

Tarea integradora

Gestión eficiente de base de datos de personas

Santiago hurtado Solís, Sebastian Morales, Cristian Morales.

Sebastian
[Fecha]

Objetivos

Unidad 3: Algoritmos y Estructuras Recursivas

OE3.1. Calcular la complejidad temporal de algoritmos recursivos.

OE3.3. Resolver ecuaciones de recurrencia que sean resultado del análisis de complejidad temporal de algoritmos recursivos.

OE3.5. Aplicar la técnica de diseño de algoritmos Dividir y Conquistar en la solución de problemas.

OE3.6. Utilizar estructuras recursivas de datos para representar la información del modelo de datos cuando sea conveniente.

OE3.7. Evaluar la utilidad del concepto de orden en un árbol binario de búsqueda para la solución de problemas.

OE3.8. Evaluar la utilidad del concepto de balanceo en un árbol binario de búsqueda para la solución de problemas.

OE3.9. Desarrollar estructuras de datos recursivas ABB.

Opcional: OE3.10. Desarrollar estructuras de datos recursivas R & N.

OE3.11. Desarrollar estructuras de datos recursivas AVL.

OE3.14. Desarrollar las pruebas unitarias de cada una de las estructuras de datos recursivas implementadas.

Gestión Eficiente de Base de Datos de Personas

Enunciado

Después del excelente rendimiento en sus trabajos académicos previos, sus profesores de algoritmos los han recomendado a diversos profesores de la Facultad de Ingeniería, para que trabajen en sus proyectos de investigación. Su equipo ha sido contratado para una monitoría en un proyecto de investigación interna de la Universidad, como parte del Equipo VIP de Simulación¹.

Generación de los Datos

El sub-proyecto que le ha sido asignado, consiste en el desarrollo de un **prototipo de software que permita gestionar eficientemente las operaciones CRUD sobre una base de datos de personas de nuestro continente**. La población del continente americano se estima, en 2020, en poco más de mil millones de personas², representando cerca del 13% del total mundial. Por tanto, usted debe **simular la creación de** (generar) un número similar de registros de personas, para este continente, con los siguientes datos: código (autogenerado), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía.

Usted debe desarrollar un programa que lleve a cabo la generación de todos estos registros de personas de acuerdo con las condiciones que se detallan a continuación. Los datasets que el programa utilizará como entrada para generar los datos de las personas deben ser descargados a un directorio del proyecto (/data).

Para la generación de nombres completos, se deben tomar los nombres de este [dataset de nombres de data.world](#). Los apellidos se deben tomar del archivo completo (el archivo pequeño -1000- debe ser ignorado) de este [dataset de apellidos de data.world](#). La combinación de todos los nombres con

¹ [VIP Curse in Data Intensive Processing and Simulation](#)

² <https://www.worldometers.info/geography/7-continent/>

todos los apellidos debe producir la cantidad (o similar) de personas que deseamos.

La fecha de nacimiento debe ser generada aleatoriamente, suponiendo una distribución de edad para toda América basada en [esta distribución de edad de Estados Unidos](#). La distribución de la población en sexo indicada en el enlace anterior puede ser ignorada, y se puede asumir una cantidad igual de hombres y mujeres.

La estatura debe ser generada aleatoriamente en un intervalo que tenga sentido. La nacionalidad debe ser asignada a cada persona, generada de tal forma que se mantengan los porcentajes relativos de población de cada país respecto del continente de acuerdo con [estos datos de población por países](#) (se puede también asignar exactamente la misma población de cada país, y si hay diferencia en el total, dicha diferencia -positiva o negativa- puede quedar en el país con mayor población).

Usted puede filtrar el dataset para dejar en el archivo únicamente los registros de los países necesarios. Su programa debe basar los cálculos en el archivo y no en condicionales por país quemados en el código (hardcode).

Bonus: La fotografía debe ser generada aleatoriamente de este sitio: <https://thispersondoesnotexist.com/> sin importar que su imagen no se corresponda exactamente con su edad, sexo u otros.

El programa debe tener una opción para empezar a generar los datos siguiendo las especificaciones anteriores. Debe tener una barra de progreso si el proceso tarda más de 1 segundo en terminar, y debe indicar cuánto tiempo se demoró la operación. La opción de generar debe tener un campo de texto en el cual se pueda digitar cuántos registros se desea generar. Por defecto, debe estar en el campo el máximo valor posible.

Una vez los datos se generan, debe haber una opción para guardarlos en la base de datos del programa, y así poderlos consultar posteriormente. Todos los datos del programa deben ser persistentes (es decir, si se cierra el programa, deben seguir allí una vez se inicie nuevamente).

Responda este par de preguntas: (1) ¿Es posible serializar los datos generados? (2) ¿Cuánto pesa el archivo serializado, cuando se persisten los datos, al generarse el número máximo posible de registros?

Los datos generados son posibles de serializar, el archivo serializado depende de la cantidad de usuarios que se generen, pero si se generan con las imágenes un par de gbs aproximadamente, y los datos se persisten cuando se cierra el programa.

Operaciones CRUD

En programación, se conoce como CRUD (Create, Read, Update y Delete) a las cuatro funciones básicas del almacenamiento persistente³. El programa desarrollado por su equipo debe tener la posibilidad de llevar a cabo cualquiera de estas funciones. Por tanto, la interfaz con el usuario deberá contar con un formulario para agregar a una nueva persona, otro para buscar a una persona por los criterios mencionados más adelante y uno más para actualizar los campos de una persona

³ https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

existente. Este último formulario debe también tener una opción para eliminar a una persona.

