

Contents

Identificación del problema:	1
Requerimientos:	2
TADS.....	3
Recopilación de Información:.....	8
Alternativas Creativas:.....	9
Elementos relacionados entre las Alternativas creativas escogidas:.....	11
Búsqueda de Soluciones Creativas:	11
Alternativas de solución creativas:.....	12
Ventajas y desventajas:	14
Fase 6. En este paso procedemos a.....	17
Fase 7. Evaluación y selección de la mejor solución	19
Fase 8. Diseño preliminar de cada idea no descartada: (Modelos de Simulación).....	20
Diagramas de clases:	21
Fuentes:	22

Identificación del problema:

Definición del problema:

La FIBA se ve en la necesidad de registrar y almacenar cada jugador profesional del deporte, junto con sus datos de mayor relevancia, en un sistema que le permita realizar diferentes consultas para el análisis de estos.

Causas y Síntomas:

Debido a la avalancha de estadísticas provenientes del deporte, donde, cada cifra representa un dato mínimamente útil, la Federación Internación de Baloncesto (FIBA, por sus siglas en inglés), ha considerado optimo consolidar los datos de mayor relevancia de cada uno de los jugadores profesionales del baloncesto en una aplicación.

Con la falta de disposición de un aplicativo que permita satisfacer esta necesidad, se ven en disposición de contratar un ente externo que pueda elaborar dicho software.

Requerimientos:

Requerimientos:

Nombre	Ingresar datos de forma masiva	#	1
Descripción	La aplicación debe permitir recibir datos por medio un archivo csv, o plano, con muchas entradas. Y debe almacenarlos en memoria secundaria		
Entrada	Archivo plano con los datos del jugador		
Salida	Datos ingresados en memoria secundaria (del disco)		

Nombre	Ingresar datos por medio de la interfaz	#	2
Descripción	La aplicación debe poder recibir los datos ingresados por medio de una interfaz gráfica. Y debe almacenarlos en memoria secundaria.		
Entrada	Datos de los jugadores de forma manual		
Salida	Datos ingresados en memoria secundaria (del disco)		

Nombre	Eliminar datos	#	3
Descripción	La aplicación debe permitir eliminar datos que el usuario requiera, sin alterar los demás datos ya almacenados.		
Entrada	El dato, o datos, que desea eliminar		
Salida	Datos fuera de la memoria secundaria del disco y de las estructuras		

Nombre	Modificar datos	#	4
Descripción	La aplicación debe permitir modificar datos que el usuario requiera, sin alterar los demás datos ya almacenados y verificando las especificaciones del mismo dato.		
Entrada	El dato, o datos, que se desea modificar La información con la que se desea reemplazar la ya existente		
Salida	Modificación en el dato o datos.		

Nombre	Consultas de jugadores	#	5
Descripción	La aplicación debe permitir mostrar la información que el usuario necesite dependiendo del criterio que el mismo especifica.		
Entrada	El criterio con que se filtrará la información		
Salida	Los datos consultados		

Requerimientos no funcionales:

Nombre	Disponibilidad 90%	#	1
Descripción	El programa debe estar disponible la mayor parte del tiempo.		
Entrada	Ninguna.		
Salida	Ninguna.		

Nombre	Información en memoria secundaria.	#	2
---------------	---	----------	----------

Descripción	Los datos deben tener un espacio en la memoria del disco y no ocupar la memoria RAM por completo. Esto, debido a que debe leerse muchos datos.
Entrada	Ninguno.
Salida	Ninguno.

Nombre	Consultas eficientes,	#	3
Descripción	Para los datos "índices" las consultas deben ser eficientes.		
Entrada	Consulta.		
Salida	Respuesta en tiempo no lineal.		

Nombre	Primera versión del software	#	4
Descripción	La aplicación debe incluir, O como mínimo, los siguientes datos por jugador: nombre, edad, equipo y 5 rubros estadísticos (puntos por partido, rebotes por partido, asistencias por partido, robos por partido, bloqueos por partido).		
Entrada	Nombre, edad, equipo y 5 rubros estadísticos por cada jugador		
Salida	Dicha información almacenada en el programa por cada jugador		

TADS

Nombre: Tree Red and Black
<p>Dibujo:</p> <pre> graph TD 13((13)) --> 8((8)) 13 --> 17((17)) 8 --> 1((1)) 8 --> 11((11)) 1 --> NIL1[NIL] 1 --> 6((6)) 11 --> NIL2[NIL] 11 --> NIL3[NIL] 17 --> 15((15)) 17 --> 25((25)) 15 --> NIL4[NIL] 15 --> NIL5[NIL] 25 --> 22((22)) 25 --> 27((27)) 22 --> NIL6[NIL] 22 --> NIL7[NIL] 27 --> NIL8[NIL] 27 --> NIL9[NIL] </pre>
<p>Invariante:</p> <ol style="list-style-type: none"> 1 Todo nodo es rojo o negro. 2 La raíz es negra. 3 Toda hoja (nil) es negra. 4 Si un nodo es rojo, entonces sus hijos son negros. 5 Cada camino de un nodo a sus hojas descendientes contienen el mismo número de nodos negros. Sea x un nodo del árbol. <p>Si y es un nodo en el subárbol izquierdo de x; entonces $key[y] \leq key[x]$:</p> <p>Si y es un nodo en el subárbol derecho de x; entonces $key[y] \geq key[x]$:</p>

Operaciones Primitivas:

- **ArbolRojoNegro** - \rightarrow **ArbolRojoNegro**
- **insert** Element \rightarrow **ArbolRojoNegro**
- **RBInsertFixUp** NodoArbolRojoNegro \rightarrow **ArbolRojoNegro**
- **RBDeleteFixUp** NodoArbolRojoNegro \rightarrow **ArbolRojoNegro**
- **deleteItem** Element \rightarrow **ArbolRojoNegro**
- **search** Element \rightarrow **NodoArbolRojoNegro**
- **leftRotate** Element-Value \rightarrow **NodoArbolRojoNegro**
- **rightRotate** NodoArbolRojoNegro \rightarrow **Element**

ArbolRojoNegro \rightarrow ArbolRojoNegro

“Contruye el Arbol Rojo y Negro”

Precondiciones: -

Postcondiciones:

ArbolRojoNegro a = \emptyset

ArbolRojoNegro \rightarrow ArbolRojoNegro

“Se encarga de construir el arbol”

Precondiciones: -

Postcondiciones: *ArbolRojoNegro a = \emptyset*

Insert Element \rightarrow ArbolRojoNegro

“Se encarga de aegregar un elemento al arbol”

Precondiciones: El elemento es valido

Postcondiciones: El elemento ha sido agregado correctamente

InsertFixUp NodoArbolRojoNegro \rightarrow **ArbolRojoNegro**

“Se encarga de organizar el árbol después de haber ingresado un elemento”

Precondiciones: El árbol debe de estar balanceado

Postcondiciones: El árbol esta correctamente organizado

DeleteFixUp NodoArbolRojoNegro
 \rightarrow **ArbolRojoNegro**

“Se encarga de reorganizar el árbol despues de haber elimiando el elemento”

Precondiciones: El árbol no se encuentra balanceado

Postcondiciones: El árbol se encuentra correctamente organizado

deleteElement Element \rightarrow **ArbolRojoNegro**

“Se encarga de eliminar un elemento en el arbol”

