SIApp, una forma eficiente de planear horarios académicos

Cristian Fabian Martinez Bohórquez, Edward Jeisen Jair Arévalo Peña, Juan David Cruz Giraldo

No. de equipo de trabajo: F

I. INTRODUCCIÓN

En la universidad Nacional de Colombia, planear horarios de forma eficiente es un desafío recurrente de los estudiantes en el momento de iniciar un nuevo semestre, principalmente por las fallas que tiene la plataforma del portal de servicios académicos. Para abordar este problema surge SIApp, una aplicación diseñada para planificar horarios académicos de forma eficiente, mediante el uso de estructuras de datos. En este trabajo se presentará una descripción detallada del diseño y desarrollo de la aplicación.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

La plataforma SIA de la universidad, encargada de proporcionar información vital como historial académico, datos generales, disponibilidad de materias y trámites educativos, enfrenta frecuentes caídas y fallos, especialmente en fechas críticas para los estudiantes. Esta situación dificulta enormemente la planificación adecuada de horarios y puede provocar estrés adicional al impedir el acceso oportuno a la información necesaria. La falta de disponibilidad para acceder a la plataforma puede resultar en la incapacidad de realizar ajustes necesarios en los horarios académicos, lo que afecta negativamente el progreso de los estudiantes.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

SIApp está diseñada específicamente para estudiantes de la Universidad Nacional de Colombia.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Funcionalidad: Búsqueda de Materias

Descripción: Permitir a los usuarios buscar y visualizar información detallada sobre las materias disponibles en la universidad.

Acciones iniciales y comportamiento esperado:

1. El usuario ingresa el nombre de la materia o parte del nombre en el campo de búsqueda.
2. El sistema muestra una lista de resultados coincidentes con el término de búsqueda.
3. El usuario puede seleccionar una materia de la lista para ver información detallada como cupos disponibles, profesor asignado, horarios de clases, entre otros.
4. Si la materia es de interés para el estudiante, puede guardarla en materias favoritas.

Funcionalidad: Creación de Horario

Descripción: Permitir a los usuarios crear y personalizar su horario académico agregando materias de su interés.

Acciones iniciales y comportamiento esperado:

1. El usuario accede a la función de creación de horario desde el menú principal de la aplicación.
2. El sistema muestra una interfaz donde el usuario puede ver un horario semanal vacío con los días de la semana y franjas horarias.
3. El usuario puede seleccionar una materia de su lista de materias favoritas o buscar una nueva y agregarla al horario.
4. El sistema valida que no existan conflictos de horario con otras materias ya agregadas y muestra un mensaje de error si se detecta algún conflicto.
5. El usuario puede ajustar el horario según sus preferencias, agregar más materias o eliminar materias ya agregadas.
6. El sistema guarda automáticamente los cambios realizados en el horario y permite al usuario acceder y editar el horario en cualquier momento.

Funcionalidad: Control de horas de estudio.

Descripción: Permitir a los usuarios registrar y gestionar el tiempo dedicado al estudio de cada materia en base a los créditos correspondientes.

Acciones iniciales y comportamiento esperado:

1. El usuario selecciona una materia de su lista de materias.
2. El usuario ingresa la cantidad de horas dedicadas al estudio de esa materia en un día específico o activa un contador que lleva la cantidad de horas que llega estudiando.
3. El sistema registra y guarda esta información para el análisis posterior del tiempo de estudio por materia y día.

Funcionalidad: Calendario Integrado

Descripción: Integrar un calendario dentro de la aplicación para que los usuarios puedan organizar sus horarios académicos y eventos importantes.

Acciones iniciales y comportamiento esperado:

1. El usuario puede visualizar un calendario mensual con eventos destacados como clases, exámenes, fechas límite de proyectos, etc.

2. El usuario puede agregar nuevos eventos al calendario, especificando la fecha, hora, título y descripción del evento.

3. El sistema muestra notificaciones de eventos próximos o recordatorios de actividades importantes.

Funcionalidad: Cálculo de promedio y seguimiento de notas.

Descripción: Permitir a los usuarios calcular su promedio académico, llevar un registro de las notas en cada materia, y conocer cuánto deben sacar en su próxima nota para pasar la materia.