Crear, Actualizar y Eliminar

El formulario para agregar debe tener todos los campos requeridos (menos el código, que es autogenerated) para la información de una persona y la opción de Guardar.

El formulario para actualizar una persona, debe tener todos los campos editables (menos el código, que no se puede actualizar) de información de una persona, cargados con su información actual, la opción de Actualizar (para guardar los cambios, si hubo) y la opción Eliminar (si se desea eliminar a esta persona).

Buscar (la estrella de la solución)

El formulario para buscar debe tener la posibilidad de realizar la búsqueda por cualquiera de los siguientes criterios, de forma excluyente (es decir, buscas por un criterio o por otro, pero no por varios al tiempo):

1. Nombre
2. Apellido
3. Nombre Completo (Nombre + " " + Apellido)
4. Código

El programa debe permitir que, en la medida en que se digite los caracteres de búsqueda en el campo (para los criterios 1 a 3), vayan apareciendo en una lista **emergente** debajo del campo, máximo 100 (parametrizable) criterios de la base de datos, que empiecen con los caracteres digitados hasta el momento.



Ejemplo de la búsqueda y la lista emergente

Para cada una de las cuatro búsquedas, usted debe mantener un árbol binario de búsqueda autobalanceado y persistente. El árbol debe ser implementado completamente por su equipo de desarrollo.

Bonus: si uno de los árboles binarios de búsqueda autobalanceado es un Árbol Rojo y Negro.

Mientras se hace la búsqueda, debe mostrarse al lado del campo donde se digita la cadena, un número que indica la cantidad total de elementos que hasta el momento coinciden con el prefijo digitado.

En el momento en que haya 20 (parametrizable) o menos elementos que coinciden con la búsqueda, debe desplegarse un listado con los registros que coinciden y un botón al lado de cada registro con la opción de Editar, que nos llevará al formulario con la posibilidad de modificar o eliminar ese registro.

El programa debe tener el diseño e implementación de pruebas unitarias automáticas de las estructuras de datos implementadas y de las operaciones principales de las clases del modelo.

El programa debe contar con una interfaz gráfica con el usuario, tener un componente menú en la parte superior de la ventana que permite cambiar todos los elementos de la ventana cada vez que se esté trabajando en opciones diferentes. Es decir, **no** se desea tener una ventana llena al mismo tiempo de elementos que se usarán en momentos diferentes.

Entregas

Usted debe entregar los siguientes artefactos en las condiciones indicadas:

1. **[15%]** Desarrollo completo del Método de la Ingeniería.
 - a. La etapa 1 debe incluir la Especificación de Requerimientos Funcionales.
 - b. En las etapas 3, 4 y 5 se deben concentrar en el problema de cómo generar sugerencias al digitar texto.
2. **[25%]** Diseño.
 - a. **[20%]** Diseño completo del diagrama de clases, incluyendo, las estructuras de datos, el paquete del modelo, de la interfaz con el usuario y pruebas.
 - b. **[5%]** Diseño de los casos de la única prueba, incluyendo los escenarios.
3. **[50%]** Implementación.
 - a. **[15%]** Implementación completa de las estructuras de datos y sus pruebas.
 - b. **[35%]** Implementación completa y correcta del modelo, la ui y las pruebas.
4. **[10%]** Usted debe entregar el enlace del repositorio en GitHub o GitLab con los elementos anteriores. Su repositorio debe corresponder con un proyecto de eclipse. Cada integrante del equipo debe tener al menos 10 commits con diferencia de 1 hora entre cada uno de ellos. En el repositorio o proyecto de eclipse debe haber un directorio llamado **docs/** en el cual deberán ir cada uno de los documentos del diseño.
5. **[5%]** **Bonus** de la imagen de la persona.
6. **[10%]** **Bonus** de Árbol Rojinegro (incluyendo diagrama de clases, implementación, pruebas y uso en el modelo).

Los requerimientos funcionales, el diagrama de clases y el diseño de casos de prueba deben entregarse en un mismo archivo en formato **pdf**, bien organizado por secciones y títulos, con hoja de portada.

Importante: su repositorio debe ser privado hasta la hora y fecha de entrega, después de la cual usted debe hacerlo público para que pueda ser revisado y compartido a la comunidad que estará ansiosa de conocerlo!.

Nota: La rúbrica con la que se evaluará esta tarea se encuentra en el listado de notas de seguimientos y tareas integradoras en la pestaña TI2. Se recomienda revisar la rúbrica con la que será evaluada su entrega.

Método de la Ingeniería

1. IDENTIFICACIÓN DE NECESIDADES Y ESPECIFICACIÓN DEL PROBLEMA

○ **necesidades del problema**

En el siguiente programa se identificaron las siguientes necesidades:

- Manejo de grandes cantidades de datos.
- Búsqueda sensitiva o filtrada de los datos
- Cálculo de datos por países de américa latina.
- Ejecución de las operaciones CRUD en los registros de datos.
 - Agregar nuevos usuarios.
 - Editar información de los usuarios existentes.
 - Eliminación de usuarios existentes.
- Manejo de una interfaz gráfica completa, e intuitiva en el manejo de las operaciones.
- Guardar la cantidad de datos generados aleatoriamente.
- Generación de gran cantidad de datos.

