**Objetivos**

**Unidad 3: Estructuras recursivas**

OE2.1. Proponer y justificar un diseño para implementar una estructura de datos, siguiendo una metodología y considerando la flexibilidad en los tipos de datos y la complejidad temporal de las operaciones.

OE2.2. Implementar estructuras de datos extensibles y generales utilizando interfaces, herencia y tipos de datos genéricos.

OE2.3. Escribir el invariante de una clase e implementar los métodos necesarios para su verificación utilizando los elementos apropiados del lenguaje.

OE3.3. Utilizar estructuras recursivas de datos para representar la información del modelo del mundo, cuando sea conveniente por razones de eficiencia o por el tipo de problema sobre el cual se trabaja.

OE3.4. Diferenciar el concepto de orden del concepto de balanceo de un árbol, y clasificar los árboles binarios entre árboles ABB, R&B y heaps binarios de acuerdo a sus características de orden y balanceo.

OE3.5. Implementar las estructuras de datos para representar árboles binarios y árboles n-arios

**Resumen**

Desarrollar un programa de computador que sea capaz de recuperar jugadores de acuerdo a la categoría de búsqueda seleccionada y el valor dado para ella. Este programa debe manejar información de gran tamaño, por esta razón se utilizaran arboles binarios de búsqueda balanceados para acceder rápidamente a los datos del jugador y así su búsqueda tome un tiempo de O(log n).

**Método de la ingeniería.**

**Fase 1:** identificación del Problema

Una base de datos o banco de datos es, a grandes rasgos, un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso, el tiempo que se requiere a la hora de manipular la información en una base de datos depende la forma en que está almacenada la información. Las operaciones de añadir, borrar, actualizar, buscar o combinar registros, se llevan a cabo millones de veces al día en bases de datos de diversos tamaños, y debido a la relevancia de estas operaciones se ha desarrollado varios métodos para representar dichas bases de datos de acuerdo con el modelo de administración de datos. Una compañía le ha pedido el manejo de información de gran tamaño que permita la búsqueda de jugadores de forma eficiente.

¿Qué sucede si se desea borrar un valor de un campo en determinado registro?, es necesario borrar todo el registro?

¿Qué sucede si quiero hacer una búsqueda por nombre, pero hay varias personas con nombres iguales?

* Definición del Problema: Implementar arboles binarios de búsqueda balanceados que nos permita acceder rápidamente a los datos del jugador de manera eficiente, para posteriormente buscar los demás atributos si se desea de manera lineal.

Algunas restricciones:

* La información a guardar en los arboles binarios deben de tener un formato específico.
* La base de datos deberá guardarse en memoria secundaria ya que por su tamaño es imposible tenerla en memoria principal. Para llevar a cabo su labor se deberán generar archivos txt en los cuales se guardará cada registro de una manera mencionada posteriormente.
* la complejidad temporal no puede ser lineal, en lo que a la búsqueda de registros se refiere, ya que sería muy lento, teniendo en cuenta los millones de registros guardados en el banco de datos.
* El sistema que se desarrolle debe estar en la capacidad de recuperar registro(s) de acuerdo el campo de búsqueda seleccionado y el valor dado para dicho campo

**Identificación de necesidades o requerimientos:**

* La compañía encargada de análisis de datos ha solicitado la implementación de un ABB balanceado que maneje gran volumen de información y permita realizar las operaciones de consultas sobre el árbol.
* Las consultas se deben realizar de forma eficiente debido al almacenamiento de millones de datos, ya que se necesitan obtener de forma rápida.
* R1. Realizar cuatro consultas de registros de acuerdo a cuatro criterios.
* R2. Implementar dos consultas con una misma estructura de datos recursiva.
* R3. Realizar el proceso de desarrollo utilizando el método de la ingeniería.
* R4. Implementar la búsqueda en un ABB balanceado a partir de los archivos txt en los cuales se guardará cada registro.
* R5. Visualizar en la interfaz grafica el procedimiento de la consulta a realizar.