Precondiciones: El elmento que va a ser borrado es valido

Postcondiciones: Elimina el elemento en el arbol

search Element \rightarrow **NodoArbolRojoNegro**

“Se encarga de buscar un elemento dentro del arbol”

Precondiciones: El elemento debe de estar en el arbol

Postcondiciones: El elemento se retorna si se encuentra dentro del arbol

leftRotate Element-Value \rightarrow
NodoArbolRojoNegro

“Rota el nodo del árbol hacia la izquierda”

Precondiciones: El nodo debe de estar en el arbol

Postcondiciones: Arbol con el nodo rotado

rightRotate NodoArbolRojoNegro → Element
“Rota el nodo del árbol hacia la derecha”
Precondiciones: El nodo debe de estar en el arbol
Postcondiciones: Arbol con el nodo rotado

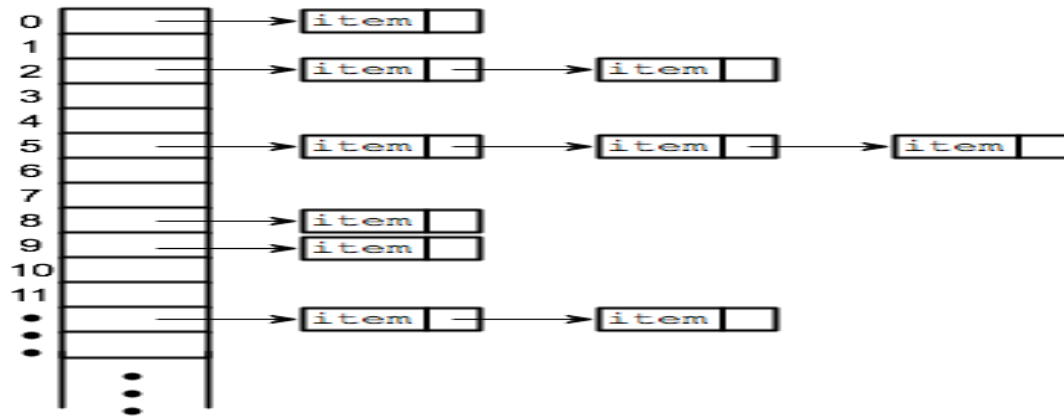
Nombre: ARBOL AVL
<div> Dibujo: </div>
Invariante: X y C son disyuntos, todos los elementos de X son menores que Y, todos los elementos de C son mayores que Y, x y c son ordenados $ altura(x) - altura(c) < 1$, x y c son AVL
Operaciones Primitivas: <div> - IniciarAVL → AVL (Constructora) - InsertarAVL: AVLxTipoAVL → AVL (Modificadora) - </div>

AVL iniciarAVL (Void)
“Crea un Arbol AVL vacio”
Precondiciones: -
Postcondiciones: $IniciarAVL = \emptyset$

AVL insertarAVL (AVL a, TipoAVL elemento)
“Agrega un elemento en el árbol AVL”
Precondiciones: A es diferente de null
Postcondiciones: $IniciarAVL = AU(Elemento)$

Nombre: HASH TABLE

Dibujo:



Invariante:

{ Inv : Para cada key, la función hash $h(k)$ devuelve la posición en que se encuentra ubicado en la tabla hash }

Operaciones Primitivas:

- **Hast Table** - \rightarrow **HashTable**
- **Put Key x Value** \rightarrow **HashTable**
- **getKey** \rightarrow Value
- **Remove key** \rightarrow value
- **isEmpty HashTable** \rightarrow booeIan

HashTable \rightarrow HashTable

“Construye una tabla Hash Vacía”

Precondiciones:

Postcondiciones:

$HashTable\ h = \emptyset$

Put Key x Value \rightarrow HashTable

“Inserta en la tabla una clave asociada a un valor”

Precondiciones: key es una llave valida

Postcondiciones:

La pareja insertada en la tabla hash

get Key \rightarrow Value

“recupera el valor asociado a una clave de una tabla hash”

Precondiciones: pareja(key, value) que esta en la tabla con algún valor

Postcondiciones:

El valor retornado correctamente

Remove Key \rightarrow Value

“Recupera el valor del elemento en la parte de enfrente de la cola”

Precondiciones: key es una llave valida

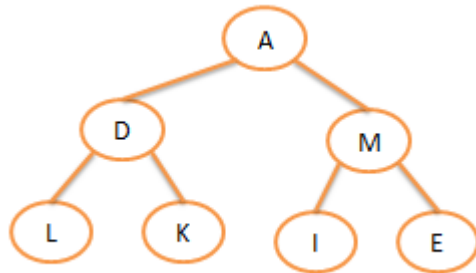
Postcondiciones:

Remueve el elemento al que le corresponde esa llave

IsEmpty HashTable → Boolean
“Determina si la tabla hash está vacía o no”
Precondiciones: tablahash h
Postcondiciones: Verdadero si $q = \emptyset$ Falso si q es diferente de \emptyset

Nombre: ARBOL BB

Dibujo:



Invariante:

{ D y M con sus respectivos nodos son disyuntos, todos los elementos de D son menores que A, todos los elementos de M son mayores que A, D y M son ordenados }

Operaciones Primitivas:

- **IniciarArbolBinario** - \rightarrow ArbolBinario (Constructora)
- **InsertarArbolBinario** ArbolBinario x TipoAO \rightarrow ArbolBinario (Modificadora)
- **EliminarArbolBinario** ArbolBinario X TipoAO \rightarrow ArbolBinario (Modificadora)

ArbolBinario IniciarArbol (Void)
“Crea un Arbol Binario el cual es vacio”
Precondiciones: -
Postcondiciones: $IniciarArbolBinario = \emptyset$

ArbolBinario EliminarArbol (ArbolBinario a, TipoAO element)
“Elimina un elemento de un árbol ”
Precondiciones: Elemento \in a, $a=A$
Postcondiciones: $IniciarArbolBinario = A - ELEMENT$

ArbolBinario InsertarArbol (ArbolBinario a, TipoAO element)
“Agrega un elemento a un árbol binario ordenado”
Precondiciones: Elemento \in a, $a=A$
Postcondiciones: $IniciarArbolBinario = A \cup ELEMENT$

Recopilación de Información:

Baloncesto:

baloncesto o básquetbol (del inglés basketball) es un deporte en el cual compiten dos equipos de cinco jugadores cada uno. El objetivo es introducir la pelota (balón) en el aro (cesta o canasta) del equipo contrario, que se encuentra ubicado a 3,05 metros de altura. Por eso, el baloncesto suele ser jugado por personas de gran estatura.

Arboles Rojos y Negros

Un árbol Rojo-Negro es una representación en árbol binario de un árbol 2-3-4. Los hijos de un nodo en un árbol Rojo-Negro son de dos tipos: Rojos y Negros. Si el hijo ya existía en el árbol 2-3-4 original será Negro, sino será Rojo

Propiedades

es un árbol binario de búsqueda cada camino desde la raíz hasta las hojas tiene el mismo número de hijos negros (esto es debido a que todos los nodos externos en un árbol 2-3-4 están en el mismo nivel y los hijos negros representan los hijos originales) ningún camino desde la raíz a las hojas tiene dos o más hijos rojos consecutivos

Arboles Binarios:

Los árboles binarios son estructuras de datos muy similares a las listas doblemente enlazadas, en el sentido que tienen dos punteros que apuntan a otros elementos, pero no tienen una estructura lógica de tipo lineal o secuencial como aquellas, sino ramificada. Tienen aspecto de árbol, de ahí su nombre.

