Software de gestión de producción con enfoque en la calidad del producto

Fabian Galindo, Sebastián Hernández, David Viracachá
No. de Equipo Trabajo: 9

I. INTRODUCCIÓN

La calidad es un parámetro fundamental que debe estar presente en cualquier bien o producto de consumo. Esta se define por la norma ISO 9000 como el grado en el que las características específicas de un objeto cumplen con un conjunto de requisitos. Teniendo en cuenta que la calidad puede estar implícita en muchos aspectos como los proveedores de las materias primas, los procesos de producción y su distribución, surge la necesidad de encontrar formas de garantizarla. Para este propósito, se han adquirido distintas metodologías a nivel mundial las cuales requieren de, entre otras acciones, de obtener, monitorear y procesar información en distintas etapas clave del proceso de producción; por esta razón la informática puede ser especialmente útil en términos de la eficiencia y confiabilidad con la que se maneja este conjunto de datos, así como en su presentación a un empresario que busca asegurar la calidad de su producto.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

Las empresas productoras deben asegurar la calidad de sus productos para cumplir con estándares gubernamentales e internacionales que garantizan la seguridad del consumidor, sin mencionar que, también, la calidad mantiene a un producto competitivo en el mercado. Este ejercicio requiere de un manejo adecuado y preciso de información relacionada al proceso de producción, por esta razón, se busca implementar un software cuyo propósito general es monitorear este conjunto de procesos y factores, brindando así una visión general de la producción que facilite la toma de decisiones destinadas a lograr la calidad del producto, además de permitir la exportación de la información obtenida para simplificar el proceso de certificación de calidad ante los entes de control respectivos.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

Habrá tres tipos de usuario.

Administrador y operario
Tiene acceso completo a las funcionalidades, es decir, puede crear y gestionar producciones, agregar materias primas y crear registros.
Puede acceder a la información del proceso y avanzar en sus etapas.

Estándar
Puede ver el estado de las producciones y generar registros de ser necesario.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Crear producción:

Se realiza la creación de una producción, con sus respectivas etapas, parámetros de calidad y materias primas. Un usuario accede a la funcionalidad a partir de un menú, se le facilita al usuario una plantilla para que ingrese los datos de manera correcta. La producción se guardará en una lista de producciones.
Requerimientos funcionales:
Permitirle al usuario ingresar información para llevar a cabo la creación una producción, sus etapas de producción, parámetros y materias primas.

Crear registro:

Se realiza un resumen de la producción que se haya escogido.
El usuario puede ver todos los detalles la respectiva producción.
Este registro no se puede editar.
Requerimientos funcionales:
Se obtienen los datos de la clase producción y se debe exportar a un formato legible para el usuario.

Gestionar producción:

Se le brinda al usuario una interfaz en la que se muestra una visión general de la producción y sus etapas y en donde podrá seleccionar entre diferentes acciones para realizar sobre su producción como abrir y cerrar etapas, generar reportes parciales o añadir comentarios.
Requerimientos funcionales.
Se debe seleccionar una producción, imprimir su estado general y dar la capacidad a un usuario de ejercer acciones sobre la producción.

Validar parámetros de calidad:

El valor ingresado debe ser comparado con los rangos establecidos previamente, si el valor está en el rango cumple con el parámetro, de lo contrario no cumple, y se le informa al usuario.
Requerimiento funcional:
El usuario ingresa un valor, se le notifica si cumple con el parámetro. De lo contrario, no podrá avanzar, se obliga al usuario a revisar el proceso de producción y a corregirlo.

Crear Usuario:

Se realiza la creación del usuario. Una persona que no posea un usuario tiene la opción de la creación de un usuario. Se le pide al usuario ingresar algunos datos y una contraseña. Se realiza esta acción una única vez. Si el usuario ya existe no se permite la acción. Se pueden crear usuarios que tengan diferentes permisos para realizar acciones dentro del programa
Requerimientos funcionales:
Se debe crear un usuario y agregarlo en la base de datos de usuarios. Se debe comprobar que el usuario creado no exista previamente. Se debe asignar un rol al usuario.