**Fase 2:** Recopilación de la información necesaria.

Con el objetivo de tener claridad en algunos conceptos o términos, a continuación, se plantean una serie de definiciones relacionadas con el problema a tratar.

**¿Qué es un dato?:** Un dato nos permite describir un objeto. Dicho objeto podemos llamarlo entidad, por ejemplo, una casa en la que viven personas. La casa es la entidad y la cantidad de personas que viven en la casa son un dato, que en este caso es numérico.

Hay diferentes tipos de datos que se pueden tener en una base de datos: caracteres, numéricos, imágenes, fechas, monedas, texto, bit, decimales y varchar. Hay bases de datos más o menos adecuadas según el tipo de dato, y hay un proceso para convertir un dato en información, pues**un solo dato por sí mismo no representa nada si no lo vemos en contraste con otros.** Así se identifican comportamientos.

Las empresas y las personas necesitan de las bases de datos en su vida diaria. Las utilizamos más de lo que nos imaginamos, cada vez que usamos el teléfono, hacemos una compra o cuando hacemos una transacción bancaria. Si estás empezando un proyecto nuevo o vas a aprender a programar vale la pena tener muy claro el funcionamiento de las bases de datos.

https://platzi.com/blog/bases-de-datos-que-son-que-tipos-existen/

**Base de datos:** Una base de datos se puede definir como un conjunto de información relacionada que se encuentra agrupada o estructurada.

Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro.

<http://www.maestrosdelweb.com/que-son-las-bases-de-datos/>

**Tabla( Base de Datos):** Tabla en las [bases de datos](https://es.wikipedia.org/wiki/Bases_de_datos), se refiere al tipo de modelado de datos, donde se guardan y almacenan los datos recogidos por un programa. Su estructura general se asemeja a la vista general de un programa de [hoja de cálculo](https://es.wikipedia.org/wiki/Hoja_de_c%C3%A1lculo).

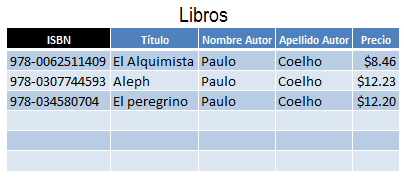
Una tabla es utilizada para organizar y presentar información. Las tablas se componen de filas y columnas de celdas que se pueden rellenar con textos y gráficos.

Las tablas se componen de dos estructuras:

* [Registro](https://es.wikipedia.org/wiki/Registro_(base_de_datos)): es cada una de las filas en que se divide la tabla. Cada registro contiene datos de los mismos tipos que los demás registros. Ejemplo: en una tabla de nombres, direcciones, etc. cada fila contendrá un nombre y una dirección.
* [Campo](https://es.wikipedia.org/wiki/Campo_(base_de_datos)): es cada una de las columnas que forman la tabla. Contienen datos de tipo diferente a los de otros campos. En el ejemplo anterior, un campo contendrá un tipo de datos único, como una dirección, o un número de teléfono, un nombre, etc.

A los campos se les puede asignar, además, propiedades especiales que afectan a los registros insertados. El campo puede ser definido como índice o autoincrementable, lo cual permite que los datos de ese campo cambien solos o sean el principal a la hora de ordenar los datos contenidos.

Cada tabla creada debe tener un nombre único en la Base de Datos, haciéndola accesible mediante su nombre o su seudónimo (Alias) (dependiendo del tipo de base de datos elegida). La estructura de las tablas viene dada por la forma de un archivo plano, los cuales en un inicio se componían de un modo similar.

Las tablas son los objetos principales de bases de datos que se utilizan para guardar datos.

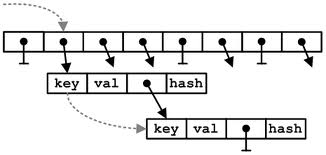
<https://es.wikipedia.org/wiki/Tabla_(base_de_datos)>

**Estructura de datos:**

En [ciencias de la computación](https://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n), una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente.

Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de [aplicaciones](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica), y algunos son altamente especializados para tareas específicas.