REQUERIMIENTOS FUNCIONALES

El sistema debe permitir:

1. Generar n cantidad de registros de usuarios, donde cada usuario debe tener código, nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y una fotografía que acompañe el registro del usuario.
 - Crear un código auto generado para cada usuario, junto a su fecha de nacimiento.
 - Generar la estatura en un intervalo determinado.
 - Generar automáticamente la nacionalidad de los usuarios, donde aparezcan los países correspondientes a Latinoamérica tales como Colombia, Argentina, Brasil, Ecuador, Venezuela, Uruguay, Paraguay, Perú, Chile y Bolivia. La nacionalidad de cada usuario debe ser generada de tal forma que se mantengan los porcentajes relativos de población de cada país.
 - Generar una foto aleatoria para cada usuario.
2. Mostrar una barra de progreso si el proceso tarda más de 1 segundo en terminar, y debe indicar cuánto tiempo se demoró la operación. La opción de generar debe tener un campo de texto en el cual se pueda digitar cuántos registros se desea generar. Por defecto, debe estar en el campo el máximo valor posible.
3. Guardar los datos generados para consultarlos después. (Persistencia)
4. Agregar y guardar un usuario con nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y una fotografía que lo identifique, además este usuario debe tener un código el cual es autogenerado.
5. Editar y guardar los datos de un usuario ya existente, los cuales se pueden actualizar todos, menos su código.
6. Eliminar un usuario que ya existe.
7. Buscar un usuario por un único criterio, ya sea por nombre, apellido, nombre completo y

código.

- Además esta búsqueda debe permitir que en la medida en que se digite los caracteres a buscar, para los criterios de nombre, apellido o nombre completo, vayan apareciendo en una lista emergente, máximo 100 elementos de la base de datos, estos deben empezar con los caracteres digitados, mostrando las coincidencias.
 - Se debe indicar la cantidad de elementos que coinciden con los caracteres digitados.
 - Cuando se realice la búsqueda, debe mostrarse al lado del campo donde se digita la cadena, un número que indica la cantidad total de elementos que hasta el momento coinciden con el prefijo digitado.
 - En el momento en que haya 20 (parametrizable) o menos elementos que coinciden con la búsqueda, debe desplegarse un listado con los registros que coinciden y un botón al lado de cada registro con la opción de Editar, que nos llevará al formulario con la posibilidad de modificar o eliminar ese registro.
8. El programa debe contar con una interfaz gráfica con el usuario, tener un componente menú en la parte superior de la ventana que permite cambiar todos los elementos de la ventana cada vez que se esté trabajando en opciones diferentes. Es decir, **no** se desea tener una ventana llena al mismo tiempo de elementos que se usarán en momentos diferentes.

REQUERIMIENTOS NO FUNCIONALES

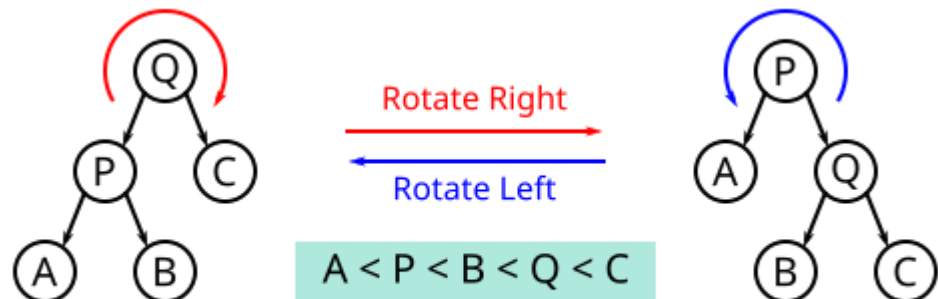
1. Implementar un árbol de búsqueda persistente y auto balanceado
 2. Los registros de los usuarios generados deben guardarse en una carpeta data/set.
 3. Implementar un árbol de búsqueda persistente y auto balanceado.
 4. El programa debe tener el diseño e implementación de pruebas unitarias automáticas de las estructuras de datos implementadas y de las operaciones principales de las clases del modelo.
 5. El árbol implementado debe ser de tipo avl, en su caso opcional rojo y negro.
- **Definición del problema**
 - El equipo VIP de simulación, requiere un software que le permita simular una base de datos de todas las personas residentes en Latinoamérica, donde en la base de datos se puedan desarrollar las operaciones crud.

2. Recopilación de Información

- Con el objetivo de encontrar una manera óptima de almacenar datos y entender el concepto de arboles auto balanceado, se ha idecido buscar informacion sobre su funcionamiento y propiedades para comprender mas claramente como funcionan y porque son utiles en el caso, ademas de investigar tambien sobre el arbol rojo-negro, el cual es el recomendado para desarrollar este proyecto.
- **Definiciones**
 - En ciencias de la computación, un **árbol binario de búsqueda auto-balanceable** o **equilibrado** es un árbol binario de búsqueda que intenta mantener su *altura*, o el número de niveles de nodos bajo la raíz, tan

pequeños como sea posible en todo momento, automáticamente. Esto es importante, ya que muchas operaciones en un árbol de búsqueda binaria tardan un tiempo proporcional a la altura del árbol, y los árboles binarios de búsqueda ordinarios pueden tomar alturas muy grandes en situaciones normales, como cuando las claves son insertadas en orden. Mantener baja la altura se consigue habitualmente realizando transformaciones en el árbol, como la rotación de árboles, en momentos clave.

- En matemáticas discretas, **Rotación de árboles** es una operación en un árbol binario que cambia la estructura sin interferir con el orden de los elementos. Un árbol de rotación se mueve hasta un nodo en el árbol y un nodo hacia abajo. Se utiliza para cambiar la forma del árbol, y en particular para disminuir su altura moviendo subárboles más pequeños hacia abajo y subárboles más grande, lo que resulta en un mejor rendimiento de muchas operaciones de los árboles.