V.DESCRIPCIÓN DE LA INTERFAZ DE USUARIO FINAL

Se realiza el mockup del proyecto utilizando la herramienta Balsamiq. Con ayuda de esta herramienta se planteó como se vería el programa y de igual manera de implemento como se muestra en las imágenes. Todas las funcionalidades mostradas funcionan correctamente. Para la tercera entrega se dejo la misma base que la segunda, y se realizaron unos retoques estéticos además de la implementación de nuevas estructuras en las funcionalidades como se explicara más adelante.
En las imágenes se muestra algunas funcionalidades del Software de la siguiente manera:

- Inicio de sesión (figura 1)
- Panel de control (figura 2)
- Crear producción (figura 3)
- Buscador de producción (figura 4)
- Gestión de producciones (figura 5)
- Gestión de parámetros (figura 6)
- Creación de registro de producción (figura 7)
- Perfil de usuario (figura 8)

Figura 1 Iniciar sesión

Figura 2 Panel de control

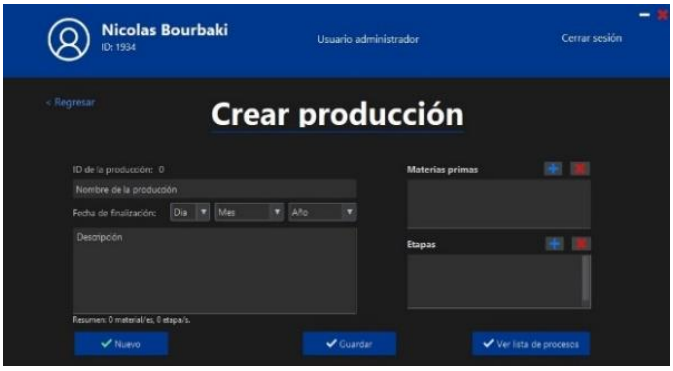


Figura 3 Crear producción

Figura 4 Buscador de producción

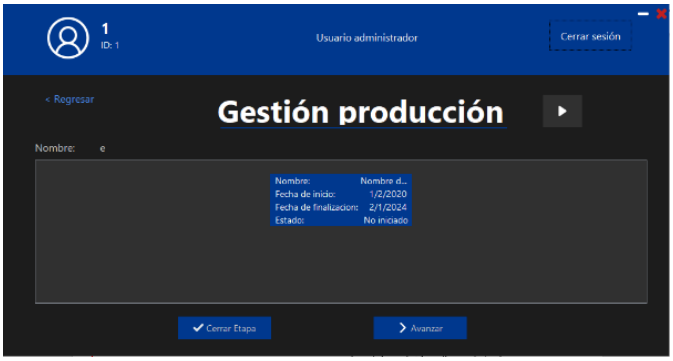


Figura 5 Gestión de producciones

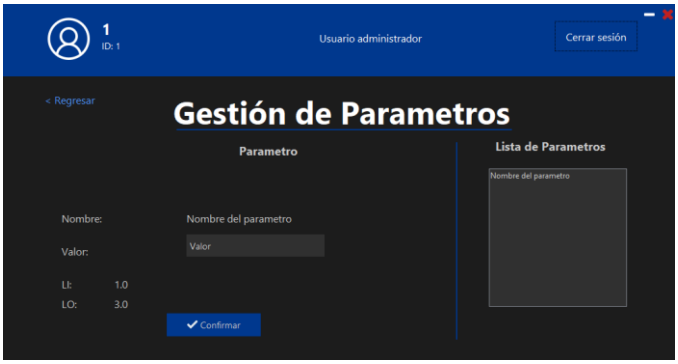


Figura 6 Gestión de parámetros

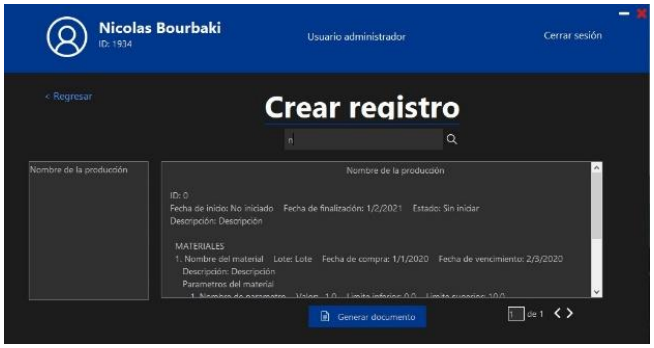


Figura 7 Creación de registro de producción

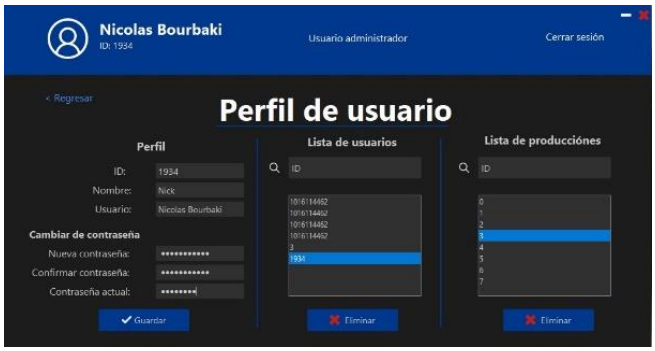


Figura 8 Perfil de usuario

VI. Entornos de desarrollo y de operación

