UNIVERSIDAD
NACIONAL
DE COLOMBIA

Sistema de Gestión de Tutorías Académicas UNAL

Estructuras de Datos

Samuel Benitez Culma
Sebastián Contreras Tellez
Adrian David Espitia Bayona
Yessica Vanessa Trujillo Ladino
Gabriel Santiago Velez Gonzalez

Facultad de Ingeniería
Departamento de Sistemas e Industrial
Bogotá, Colombia
10 de diciembre, 2025

Índice

Índice de figuras

1. Problema a abordar

Se tiene un sistema sencillo de asignación de tutorías estudiantiles. Dicho sistema se apoya con el uso de Formularios de Google, en los cuales los estudiantes pueden registrar sus datos, tales como: número de documento, nombre y asignatura de la cual desea recibir asesoría. Después, el estudiante elige una hora que se acomode a su horario y elige su tutor, dentro de los que están disponibles.
Sin embargo, este sistema de tutorías, aunque es simple, no permite un seguimiento óptimo de los procesos académicos. Además, no ofrece una visualización cómoda para el estudiante, ni un uso intuitivo al ser un formulario en línea. Tampoco permite una organización eficiente del histórico de tutorías.
Se desea crear un programa sencillo, pero eficiente e intuitivo, que permita automatizar la solicitud de tutorías académicas; la administración de tutorías agendadas por un estudiante y la cancelación de las mismas, junto con un histórico de tutorías completadas.

Organización de los deberes.

En la Tabla 1 se presenta la organización de los deberes del equipo.

Integrantes	**Roles**	**Actividades Asignadas**
Samuel Benítez	Desarrollador - Gestor	▪ Diseñar el diagrama de flujo general de sistema. ▪ Implementar la clase Nodo. ▪ Crear interfaces para LinkedList y MaxHeap. ▪ Implementar Controlador GestorTutorías y sus métodos.
Sebastián Contreras	Desarrollador - Diseñador	▪ Diseñar MockUp de la interfaz gráfica. ▪ Corregir implementaciones de ED usadas. ▪ Realizar la interfaz gráfica del programa.
Adrian Espitia	Desarrollador - Analista	▪ Diseñar diagrama de secuencia UML. ▪ Implementar HashMap. ▪ Realizar análisis de complejidad y pruebas.
Yessica Trujillo	Desarrolladora - Diseñadora	▪ Definir estructuras de datos a utilizar. ▪ Crear interfaz de GestorTutorías para posterior implementación. ▪ Realizar la interfaz gráfica del programa.
Santiago Vélez	Desarrollador - Diseñador	▪ Diseñar el diagrama de clases UML. ▪ Crear las clases Tutoría, Tutor, Estudiante. ▪ Preparar archivo Main e instanciar objetos. ▪ Implementar el ArrayMaxHeap.

Cuadro 1: Organización de los deberes.

Repositorio del proyecto y video demostrativo

Como resultado del desarrollo realizado, el repositorio remoto del proyecto en GitHub se encuentra documentado en [1].
Asimismo, se cuenta con un video breve en YouTube que presenta una demostración del funcionamiento y la apariencia del producto desarrollado, el cual se referencia en [3].

2. Objetivos

Diseñar e implementar un sistema que permita gestionar de manera eficiente las tutorías académicas, mejorando el registro de estudiantes y tutores, la asignación automática según criterios de prioridad y disponibilidad.

Objetivos Específicos

1. Registrar y almacenar la información de las tutorías y del estudiante, garantizando un acceso rápido y estructurado a los datos.
2. Automatizar la asignación de tutorías considerando la urgencia del estudiante, la asignatura requerida y la disponibilidad del tutor.
3. Facilitar la consulta y visualización de las tutorías, permitiendo identificar estudiantes pendientes, tutores asignados y tutores disponibles.
4. Mantener un historial organizado de las tutorías realizadas, incluyendo fechas y observaciones relevantes para el seguimiento académico.
5. Permitir la actualización y eliminación de información sobre tutorías, preservando la consistencia del sistema.

3. Descripción general del MVP.

El MVP desarrollado corresponde a una primera versión funcional del sistema gestor de tutorías, enfocado en permitir que el estudiante organice, visualice y administre sus tutorías mediante una interfaz gráfica, simple e intuitiva. Esta versión incorpora las funcionalidades esenciales para tener una experiencia de uso en un entorno real.
El sistema se estructura en cinco secciones principales:

1. **Solicitar una tutoría.**
   El estudiante puede seleccionar la materia de interés, asignar una prioridad(alta, media o baja), registrar observaciones y elegir el tutor y el horario disponible.
2. **Ver el historial de tutorías.**
   Presenta una lista con el histórico de listas que ha tenido un estudiante y que han sido marcadas como finalizadas.

3. **Ver las tutorías programadas.**
   Muestra una lista con las tutorías que el estudiante tiene actualmente agendadas y aún no se han realizado.

4. **Cancelar una tutoría.**
   Permite al estudiante visualizar sus tutorías activas y seleccionar cuál de ellas quiere cancelar. Al confirmar la selección, el sistema elimina la tutoría programada.

5. **Finalizar una tutoría.**
   Le muestra al estudiante la lista de sus tutorías activas y le permite escoger cual desea marcar como finalizada, después de haber tenido la sesión de tutoría. Al confirmar la selección, la tutoría se elimina de sus tutorías activas y pas a sus tutorías históricas.

En conjunto, el MVP permite comprobar tanto la viabilidad técnica como práctica del sistema y obtener retroalimentación temprana sobre la interacción real de los estudiantes. Aunque es un versión inicial, es una base sólida para ampliar funcionalidades y evolucionar hacia una plataforma más completa.

4. Implementación y diseño.

En esta sección se describe cómo fue implementado el sistema de gestión de tutorías académicas, organizando la explicación en las estructuras de datos empleadas y en el diseño de la interfaz gráfica de usuario.

4.1. Los entornos de desarrollo y ejecución.

Dentro del equipo de trabajo, se utilizaron diferentes herramientas y Entornos de Desarrollo Integrado ***IDE***, según el gusto y la comodidad de cada quién. Entre los utilizados, se encuentran:

- Entorno de Desarrollo Integrado - Intellij IDEA.
- Entorno de Desarrollo Integrado - Apache NetBeans.
- Editor de código - Visual Studio Code.
- Git y GitHub como herramientas de control de versiones local y remota.

4.2. Estructuras de datos utilizadas

En esta subsección se presentan las principales estructuras de datos que se emplearán en el sistema de gestión de tutorías académicas. Estas estructuras se eligieron con el objetivo de soportar de manera eficiente las operaciones de registro, consulta, asignación y cancelación de tutorías, tal como se presenta en los diagramas de sistema y en el diagrama de flujo de la vista del estudiante.
Las estructuras principales son las siguientes:

1. **HashMap de tutores por asignatura (`tutores`).**
   Se usará un mapa asociativo donde la clave corresponde al nombre de la asignatura y el valor es una colección de tutores que pueden dictar dicha materia. A partir de esta estructura, el sistema puede obtener rápidamente la lista de tutores disponibles cuando el estudiante selecciona una asignatura en el proceso de solicitud de una nueva tutoría.

2. **HashMap de tutorías por estudiante (`tutorias_estudiante`).**
   Esta estructura almacena, para cada estudiante identificado por su número de documento, una lista enlazada de tutorías que tiene programadas. Se utiliza tanto para mostrar las tutorías vigentes de un estudiante como para permitir la cancelación de una tutoría específica, recorriendo y actualizando la lista correspondiente.

3. **HashMap de tutorías pendientes por tutor (`tutorias_pendientes`).**
   Desde la perspectiva de cada tutor, se manejará un mapa cuya clave es el identificador del tutor y cuyo valor es una cola de prioridad (Max-Heap) con las tutorías que tiene asignadas. La prioridad de cada tutoría estará dada por la urgencia del estudiante u otros criterios definidos. De esta forma, el sistema puede determinar de manera eficiente qué tutoría debe ser atendida primero por cada tutor.

4. **Lista de histórico de tutorías (`historico_tutorias`).**
   Para registrar las tutorías ya realizadas se empleará una lista que almacenará el historial general del sistema. Cada vez que una tutoría pase de estar pendiente a estar completada, se agregará un registro a esta lista, lo que permitirá consultar posteriormente el historial de tutorías de un estudiante.

5. **Estructuras auxiliares.**
   Además de las estructuras anteriores, se manejarán listas simples para almacenar las asignaturas ofrecidas por el programa de tutorías y las opciones del menú principal de la interfaz. Estas listas se cargan al inicio de la ejecución y facilitan la presentación de opciones al estudiante.

4.3. Diseño e interfaz gráfica de usuario

La interfaz gráfica de usuario se diseñó pensando en las necesidades de los estudiantes. La aplicación se implementó en Java Swing como una única ventana principal titulada *Manejo de tutorías*. Desde esta ventana, el estudiante encuentra las opciones más importantes: solicitar una nueva tutoría, ver las tutorías programadas, cancelar una sesión y consultar su historial. Cada opción se presenta como un botón claramente identificado y conduce a una pantalla específica con instrucciones simples.
En la opción de programar tutoría se muestra una pantalla en la que el estudiante puede elegir la asignatura y la prioridad de la solicitud mediante listas desplegables, y añadir si lo desea algunas observaciones en un cuadro de texto. A partir de esa información, el sistema guía al usuario paso a paso, utilizando cuadros de diálogo para completar la selección del tutor y del horario, y mostrando mensajes de confirmación cuando la tutoría se ha registrado correctamente.
Las opciones de historial, tutorías programadas y cancelación presentan la información en listas o áreas de texto desplazables, lo que permite revisar y gestionar de manera sencilla las tutorías registradas. En todo momento se dispone de un botón para volver a la ventana principal, lo que ayuda a mantener una navegación simple para el estudiante.

5. Pruebas y análisis de complejidad

Los procesos principales dentro del sistema se dan al momento de agendar, finalizar y cancelar tutorías, por esta razón se ejecutan las pruebas sobre los métodos `GestorTutorias.solicitarTutoria`, `GestorTutorias.finalizar` y `GestorTutorias.cancelarTutoria`.

5.1. Hipótesis y tiempos de ejecución teóricos

A partir de las estructuras de datos utilizadas y de las operaciones que ejecuta cada método, se puede anticipar su comportamiento temporal en el peor de los casos. En particular, se espera que `GestorTutorias.solicitarTutoria` y `GestorTutorias.finalizar` presenten tiempos de ejecución dominados por las operaciones sobre el montículo, con una complejidad asintótica del orden de $O(\log n)$, mientras que `GestorTutorias.eliminarTutoria` debería exhibir un comportamiento lineal $O(n)$ al requerir búsquedas y eliminaciones secuenciales.
Sin embargo, debido al rango acotado de prioridades (entre 1 y 5) y al tamaño moderado de las listas manejadas en el contexto del sistema de tutorías, se plantea como hipótesis que, en la práctica, los tiempos de ejecución observados para `GestorTutorias.solicitarTutoria` y `GestorTutorias.finalizar` crecerán muy lentamente con respecto a n, acercándose a un comportamiento casi constante en los rangos de entrada evaluados, mientras que `GestorTutorias.eliminarTutoria` mostrará una tendencia aproximadamente lineal.

5.1.1. `GestorTutorias.solicitarTutoria`

El método principal para solicitar una tutoría ejecuta una operación de `push front` en una lista enlazada (tiempo $O(1)$) y otra operación de inserción en una cola prioritaria (tiempo $O(\log n)$, donde n es el tamaño del montículo). No obstante, el rango de prioridades es muy limitado, pues solo se permiten enteros entre 1 y 5, por lo que las operaciones `sift up` y `sift down` no recorren demasiados niveles del montículo antes de parar. Esto puede propiciar que el tiempo de ejecución sea más corto de lo esperado.

5.1.2. `GestorTutorias.finalizar`

Este método ejecuta una operación de eliminación en una lista enlazada (tiempo $O(m)$ donde m es el tamaño de la lista) y una operación de `extractMax` en un montículo (tiempo $O(\log n)$). Sin embargo, también se esperan tiempos de ejecución más cortos por el rango limitado de prioridades y los tamaños reducidos de las listas enlazadas (estimados al rededor de $n/20$ siendo n el tamaño del montículo).

5.1.3. `GestorTutorias.eliminarTutoria`

Este método ejecuta una operación de eliminación en una lista enlazada (tiempo $O(n)$) y una operación de eliminación en un montículo sin conocer la posición del elemento (tiempo $O(n)$ pues se debe buscar el elemento primero). Se espera un tiempo de ejecución lineal para este método.

5.2. Diseño experimental

Con el fin de contrastar las complejidades teóricas con el comportamiento real del sistema, se diseñó un experimento de medición de tiempos de ejecución sobre los tres métodos de interés. La idea general

consiste en construir instancias del `GestorTutorias` con diferentes tamaños de entrada y registrar el tiempo que tarda cada método en ejecutarse, manteniendo controladas las demás condiciones del entorno. Para ello se generaron automáticamente los datos de prueba (estudiantes, asignaturas, tutorías y prioridades) y se definió un conjunto de tamaños n sobre los cuales se repite la medición. A partir de estos datos se construyen las estructuras internas del sistema y se realizan múltiples llamadas a cada método, promediando los tiempos obtenidos. De esta manera es posible comparar las curvas empíricas con las complejidades esperadas y analizar si el comportamiento observado coincide con las hipótesis planteadas.

5.2.1. Mock Data

Para generar los datos de prueba se toman listas de nombres, apellidos, materias y horarios, y se generan combinaciones aleatorias para tener un total de 20 estudiantes, un tutor y 1000200 tutorías. Se fija un solo tutor para explorar el peor caso del tamaño del montículo. Las tutorías se generan tomando una fecha aleatoria en el rango de un año y un nivel de prioridad aleatorio entre comprendido entre 1 y 5. La clase `MockDataGenerator` genera estos datos y ofrece la posibilidad de ajustar la cantidad de estudiantes, tutores y tutorías.

5.2.2. Pruebas de tiempo de ejecución

Se genera un conjunto de 20 puntos n equidistantes en una escala logarítmica entre 10^2 y 10^6. Para cada n se instancia un objeto `GestorTutorias` y se insertan n tutorías para tener un montículo de dicho tamaño. Posteriormente, con el objetivo de reducir el efecto del ruido y tener medidas más significativas, se mide el tiempo de ejecución promedio de 100 llamadas al método. La medición se hace utilizando `System.nanoTime`. Este proceso se repite para cada uno de los 3 métodos.

5.3. Resultados y discusión

Los resultados obtenidos tras realizar las mediciones de los tiempos de ejecución se muestran en la Figura 1. Para el método `GestorTutorias.solicitarTutoria` no se observa un aumento significativo del