- Un **árbol rojo-negro** es un tipo abstracto de datos. Concretamente, es un árbol binario de búsqueda equilibrado, una estructura de datos utilizada en informática y ciencias de la computación. Es complejo, pero tiene un buen peor caso de tiempo de ejecución para sus operaciones y es eficiente en la práctica. Puede buscar, insertar y borrar en un tiempo $O(\log n)$, donde n es el número de elementos del árbol. Un árbol rojo-negro es un árbol binario de búsqueda en el que cada nodo tiene un atributo de color cuyo valor es rojo o negro. En adelante, se dice que un nodo es rojo o negro haciendo referencia a dicho atributo. 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:
 - Todo nodo es o bien rojo o negro.
 - La raíz es negra.
 - Todas las hojas (NULL) son negras.
 - Todo nodo rojo debe tener dos nodos hijos negros.

- Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.
 - Estas reglas producen una regla crucial para los árboles rojo-negro: el camino más largo desde la raíz hasta una hoja no es más largo que dos veces el camino más corto desde la raíz a una hoja. El resultado es que dicho árbol está aproximadamente equilibrado.
 - Dado que las operaciones básicas como insertar, borrar y encontrar valores tienen un peor tiempo de ejecución proporcional a la altura del árbol, esta cota superior de la altura permite a los árboles rojo-negro ser eficientes en el peor caso, a diferencia de los árboles binarios de búsqueda.
- Los **filtros de búsqueda** son una herramienta para perfeccionar la estrategia de búsqueda y están constituidos por una combinación de términos o descriptores y diseñadas para localizar tipos de estudios determinados. Estas estrategias prediseñadas se han de combinar con el término/s o descriptor/es de lo que deseamos buscar permitiéndonos una recuperación de la información con un alto grado de exactitud.
- El concepto CRUD está estrechamente vinculado a la gestión de datos digitales. CRUD hace referencia a **un acrónimo** en el que se reúnen las primeras letras de las cuatro operaciones fundamentales de aplicaciones persistentes en sistemas de bases de datos:

- Create (Crear registros)
- Read bzw. Retrieve (Leer registros)
- Update (Actualizar registros)
- Delete bzw. Destroy (Borrar registros)

En pocas palabras, CRUD resume las funciones requeridas por un usuario para crear y gestionar datos. Varios procesos de gestión de datos están basados en CRUD, en los que **dichas operaciones están específicamente adaptadas a los requisitos del sistema y de usuario**, ya sea para la gestión de bases de datos o para el uso de aplicaciones.

- Una **base de datos** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto, se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

3. Búsqueda de Soluciones creativas

- **Generación base de datos**

1. Generar la cantidad de datos, mediante una página web, el cual concatena toda la información de usuario, la página es <http://www.generatedata.com/?lang=es>.
2. Generar la cantidad de datos requeridas, de diferentes páginas las cuales están en el documento, teniendo en cuenta la distinta información de los usuarios.
3. Tomar una base de datos ya realizada.
4. Que el usuario ingrese la información de varios usuarios.

- **Generación de las imágenes**

1. Buscar un api que permita enlazar una imagen a nuestro proyecto mediante la página tal: <https://thispersondoesnotexist.com/image>
2. Tomar una dirección URL de una imagen y agregarla como atributo del usuario.
3. Tener un data sets de imágenes de personas, para agregarlas a cada usuario.
4. Que el usuario agregue las imágenes manualmente.
5. Limitar la funcionalidad a solo datos del usuario, sin imagen.

- **Digitar Texto y búsqueda de usuario.**

1. Cada vez que el usuario digite un carácter, agregar los 20 o menos primeros usuarios a un arreglo que tengan ese carácter, y la cantidad de usuarios encontrados es mayor a 20, el arreglo se saldrá vacío y mostrar el arreglo en la pantalla, para mostrar una tabla que permite seleccionar el usuario a buscar para editar o eliminar sus datos, además de mostrar la cantidad de usuarios que coinciden con ese nombre, ejecutándose así en un segundo plano mediante un hilo, para que el programa no se quede en stop mientras se realiza la búsqueda.
2. Cada vez que el usuario digite un conjunto de caracteres, utilizando una cola se almacenen las coincidencias y se vaya actualizando cada vez que el usuario digite un carácter, además está de irse autocompletando el texto con los nombres o apellidos de la persona a buscar, estos guardados en una lista, para que al final al usuario le aparezca el que busque y pueda actualizar o eliminar la información de este.
3. Implementar una búsqueda sensitiva mediante (un contenedor en la interfaz el cual

contenga opciones de los diferentes nombres, o apellidos de la persona a buscar) donde se filtren estos datos, estos aparecerán de acuerdo con el carácter escrito, las coincidencias a buscar se guardarán en una pila, mostrando los resultados en un table view, por orden alfabético.

- **Barra de progreso**

1. Realizar una barra de progreso mediante javafx, con el manejo de hilos, y diseño en scene builder.
2. Realizar un círculo de progreso el cual muestre en porcentaje el valor por el que va cargando.
3. No realizar barra de progreso, y mostrar en porcentaje lo que lleva hasta finalizar la operación de generación de datos.
4. omitir este punto.

- **Diseño de la interfaz**

1. Diseño de la interfaz en código fxml.
2. Uso de la librería gráfica de java, java Swing.
3. Uso de JavaFx, diseñando las interfaces mediante SceneBuilder.
4. Mostrar la funcionalidad del programa mediante la consola.

- **Agregar los usuarios**

1. Otra solución es cuando se generen los n usuarios, estos se guarden directamente en el árbol AVL, siendo nodos de este.
2. Una solución a la hora de agregar usuarios puede ser que cuando se generen una cantidad de usuarios, estos se guarden directamente en una lista, la cual al darle la opción de guardar estos migren al árbol, haciendo esto en segundo plano.
3. Otra solución sería que los guarde en una lista y al guardarlos en esta pasen al árbol rojo y negro, mediante uso de hilos.