Un árbol binario es una estructura de datos no lineal en la que cada nodo puede apuntar a uno o máximo a dos nodos. También se suele dar una definición recursiva que indica que es una estructura compuesta por un dato y dos árboles. Esto son definiciones simples. Este tipo de árbol se caracteriza porque tienen un vértice principal y de él se desprende dos ramas. La rama izquierda y la rama derecha a las que también se les conoce como subárboles.

Archivos CSV:

os archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el separador decimal: Chile, Perú, Argentina, España, Brasil...) y las filas por saltos de línea.

El formato CSV es muy sencillo y no indica un juego de caracteres concreto, ni cómo van situados los bytes, ni el formato para el salto de línea. Estos puntos deben indicarse muchas veces al abrir el archivo, por ejemplo, con una hoja de cálculo.

Arboles N-Arios:

Un árbol n-ario es una estructura recursiva, en la cual cada elemento tiene un número cualquiera de árboles n-arios asociados. Estos árboles corresponden a la generalización de un árbol binario. La diferencia radia en que esta estructura puede manejar múltiples subárboles asociados a cada elemento, y no solamente 2, como en el caso de los árboles binarios.

Construcción Árboles N-Arios

Añadir nodos a un Árbol:

- como Hijo más a la Izquierda.
- como Hermano Derecho.

Utilizar directamente en la definición recursiva de Árbol n-ario:

- Entradas: A_1, \dots, A_n Nodo Raiz
- Salida: El nuevo Árbol n-ario

Tipo abstracto de datos: (TAD):

- A. Un conjunto de valores y operaciones asociadas
- B. especificados de manera precisa e independiente de la implementación

Notación:

1. El estado de un TAD viene dado por el número de operaciones dado sobre el.
2. La definición de las operaciones suele darse mediante axiomas de reglas lógicas

Un tipo de dato abstracto (TDA) o Tipo abstracto de datos (TAD) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo. Algunos ejemplos de utilización de TDA's en programación son:

Conjuntos: Implementación de conjuntos con sus operaciones básicas (unión, intersección y diferencia), operaciones de inserción, borrado, búsqueda...

Árboles Binarios de Búsqueda: Implementación de árboles de elementos, utilizados para la representación interna de datos complejos. Aunque siempre se los toma como un TDA separado son parte de la familia de los grafos.

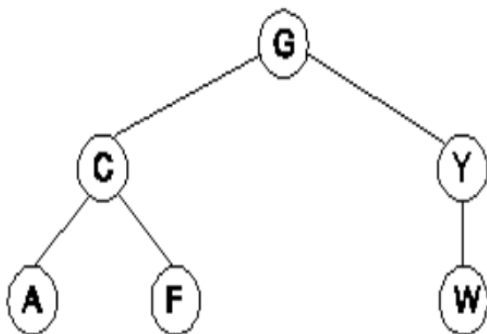
Pilas y Colas: Implementación de los algoritmos FIFO y LIFO.

Grafos: Implementación de grafos; una serie de vértices unidos mediante una serie de arcos o aristas.

.bb

Alternativas Creativas:

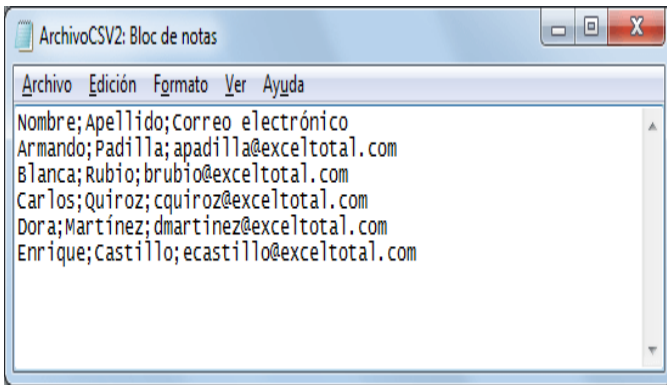
Árbol Binario para los jugadores:



A los árboles ordenados de grado dos se les conoce como árboles binarios ya que cada nodo del árbol no tendrá más de dos descendientes directos, los cuales se pueden llamar izquierda o derecha. Las aplicaciones de los árboles binarios son muy variadas ya que se les puede utilizar para representar una estructura en la cual es posible tomar decisiones con dos opciones en distintos puntos. Esta estructura de datos nos va a facilitar un poco a la hora de introducir los datos al Sistema, ya que nos estará organizando, tanto los enteros como los números de coma flotante en una jerarquía de un árbol.

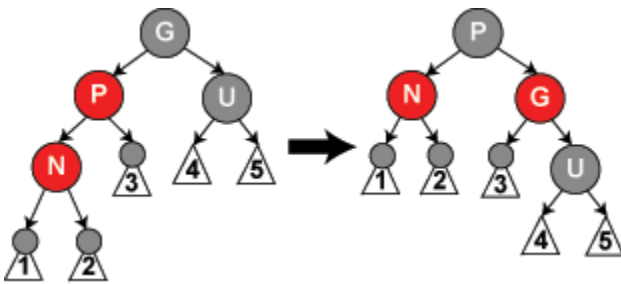
En este proyecto utilizaremos varias alternativas creativas, entre varias de ellas encontramos el uso de los árboles como herramienta indispensable a la hora de hallar una solución al problema planteado. En este caso utilizaremos un árbol binario en el cual agregaremos a los jugadores a la plataforma de Baloncesto. Los jugadores irán agregándose de una manera ordenada.

Archivos CSV:



Los archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el separador decimal: Chile, Perú, Argentina, España, Brasil...) y las filas por saltos de línea. Por otra parte haremos uso de los archivos CSV, ya que estaremos manejando grandes volúmenes de información y esto nos llevaría a ingresar a los jugadores por medio de estos archivos, (Los jugadores en esta etapa no estarán ordenados).

Árbol Rojo y Negro:

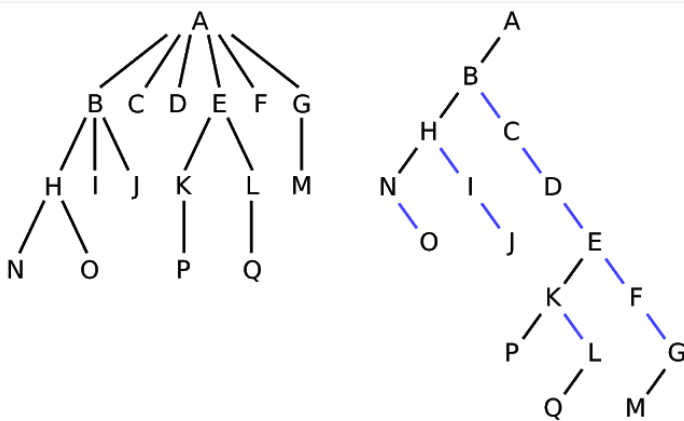


Árbol binario estricto (los nodos nulos se tienen en cuenta en la definición de las operaciones \Rightarrow todo nodo hoja es nulo)

- Cada nodo tiene estado rojo o negro
- Nodos hoja (nulos) son negros
- La raíz es negra (esta condición se impone para simplificar algunas operaciones)

Se debe de implementar un árbol rojo y negro para los jugadores, ya que esto nos permitirá agregar un nuevo jugador, encontrarlo o ya sea eliminarlo del árbol rojo y negro, teniendo en cuenta las reglas de ordenamiento planteadas para esta clase de árboles.