Acciones iniciales y comportamiento esperado:

1. El usuario selecciona las materias cursadas y registra las calificaciones obtenidas en cada una.

2. El sistema calcula automáticamente el promedio general del estudiante, mostrando el resultado en pantalla.

3. Adicionalmente, el usuario puede seleccionar la opción “nota final para pasar” qué hará el cálculo de cuánto debe sacar en la próxima nota para pasar la asignatura.

Funcionalidad: Cola prioritaria de tareas

Descripción: Permite a los usuarios crear una cola prioritaria de tareas y parciales, lo que les permite organizar y gestionar eficientemente sus actividades académicas.

Acciones iniciales y comportamiento esperado:

1. El usuario accede a la función de cola prioritaria desde el menú principal de la aplicación.

2. El sistema muestra una interfaz donde el usuario puede ver una lista de todas las tareas y parciales pendientes.

3. El usuario puede agregar nuevas tareas o parciales especificando el nombre, la fecha de vencimiento y la prioridad de cada una.

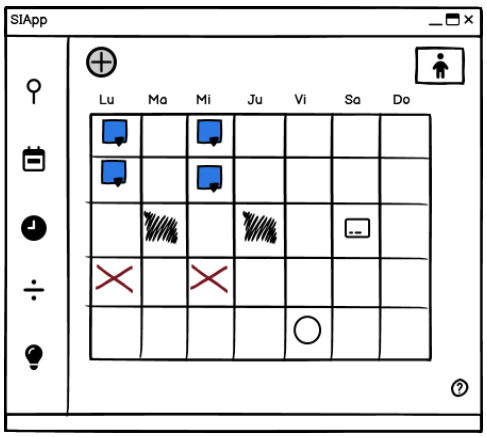
4. El sistema ordena automáticamente las tareas y parciales en la cola según su prioridad, con las tareas más urgentes en la parte superior de la lista.
5. El usuario puede editar o eliminar tareas y parciales de la cola según sea necesario.

6. El sistema notifica al usuario sobre las tareas o parciales próximos a vencerse, ayudándoles a gestionar su tiempo de manera efectiva.

7. El usuario puede marcar las tareas o parciales completadas, lo que los elimina de la cola o los mueve a una sección de tareas completadas.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

8. La interfaz de usuario de SIApp está diseñada para ser intuitiva y fácil de usar. **En la parte izquierda de la pantalla**, se encontrará un panel lateral que contiene botones para acceder a cada funcionalidad de la aplicación. Cada botón representa una funcionalidad específica, como "Buscar Materias", "Calendario", etc. Al hacer clic en uno de estos botones, se abrirá la función respectiva en el área principal de la interfaz. **En la esquina superior derecha** de la pantalla, estará ubicado un menú de configuración del usuario. para acceder a opciones como editar su nombre, correo institucional u otras preferencias de la cuenta. El resto de la interfaz está dedicada a mostrar la funcionalidad seleccionada. Por ejemplo, si estás utilizando la función de "Horario", verás una representación visual del horario de tus clases.



VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

Lenguaje: Java

Entorno de desarrollo: Para el desarrollo del software haremos uso de la IDE intelliJ, además de algunas herramientas adicionales como git.

Sistemas operativos compatibles: La aplicación debe ser compatible con windows desde una versión igual o superior a 7, linux y macOs.

Configuración específica: La aplicación será fácil de instalar y ejecutar siempre y cuando cumpla con requisitos mínimos de hardware y con tener instalado el entorno de ejecución de ja**va.**

VII. PROTOTIPO DE SOFTWARE INICIAL

El prototipo inicial de software cuenta con las siguientes funcionalidades incorporadas:

Adición, búsqueda y eliminación de asignaturas de la base de datos:
Nuestro programa permite agregar, buscar o eliminar una asignatura específica de la base de datos al proporcionar los atributos respectivos.

Creación de horario para usuario:
Se creará un horario como una lista de estructura de datos, diseñada para almacenar asignaturas con los atributos que mejor se adapten a las preferencias del usuario.