El software se desarrollada en NetBeans 11.0 o mayor e Intellij IDEA en el lenguaje de programación Java 14.0.1(Java HotSpot(TM) 64-Bit Server VM 14.0.1+7). También se usa la plataforma GitKraken para comparar los aportes de cada integrante del equipo de trabajo y decidir qué elementos pasaran a ser parte de la versión del prototipo. Adicional, esta plataforma permite almacenar el proyecto en un repositorio de GitHub con el fin de que cada integrante tenga el código software actualizado.

A continuación, se dan a conocer los requisitos mínimos del sistema para el software:

Sistema Operativo: Windows 10 para 64 bits.
Procesador: Procesador a 3.0 GHz o más rápido.
RAM: 4GB.
Espacio en disco duro: 100 MB.
Tarjeta gráfica: DirectX 9 o posterior con controlador WDDM 1.0
Pantalla:1024x600

VII. PROTOTIPO DE SOFTWARE INICIAL

Para esta entrega se manejó el siguiente repositorio de GitHub: https://github.com/fsgalindope/Proyecto_estructuras_de_datos_2020-1_grupo_9/tree/Tercera_Entrega

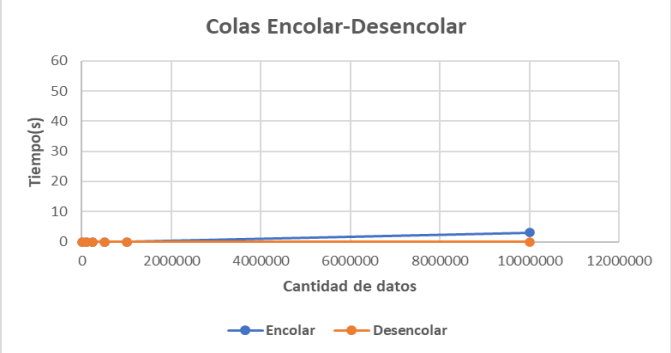
VIII. PRUEBAS DEL PROTOTIPO Y ANALISIS COMPARATIVO

Pruebas Primera Entrega:

Se escogió 3 estructuras de datos que se adaptan mejor a las funcionalidades del proyecto que están ahora planteadas, estas son colas, lista de arreglos y listas enlazadas.
Se realizaron pruebas a las siguientes funcionalidades por estructura de datos.
- Colas: encolar y desencolar n valores, mostradas en la tabla 1 y grafica 1.
- Listas enlazadas y lista de arreglos: Insertar n valores al final, eliminar n valores del final, buscar un valor al final de una lista de tamaño n, mostradas en las tablas 2, 3, 4, y las gráficas 2, 3, 4, respectivamente.