Árbol N-Ario:



Es una estructura de datos donde cada nodo posee un número indeterminado de hijos. es una estructura recursiva, y corresponde a la generalización de un árbol binario de cuyos nodos pueden desprenderse múltiples árboles binarios. las reglas que aplican a los árboles binarios pueden ser fácilmente transpoladas a los árboles n-arios así como los consejos base.

Este árbol lo vamos a utilizar a la hora de agregar el puntaje a los jugadores, en este caso estamos hablando de los puntos anotados por partido, los rebotes hechos en cada partido, asistencia por partidos, robos y bloqueos. Nos servirá de mucha ayuda a la hora de buscar elementos dentro del árbol.

Recursividad:

```
(define (factorial x)
  (if (= x 0)
      1
      (x * (factorial (- x 1)))))
```

Ya hemos visto algunos ejemplos de funciones recursivas. Una función es recursiva cuando se llama a si misma. Una vez que uno se acostumbra a su uso, se comprueba que la recursión es una forma mucho más natural que la iteración de expresar un gran número de funciones y procedimientos.

Recordemos el ejemplo típico de función recursiva, el factorial: La formulación matemática de la recursión es sencilla de entender, pero su implementación en un lenguaje de programación no lo es tanto. El primer lenguaje de programación que permitió el uso de expresiones recursivas fue el LISP.

Elementos relacionados entre las Alternativas creativas escogidas:

Una de las relaciones que Podemos encontrar entre las soluciones creativas es que todas tratan de organizar elementos dentro de una estructura de datos, todas ellas están pensadas para la necesidad de satisfacer los problemas de ordenamiento a la hora de buscar, encontrar o eliminar un jugador de baloncesto. Las practicas que se realizan en cada una varían demasiado entre cada alternativa. Pero algo en común de todas las alternativas escogidas es que podemos manejar un gran volumen de datos a la hora de modificar, agregar, etc. Estos nos facilitaría mucho a la hora de la implementación algorítmica. Las alternativas escogidas son compatibles con la solución del problema, lo que lo hace algo no complejo de tratar

Búsqueda de Soluciones Creativas:

método generador de ideas:

Para desarrollar bien las alternativas creativas usamos el método de lluvia de ideas. Primero, dividimos el problema en tres categorías: entradas, procesos, salidas. Luego, en cada categoría, enlistamos ideas abstractas que nos sirven como sustantivos para la siguiente etapa. Categorías como “procesos” tiene ya predefinidas las alternativas investigadas en el paso anterior de recolección de información. Con las categorías de entrada y salida lo que buscamos es buscar componentes que nos permitan darnos una idea de como recibir y mostrar la información. A continuación, cada componente enlistado en cada categoría es indexado. Por ejemplo, las ideas que salieron en la categoría de entrada están enumeradas; las categorías en proceso están ordenados alfabéticamente y en la categoría de salidas están enlistadas con números romanos.

1. Archivos CSV	A. Arbol Rojo y Negro	I) Jlist
	B. Arbol Binario	II) Campo de Texto
	C. Arbol N-Ario	III) Archivo
2. Campo de texto	D. Estrucutras Recursivas	IV) Ventana Emergente
	E. Tablas hash	

Una vez teniendo llena esta tabla, cada uno procede a escribir combinaciones tomando componentes de cada categoría. Decidimos poner dos condiciones, la primera es solo tomar un componente de la categoría de entrada, y tres, diferentes, de la categoría de procesos. De esta forma, nos aseguramos tener un solo componente de entrada y 3 elementos de la categoría de procesos. Con respecto a la salida decidimos seleccionar más de uno pues nos daría una general, o alternativas, para mostrar el resultado.

Un ejemplo del procedimiento es el siguiente:

1,a,b,c,II,III esto quiere decir que la idea se arma con el sustantivo 1 de la categoría de entrada, primero, segundo y tercero de la categoría de proceso y el componente nombrado con los números romanos de la categoría de salida.

Con este formato, procedemos a armar oraciones que representen una idea para solucionar nuestro problema. Continuamos seleccionado las 7 ideas postuladas por cada uno, las unimos para mejorarlas y sacamos las 7 ideas que serán postuladas para el siguiente proceso el cual es g.

Alternativas de solución creativas:

Alternativa 1:

El usuario tiene la opción de ingresar los Datos por medio de los archivos planos. Una vez los elementos sean ingresados al sistema el usuario procederá a continuar con la opción requerida. Luego cuando el usuario decida consultar elementos sobre los jugadores de baloncesto, por ejemplo: si el usuario quiere buscar a los jugadores cuyos puntajes hayan sido 10 por partido o más de 20 rebotes por partido, el sistema acudirá a crear el árbol rojo y negro, en el cual se organizaran primero todos jugadores que hayan sido ingresados por medio de los archivos CVS, una vez tengamos el árbol rojo y negro, el usuario ejecutara la opción que más desee, como ya lo habíamos mencionado anteriormente, como consultar los puntajes, los rebotes, etc. Cuando el usuario decida efectuar esta opción se creará otro árbol, pero en este caso será un árbol binario en el cual se guardarán los elementos que este decida buscar en el árbol rojo y negro. Ya sea el puntaje, los rebotes, asistencia por partidos, robos por partido, y bloqueos por partido. Una vez el usuario escoja una de las opciones anteriormente descritas los elementos serán ordenados y mostrados por una lista, la cual se mostrará en la interfaz del programa. Por otra parte, el programa también le permitirá mostrar el tiempo de búsqueda que se tardó en dar una respuesta en las estructuras de datos.

Alternativa 2:

El usuario ingresará los datos por medio de un campo de texto, una vez el usuario indique la opción para ingresar los elementos en el campo de texto, se abrirá una ventana en la cual le pedirá al usuario ubicar los elementos a ingresar al sistema. Una vez los elementos sean ingresados, esta acción hará llamado al algoritmo que se va a encargar de crear el Árbol binario en donde ira agregando los elementos uno por uno. Una vez los elementos sean ingresados completamente en el sistema, también serán ingresados en una tabla hash. Dependiendo de la opción que el usuario escoja se buscaran los elementos ya sea por medio de la tabla hash o en el árbol binario. Las opciones que el usuario podrá consultar son, por ejemplo: los jugadores que hayan obtenido un puntaje por partido de 10 puntos o más de 20 rebotes en el partido. Por otra parte, el usuario también contara con la opción de buscar los elementos de puntos por partido, asistencia de por partido del jugador, robos por partido, bloqueos por partido, etc. Los mismos elementos estarán organizados en dos tipos de estructuras diferentes, pero dependiendo de lo que el usuario desee buscar, se hará ya sea en el árbol binario o en la tabla hash. Una vez el usuario escoja la opción a realizar se mostrará los elementos organizados en un campo de texto. También, el programa le permitirá al usuario ver el tiempo que se demoró en buscar los elementos dentro de las estructuras.

