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.

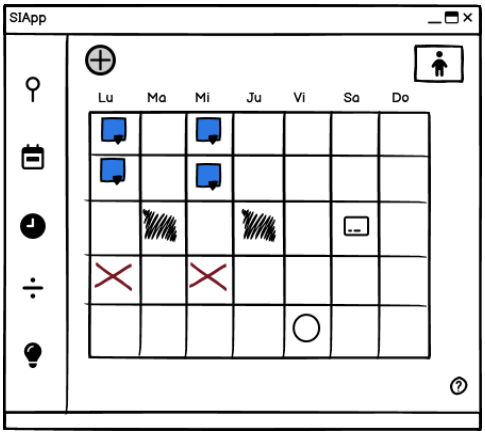
4. El sistema ordena automáticamente las tareas y parciales en la cola según su prioridad, con las tareas más urgentes en la parte superior de la lista.
5. El usuario puede editar o eliminar tareas y parciales de la cola según sea necesario.

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

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. Lista enlazada: Para implementar la lista enlazada usamos dos clases, Node y LinkedList, la clase Node se encuentra dentro de la clase LinkedList y contiene los siguientes atributos y métodos:

Node:

Atributos:

- **value**: Valor asociado al nodo.
- **Node next:** Referencia al siguiente nodo en la lista.

Métodos:

- **getData()**: Devuelve el valor asociado al nodo.
- **getNext():** Retorna el siguiente nodo.
- **setValue(item)**: Establece el valor asociado al nodo.
- **setNext(GenericNode node):** Establece la referencia al siguiente nodo.

LinkedList:

Atributos:

- **Node head:** Primer nodo de la lista.
- **Node tail:** Último nodo de la lista.
- **int size**: Tamaño máximo de la lista.
- **int count:** Cantidad de nodos en la lista.

Métodos:

- **isEmpty()**: Verifica si la lista está vacía.
- **isFull():** Verifica si la lista está llena.
- **length():** Devuelve el tamaño máximo de la lista.
- g**etCount()**: Retorna la cantidad de nodos en la lista.
- **pushFront(item**): Agrega un nodo al inicio de la lista.
- **keyTopFront():** Retorna el valor del primer nodo.
- **popFront():** Devuelve y elimina el valor asociado al primer nodo.
- **pushBack(item)**: Agrega un nodo al final de la lista.
- **keyTopBack()**: Retorna el valor del último nodo.
- **popBack():** Devuelve y elimina el valor asociado al último nodo.
- **find(item):** Verifica si un elemento está presente en la lista.
- **erase(item):** Elimina un nodo que contiene el elemento especificado.
- **addBefore(item, int position):** Agrega un nuevo nodo antes de una posición especifica.
- **addAfter(item, int position):** Agrega un nuevo nodo después de la posición indicada,
- **getValue(int position):** Retorna el valor asociado al nodo en la posición indicada.
- **toString():** Retorna una representación en forma de cadena de la lista.

2. Pila: Para implementar la pila, hacemos uso de un objeto de lista enlazada para poder usar los métodos de linkedlist para crear los métodos de Stack.

Stack:

Atributos:

- **Stack=** Objeto de la clase linkedList.

Métodos:

- **Push(key):** Agrega un nuevo elemento al inicio de la lista enlazada usando pushFront.
- **Pop():** Elimina el elemento que se encuentra al inicio de la linkedlist.
- **Top():** Retorna el valor asociado al nodo que se encuentra en el inicio de la linkedList.
- **isEmpty(): Verifica si se encuentra vacía.**
- **isFull():** Verifica si la lista está llena.
- Display(): Imprima una representación de la pila.

3. Cola: Para implementar la cola, hacemos uso de un objeto de lista enlazada para poder usar los métodos de linkedlist para crear los métodos de Queue.

Queue:

Atributos:

- **Stack =** Objeto de la clase linkedList.

Métodos:

- **Enqueue(key):** Agrega un nuevo elemento al final de la lista enlazada usando pushBack.
- **Dequeue():** Elimina el elemento que se encuentra al final de la linkedlist.
- **isEmpty():** Verifica si se encuentra vacía.
- **isFull():** Verifica si la lista está llena.

Utilidad: Hacemos uso de pilas para almacenar la información de las materias, como el nombre o el código que tienen.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

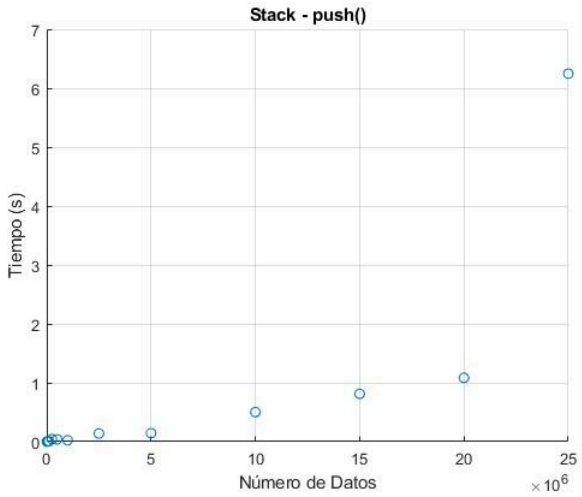
Para el análisis de eficiencia y pruebas del prototipo, se seleccionó la opción de agregar y eliminar las distintas características de nuestro horario, ya sean el nombre de la materia, código, créditos, docente, salon, hora, etc.

Para estas pruebas se decidió hacer la implementación haciendo uso tanto de un ***stack*** como una ***queue***. Esto con el objetivo de comparar la eficacia de ambas estructuras cuando se ven enfrentadas a grandes cantidades de datos. A primera vista, se esperan resultados similares para ambas estructuras, ya que, para el caso de un ***stack,*** el hacer **push()** y **pop()** tiene una complejidad constante O(1). Para el caso de una ***queue*** la situación es la misma, los métodos **enqueue()** y **dequeue()** son de complejidad constante O(1).

Para el análisis del tiempo que emplea cada funcionalidad en realizarse, se hizo uso de la librería **System** propia del lenguaje Java, el cual cuenta con el método .**nanoTime(),** el cual permite calcular el tiempo empleado en nanosegundos para ejecutarse una acción en concreto.

A continuación se muestran los resultados obtenidos al momento de realizar **push()** de acuerdo a la cantidad de datos establecida.

Stack - push()	
Cantidad de Datos	**Tiempo Empleado (ns)**
10000	2193900
25000	3840300
50000	5671000
100000	7906200
250000	43664900
500000	44277200
1000000	23726400
2500000	152333300
5000000	47864100
5000000	665430100
15000000	827031600
20000000	1238205800
25000000	7415442800

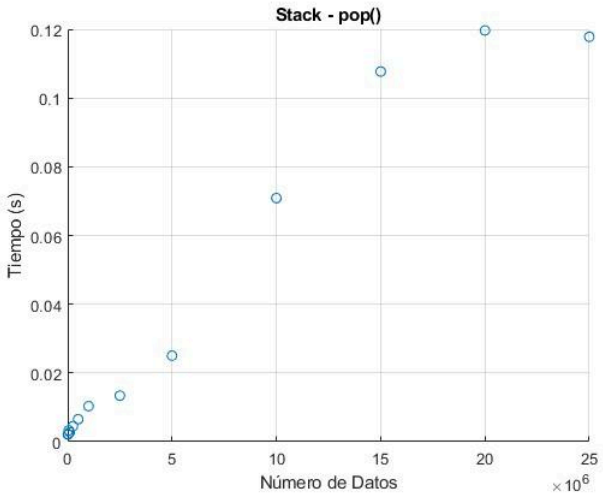


Al momento de realizar la gráfica, se observa que el comportamiento aproximadamente lineal, tal como se esperaba, ya que el método **push()** por si solo tiene una complejidad constante, y al repetir este proceso para **n** cantidades de datos, la complejidad es lineal O(n).

Para el caso del método **pop(),** el procedimiento que se realizó fue exactamente el mismo. Se calculó el tiempo requerido para para eliminar el dato del stack, obteniendo lo siguiente:

Stack - pop()	
Cantidad de Datos	**Tiempo Empleado (ns)**
10000	2059100
25000	2184700
50000	3290400
100000	2735000
250000	4517300
500000	6536900
1000000	10338700
2500000	13391700

5000000	25030200
5000000	70871700
15000000	107704600
20000000	119665000
25000000	117808800

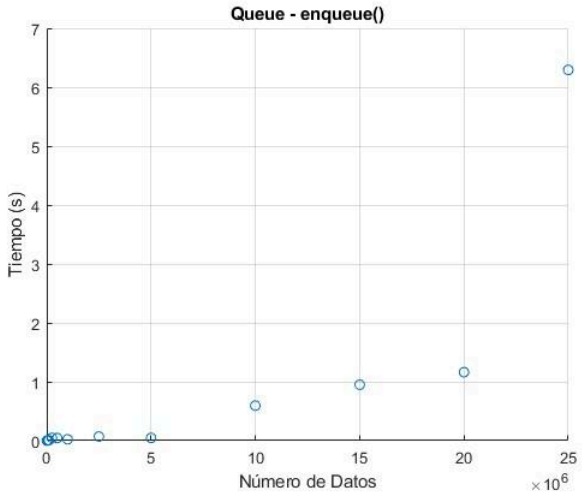


Tal como se esperaba, el método **pop()** también tiene un comportamiento lineal, cabe resaltar que los tiempos empleados para realizarlos disminuyeron a comparación del método **push(),** esto debido a que la lista ya está creada con los **n** datos generados aleatoriamente, por lo que solo es necesario eliminar uno por uno.

Posteriormente, se empleó una queue para analizar y comparar la efectividad para ejecutar las mismas funcionalidades. Empezando con el método **enqueue(),** se calculó el tiempo que este tardó en ejecutarse para **n** cantidades de datos, obteniendo los siguientes resultados:

Queue - enqueue()	
Cantidad de Datos	**Tiempo Empleado (ns)**
10000	1922800
25000	3117700
50000	5227500
100000	7924500
250000	47820900
500000	45364000
1000000	22532700
2500000	70862000
5000000	46642800
5000000	594903900
15000000	949578900
20000000	1162466200
25000000	6293200100

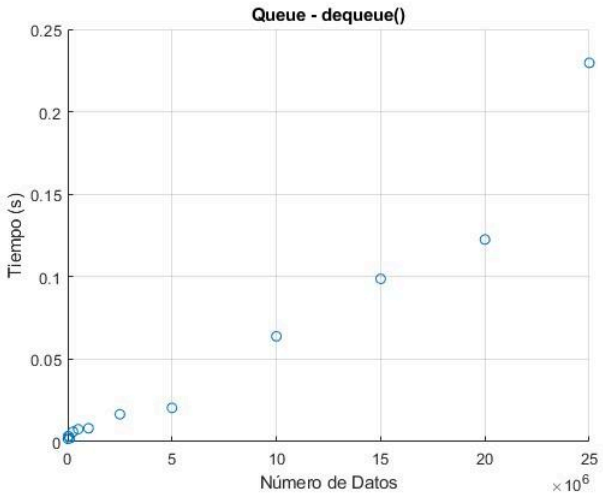
Para el caso del método **enqueue(),** el método tiene una complejidad constante de O(1), al realizarse para **n** cantidades de datos, la complejidad total pasa a ser lineal de O(n), tal como se puede apreciar en la gráfica a continuación:



Para el caso del método **dequeue(),** se siguió el mismo procedimiento:

Queue - dequeue()	
Cantidad de Datos	**Tiempo Empleado (ns)**
10000	1674200
25000	3356600
50000	3096600
100000	1990500
250000	6025000
500000	7565600
1000000	8104300
2500000	16619600
5000000	20515500
5000000	63887900

15000000	98710200
20000000	122568700
25000000	229625500



Al analizar la gráfica obtenida, se observa un comportamiento lineal O(n), tal como se había anticipado.

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 Jair Arévalo Peña	Observador	Pruebas técnicas.
		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.