4. Transición de las Ideas a los Diseños Preliminares

- **Generación de base de datos:**

Alternativa 1: Esta idea es viable, pero el programa que utilizaremos allí limita la generación de usuarios como hasta 1000, lo cual tocaría generarlos muchas veces y unir estos archivos, para ya luego programar y que se generan n cantidad de registros.

Alternativa 2: Esta idea es una de las más completas, y tendríamos que desarrollar un método para cada característica para unirlas a cada usuario, tal como la fecha de cumpleaños, la nacionalidad y demás características, en consecuencia, obtendremos una base de datos gigante cumpliendo con los requerimientos, esta generación de usuarios se puede ejecutar en segundo plano, para mejorar su tiempo de ejecución.

Alternativa 3: Es otra idea que se puede desarrollar fácilmente, pero las que encontramos la cantidad de los registros era muy limitada.

Alternativa 4: es una de las formas de desarrollar nuestra base de datos, pero se demoraría mucho tiempo a la hora de tener casi mil millones de registros como lo piden en los requerimientos, por lo cual la descartaríamos.

- **Generación de la imagen**

Alternativa 1: Es una solución viable, que cumple con todos los requerimientos, realizando una petición https para obtener la imagen cada vez que se busca, obteniendo que los usuarios tendrían una imagen mediante el enlace de la página dada anteriormente, lo cual permitiría que los usuarios salgan diferentes, además cuando se crea un usuario este se le asigna imagen automáticamente.

Alternativa 2: Realizar una petición cada vez que se genere un usuario en la base de datos, la cual se cargaría con el resto de las características, utilizando la librería de java.awt la cual hay un método el cual se le puede pasar la dirección como una url, podría ser una solución viable, pero no conocemos mucho el manejo de esta.

Alternativa 3: Cargaremos un dataset de imágenes el cual lo conectaremos con los usuarios a generar en la base de datos, pero la desventaja sería que algunas imágenes saldrían repetidas.

Alternativa 4: Es una idea que descartaríamos, porque esta solución viene incluida directamente en la alternativa 1, cuando se agregan los usuarios manualmente, además si vamos a poner una imagen para n usuarios, tomaría mucho tiempo para el usuario.

5. Es una alternativa la cual pensamos, si vemos un problema de tiempo, al implementarlo.

Digitar texto

1. Esta opción nos parece muy viable, y la verdad eficaz ya que, a la hora de correr el programa, el cual va a tener muchos datos, es necesario que esto se corra en segundo plano, para así poder realizar otras búsquedas, y la idea que se plantea es mediante el método divide y vencerás, ir partiendo los arreglos, hasta llegar a las coincidencias más acertadas. Además de tener un contador que cuenta las coincidencias encontradas, cada búsqueda se realizara en un árbol avl, por código, por nombre, por apellido y nombre completo, todo esto con el fin de optimizar la búsqueda de los datos.
2. La idea es viable, algo menos compleja, pero a la hora de buscar un usuario en millones de datos, el programa puede tender a presentar fallas y podríamos tener fallas a la hora de conectarlo con el table view.
3. Es una solución la cual puede ser viable, mediante la filtración de los datos por el criterio, mostrando los datos ordenados por orden alfabético, lo cual generaría un problema ya que no mostraría las coincidencias correctas, solo mostraría una tabla con los datos ordenados correctamente.

Barra de progreso:

1. Es una idea que, al evaluarla, funciona bien con los requerimientos y demás, que es eficiente a la hora de ir mostrando El Progreso de la generación de datos,

haciendo que no se quede en stop el programa, la barra de progreso se realizaría mediante scene builder, la opción de la librería javaFX.

2. Es una idea que no consideramos viable, ya que lo nos piden es una barra de progreso la cual está ligada al tiempo de generación de los usuarios.
3. Es algo que nos indicaría el progreso de la generación de los usuarios, pero no cumpliría con los requerimientos ya que nos piden una barra de progreso.

Diseño de la interfaz

1. Diseñar las interfaces en fxml es algo largo, y tedioso, además debemos aprender algo del lenguaje para realizarlo.
2. Diseñar las interfaces con la opción de la librería de Java swing, es una idea viable, brindaría una opción a la hora de cargar las imágenes, siendo esto una ventaja para los requerimientos.
3. Es la opción más viable, ya que scene builder nos ayuda mucho en cuanto al diseño y demás, la cantidad de componentes a usar, lo cual nos facilita mucho a la hora de programar y tanto para la usabilidad del usuario, y el manejo de todas las cosas que nos piden.
4. Es una opción que descartaríamos porque al mostrarlo en consola, puede mostrar algunas funcionalidades, pero no todas entre ellas la búsqueda sensitiva.

Agregar los usuarios

1. Es la solución óptima, ya que al implementar el árbol avl, heredado de árbol binario este cuenta con sus pruebas de búsqueda, inserción.
2. Es una solución viable, pero esta no cumpliría con los criterios de la rúbrica, aunque puede ser más optima en tiempo de ejecución, pero no cumple con los requerimientos.
3. El arbol rojo y negro, nos brinda una solución optima para este problema de la generación de la base de datos, pero conocemos poco de este lo cual es una desventaja a la ahora de implementarlo.