Colas de Materias primas		
Datos	Tiempo(s)	
	Encolar	Desencolar
10000	0	0
100000	0	0
250000	0	0
500000	0	0
1000000	0	0
10000000	3	0

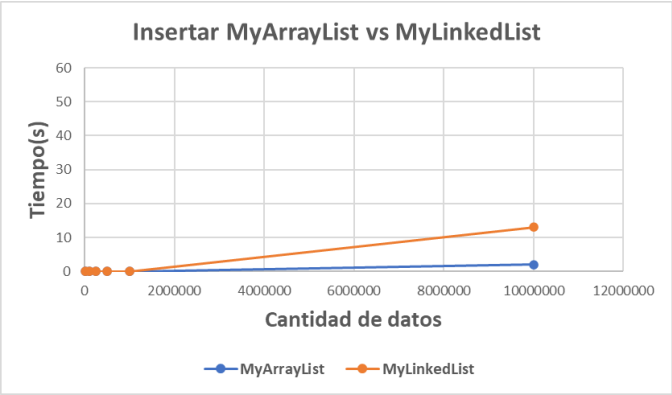
Tabla 1 Colas de materias primas.



Grafica 1 Colas Encolar-Desencolar

Insertar MyArrayList vs MyLinkedList		
Datos	Tiempo(s)	
	MyArrayList	MyLinkedList
10000	0	0
100000	0	0
250000	0	0
500000	0	0
1000000	0	0
10000000	2	13

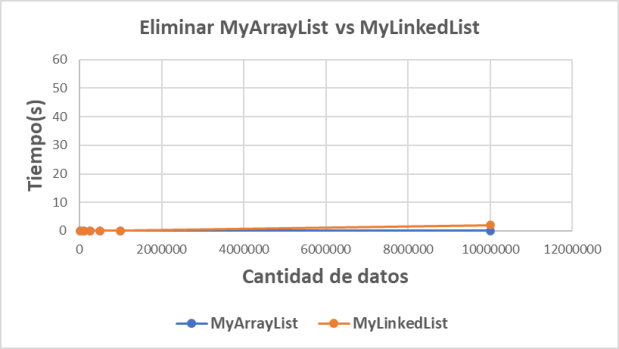
Tabla 2 Insertar MyArrayList vs MyLinkendList



Grafica 2 Insertar MyArrayList vs MyLinkendList

Eliminar MyArrayList vs MyLinkedList		
Datos	Tiempo(s)	
	MyArrayList	MyLinkedList
10000	0	0
100000	0	0
250000	0	0
500000	0	0
1000000	0	0
10000000	0	2

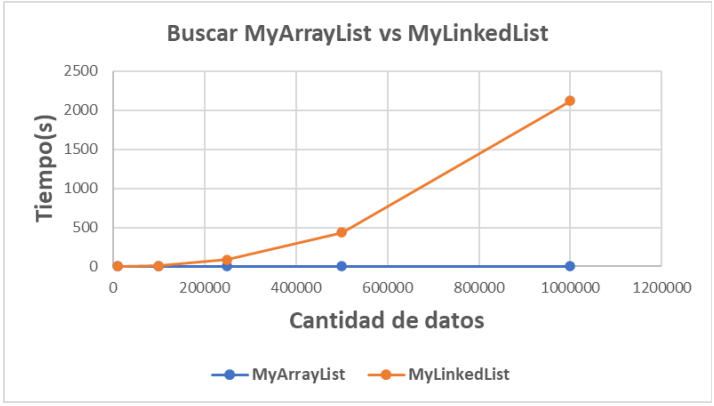
Tabla 3 Eliminar MyArrayList vs MylinkedList



Grafica 3 Eliminar MyArrayList vs MyLinkendList

Buscar MyArrayList vs MyLinkedList		
Datos	Tiempo(s)	
	MyArrayList	MyLinkedList
10000	0	0
100000	0	8
250000	0	89
500000	0	434
1000000	0	2118

Tabla 4 Tabla 4. Buscar MyArrayList vs MyLinkendList



Grafica 4 Buscar MyArrayList vs MyLinkendList

Análisis Pruebas Primera Entrega:

De la tabla 1, se puede concluir que el gasto computación de encolar y desencolar es de O(1).
Por otro lado, la funcionalidad de insertar a arrayList y linkedList mostrada en la tabla 2 y en la gráfica 2, se puede observar que también es de O(1), sin embargo, linkedList tiene un aumento considerable al ingresar valores grandes, cuando se realizaron pruebas con valores mayores a los mostrados en las tablas, la consola envió un error de memoria. Por esto, concluimos que el aumento del tiempo en la linkedList para valores altos es por su gasto en memoria ya que la maquina se demora en encontrar espacio disponible y llegado el caso en retornar un error. También se debe tener en cuenta que se realizaron las pruebas con un computador con 20GB de memoria RAM así que se forzó el error realizando pruebas para más de 10 millones de datos. En la gráfica 3, y tabla 3, eliminar el último elemento se comportó como esperábamos con un gasto computacional constante en ambas estructuras (MyArrayList y MyLinkedList) de O(1).
Por otro lado, buscar en elemento en la lista como se muestra en la tabla 4 y la gráfica 4, se comportó de una manera muy distinta, el arreglo dinámico tuvo un comportamiento constante (O(1)) mientras que la lista enlazada se comporta de manera cuadrática ($O(n^2)$), analizando el código, la lista enlazada que se utilizó y la forma que se realizó la búsqueda se puede decir que al llamar un método que se utiliza en el momento de la búsqueda, se realiza internamente n iteraciones por n comparaciones, situación que se corrigió para las siguientes entregas.
Teniendo en cuenta este análisis y las funcionalidades del proyecto planteadas se puede estar seleccionando la lista que mejor se acomode a estas variables. Se ira revisando con forme se vaya avanzando en el proyecto.

Pruebas Segunda Entrega:

En la primera entrega se planteó usar entre lista enlazadas o listas implementadas por arreglos dinámicos para almacenar usuarios y producciones, lo cual como no se van a realizar eliminaciones se escoge los arreglos dinámicos.
Para esta entrega se implementó una nueva estructura de datos que es el árbol AVL. El cual se va a probar las siguientes funcionalidades:

- Insertar n elementos al árbol (grafica 5 y tabla 5).
- Eliminar una hoja, Inicio de sesión (grafica 6 y tabla 6).
- Encontrar el mínimo de un árbol de n elementos (grafica 7 y tabla 7).
- Encontrar el máximo de un árbol de n elementos (grafica 8 y tabla 8).
- Encontrar un elemento aleatorio de un árbol de n elementos (grafica 9 y tabla 9).

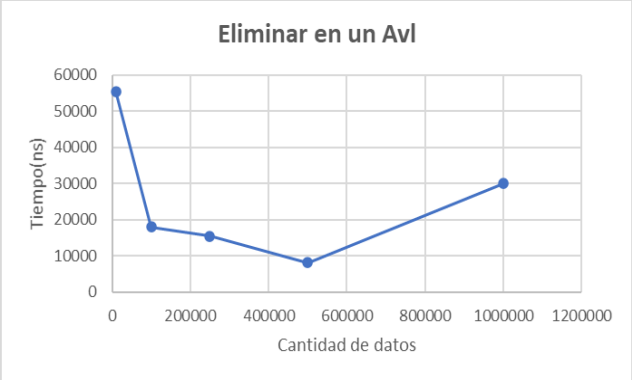
Grafica 5. Insertar en un AVL Datos vs Tiempo(ms)

Insertar en un Avl	
Datos	**Tiempo(ms)**
10000	542
100000	107142
250000	683697
500000	2920283
1000000	11879777
10000000	No aplica

Tabla 5.Insertar en un AVL Datos vs Tiempo(ms)

Eliminar en un Avl	
Datos	**Tiempo(ns)**
10000	55400
100000	18000
250000	15500
500000	8100
1000000	30000
10000000	No aplica

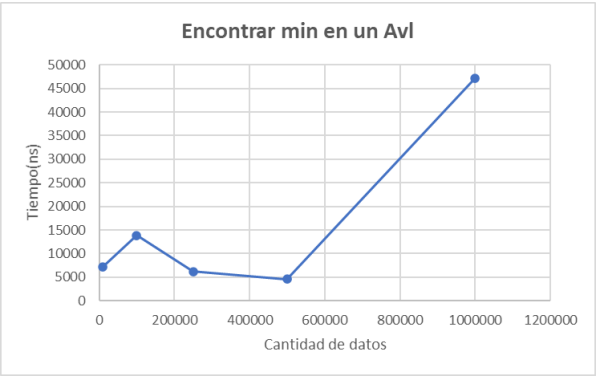
Tabla 6. Eliminar un elemento en un AVL Datos vs Tiempo(ns).