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

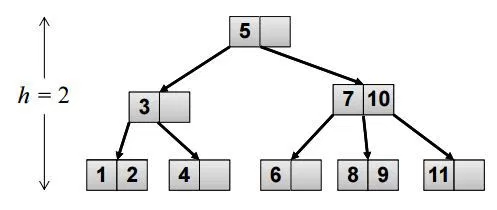
**Estructura de datos HashMap:**

Un HashMap permite guardar elementos, donde cada elemento es un par clave/valor. A diferencia de un array simple donde se guarda el valor en un índice en concreto, un HashMap determina el índice él mismo basándose en el valor hash (hashcode) generado a partir de la clave.

**Estructura de datos Árbol:**

En [ciencias de la computación](https://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n) y en [informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica), un árbol es un [tipo abstracto de datos (TAD)](https://es.wikipedia.org/wiki/Tipo_abstracto_de_datos) ampliamente usado que imita la [estructura jerárquica de un árbol](https://es.wikipedia.org/wiki/Topolog%C3%ADa_arb%C3%B3rea), con un valor en la raíz y subárboles con un nodo padre, representado como un conjunto de [nodos](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)) enlazados.

Una estructura de datos de árbol se puede definir de forma [recursiva](https://es.wikipedia.org/wiki/Recursi%C3%B3n) (localmente) como una colección de nodos (a partir de un nodo raíz), donde cada nodo es una estructura de datos con un valor, junto con una lista de referencias a los nodos (los hijos) , con la condición de que ninguna referencia esté duplicada ni que ningún nodo apunte a la raíz.

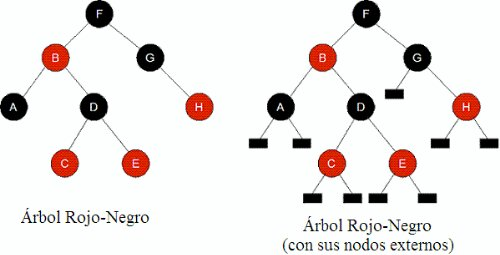
**Arboles B:**

En las [ciencias de la computación](https://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n), los árboles-B o B-árboles son [estructuras de datos de árbol](https://es.wikipedia.org/wiki/%C3%81rbol_(programaci%C3%B3n)) que se encuentran comúnmente en las implementaciones de [bases de datos](https://es.wikipedia.org/wiki/Bases_de_datos) y [sistemas de archivos](https://es.wikipedia.org/wiki/Sistemas_de_archivos). Al igual que los [árboles binarios de búsqueda](https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda), son árboles balanceados de búsqueda, pero cada nodo puede poseer más de dos hijos.[1](https://es.wikipedia.org/wiki/%C3%81rbol-B#cite_note-1)​ Los árboles B mantienen los datos ordenados y las inserciones y eliminaciones se realizan en tiempo logarítmico amortizado.

**Arboles rojinegros:**

Un árbol rojo-negro es un tipo especial de [árbol binario](https://es.wikipedia.org/wiki/%C3%81rbol_binario) usado en [informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica) para organizar información compuesta por datos comparables (por ejemplo, números). En los árboles rojo-negro las [hojas](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)#Nodo_hoja) no son relevantes y no contienen datos.

En los árboles rojo-negro, como en todos los [árboles binarios de búsqueda](https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda), es posible moverse ordenadamente a través de los elementos de forma eficiente si hay forma de localizar el padre de cualquier nodo. El tiempo de desplazarse desde la raíz hasta una hoja a través de un árbol equilibrado que tiene la mínima altura posible es de O(log n).

Al implementar esta estructura es posible utilizar un único [nodo centinela](https://es.wikipedia.org/wiki/Nodo_centinela). Este cumple la función de hoja para todas las ramas del árbol. Así, todos los [nodos internos](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)#Nodo_interno) que finalicen en una hoja tienen referencia a este único nodo centinela. Esto no es necesario, ya que puede hacerse una referencia nula (NIL) en el final de cada rama

Además de los requisitos impuestos a los árboles binarios de búsqueda convencionales, se deben satisfacer las siguientes reglas para tener un árbol rojo-negro válido:

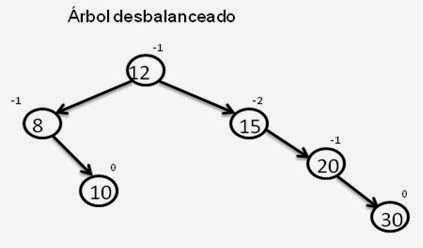
1. Todo [nodo](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)) es o bien rojo o bien negro.
2. La [raíz](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)#Nodo_ra%C3%ADz) es negra.
3. Todas las [hojas](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)#Nodo_hoja) (NULL) son negras.
4. Todo nodo rojo debe tener dos nodos hijos negros.
5. Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

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

**Arboles AVL:**

El árbol AVL toma su nombre de las iniciales de los apellidos de sus inventores, [Georgii Adelson-Velskii](https://es.wikipedia.org/wiki/Georgii_Adelson-Velskii) y [Yevgeniy Landis](https://es.wikipedia.org/wiki/Yevgeniy_Landis). Lo dieron a conocer en la publicación de un artículo en [1962](https://es.wikipedia.org/wiki/1962) (Un algoritmo para la organización de la información).

Los árboles AVL están siempre equilibrados de tal modo que, para todos los nodos, 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](https://es.wikipedia.org/wiki/Complejidad_computacional) [O](https://es.wikipedia.org/wiki/Cota_superior_asint%C3%B3tica)(log n). El factor de equilibrio 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](https://es.wikipedia.org/w/index.php?title=Rotaci%C3%B3n_de_%C3%A1rbol&action=edit&redlink=1)

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

A parte de los métodos ya existentes, se podrían realizar algunas implementaciones propias, que requerirían al menos conceptos básicos de cómo funcionan las estructuras mencionadas previamente.

Información adicional:

* Limitaciones de Java para este problema.
* Alternativas que propone Java.
* Rango de datos para millones de datos en la base.

**Fase 3:** Búsqueda de soluciones creativas.

Habiendo investigado varias alternativas de estructuras de datos para darle solución al problema, se debe encontrar o perfeccionar una que se adecue a la naturaleza de este.

A continuación, se realiza una lista de revisión de las estructuras en la cual se examinan diferentes puntos, áreas y posibilidades de diseño para la solución del problema planteado:

* HashMap:
  + usa los métodos equals y hashCode para identificar la llave.
  + no mantiene un orden secuencial de los datos registrados.
  + Almacena datos asociando una llave a un valor.
  + Cada llave tiene que estar asociada a un valor como máximo.
* Arboles B:
  + Son arboles balanceados de búsqueda.
  + Cada nodo puede poseer más de dos hijos.
  + Mantienen los datos ordenados.
  + No necesitan re balancearse tan frecuentemente como los ABB.
  + Requiere que todos los nodos hoja se encuentren a la misma altura.
* Arboles Roji-Negros:
  + Puede buscar, insertar y borrar en O(log n).
  + Se usa para organizar información compuesta por datos comparables.
  + Existen las hojas NIL que no son relevantes y no contienen datos
  + Son auto balanceables, y cada nodo tiene un atributo de color.
  + La búsqueda y devolución del resultado es mas lenta que la de los AVL.
* Arboles AVL:
  + Es un tipo de árbol binario de búsqueda auto-balanceable.
  + Están siempre equilibrados.
  + Tiene complejidad de búsqueda O(log n).
  + Tiene un factor de equilibrio en cada nodo.
  + La inserción y eliminación es más lenta que la de los RojiNegros, porque esta equilibrado de forma más rígida.
* A partir de la lista de revisión, se puede dar solución al problema mediante las siguientes ideas de diseño:
  + 1. Con la estructura de hashMap u otra implementación de los Map de java, se puede implementar la base de datos tal como la propone el enunciado, usando el valor para el campo de la BD como el valor del hashMap y el índice del registro como la clave.
    2. Con las estructuras de arboles AVL, ARN, y AB, se puede implementar la base de datos, de forma organizada como lo piden en los que cada nodo tendrá un par de valores (c, r) en donde ( c ) es un valor para el campo y ( r ) el índice de registro.