5. Evaluación de alternativas y escogencia de la mejor solución.

Escala de 1 a 5

Alternativa	FACTIBLE	EFICIENTE	USO Y MANEJO	CONOCIMIENTO DEL TEMA	COMPLEJIDAD	INTEGRAL	Total/Validada
Generación de la base de datos							
Alternativa 1	3	5	3	4	5	2	22
Alternativa 2	4	3	4	4	4	5	24 Validada
Generación de la imagen							
Alternativa 1	4	3	4	5	4	3	23 Validada
Alternativa 2	4	4	5	2	3	2	20
Alternativa 3	4	2	4	4	3	3	20
Digitar Texto y busqueda							
Alternativa 1	5	5	4	3	5	5	27 Validada
Alternativa 2	3	2	2	4	5	4	20
Barra de Progreso							
Alternativa 1	4	5	5	4	3	5	26 Validada
Alternativa 2	3	3	5	4	4	3	22
Alternativa 3	2	3	4	4	1	3	17
Diseño de la interfaz							
Alternativa 1	2	2	2	1	5	3	15
Alternativa 2	3	3	2	3	4	3	18
Alternativa 3	5	5	4	5	3	5	27 Validada

Para seleccionar las ideas mas viables, se tuvieron en cuenta factores como si era viable y complejo de realizar, si el uso de esta solución era eficiente, además del uso y el manejo del tema, por otro lado, también la complejidad que este requiere, y si esta solución era integral, abarcando los requerimientos.

La generación de la base de datos se escogió la solución donde tendríamos que desarrollar un método para cada característica para unirlos a cada usuario, tal como la fecha de cumpleaños, la nacionalidad y demás características, en consecuencia, obtendremos una base de datos gigante cumpliendo con los requerimientos, esta generación de usuarios se puede ejecutar en segundo plano, para mejorar su tiempo de ejecución.

Por otro lado, para la generación de la imagen como bonus, es realizando una petición https para obtener la imagen cada vez que se busca, obteniendo así está en la interfaz.

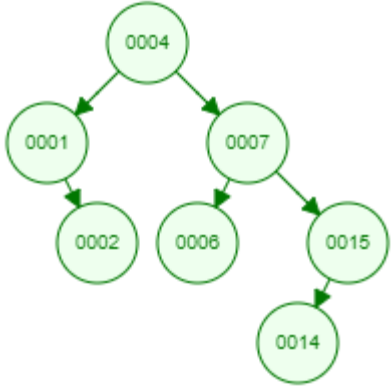
Otro lado importante que se vio para desarrollar esta solución sobre la búsqueda y el digitar es basada en arreglos, ordenarlos y buscarlos mediante la tabla, lo cual cumpliría con el requerimiento que nos piden, esta se realizaría en segundo plano.

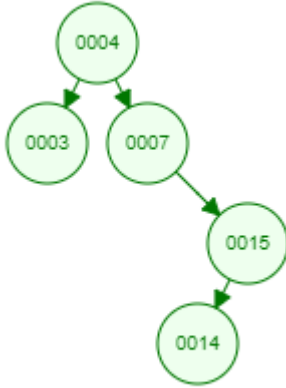
En adición, la barra de progreso la podemos hacer mediante scenebuilder, y el uso de hilos, y por último el diseño de la interfaz mediante scenebuilder el cual brinda una usabilidad al usuario más usado.

Diseño de casos de pruebas del árbol BST

Escenarios

Nombre	clase	Escenario
Setup 1	TestBST	"ARBOL VACIO "

Nombre	clase	Escenario
Setup 2	TestBST	 <pre> graph TD 0004((0004)) --> 0001((0001)) 0004 --> 0007((0007)) 0001 --> 0002((0002)) 0007 --> 0006((0006)) 0007 --> 0015((0015)) 0015 --> 0014((0014)) </pre>

Nombre	clase	Escenario
Setup 3	TestBST	 <pre> graph TD 0004((0004)) --> 0003((0003)) 0004 --> 0007((0007)) 0007 --> 0015((0015)) 0015 --> 0014((0014)) </pre>

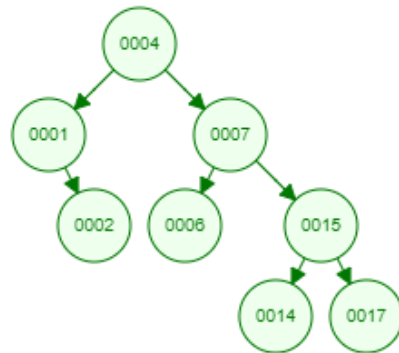
Agregar

Agregar un nodo por la derecha

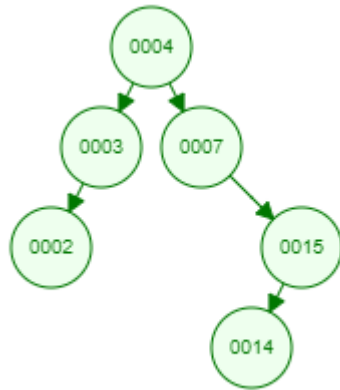
Agregar un nodo por la izquierda

Agregar un nodo cuando no hay nodos en el árbol

Objetivo de la prueba	Añadir correctamente un nodo en el árbol por la derecha.			
Clase	método	escenario	entrada	resultado
TestBST	insert	Setup 2	17	El nodo se inserta correctamente en el árbol por la derecha.



Objetivo de la prueba	Añadir correctamente un nodo en el árbol por la izquierda.			
Clase	método	escenario	entrada	resultado
TestBST	insert	Setup 3	2	El nodo se inserta correctamente en el árbol por la izquierda.



Objetivo de la prueba	Añadir correctamente un nodo en el árbol, este se agrega como raíz.			
Clase	método	escenario	entrada	resultado
TestBST	insert	Agregar un nodo en el árbol. Setup1	15	El nodo se inserta correctamente en el árbol vacío.