Grafica 6 Eliminar un elemento en un AVL Datos vs Tiempo(ns).

Encontrar min en un Avl	
Datos	**Tiempo(ns)**
10000	7200
100000	13900
250000	6200
500000	4600
1000000	47200
10000000	No aplica

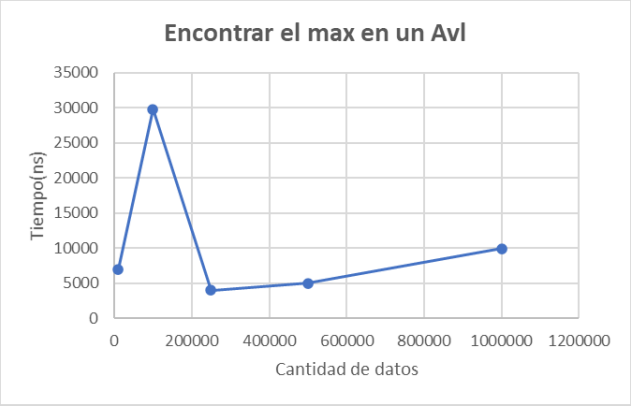
Tabla 7 Encontrar el min elemento del AVL Datos vs Tiempo(ns).



Grafica 7 Encontrar el min elemento del AVL Datos vs Tiempo(ns).

Encontrar el max en un Avl	
Datos	**Tiempo(ns)**
10000	7000
100000	29800
250000	4000
500000	5000
1000000	9900
10000000	No aplica

Tabla 8 Encontrar el max elemento del AVL Datos vs Tiempo(ns).



Grafica 8 Encontrar el max elemento del AVL Datos vs Tiempo(ns).

Encontrar aleatorio en un Avl	
Datos	**Tiempo(ms)**
10000	45300
100000	17500
250000	9900
500000	2400
1000000	6500
10000000	No aplica

Tabla 9 Encontrar un elemento aleatorio del AVL Datos vs Tiempo(ns).

Grafica 9 Encontrar un elemento aleatorio del AVL Datos vs Tiempo(ns).

No se realizó pruebas para un árbol de búsqueda binario porque el balance del árbol AVL permite asegurar que tendrá un mejor comportamiento es gasto computacional.
El árbol AVL de producciones se utiliza en el programa para hacer una búsqueda a partir del nombre de producciones para que sea eficiente al generar registros.

Análisis Pruebas Segunda Entrega:

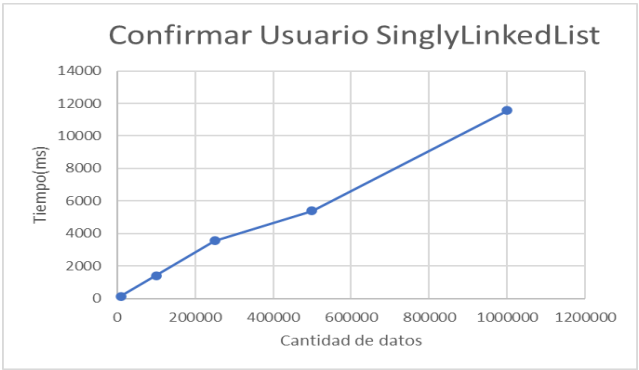
En la gráfica 5. Insertar en un AVL Datos vs Tiempo(ms)
Se puede observar que se comporta de manera casi cuadrática. Pero haciendo un análisis más preciso se puede concluir que el gasto computacional es O (n *$\log_2$ n), $\log_2$ n por cada inserción y n que es la cantidad de datos que se ingresa. Es la funcionalidad que más gasto tiene computacionalmente. Sin embargo, para este caso no se tiene problema ya que nunca se realiza una inserción de una cantidad n de elementos en un mismo momento.
Las gráficas 6, 7, 8 y 9 se encuentran a escalas de nanosegundo inclusive para un árbol con 1 millón de datos, esto es bueno en nuestra implementación que se basa en la búsqueda de elementos. Las gráficas presentan un comportamiento algo diferente a lo esperado que es un O ($\log_2$ n) pero si se aumenta la escala no se observa un crecimiento mayor al esperado.
Al final se implementó el árbol AVL para realizar búsquedas por nombre de producciones.