Adición y eliminación de asignaturas en el horario:
Los usuarios podrán añadir o eliminar asignaturas de su horario según su conveniencia, lo que les proporcionará un mejor panorama para la selección de materias.

El software desarrollado registra en el siguiente repositorio de github: https://github.com/cmartinezbo/ProjectDataStructures

VIII. DISEÑO, IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Estructuras de datos implementadas:

1. Binary Search Tree (BST): Para implementar el bst usamos dos clases, Node y BinarySearchTree, la clase Node se encuentra dentro de la clase BinarySearchTree y contiene los siguientes atributos y métodos:

Node:

Atributos:

- **Key;** Valor asociado al nodo.
- **Node Left:** Referencia al hijo nodo de la izquierda.
- **Node Right:** Referencia al hijo nodo de la derecha.
- **Node Parent:** Referencia al nodo padre.

Métodos:

- **toString()**: Permite imprimir el nodo directamente para saber el valor asociado al mismo.

Binary Search Tree:

Atributos:

- **Node Root:** Raiz del árbol.

Métodos:

- **Height():** Retorna la altura del árbol.
- **size():** Retorna la cantidad de nodos que tiene el árbol.
- **inOrder():** Recorre el arbol imprimiendo los nodos de forma “in-order”.
- **preOrder()**: Recorre el árbol imprimiendo los nodos en forma “pre-order”.
- **postOrder()**: Recorre el árbol imprimiendo los nodos en forma “post-order”.
- **LevelTraversal()**: Recorre el árbol imprimiendo los nodos en forma “breadth-first”.
- **Find(key)**: Verifica si un elemento ya se encuentra en el árbol.
- **leftSon:** Verifica si un nodo es hijo derecho o hijo izquierdo.
- **Next(key):** Encuentra el siguiente nodo con valor mayor mas cercano a la llave y lo retorna.
- **insert(key**): Agrega un nodo al árbol.
- **promote(Node):** Hace que un determinado nodo ocupe la posición del padre.
- **delete(key):** Devuelve el nodo que tenga ese valor asociado.

2. AVL: Para implementar el avl, hacemos uso de un objeto de dos clases, Node y AVL, la clase Node se encuentra dentro de la otra, contenindo cada una los siguientes atributos:

Node:

Atributos:

- **Key;** Valor asociado al nodo.
- **Node Left:** Referencia al hijo nodo de la izquierda.
- **Node Right:** Referencia al hijo nodo de la derecha.
- **Node Parent:** Referencia al nodo padre.

Métodos:

- **size():** Retorna la cantidad de nodos que tiene el árbol.
- **inOrder():** Recorre el arbol imprimiendo los nodos de forma “in-order”.
- **preOrder()**: Recorre el árbol imprimiendo los nodos en forma “pre-order”.
- **postOrder()**: Recorre el árbol imprimiendo los nodos en forma “post-order”.
- **LevelTraversal()**: Recorre el árbol imprimiendo los nodos en forma “breadth-first”.
- **Find(key)**: Verifica si un elemento ya se encuentra en el árbol.
- **leftSon:** Verifica si un nodo es hijo derecho o hijo izquierdo.
- **Next(key):** Encuentra el siguiente nodo con valor mayor mas cercano a la llave y lo retorna.
- **insert(key**): Agrega un nodo al árbol.
- **promote(Node):** Hace que un determinado nodo ocupe la posición del padre.

- **delete(key):** Devuelve el nodo que tenga ese valor asociado.
- **rotateRight(Node):** Realiza una rotación hacía la derecha con el nodo indicado.
- **rotateLeft(Node):** Realiza una rotación hacía la izquierda con el nodo indicado.
- **RebalanceRight(Node):** Rebalancea el árbol hacía la derecha.
- **RebalanceLeft(Node):** Rebalancea el árbol hacia la izquierda.
- **Rebalance(Node):** Determina la rotación que se debe realizar cada que se inserta un nuevo elemento.

Utilidad: El uso que le damos al avl, es el de realizar operaciones con el código de las materias de forma eficiente**,** como por ejemplo obtener las materias, eliminarlas de los horarios o buscarlas en base a su nombre.