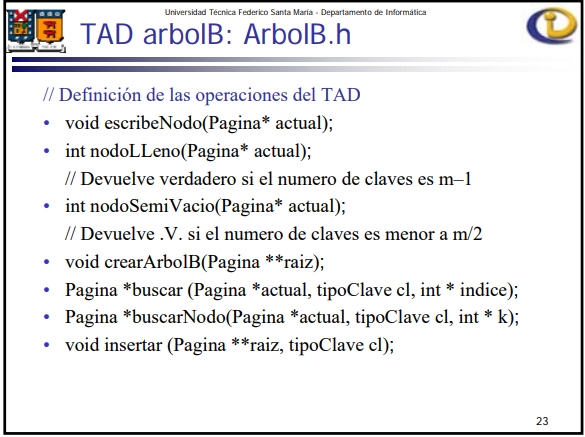
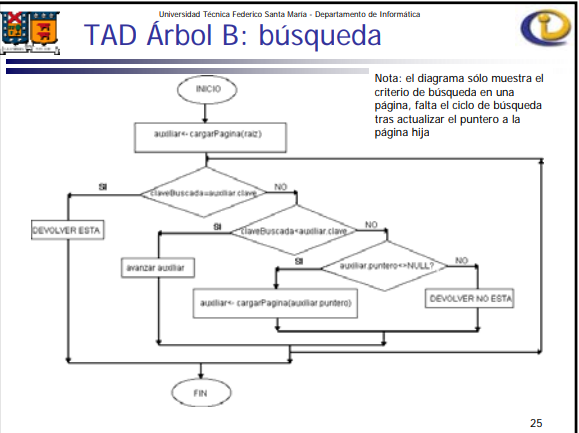
**Fase 4:**  Transición de la formulación de ideas a diseños preliminares.

Lo primero que se hará en esta etapa es proceder a descartar las ideas que no son factibles. En este sentido, descartamos la idea de usar estructuras como HashMap, porque se necesita un orden de jerarquía tipo árbol para acceder fácilmente a las búsquedas de los millones de datos de manera eficiente.

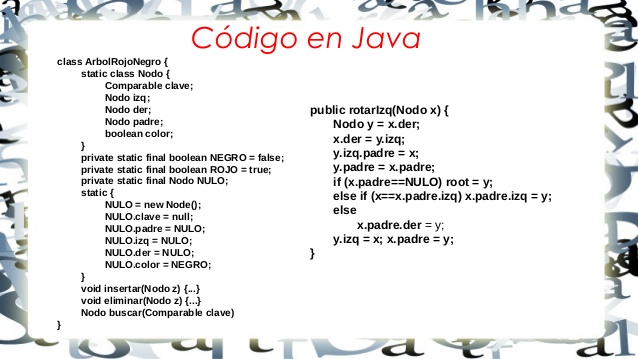
Diseños preliminares:

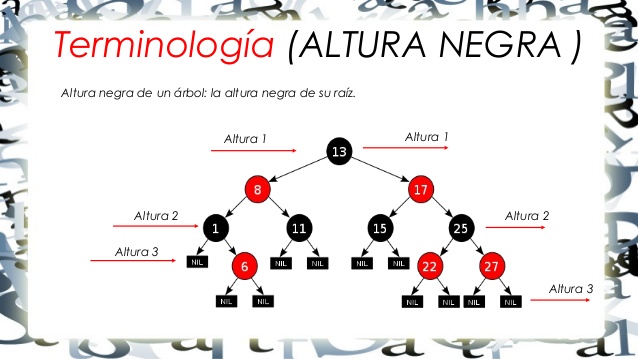
* **Arboles B**: se muestra un diagrama de flujo del criterio búsqueda, y también se define el TAD con métodos sujetos a cambios.