Pruebas Entrega Final:

En esta entrega se implementó las tablas hash para remplazar las listas enlazadas utilizadas en la gestión de usuarios, con el objetivo de disminuir el tiempo de comparación de usuarios al iniciar sesión o crear un nuevo usuario. Para esto se realiza las pruebas de comparación de usuarios para la lista enlazadas SinglyLinkedList y la tabla hash MyHashTable, y además de algunas otras funcionalidades de la tabla hash, como insertar o eliminar un elemento de la tabla hash. En la tabla 10 y gráfica 10 Confirmar Usuario SinglyLinkedList Datos vs Tiempo(ms) se observa el comportamiento de la comparación de cada usuario en la lista enlazada, y en la gráfica 11, y tabla 11 Confirmar Usuario Hash Datos vs Tiempo(ns), en la tabla hash, mientras que en las gráficas y tablas 12 y 13, se observa las demás funcionalidades de la tabla hash, como insertar y eliminar.

Confirmar usuario SinglyLinkedList	
Datos	**Tiempo(ms)**
10000	146
100000	1410
250000	3558
500000	5386
1000000	11558
10000000	139917

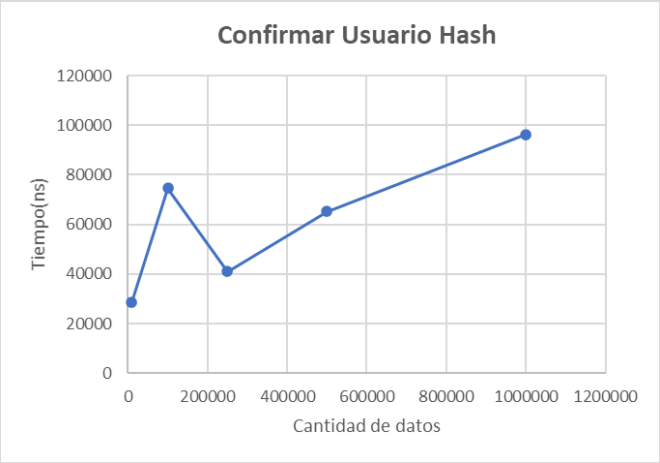
Tabla 10 Confirmar Usuario SinglyLinkedList Datos vs Tiempo(ms)



Grafica 10 Confirmar Usuario SinglyLinkedList Datos vs Tiempo(ms)

Confirmar Usuario Hash	
Datos	**Tiempo(ns)**
10000	28700
100000	74600
250000	41000
500000	65200
1000000	96200
10000000	12501

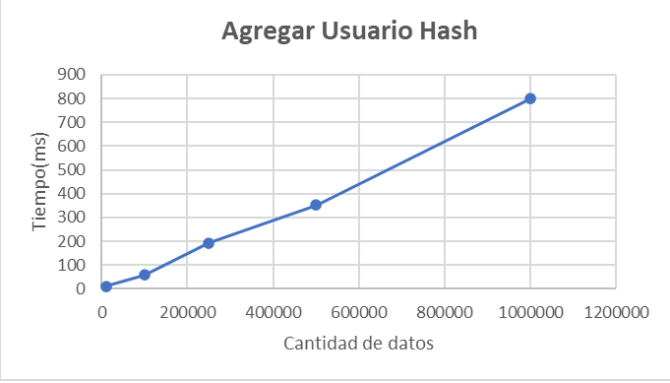
Tabla 11 Confirmar Usuario Hash Datos vs Tiempo(ns)



Grafica 11 Confirmar Usuario Hash Datos vs Tiempo(ns)

Agregar Usuario Hash	
Datos	**Tiempo(ns)**
10000	12
100000	58
250000	193
500000	353
1000000	798
10000000	8306