Alternativa 3:

El usuario ingresara los elementos por medio de un archivo CSV, una vez el usuario haya ingresado los elementos, se harán los llamados respectivos en el programa para agregarlos en el sistema, en este caso se hará el llamado respectivo para crear un árbol N-ario, en el cual estarán ubicados todos los elementos respectivos. Una vez los elementos hayan sido agregados, el usuario también tendrá la opción de agregar a otro jugador de baloncesto sin necesidad de ser ingresado por medio de un archivo plano. Solamente se tendrán en cuenta los atributos respectivos a este. Por otra parte cuando el usuario decida utilizar la búsqueda dentro

del árbol N-Ario, ya sea para buscar los jugadores que hayan obtenido un puntaje por partido de 10 puntos o más de 20 rebotes en el partido, puntos por partido, asistencia de por partido del jugador, robos por partido, y bloqueos por partido, se harán llamado a los algoritmos de recursividad, ya que esto permitiría que la búsqueda del elemento no sea de manera lineal, ya que es un gran volumen de elementos que serán ingresados al sistema, se tendrán en cuenta lo que es la recursividad a la hora de la búsqueda o la eliminación de un elementos dentro del árbol. Una vez el usuario haya escogido cualquiera de las opciones anteriormente descritas, el programa le devolverá la respuesta por medio de un archivo que se exportara. En esta alternativa el programa cada vez que el escoja una de las opciones anteriormente descritas le mostrara por medio del archivo el tiempo que se demoró en segundos para obtener una respuesta.

Alternativa 4:

El usuario ingresará los elementos por medio de un campo de texto y cuando los elementos sean ingresados al sistema este procederá a crear una tabla hash con los elementos ingresados. Como los elementos serán de gran volumen, ya que estaremos ingresados demasiados elementos al mismo instante, procedemos a utilizar la estructura de datos de una tabla hash, y también serán organizados por medio de un Árbol rojo y negro. Una vez los datos estén dentro del sistema el usuario tendrá varias opciones en la interfaz lo cual le permitirá conocer más a fondo sobre los jugadores que este con anterioridad ha ingresado. Estas son las opciones que el usuario podrá consultar dentro del sistema: los jugadores que hayan obtenido un puntaje por partido de 10 puntos o más de 20 rebotes en el partido, puntos por partido, asistencia de por partido del jugador, robos por partido, y bloqueos por partido. Una vez el usuario haya escogido la acción que desea hacer esta será mostrada por medio de una ventana emergente. En esta alternativa, cada vez que el usuario escoja una de las opciones anteriormente descritas, este le mostrara por medio de una ventana emergente el tiempo que se demoró a la hora de dar la respuesta.

Alternative 5:

El usuario en este caso ingresara los elementos por medio de un cuadro de texto, una vez los datos sean ingresados estos serán agregados en un árbol rojo y negro con todas sus características correspondientes y al mismo tiempo serán agregados en un árbol AVL con sus respectivas características. Los agregaremos en dos tipos de estructuras de datos porque dependiendo de la opción que el usuario vaya a escoger se escogerá la estructura de datos indicada, nos referimos a la complejidad de tiempo. Una vez los elementos sean agregados al sistema, el programa le permitirá escoger entre las opciones tales como, los jugadores que hayan obtenido un puntaje por partido de 10 puntos o más de 20 rebotes en el partido, puntos por partido, asistencia de por partido del jugador, robos por partido, y bloqueos por partido. EL usuario tendrá la oportunidad de escoger cualquiera de las opciones anteriormente mencionadas. Cualquiera de las opciones anteriormente mencionadas será mostrado por medio de una lista. En esta alternativa el programa le mostrara el tiempo de ejecución que tomo en la acción que escogió el usuario

Alternativa 6:

En esta alternativa el usuario podrá ingresar los elementos por medio de dos manera, un archivo CVS y por un archivo en donde tenga los jugadores separados por espacio. Una vez los elementos sean ingresados al sistema independientemente de que opción haya escogido anteriormente serán guardados en una hash table con el nombre del jugador y el TXT index que le corresponde, luego saca al jugador en un archivo txt con el índice que fue asignado con todos sus atributos correspondientes, luego las estadísticas del jugador se proceden a guardar en los árboles. En el árbol rojo y negro se proceden a guardar los atributos correspondientes en este caso en este árbol se guardaran los atributos estadísticos: FIELD GOALS PERCENTAGE y THREE POINT FIELD GOALS PERCENTAGE, en el árbol AVL se guardaran los atributos FREE THROW

PERCENTAGE y PERSONAL FOULS. Por otra parte en el árbol binario de búsqueda se guardarán todos los atributos anteriormente descritos. Por otro parte, si queremos borrar un elemento este tendrá que ser borrado en el hash map teniendo en cuenta su referencia y también será borrado en los árboles (Su referencia). Cada vez que se inicia el programa se cargarán todos los serializables; el hash map y los árboles.

Por otra parte, el programa también contará con las opciones de búsqueda estadísticas, en las cuales tendremos 4 opciones de búsqueda las cuales son; FIELD GOALS PERCENTAGE, THREE POINT FIELD GOALS PERCENTAGE, FREE THROW PERCENTAGE y PERSONAL FOULS. Estas opciones de búsqueda contarán con dos tipos de opciones, la primera es en donde el usuario podrá escoger un rango de búsqueda y la otra en donde el usuario podrá ingresar un número entero.

En esta alternativa también se contarán con las opciones de búsqueda por nombre, agregar nuevo jugador manual, modificar el jugador y eliminar el jugador. El programa también contará con una lista en donde se mostrarán los elementos buscados de acuerdo al criterio escogido con su respectivo rango o su número entero.

Ventajas y desventajas:

Alternativa 1:

En esta alternativa encontramos que el usuario tendrá la opción de ingresar los datos por un archivo plano. Una de las ventajas de ingresar los elementos por un archivo plano es que este ya viene con unos criterios de lectura y el usuario no tendrá que escribirlos en un cuadro de texto, esto reduciría el margen de error a la hora de la lectura de archivos. Una vez los datos ya sean ingresados al sistema se creará el árbol rojo y negro. Una de las ventajas de usar esta clase de estructuras de datos es que ventajas, todas las operaciones son $O(\log n)$. Se mantienen más balanceados que otras estructuras, y permite organizar un listado de números de manera sencilla. Por otra parte, en las desventajas podemos encontrar que su costo espacial es mayor que el de otros árboles por el uso de nodos centinelas. La rotación para conservar las propiedades que debe cumplir todo árbol rojo-negro, en ciertos casos de la inserción y la eliminación será necesario reestructurar el árbol, si bien no debe perderse la ordenación relativa de los nodos, lo que conllevará a un poco más de tiempo.