Eliminar

Eliminar el nodo raíz

Objetivo de la prueba	Eliminar nodo raíz			
Clase	método	escenario	entrada	resultado
TestBST	deleteTest1	Setup2	0004	True, nodo correctamente eliminado

Eliminar el nodo derecho

Objetivo de la prueba	Eliminar ultimo nodo derecho			
Clase	método	escenario	entrada	resultado
TestBST	deleteTest2	Setup2	0015	True, nodo correctamente eliminado

Eliminar el nodo izquierdo

Objetivo de la prueba	Eliminar ultimo nodo izquierdo			
Clase	método	escenario	entrada	resultado
TestBST	deleteTest3	Setup3	0003	True, nodo correctamente eliminado

Buscar

Buscar un nodo que este en el árbol

Objetivo de la prueba	Buscar el nodo indicado en arbolBST			
Clase	método	escenario	entrada	resultado

TestBST	search	Setup2	3	TRUE.
---------	--------	--------	---	-------

Buscar un nodo que no existe en el árbol

Objetivo de la prueba	Buscar el nodo que no existe en el arbolBST			
Clase	método	escenario	entrada	resultado
TestBST	search	Setup2	10	La prueba falla, el resultado sería FALSE

Diseño de los casos de prueba Arbol AVL

Escenario de pruebas

Nombre	clase	Escenario
Setup 1	TestAVLTree	" ARBOL VACIO "

Nombre	clase	Escenario
Setup 2	TestAVLTree	1,"1234"

Nombre	clase	Escenario
Setup 3	TestAVLTree	"1,2", "1"

Nombre	clase	Escenario
--------	-------	-----------

Setup 4	TestAVLTree	1, "01" - 2, "02" - 5, "05"
---------	-------------	-----------------------------

Diseño de casos de pruebas ArbolAVL

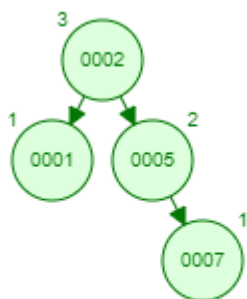
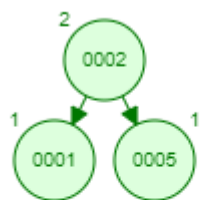
Objetivo de la prueba	Añadir correctamente un nodo en el árbol.			
Clase	método	escenario	entrada	resultado
AVLTree	insert	Agregar un nodo en el árbol. Setup1	1, "1234"	El nodo se inserta correctamente en el árbol vacío.

Objetivo de la prueba	Agregar un elemento duplicado al árbol			
Clase	método	escenario	entrada	resultado
AVLTree	insert	Setup2	1, "1234"	Mensaje de error, que el elemento no se puede agregar.

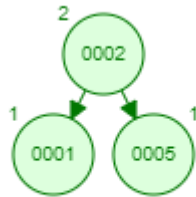
Objetivo de la prueba	Agregar un elemento que no corresponde al árbol			
Clase	método	escenario	entrada	resultado
AVLTree	Insert	Setup3	"1,2", "1"	El elemento para insertar no corresponde al formato establecido.

Objetivo de la prueba	Añadir correctamente un nodo en el árbol por la derecha.			
Clase	método	escenario	entrada	resultado

AVLTree	insert	Setup 4	7,"07"	El nodo se inserta correctamente en el árbol por la derecha quedando balanceado.
---------	--------	---------	--------	--



Objetivo de la prueba	Añadir correctamente un nodo en el árbol por la izquierda.			
Clase	método	escenario	entrada	resultado
AVLTree	insert	Setup 4	2,"02"	El nodo se inserta correctamente en el árbol por la izquierda quedando balanceado.



Escenarios de pruebas unitarias método buscar

Nombre	clase	Escenario
Setup 1	TestAVLTree	1,"01" 2,"02" 3,"03" 4,"04"

Nombre	clase	Escenario
Setup 2	TestAVLTree	1,"01" 2,"02" 4,"04"

Objetivo de la prueba	Buscar el nodo indicado en arbolAVL			
Clase	método	escenario	entrada	resultado
AVLTree	search	Setup1	3,"03"	TRUE.

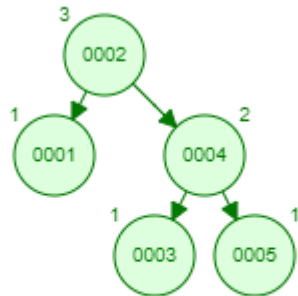
Objetivo de la prueba	Buscar el nodo que no existe en el arbolAVL.			
Clase	método	escenario	entrada	resultado
AVLTree	search	Setup2	3,"03"	La prueba falla, el resultado sería FALSE

Eliminar

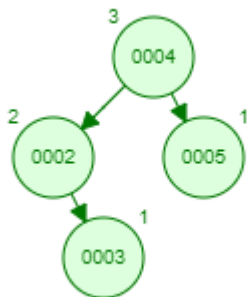
Estado inicial del árbol

Nombre	clase	Escenario
Setup 1	TestAVLTree	1,"01" 2,"02" 3,"03" 4,"04" 5,"05"

Eliminar el nodo 1 del árbol



Objetivo de la prueba	Eliminar el nodo izquierdo del árbol			
Clase	método	escenario	entrada	resultado
AVLTree	deleteTest1	Setup1	1,"01"	TRUE



Objetivo de la prueba	Eliminar el ultimo nodo derecho del árbol			
Clase	método	escenario	entrada	resultado
AVLTree	deleteTest2	Setup1	5,"05"	TRUE