3. Heap: Para implementar el heap, solo creamos una clase que contiene todos los atributos y métodos necesarios, los cuales son los siguientes:

Atributos:

- **Array =** Array de enteros en donde almacenamos todos los elementos del heap.
- **maxSize=**Representa el tamaño máximo que puede tener el array.
- **N:** Indica la cantidad de elementos que tiene el heap hasta ese momento.

Métodos:

- **Heap(n):** Es constructor de la clase y recibe el entero n con el tamaño que esperamos que tenga el heap.
- **Parent(i);** Retorna la posición del padre que tiene el i, donde el i representa la posición que tiene el elemento del array.
- **leftSon(i);** Retorna la posición del hijo izquierdo que tiene el i, donde el i representa la posición que tiene el elemento del array.
- **RightSon(i);** Retorna la posición del hijo derecho que tiene el i, donde el i representa la posición que tiene el elemento del array.
- **getMin():** Retorna el elemento mas pequeño que tiene el heap.
- **sitfUp(i):** Va subiendo el elemento recién agregado hasta que el valor de su padre sea menor.
- **siftDown(i):** Baja el elemento que se pone en la raíz cuando se usa getMin para luego ponerlo en la posición correcta.
- **Insert(i):** Inserta un nuevo elemento al heap.

Utilidad: Hacemos uso del heap para la implementación de la cola prioritaria de tareas, en donde usamos la diferencia de días entre la fecha de la entrega y la fecha actual, y así sabemos cual es la tarea o trabajo que debemos hacer con mas urgencia.

4. DisjoinSets: Para implementar los conjuntos disjuntos, solo creamos una clase que contiene todos los atributos y métodos necesarios, los cuales son los siguientes:

Atributos:

- **Parent():** Array que almacena el padre de cada conjunto. Inicialmente, cada conjunto es su propio padre.
- **rank ():** Array que almacena la profundidad del árbol para optimizar las uniones. Se utiliza en la técnica de unión por rango.
- **size (int n):** Almacena el tamaño del conjunto disjunto.

Métodos:

- **DisjointSets(int size):** Constructor que inicializa los arreglos parent y rank con el tamaño proporcionado.
- **makeSet(int i):** Inicializa el conjunto en el índice i, estableciendo que el padre de i es i mismo y su rango es 0.
- **find(int i):** Encuentra el representante (padre) del conjunto que contiene al elemento i. Implementa la compresión de caminos, lo que significa que durante la búsqueda, se actualizan los padres para apuntar directamente al representante.
- **union(int i, int j):** Une dos conjuntos que contienen a los elementos i y j. Implementa la unión por rango, es decir, el conjunto con menor rango se convierte en hijo del conjunto con mayor rango. Si ambos tienen el mismo rango, uno se convierte en hijo del otro, y el rango del nuevo padre aumenta en uno.

Utilidad: Hacemos uso de los conjuntos para verificar si no existen conflictos entre las materias al agregarlas al horario, como por ejemplo que se crucen a la misma hora o que alguna sea prerrequisito de la otra.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

- **BST vs AVL:**

  - **Hipótesis:**

Es claro que el tiempo de inserción de un árbol AVL es mayor que en un BST, esto debido al costo adicional de mantener el equilibrio, pero el AVL mantiene una eficiencia de búsqueda más consistente que el BST, especialmente en el caso de secuencias de inserciones desbalanceadas.

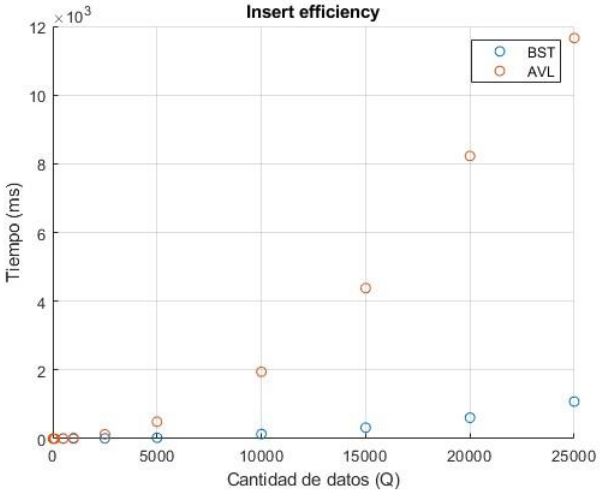
  - **Análisis de resultados:**

Para medir la eficiencia de ambas estructuras de datos, se eligieron los métodos *find* e *insert*, se añadieron una cantidad Q de números aleatorios a las estructuras y se midió el tiempo que se tarda en ejecutar tanto la inserción como la búsqueda de los elementos.

A continuación se presentan los resultados obtenidos:

Cantidad de números (Q)	Tiempo Insert BST (ms)	Tiempo Insert AVL (ms)

10	1	1
50	1	1
100	1	1
500	1	8
1000	2	22
2500	9	127
5000	25	494
10000	135	1947
15000	320	4380
20000	609	8226
25000	1080	11657



- De estos resultados se puede comprobar que el tiempo de inserción para BST se mantiene relativamente bajo a medida que aumenta la cantidad de datos .Incluso con 25,000 datos, el tiempo de inserción para BST apenas supera los 1,000 ms.

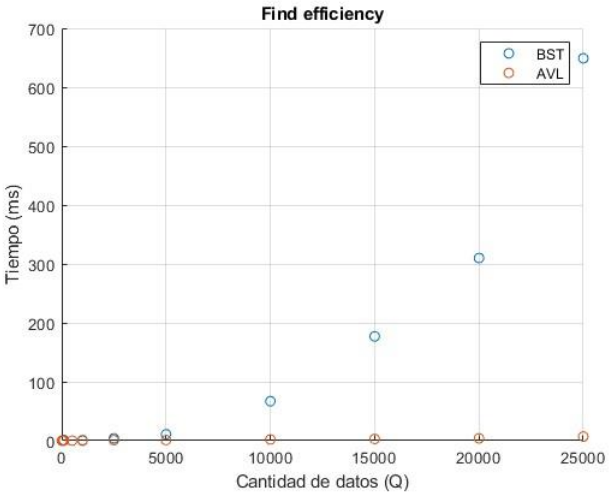
- Por otra parte, el tiempo de inserción para AVL aumenta de manera más pronunciada a medida que crece la cantidad de datos. Con 25,000 datos, el tiempo de inserción para AVL alcanza casi los 12,000 ms.

- Para cantidades pequeñas de datos (menos de 5,000), ambas estructuras muestran tiempos de inserción similares y bajos. A medida que aumenta la cantidad de datos, la diferencia en el rendimiento se vuelve más evidente. BST mantiene una eficiencia de inserción superior en comparación con AVL para conjuntos de datos más grandes.

- Resumiendo, estos resultados sugieren que para operaciones de inserción, el BST es más eficiente que el AVL, especialmente cuando se trabaja con grandes cantidades de datos.

Por otra parte, para el tema de búsqueda, se obtuvieron los siguientes resultados:

Cantidad de números (Q)	Tiempo Find BST (ms)	Tiempo Find AVL (ms)
10	0	0
50	0	0
100	1	0
500	0	0
1000	1	0
2500	4	1
5000	11	1
10000	67	2
15000	177	3
20000	310	4
25000	649	7



- El tiempo de búsqueda para BST aumenta significativamente a medida que crece la cantidad de datos. Con 25,000 datos, el tiempo de búsqueda para BST llega aproximadamente a 650 ms.

- El tiempo de búsqueda para AVL se mantiene consistentemente bajo, casi un comportamiento lineal. Incluso con 25,000 datos, el tiempo de búsqueda para AVL apenas es menor a 1 ms.

- Para cantidades pequeñas de datos (menos de 5,000), ambas estructuras muestran tiempos de búsqueda similares y bajos. Pero a medida que aumenta la cantidad de datos, la diferencia en el rendimiento se vuelve drásticamente evidente. El AVL mantiene una eficiencia de búsqueda superior en comparación con BST para conjuntos de datos más grandes.

- Esta diferencia en rendimiento se debe a la naturaleza autobalanceada del árbol AVL, que mantiene una altura óptima y por lo tanto garantiza tiempos de búsqueda logarítmicos, mientras que el BST puede degenerar en estructuras menos eficientes para la búsqueda si no está balanceado.

- **MaxHeap Binary vs MaxHeap Ternary:**

    - **Hipótesis:**

    Un MaxHeap ternario es más eficiente que un MaxHeap binario al momento de comparar la eficiencia de los métodos Insert y ExtractMax, debido a su menor altura. Las comparaciones adicionales que realiza el MaxHeap ternario no son suficientes para igualar o superar en tiempo a las del MaxHeap binario.

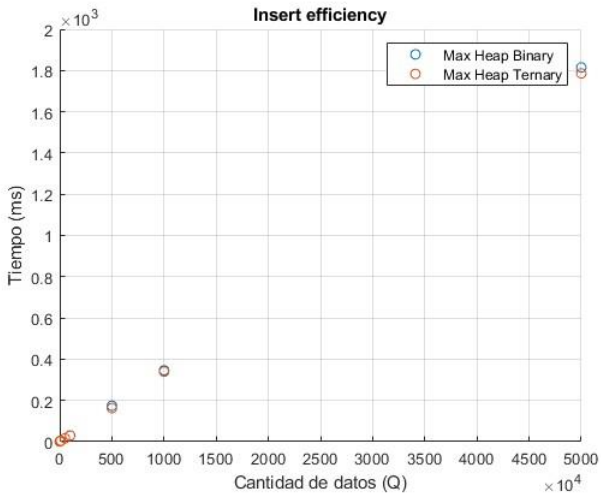
    Por otra parte, el HeapSort implementado con un MaxHeap ternario podría reducir el número de niveles a recorrer durante la construcción del heap y la extracción del máximo, pero debido a un mayor número de comparaciones por nivel, el MaxHeap binario podría ser más eficiente en términos de tiempo asintótico general.

    - **Análisis de resultados:**

    Para medir la eficiencia de las estructuras Max Heap Binary y Ternary, se eligieron los métodos Insert, ExtractMax y HeapSort para comparar los tiempos de ejecución.

    A continuación se muestran los resultados:

Cantidad de números (Q)	Tiempo Insert MaxHeapBinary (ms)	Tiempo Insert MaxHeapTernary (ms)
10000	1	1
50000	3	4
100000	5	5
500000	16	16
1000000	30	29
5000000	174	163
10000000	346	340
50000000	1815	1786

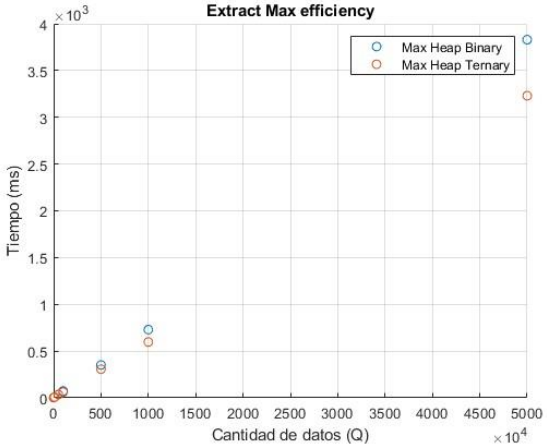


- Ambas estructuras muestran un aumento en el tiempo de inserción a medida que aumenta la cantidad de datos. A su vez, ambas muestran tiempos muy similares en toda la gráfica.

- La diferencia en rendimiento entre las dos estructuras es mínima para la operación de inserción.

- Estos resultados sugieren que en la inserción, ambas estructuras se comportan de manera muy similar. Sin embargo, las diferencias no son drásticas y podrían variar dependiendo de la implementación específica y el contexto de uso.

Cantidad de números (Q)	Tiempo ExtractMax MaxHeapBinary (ms)	Tiempo ExtractMax MaxHeapTernary (ms)
10000	2	2
50000	6	6
100000	8	8
500000	36	35
1000000	73	61
5000000	351	306
10000000	729	597
50000000	3830	3231



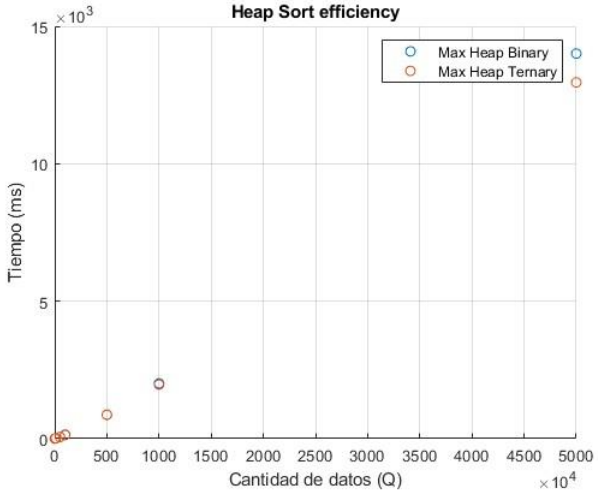
- Ambas estructuras muestran un aumento en el tiempo de ejecución a medida que aumenta la cantidad de datos. Para cantidades pequeñas de datos, el rendimiento es similar.

- La diferencia es más notable en el punto máximo (50,000 datos), donde Binary toma cerca de 3,800 ms y Ternary alrededor de 3,300 ms.

- Aunque para tamaños pequeños de datos no hay una diferencia clara entre el MaxHeap binario y el MaxHeap ternario, para tamaños más grandes, el MaxHeap ternario es más eficiente, ya que tarda menos tiempo en ejecutar el método *ExtractMax*.

- Esto respalda la idea de que la menor altura del MaxHeap ternario ofrece ventajas en operaciones que implican repetidas extracciones, como ExtractMax.

Cantidad de números (Q)	Tiempo HeapSort MaxHeapBinary (ms)	Tiempo HeapSort MaxHeapTernary (ms)
10000	2	2
50000	9	9
100000	16	16
500000	66	64
1000000	144	131
5000000	867	875
10000000	2009	1970
50000000	14008	12958



- Ambas estructuras muestran un aumento significativo en el tiempo de ejecución con el aumento de datos.

- La diferencia en rendimiento se hace más evidente a medida que aumenta la cantidad de datos.

- Aunque ambos heaps tienen un rendimiento similar para cantidades pequeñas de datos, el MaxHeap ternario se vuelve más eficiente que el MaxHeap binario cuando el tamaño de la entrada es grande, tal como se observa en el tiempo de ejecución de *HeapSort* para cantidades mayores de 40,000.

- Esto confirma que el MaxHeap ternario puede ser más ventajoso en situaciones donde se trabaja con grandes volúmenes de datos debido a su menor altura, lo que reduce la cantidad de operaciones necesarias para reorganizar el heap durante el ordenamiento.

- **Disjoint Set (Path compression) vs Disjoint Set (Basic):**

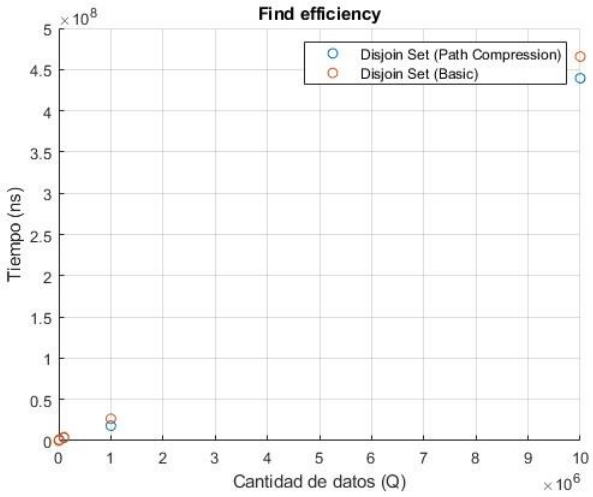
    - **Hipótesis:**

El método *Find* en un Disjoint Set con Path Compression es significativamente más eficiente que en un Disjoint Set normal, especialmente en estructuras con múltiples operaciones Find, debido a la reducción en la altura de los árboles representativos, lo que mejora el rendimiento en secuencias largas de consultas.e

    - **Análisis de Resultados:**

Cantidad de números (Q)	Tiempo Find Disjoint Set Path-Compression (ns)	Tiempo Find Disjoint Set Basic (ns)

10	15100	8100
100	54900	38700
1000	259800	111400
10000	1208500	1038700
100000	3581800	4315200
1000000	18125700	26565600
10000000	439443500	465698000

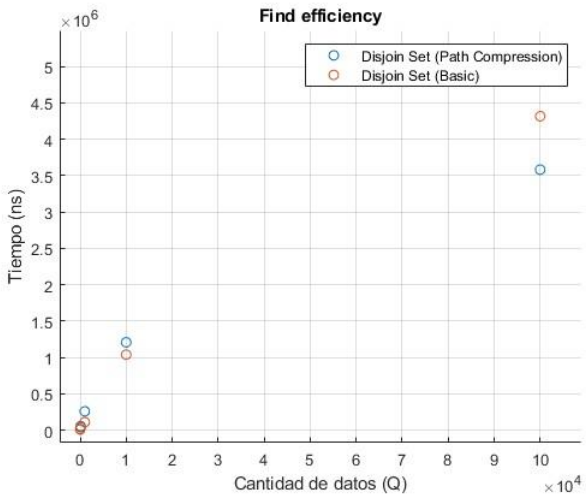


(Gráfica 1)

- Ambas implementaciones muestran un aumento en el tiempo de ejecución a medida que aumenta la cantidad de datos.

- El Disjoint Set básico (rojo) muestra un tiempo de ejecución ligeramente mayor que el Disjoint Set con compresión de camino (azul) para grandes cantidades de datos. Pero para cantidades pequeñas de datos, el rendimiento es muy similar.

A continuación una vista más detallada para cantidades de datos menores:



(Gráfica 2)

- Esta gráfica muestra un detalle más fino para cantidades menores de datos.

- El Disjoint Set con compresión de camino muestra un mejor rendimiento en la operación "find" en comparación con el Disjoint Set básico, especialmente con pocas cantidades de datos.

- La compresión de camino parece ofrecer una mejora de rendimiento consistente, aunque la magnitud de la mejora varía según la escala de los datos.

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

demostracion-entrega1.mp4

El ejecutable se presenta en la carpeta *dist* del repositorio especificado en la sección VII.

XI. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS
Cristian Fabian Martinez Bohórquez	Técnico	Pruebas técnicas.
		Implementación de estructuras.
	Coordinador	Código al prototipo del proyecto.
Edward Jeisen	Observador	Pruebas técnicas.

Jair Arévalo Peña		Código al prototipo del proyecto.
	Animador	Contacto permanente
Juan David Cruz Giraldo	Investigador	Implementación de estructuras.
		Documento escrito.
	Experto	Diseñador

XII. DIFICULTADES Y LECCIONES APRENDIDAS

Durante el proceso de desarrollo del prototipo nos enfrentamos a dificultades principalmente al realizar pruebas con conjuntos de datos muy grandes. Puesto que no eran aplicables a nuestro proyecto en ese momento. En el contexto de SIApp, donde el objetivo principal es planificar y organizar el horario de materias para los estudiantes universitarios, no tenemos 100,000 materias o profesores en el sistema de la aplicación.

XIII. REFERENCIAS BIBLIOGRÁFICAS

[1] atlassian, Learn Git with Bitbucket Cloud, atlassian, disponible en https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud, accedido el: 28 de marzo de 2024.

[2] reqtest, Why is the difference between functional and Non-functional requirements important?, reqtest, disponible en https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/, accedido el: 29 de marzo de 2024.

[3] Y. D. Chong, Estructuras de datos secuenciales, espanol.libretexts, disponible en https://espanol.libretexts.org/Fisica/Física_Matemática_y_Pedagogía/Física_Computacional_(Chong)/02%3A_Tutorial_de_Scipy_(Parte_2)/2.01%3A_Estructuras_de_datos_secuenciales, accedido el: 29 de marzo de 2024.

[4] Streib, J. T., & Soma, T. (2017). Guide to Data Structures. Springer International Publishing.