVL, los dos rubros con rojinegro y los 2 rubros con ABB puede ser de manera aleatoria.

Cabe resaltar que de igual forma se debe informar al usuario del tiempo tomado con cada árbol, y en caso de que la consulta sea realizada por dos árboles (uno balanceado y uno no balanceado) debe tenerse la comparación de los tiempos, para así tener una idea de la diferencia entre tener garantizado un O(log n) y tener abierta la posibilidad de que se convierta en O(n).

**Paso 5: Evaluación o selección de la mejor solución (Criterios y selección)**

**Manejo de información de gran tamaño**

Criterio A: Organización de la información. Los datos están ordenados de manera que es fácil entender a quién pertenecen y lo que significan (a qué rubro se refiere).

* [3] Los datos se encuentran separados por índices, como en un diccionario
* [2] Los datos se encuentran en los atributos del jugador.
* [1] Los datos se encuentran enlistados

Criterio B: Rapidez de obtención de la información. Que tan accesible es la información por jugador.

* [3] La información puede ser accedida abiertamente, no se necesita de pasar por otro archivo para llegar a los datos.
* [2] Hay que buscar línea por línea en un mismo archivo hasta llegar a la línea que contiene la información del jugador en cuestión.
* [1] Debe realizarse una búsqueda en otro árbol binario de búsqueda.

Criterio C: Persistencia de la información.

* [3] La información sigue estando disponible cuando se vuelva a ejecutar el programa
* [1] La información debe generarse de nuevo cuando se vuelve a ejecutar el programa.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Criterio A | Criterio B | Criterio C | Total |
| Archivo cvs | 1 | 2 | 3 | 6 |
| Archivos de texto | 3 | 3 | 3 | 9 |
| Nodos | 2 | 1 | 1 | 4 |

La manera en que se van a almacenar los datos es por medio de archivos de texto. De esta forma, se pretende que haya una carpeta data donde se encuentren los archivos de texto organizados por índices, y este índice hará parte de los nodos de cada árbol de búsqueda, donde Key va a ser el dato como tal, el rubro estadístico, y Value será la posición en los archivos de texto donde se encuentra el resto de la información del jugador (índice).

Adicionalmente, para hacer más rápida la inicialización del programa serán persistentes también los árboles binarios, para que así la creación de los 200000 jugadores solo debe realizarse una vez.

**Eficiencia de las consultas**

Para los 4 rubros estadísticos elegidos se tienen las siguientes opciones:

|  |  |  |  |
| --- | --- | --- | --- |
| Rubro/Árbol | ABB | AVL | RBTree |
| Rebotes por partido | X |  | X |
| Asistencias por partido |  |  | X |
| Robos por partido | X | X |  |
| Bloqueos por partido |  | X |  |

Por lo tanto, para cumplir con los requerimientos se usarán ABB y árboles rojinegros para el manejo del rubro de rebotes por partido, y así sucesivamente. Para los ítems de nombre, edad y equipo se realizará una búsqueda lineal por los archivos de texto.

**Paso 6: Preparación de informes**

**TAD Árboles Binarios de búsqueda**

**TAD Árboles AVL**

**TAD Árboles rojinegros**

**Diseño del diagrama de clases de la solución**

**Diseño de casos de pruebas unitarias**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 1: | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Diseño del diagrama de pruebas unitarias**

**Bibliografía**

<http://www.espn.com/nba/statistics/player/_/stat/scoring/sort/points>

<http://sportsmadeinusa.com/baloncesto/nba/estadisticas-baloncesto-4-factores/>

<http://www.espn.com.co/basquetbol/nba/estadisticas>

<https://www.basketball-reference.com/teams/GSW/2018.html>