Para ello, se llevan a cabo una o varias rotaciones, que no son más que reestructuraciones en las relaciones padre-hijo-tío nieto. En las ventajas que encontramos a la hora de implementar un Árbol binario es que el número de accesos al árbol es menor que en una lista enlazada, por ejemplo, en un árbol lleno que tenga n nodos el camino más largo que hay que recorrer es $\log_2(n+1)$, esto nos facilitaría mucho a la hora de buscar o encontrar un elemento en el árbol. Por otro lado, una de las ventajas de utilizar árbol binario es la simplicidad de comprensión y su gran potencia, favoreciendo la resolución de problemas de manera natural, sencilla y elegante, y facilidad para comprobar y convencer de que la solución del problema es correcta. El principal inconveniente es la ineficiencia tanto en el tiempo como en memoria, dado que la permitir sus usos es necesario transformar el programa recursivo en otro iterativo, que puede utilizar bucles y pilas para almacenar las variables. Por otra parte, a la hora de mostrar los datos nos encontramos con que la lista puede ser muy conveniente ya que nos permitiría mostrar gran volumen de información en ella.

Alternativa 2:

En esta alternativa podemos encontrar que el usuario ingresará los datos por medio de un campo emergente. Una de las ventajas que podríamos encontrar a la hora de implementar esta solución es que el usuario podrá ingresar los datos de una manera concisa y sin gasto de tiempo. También, por medio del campo de texto se le hará saber al usuario como ingresar los datos de una manera correcta reduciendo así el nivel de error. Una vez los datos sean agregados al programa se procederá a la creación de un árbol binario, y podemos encontrar que en esta clase de estructura de datos encontramos desventajas tales como: es la eliminación de la recursividad a la hora de hacer recorridos; si hay una gran profundidad, la recursividad puede llegar a un punto en el que se

acabe la memoria. Por otra parte, las desventajas que podríamos encontrar a la hora de la implementación de este son: La memoria necesaria se toma en tiempo de ejecución y todos los accesos a la información deben hacerse de forma indirecta.

A la hora de la implementación de una tabla hash podemos encontrar también tanto ventajas como desventajas en esta: Nos permiten guardar información de un mismo tipo de dato, La guardan de manera contigua (ósea junta, una al lado de la otra) en memoria. Gracias a lo anterior podemos acceder a cualquier posición que queramos mediante un simple paso, en un tiempo constante sin importar si es el primero o el décimo elemento. También podemos encontrar algunas desventajas a la hora de la implementación de esta: No es dinámico, es decir debemos especificar su tamaño cuando lo declaramos, por lo tanto, es muy muy difícil y tardado añadir o quitar elementos (es más ni siquiera es seguro que haya más espacio libre contiguo en memoria para añadir más elementos). Por otra parte, a la hora de mostrar los elementos, lo hará por medio de un campo de texto, pero este tiene más desventajas ya que por el volumen de información que estaremos manejando no serán mostrados correctamente los elementos y se vuelve algo tedioso para la persona que esté utilizando el programa en lo relacionado a lo visual.

Alternativa 3:

En esta alternativa encontramos que el usuario tendrá la opción de ingresar los datos por un archivo plano. Una de las ventajas de ingresar los elementos por un archivo plano es que este ya viene con unos criterios de lectura y el usuario no tendrá que escribirlos en un cuadro de texto, esto reduciría el margen de error a la hora de la lectura de archivos. Una de las ventajas que encontramos a la hora de aplicar los arboles N-Arios es que consiste en que existen más nodos en un mismo nivel que en los arboles binarios con lo que se consigue que si el árbol es de búsqueda, los accesos a los nodos sean más rápidos. Por otra parte, cuando hablamos de las desventajas nos encontramos con que contienen la mayor ocupación de memoria, pudiendo ocurrir que en ocasiones la mayoría de los nodos no tengan descendientes o al menos no todos los que podrían tener desaprovechándose por tanto gran cantidad de memoria. También, se hará uso de la recursividad y en esta se puede encontrar desventajas y ventajas al mismo tiempo, tales como; –Soluciona problemas recurrentes, son programas cortos y las desventajas es que crea muchas variables y puede necesitar mucha memoria. Por otra parte, cuando el usuario escoja la opción que desee, la respuesta será exportada por medio de un archivo plano. Las ventajas que podríamos encontrar al utilizar esta opción de respuesta es que se organizara la información de una manera concisa y eficaz, pero una de las desventajas que podríamos encontrar es la complejidad de manejo a la hora de la implementación.

Alternativa 4:

En esta alternativa encontramos que el usuario tendrá la opción de ingresar los datos por medio de un cuadro emergente. Una de las desventajas de usar un cuadro de texto como método para ingresar los elementos es que el margen de error sería muy grande. Una vez los datos sean ingresados se procederá a la creación de la tabla hash, las ventajas que podríamos encontrar a la hora de la implementación es que una tabla hash tiene como principal ventaja que el acceso a los datos suele ser muy rápido si se cumplen las siguientes condiciones: Una razón de ocupación no muy elevada (a partir del 75% de ocupación se producen demasiadas colisiones y la tabla se vuelve ineficiente), entre otras desventajas que podríamos encontrar son que si queremos ampliar el espacio de la tabla, el volumen de datos almacenados crece y se trata de una operación costosa, por otra parte, también es la dificultad para recorrer todos los elementos. Se suelen emplear listas para procesar la totalidad de los elementos. Desaprovechamiento de la memoria. Si se reserva espacio para todos los posibles elementos, se consume más memoria de la necesaria; se suele resolver reservando espacio únicamente para punteros a los elementos. Una de las ventajas de usar esta clase de estructuras de datos es que ventajas, todas las operaciones son $O(\log n)$. Se mantienen más balanceados que otras estructuras, y permite organizar un listado de números de manera sencilla. Por otra parte, en las desventajas podemos encontrar que su costo

espacial es mayor que el de otros árboles por el uso de nodos centinelas. La rotación para conservar las propiedades que debe cumplir todo árbol rojo-negro, en ciertos casos de la inserción y la eliminación será necesario reestructurar el árbol, si bien no debe perderse la ordenación relativa de los nodos, lo que conllevará a un poco más de tiempo. Para ello, se llevan a cabo una o varias rotaciones, que no son más que reestructuraciones en las relaciones padre-hijo-tío nieto.

Cuando vayamos a mostrar la respuesta por medio del programa se hará usando una ventaja emergente. En este caso la desventaja es que cuando se vayan a mostrar los elementos no se verán bien visualmente por el volumen de datos que estaremos usando.

Aternativa 5:

En esta alternativa encontramos que el usuario tendrá la opción de ingresar los datos por un archivo plano. Una de las ventajas de ingresar los elementos por un archivo plano es que este ya viene con unos criterios de lectura y el usuario no tendrá que escribirlos en un cuadro de texto, esto reduciría el margen de error a la hora de la lectura de archivos. Las ventajas que encontramos a la hora de usar el árbol rojo y negro es que todas sus operaciones son $O(\log n)$, se mantienen más balanceados que otras estructuras, permite organizar un listado de números de manera sencilla. Por otra parte, dentro de las desventajas que encontraremos a la hora de agregar los datos dentro de un árbol rojo y negro es que su costo espacial es mayor que el de otros arboles por el uso de nodos centinelas. Cuando vayamos a agregar los datos en el árbol AVL nos encontraremos con la ventaja de que su complejidad tanto espacial como temporal es $O(\log n)$, lo que sería de gran ayuda a la hora de usar un gran volumen de datos, como es en este caso. Por otra parte, al utilizar los arboles AVL se nos resultara no tan complejo de implementar como otras estructuras de datos y la facilidad de la búsqueda de elementos dentro de esta estructura de datos no es tan complicada tampoco.