Eliminar la raíz

Objetivo de la prueba	Eliminar la raíz del árbol			
Clase	método	escenario	entrada	resultado
AVLTree	deleteTest3	Setup1	1,"001"	TRUE

Eliminar si solo tiene un hijo

Objetivo de la prueba	Eliminar si el nodo solo tiene un solo hijo			
-----------------------	---	--	--	--

Clase	método	escenario	entrada	resultado
AVLTree	deleteTest4	Setup1	3,"03"	TRUE

Eliminar si tiene dos hijos

Objetivo de la prueba	Eliminar si el nodo tiene dos hijos			
Clase	método	escenario	entrada	resultado
AVLTree	deleteTest5	Setup1	4,"004"	TRUE

Diseño de casos de pruebas del modelo

Agregar un usuario

- Cuando no hay usuarios en la BD
Resultado: Se agrega correctamente
- Cuando ya hay usuarios en la BD
Resultado: se agrega junto las que ya están

- Cuando el usuario a agregar no tiene bien todas las características
- Cuando el usuario agregado, este duplicado

Buscar un usuario

- Cuando el elemento a buscar coincide
- Cuando el elemento a buscar no coincide
- Cuando no haya usuarios en la base de datos

Eliminar un usuario

- Cuando se elimina correctamente
- Cuando no hay un usuario a eliminar

Actualizar un usuario

- Cuando toca cambiar una característica del usuario registrado en la BD

ESCENARIOS

Nombre	clase	Escenario
Setup 1	TestBD	No hay usuarios

Nombre	clase	Escenario
Setup 2	TestBD	<p>User1= {"0001","Sebastian","Morales", "Male",1.65,"Colombia", 07/2/2001,"https: miimage.com"}}</p> <p>User2= {"0002","Cristian","Morales", "Male",1.75,"Colombia", 04/3/1999,"https: miimage.com"}}</p> <p>User3= {"0003","Santiago","Hurtado", "Male",1.80,"Brazil", 04/3/1998,"https: miimage.com"}}</p>

Objetivo de la prueba	Agregar un usuario cuando no hay usuarios en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	<u>addUserTest</u>	Setup1	"0001"," Sebastian", "Morales", "Male",1.65,"Colombia", 07/2/2001, "https: miimage.com"	Usuario agregado correctamente

Objetivo de la prueba	Agregar un usuario cuando hay usuarios en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	<u>addUserTest2</u>	Setup2	"0001","Andres","aristi", "Male",1.70,"Colombia", 07/18/1980, "https: miimage.com"	Usuario agregado correctamente en el sistema

Objetivo de la prueba	Verificar la información de un usuario cuando se va a agregar el usuario al sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	<u>addUserTest3</u>	Setup2	"0001", 11122,"aristi", "Male","170","Colombia", 07/18/1980, "https: miimage.com" Nombre incorrecto Y estatura	El Usuario no fue agregado correctamente en el sistema

BUSCAR UN USUARIO

Objetivo de la prueba	Buscar un usuario por código en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	<u>searchUserTest2</u>	Setup2	"0001"	User1= { "0001","Sebastian","Morales", "Male",1.65,"Colombia", 07/2/2001, "https: miimage.com" }

				True
--	--	--	--	------

Objetivo de la prueba	Buscar un usuario por nombre en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	searchUserTest2	Setup2	"Sebastian"	User1={ "0001","Sebastian","Morales", "Male",1.65,"Colombia", 07/2/2001,"https: miimage.com"} True

Objetivo de la prueba	Buscar un usuario por apellido en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	searchUserTest3	Setup2	"Hurtado"	User3={ "0003","Santiago","Hurtado", "Male",1.80,"Brazil", 04/3/1998,"https: miimage.com"} True

Objetivo de la prueba	Buscar un usuario cuando no existan usuarios en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	searchUserTest4	Setup1	0002	No existen usuarios en la base de datos

Objetivo de la prueba	Buscar un usuario cuando los elementos a buscar no coinciden			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	searchUserTest5	Setup2	"0010"	El usuario no está registrado en el sistema

ELIMINAR UN USUARIO

Objetivo de la prueba	Eliminar un usuario cuando no existen usuarios en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	DeleteUserTest	Setup1	"0001"	No hay usuarios en el sistema por lo tanto no se puede eliminar

Objetivo de la prueba	Eliminar un usuario por código cuando existen usuarios en el sistema			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	DeleteUserTest2	Setup2	"0001"	True Usuario eliminado correctamente

ACTUALIZAR USUARIO

Objetivo de la prueba	Actualizar el nombre de un usuario			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	UpdateUser	Setup2	NOMBRE="JUAN"	"0001","juan","Morales", "Male",1.65,"Colombia", 07/2/2001, "https: miimage.com"

Objetivo de la prueba	Actualizar dos características de un usuario			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	UpdateUser2	Setup2	NOMBRE="JUAN" Estatura = 1,70	"0001","juan","Morales", "Male",1.70,"Colombia", 07/2/2001, "https: miimage.com"

Objetivo de la prueba	Actualizar información de un usuario con características incorrectas			
Clase	método	escenario	entrada	resultado
<u>DataBase</u>	UpdateUser2	Setup2	NOMBRE=92029 Estatura = 1,70	No se puede actualizar información del usuario datos no validos.

Fuentes:

1. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/>
2. https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro
3. https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda_auto-balanceable
4. https://es.wikipedia.org/wiki/Rotaci%C3%B3n_de_%C3%A1rboles
5. <https://es.slideshare.net/marhoz/arbol-avl-codigo-java>