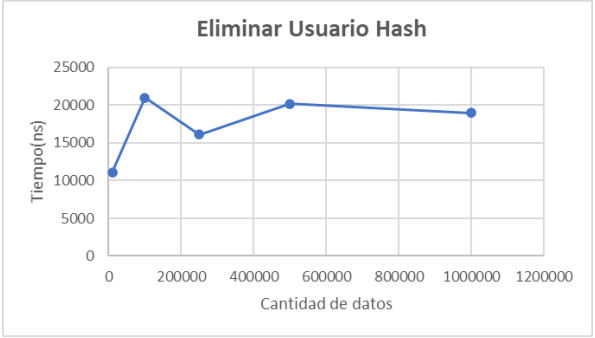
Tabla 12 Agregar Usuario Hash Datos vs Tiempo(ms)



Grafica 12 Agregar Usuario Hash Datos vs Tiempo(ms)

Eliminar Usuario Hash	
Datos	**Tiempo(ns)**
10000	11100
100000	20999
250000	16100
500000	20201
1000000	19000
10000000	16399

Tabla 13 Eliminar Usuario Hash Datos vs Tiempo(ns)



Grafica 13 Eliminar Usuario Hash Datos vs Tiempo(ns)

Análisis Pruebas Entrega Final:

Como se muestra en la gráfica 10 Confirmar Usuario SinglyLinkedList Datos vs Tiempo(ms). Se puede observar que el gasto computacional de la comparación de usuario es O(n), es el valor esperado pues tienen que comparar si el dato usuario ingresado es igual al usuario enésimo que se está comparando, el peor de los casos es que tenga que realizar la comparación n veces. Por otro lado, en la gráfica 11 Confirmar Usuario Hash Datos vs Tiempo(ns) se observa que el comportamiento es constante (O (1)), este es uno de los beneficios de usar hash. Se pueden ver algunas pequeñas diferencias en los tiempos, esto se puede explicar porque existe la posibilidad de producirse colisiones y se deba hacer la comparación en una lista para verificar que el usuario existe, sin embargo, son casos excepcionales y que no producen mayor gasto computacional. Otras funcionalidades se ven en la gráfica 12 Agregar Usuario Hash Datos vs Tiempo(ms) donde se puede apreciar que la inserción de n datos al hash tiene un gasto computacional de O(n). Otra de las funcionalidades es la de eliminación mostrada en la gráfica 13 Eliminar Usuario Hash Datos vs Tiempo(ns) se puede observar que tiene un gasto computación constante O(1) de la misma forma que la funcionalidad comparar, depende de las colisiones en algunos casos.
De esta manera se decidió cambiar las listas enlazadas por tablas hash.

IX. ROLES Y ACTIVIDADES

Sebastian Hernández

- Roles: Coordinador, Técnico
- Actividades realizadas:
  - Programación de la interfaz gráfica: Crear

usuarios, crear y buscar producción y generar registro.
    - Desarrollo, manejo y definición de la base de datos.

David Viracachá

- Roles: Investigador, Líder
- Actividades realizadas:
    - Implementación de los árboles y árboles AVL estructuras de datos.
    - Aportaciones en la interfaz grafica
    - Aportes en la creación del mockup

Fabian Galindo

- Roles: Observador, Animador
- Actividades realizadas:
    - Programación de la interfaz gráfica: Gestor de producción
    - Edición y revisión del documento
    - Realización de las pruebas
    - Aporte en el desarrollo de las colas

X. DIFICULTADES Y LECCIONES APRENDIDAS

Durante el desarrollo del proyecto se encontraron inconvenientes con los computadores ya que generaba fallas al manejar más 1 millón de datos debido su capacidad de procesamiento.

Se aprendió a manejar herramientas que facilitan el desarrollo de software tales como GitKraken adicional se adquiere conocimiento de diferentes funciones para interfaz gráfica y mejoras visuales, entre otros.

Comprendimos la importancia de definir asertivamente la estructura de datos con el fin optimizar los procesos que se llevan a cabo en el software.

Entendimos que las definiciones recursivas de los árboles, aunque no siempre resultan sencillas de conceptualizar, resultan muy útiles en su implementación computacional. También, entendemos que existen estructuras de datos que, aunque no sean tan sencillas y/o triviales y requieran de un mantenimiento más intenso, resultan especialmente útiles en términos de efectividad al momento de realizar ciertas operaciones. Con esto también podemos decir que entre estructuras hay ciertos intercambios de memoria y rendimiento los cuales deben ser tenidos en cuenta al momento de usar cualquiera de ellas.