Por otra parte, a la hora de mostrar los elementos se hará por medio de una lista, pero no tendría una buena visualización si no se implementara correctamente, ya que esto haría que la persona que estará visualizando los elementos dentro del programa no entienda correctamente como están organizados los jugadores dentro de el.

Aternativa 6:

En esta alternativa nos encontraremos con ventajas y desventajas. En las ventajas que podemos encontrar a la hora de ingresar los datos por medio de un archivo CVS es que el margen de error va a ser muy pequeño a la hora de compararlo con el método de ingreso de datos vía manual, ya que esto no permitiría que el usuario pueda ingresar un carácter no deseado en el sistema. Por otra parte, a la hora de ingresar los elementos al sistema veremos que estos serán ingresados tanto en un Árbol Rojo y negro, AVL y Binario. Las ventajas que podemos encontrar a la hora de implementar el árbol rojo y negro son el número de accesos al árbol es menor que en una lista enlazada, por ejemplo, en un árbol lleno que tenga n nodos el camino más largo que hay que recorrer es $\log_2(n+1)$, esto nos facilitaría mucho a la hora de buscar o encontrar un elemento en el árbol. Las ventajas a la hora de implementar un árbol AVL son: través de los árboles AVL llegaremos a un procedimiento de búsqueda análogo al de los ABB pero con la ventaja de garantizaremos un caso peor de $O(\log_2 n)$, manteniendo el árbol en todo momento equilibrado, por otra parte a la hora de su implementación no se vuelve tan tedioso como otras estructuras de datos. En las ventajas de usar un Árbol Binario son : La ventaja especial de utilizar un árbol es que se facilita la búsqueda, y cuando vayamos a manejar un gran volumen de datos estos no tardaran tanto en buscar el elemento y retornarlo ya que su complejidad es $O(\log n)$. Las operaciones estadísticas que se van a implementar las cuales son cuadro serán buscadas por medio de las estructuras de datos anteriormente mencionadas y sus respuestas serán mostradas por medio de una lista con sus respectivos tiempos de demora en cada operación.

Fase 6. En este paso procedemos a definir los criterios que nos permitirán evaluar las alternativas de las soluciones anteriormente propuestas y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más

Criterio 1: En el criterio 1 evaluaremos la precisión de la solución que se va a escoger, definiéndola en una escala numérica.

- A) La solución es Exacta—Valor: 2
- B) La solución es Aproximada –Valor: 1

Criterio 2: En el criterio 2 evaluaremos la facilidad que tendrá el usuario al momento de ingresar los datos.

- A) Fácil ----) Valor: 3
- B) Media ---) Valor: 2
- C) Algo Complicada ---) Valor:1

Criterio 4: En el criterio 4 evaluaremos el tiempo utilizado por cada algoritmo.

- A) Constante --) Valor: 1
- B) Logarítmica –) Valor: 6
- C) Lineal --) Valor: 5
- D) Cuadrática --) Valor: 4
- E) Cubica --) Valor: 3
- F) Exponencial --) Valor: 2

Criterio 5: En el criterio 5 evaluaremos el espacio en memoria utilizado por cada algoritmo.

- A) Constante --) Valor: 1
- B) Logarítmica –) Valor: 6
- C) Lineal --) Valor: 5
- D) Cuadrática --) Valor: 4
- E) Cubica --) Valor: 3
- F) Exponencial --) Valor: 2

Criterio 6: En el criterio 6 evaluaremos la facilidad con la que se muestran los datos en el programa.

- A) Fácil ----) Valor: 3
- B) Media ---) Valor: 2

Alternatives	Criterio 1	Criterio 2	Criterio 3	Criterio 4	Criterio 5	Criterio 6
	Precisión de la solución	Facilidad de ingreso de datos	Facilidad de implementación del mundo	Tiempo Utilizado por cada Algoritmo	Espacio de Memoria Utilizado por cada Algoritmo	Facilidad de implementación del algoritmo
Alternative 1	La solución es Exacta—Valor: 2	Cuadro Emergente : 3 Archivos de texto Plano: 2	Medio: 2	Arbol Rojo y negro: $O(\log n)$ 6 Arbol binario: $O(\log n)$ 6	Arbol rojo y negro $O(n)$: 5 Arbol binario: $O(n)$: 5	Arbol Rojo y Negro: 2 Arbol Binario: 3
Alternative 2	La solución es Exacta—Valor: 2	Campo de texto: 3	Medio 2	Arbol binario: $O(\log n)$ 6 Tablas Hash: $O(1)$: 1	Arbol rojo y Binario : 5 Tablas Hash: $O(c)$: 1	Arbol Binario: 2 Tablas Hash: 1
Alternative 3	La solución es Exacta—Valor: 2	Archivos de texto Plano: 2	Medio : 2	Arbol N ario: $O(\log n)$: 6 Arbol Rojo y negro: $O(\log n)$ 6	Arbol N Ario : 5 Arbol rojo y negro $O(n)$: 5	Arbol n Ario: 2 Arbol Rojo y negro: 2

Alternative 4	La solución es Exacta— Valor: 2	Campo de texto: 3	Medio: 2	Tablas Hash: $O(1)$: 1 Arbol Rojo y negro: $O(\log n)$ 6	Tablas Hash: $O(c)$: 1 Arbol N Ario : 5	Tablas Hash: 1 Arbol n Ario: 2
Alternative 5	La solución es Exacta— Valor: 2	Archivo de texto Plano: 2	Medio: 2	Arbol Rojo y negro: $O(\log n)$ 6 ARBOL AVL: $O(\log n)$ 6	Arbol rojo y negro $O(n)$: 5 Arbol AVL: $O(n)$: 5	Arbol Rojo y negro: 2 Arbol AVL: 3
Alternative 6	La solución es Exacta— Valor: 2	Campo de texto: 3	Medio: 2	ARBOL AVL: $O(\log n)$ 6 Arbol N Ario: $O(\log n)$: 6 Arbol Rojo y negro: $O(\log n)$ 6	Arbol AVL: $O(n)$: 5 Arbol N Ario : 5 Arbol rojo y negro $O(n)$: 5	Arbol AVL: 3 Arbol n Ario: 2 Arbol Rojo y negro: 2

Fase 7. Evaluación y selección de la mejor solución

Alternativa 4:

Categoría1(2)+Categoría2(3)+Categoría3(2)+Categoría4(6+1)+Categoría5(5+1)+Categoría6(2+1)+=23

Alternativa 5:

Categoría1(2)+Categoría 2(2)+Categoría3(2)+Categoría 4(6+6)+ Categoría5(5+5)+Categoría 6(3+2)+=33

Alternativa 6:

Categoría1(2)+Categoría 2(3)+Categoría3(2)+Categoría 4(6+6+6)+ Categoría5(5+5+5)+Categoría 6(2+3+2)+=47

En esta fase lo que procedimos a hacer fue evaluar la mejor solución, teniendo en cuenta la tabla de criterios anteriormente definida y descartar las ideas que no alcanzaran el puntaje promedio para pasar. Al final de la evaluación encontramos que las mejores soluciones son la Alternativa 4, 5 y 6, descartando así las otras alternativas. Con base en el puntaje obtenido lograremos escoger la mejor solución entre las tres restantes y así poder implementar con todos sus requerimientos correspondientes. **La mejor solución escogida dependiendo a los criterios de evaluación, es la opción que haya obtenido mayor puntaje, en este caso es la alternativa número 6.**