**TADS**

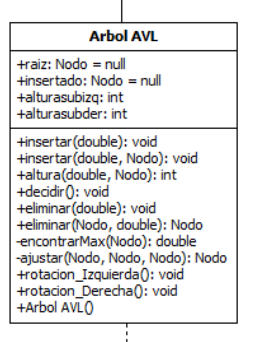


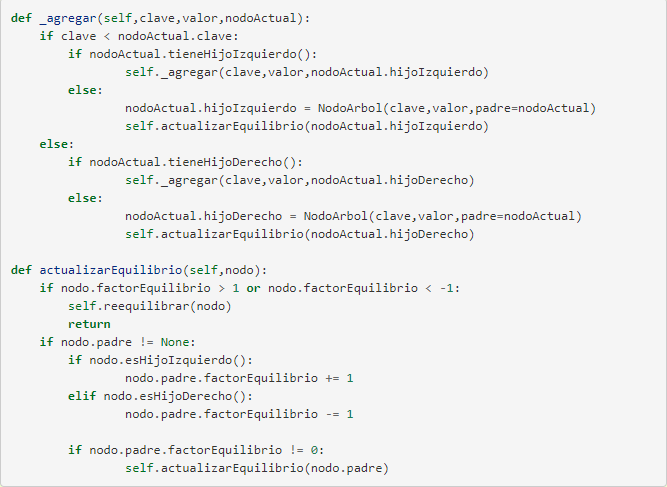
* **Arboles Roji-Negros:** se muestra una parte de la implementación realizada en java, y posteriormente un diseño de la terminología altura negra.

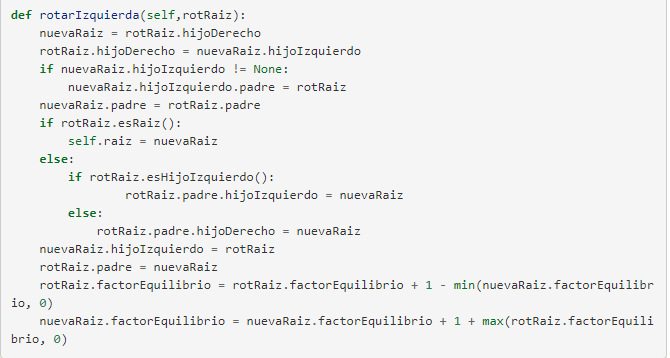




* **Arboles AVL:** se presenta el siguiente diagrama de la clase arbolAVL, junto con la implementación de los métodos agregar, actualizar equilibrio y rotar a la izquierda, sujetos a cambios posteriores







**Fase 5:** Evaluación y selección de la mejor solución.

Criterios: los siguientes criterios permitirán evaluar las diferentes alternativas o ideas y con base en este resultado se procederá a elegir las soluciones que mejor satisfacen las necesidades del problema. Al lado de cada valor del criterio hay un numero que indica cuales tienen más valor o más peso, entre mayor peso tenga, más importante será para la consideración final.

Criterio A: complejidad temporal. Se prefiere una solución más eficiente y con menor complejidad temporal que las otras. La complejidad temporal puede ser:

* Constante (5)
* Logarítmico (4)
* Lineal (3)
* Cuadrático (2)
* Exponencial (1)
* Criterio B: dificultad de implementación. La solución al problema con el algoritmo implementado tiene muchas líneas de código, declaración de variables, importaciones, ciclos, etc.
  + Poca dificultad de implementación (3)
  + Dificultad media (2)
  + Bastante dificultad (1)
* Criterio C: tiempo de respuesta. El algoritmo es muy demorado y utiliza mucha memoria en sus cálculos internos, o lo hace de manera rápida y eficiente:
  + Demorado (1)
  + Rápido (2)

Evaluación: evaluando las alternativas pre seleccionadas con los criterios mencionados anteriormente, se procede a la siguiente tabla.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Criterio A | Criterio B | Criterio C | Total |
| Solución con Arboles B | 4 | 1 | 1 | 6 |
| Solución con árboles Roji-Negros | 4 | 2 | 2 | 8 |
| Solución con árboles AVL | 4 | 2 | 2 | 8 |

Selección: de acuerdo con la tabla anterior se procede a seleccionar las alternativas de implementar la base de datos con las estructuras de árboles roji-negros y árboles AVL, ya que obtuvieron mayores puntuaciones.