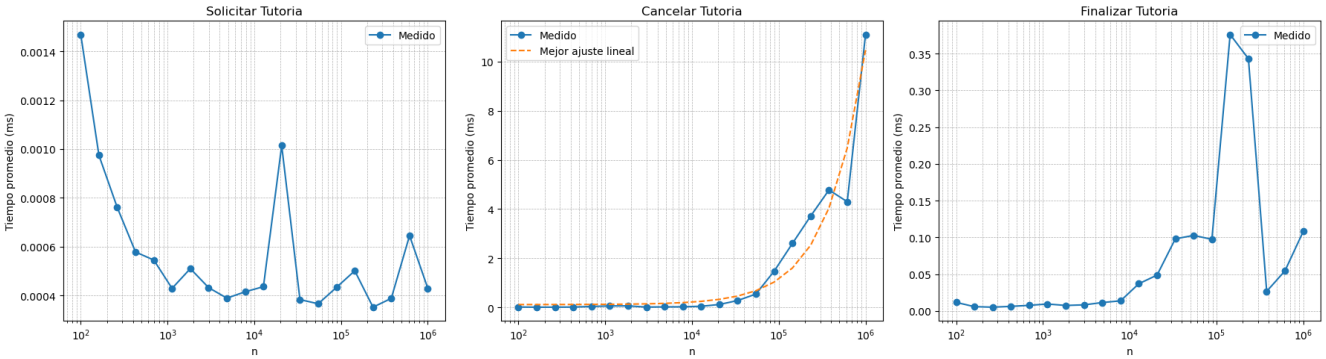

Figura 1: Resultados de las mediciones de complejidad para los tres métodos estudiados.

tiempo de ejecución con el tamaño de la entrada, lo que se identifica con un tiempo de ejecución $O(1)$. A pesar de que este método utiliza una operación de complejidad $O(\log n)$ donde n es el tamaño del montículo, el rango limitado de prioridades reduce un poco la cantidad de operaciones usadas durante los procesos de `sift up` y `sift down`, pues hay muchos nodos con prioridades repetidas. Por otro lado, el

método `GestorTutorias.eliminarTutoria` muestra un claro comportamiento lineal, lo que concuerda con las hipótesis planteadas respecto a este método. Finalmente, `GestorTutorias.finalizar` muestra un aumento del tiempo de ejecución con respecto al tamaño de la entrada, lo que concuerda con una complejidad $O(m) + O(\log n)$ con m el tamaño de la lista enlazada y n el tamaño del montículo. La reducción del tiempo en los tres últimos tamaños de prueba se puede atribuir a ruido en la medición, u otras dinámicas de las estructuras de datos. Para resultados más concluyentes se puede incrementar la densidad de puntos en el intervalo de tamaños de prueba.

6. Retos enfrentados

Durante el desarrollo del proyecto, hubo diferentes obstáculos y retos que se tuvieron que afrontar. Problemas de sintaxis, de lógica o bien, pequeños problemas de optimización que nos hicieron aprender a implementar métodos de mejor manera o a escribir un código más limpio.

Como prueba de ello, se mencionan algunos ejemplos:

- **Falta de un método:** Durante la implementación de la clase `gestorTutorias` reconocimos la ausencia de métodos importantes dentro de algunas estructuras de datos, tales como eliminación por objeto ó eliminación específica.
- **Familiaridad con el control de versiones:** Este proyecto fue un reto para aquellos integrantes que no habían trabajado a fondo con una herramienta de control de versiones. Les permitió adquirir esta valiosa habilidad mediante la práctica.
- **La sintaxis:** Ocurrieron varios inconvenientes con la sintaxis. Por ejemplo, incongruencias con el nombre de un parámetro de un método o el tipo de objeto de una clase. Estos fueron subsanados con apoyo de las indicaciones del IDE, editor de código o del mismo lenguaje.
- **El diseño:** Diseñar el sistema antes de programarlo fue un reto porque exigió definir con precisión las relaciones entre clases, responsabilidades y flujos de datos sin tener aún el código como referencia. Esto implicó anticipar cómo se comportarían los objetos, prever casos de uso, posibles errores y la interacción completa del sistema.

Referencias

[1] *ED-Proyecto*. Repositorio en GitHub. 2025. URL: `https://github.com/ytrujillol/ED-Proyecto`.

[2] James T. Streib y Takako Soma. *Guide to Data Structures. A Concise Introduction Using Java*. Undergraduate Topics in Computer Science. Cham: Springer, 2018. ISBN: 978-3-319-70083-0. DOI: `10.1007/978-3-319-70085-4`.

[3] *Video de demostración*. Video de YouTube. 2025. URL: `https://youtu.be/wT6ULALAGdE`.