Fase 8. Diseño preliminar de cada idea no descartada: (Modelos de Simulación)

Alternativa 4:

En esta alternativa encontraremos que el usuario podrá ingresar los elementos por medio de los campos de texto. En esta alternativa el margen de error seria un poco elevado ya que al ingresar un volumen grande de jugadores esto tiende a cometer errores más fáciles. Esta opción permitirá al usuario consultar si dentro del sistema se encuentra el jugador a buscar por medio del campo de texto del nombre, añadirlo, modificarlo y eliminarlo. Por otra parte, el usuario también contara con unas opciones

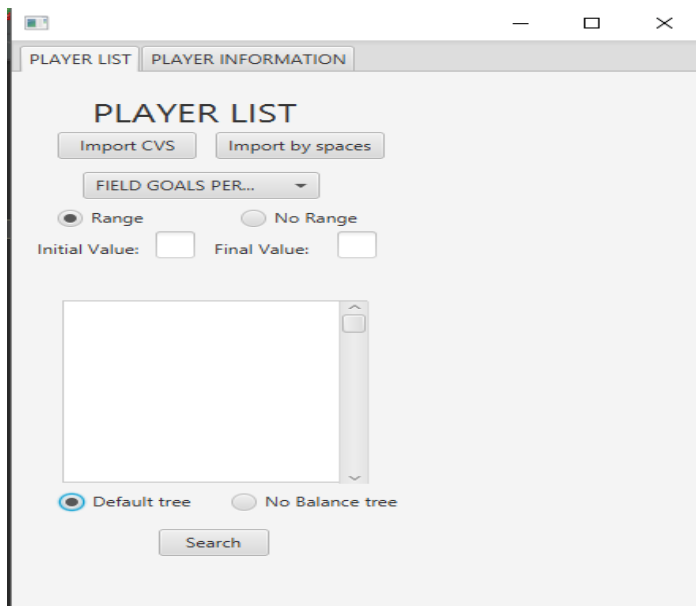
de búsqueda las cuales son: puntos por partido, rebotes por partido, asistencias por partido, robos por partido, bloqueos por partido), en donde en el campo de texto podrá ingresar el numero de robos, asistencia, puntos , etc, que le gustaría buscar dentro del programa. Estos serán mostrados por medio de un campo emergente.

Alternativa 5:

En esta alternativa el usuario contara con varias opciones tales como; buscar un jugador por su nombre, agregar un jugador ingresando la información requerida en los campos emergentes, modificar la información de un jugador encontrado y eliminar un jugador si así lo desea el usuario. Por otra parte, el usuario también contara con unas opciones de búsqueda dependiendo del criterio que este haya escogido, e ingresara el elemento entero con el cual serán buscados todos jugadores en el

árbol que tengan ese entero en su criterio de búsqueda. Por otra parte, la información encontrada será mostrada por medio de una lista que será mostrada en un campo emergente.

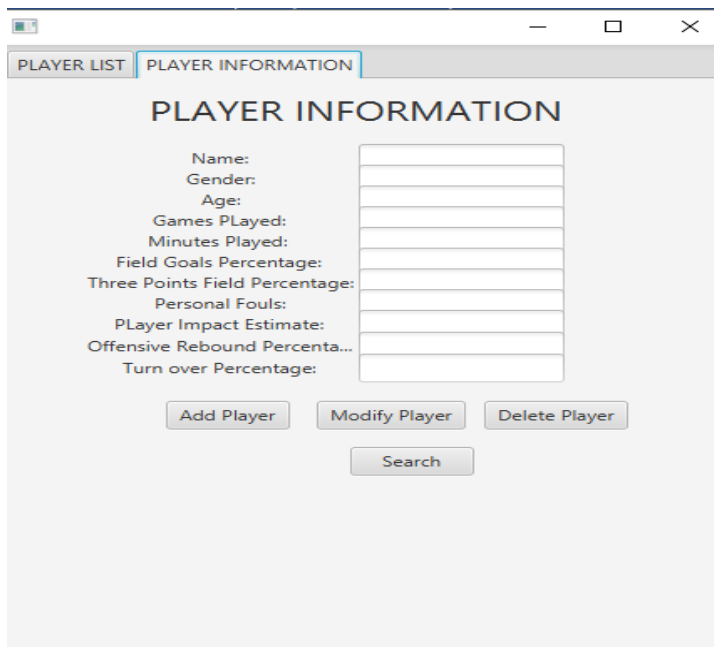
Alternativa 6:



En esta alternativa escogida como mejor solución tendremos varias opciones con las cuales se logrará satisfacer las necesidades del cliente. En esta alternativa los elementos serán ingresados por medio de dos maneras, por medio de un archivo CVS y por medio de un archivo plano en el cual cada elemento será separado por un espacio. Por otra parte, una vez los elementos sean ingresados al sistema, el usuario contara con otras opciones. En el primer Tab podemos ver que tiene un nombre denominado Player List, es aquí donde realizaremos la búsqueda de los componentes estadísticos, estoy hablando de: FIELD GOALS PERCENTAGE, THREE POINT FIELD GOALS PERCENTAGE, FREE THROW PERCENTAGE y PERSONAL FOULS.

Cada uno de los criterios anteriormente mencionado serán buscados en el sistema, pero antes de ello el sistema le preguntara al usuario si desea que el criterio sea buscado por rangos o por un numero entero. En cualquiera de los dos

casos el usuario tendrá que ingresar los números de intervalo o el numero entero que le gustaría buscar por medio del criterio escogido. Cuando el usuario ya haya llenado ese criterio el sistema le preguntara que si le gustaría buscar el criterio seleccionado en el árbol rojo y negro o en un árbol no balanceado binario. Una vez el usuario decida se podrá hacer la búsqueda de los elementos que cumplen con ese criterio.



Por otra parte, el usuario también contará con un segundo tab en el cual se encontrará la información del jugador. En este tab tendremos varias opciones también, tales como; agregar un jugador por medio de los campos de texto , modificar el jugador, eliminarlo y buscar un jugador por el nombre cuyos parámetros serán mostrados en los campos de texto.

Diagramas de clases:

Puedes pulsar en la imagen que te dirigirá al archivo jpg.

Diagrama del paquete de la vista:

https://rua.ua.es/dspace/bitstream/10045/16037/7/ped-09_10-tema3_5.pdfhttps://jarroba.com/tablas-hash-o-tabla-de-dispersion/

<http://www.dccia.ua.es/dccia/inf/asignaturas/LPP/2010-2011/teoria/tema4.html>

<https://prezi.com/c-mqwfu7n2q2/arboles-n-arios/>

<http://estructdatos2incca.blogspot.com/2015/10/arboles-n-arios.html>

https://www.google.com.co/search?q=Arboles+Rojos+Y+negros&rlz=1C1MKDC_enUS772US772&tbm=isch&source=iu&ictx=1&fir=IORLGhBhFFzawM%253A%252CvoqA77sbAVKFLM%252C_&usq=AI4_-kRdX86N0vF1U7ZLXuxdEzYy6NSPZQ&sa=X&ved=2ahUKewjHq8mT1YPeAhVKwFkKHdRVBfYQ9QEwBHoECAUQCg#imgsrc=iZA_yDuDjr0CZM: