MOMENTO EVALUATIVO #3

Por:

Julián Felipe Vélez Medina

Luz Natalia Ríos Serna

Juan Pablo Salazar Ceballos

Nilzon Alejandro Gómez Maya

CURSO DE ESTRUCTURAS DE DATOS
INSTITUTO TECNOLÓGICO METROPOLITANO

GESTIÓN DE SOLICITUDES DE TUTORÍAS UNIVERSITARIAS

FORMULACIÓN

CONTEXTO

En la universidad, los estudiantes frecuentemente solicitan tutorías académicas con docentes o monitores de diversas asignaturas. Sin embargo, la atención de dichas solicitudes suele verse afectada por listas de espera extensas y solicitudes urgentes que requieren atención prioritaria.
Con el fin de optimizar este proceso, se plantea el desarrollo de un sistema de gestión de solicitudes de tutorías, basado en el uso de estructuras de datos tipo Pila y Cola, para simular el flujo real de atención.

OBJETIVO DEL SISTEMA

Diseñar e implementar un sistema que permita registrar, gestionar, priorizar y atender solicitudes de tutorías académicas, aplicando estructuras de datos Pila y Cola construidas mediante clases y nodos enlazados, garantizando un flujo organizado de atención y la trazabilidad de las operaciones realizadas.

ENTRADAS

- Archivo de texto con las solicitudes iniciales, donde cada línea contiene: nombre_estudiante, materia, fecha, tipo_solicitud, prioridad
- Solicitudes nuevas ingresadas manualmente por el usuario.
- Criterios de búsqueda o eliminación, ingresados por consola (por ejemplo: nombre del estudiante o materia).

SALIDAS

- Listado ordenado de solicitudes pendientes (en consola).
- Archivo de texto con las solicitudes atendidas.
- Resultados de operaciones de búsqueda, eliminación y ordenamiento.

ESTRUCTURAS DE DATOS UTILIZADAS

El sistema se construye utilizando listas simples enlazadas como base para representar las estructuras de datos Cola y Pila, garantizando un tamaño dinámico y operaciones controladas mediante nodos.
Cada estructura está implementada como una clase independiente que administra sus propios nodos, evitando el uso de listas nativas de Python.

- **Cola (Queue) – Basada en lista simple enlazada:** La cola gestiona las solicitudes normales de tutorías en el orden en que fueron registradas, aplicando el principio FIFO (First In, First Out). Internamente, se representa mediante una lista simple enlazada con referencias al frente (primer nodo) y al final (último nodo).
  Cada solicitud se encapsula en un objeto Nodo, que contiene el dato (Solicitud) y una referencia al siguiente nodo.

  Operaciones implementadas:
  - enqueue(): Inserta una nueva solicitud al final de la lista enlazada.
  - dequeue(): Elimina la solicitud ubicada al inicio (la más antigua).
  - buscar(criterio): Recorre la lista para ubicar una solicitud por nombre o materia.
  - mostrar(): Permite visualizar el contenido actual de la estructura sin usar iteradores ni listas nativas de Python, cumpliendo las restricciones del curso.
  - isEmpty(): Permite validar condiciones previas antes de operaciones de extracción o eliminación, previniendo errores de ejecución.

  Uso en el sistema:
  Permite administrar de forma secuencial las solicitudes comunes de tutoría, simulando una lista de espera real, donde las solicitudes se atienden en el orden de llegada.

- **Pila (Stack) – Basada en lista simple enlazada:** La pila se utiliza para registrar solicitudes urgentes aplicando el principio LIFO (Last In, First Out).
  Internamente, se implementa como una lista simple enlazada en la que solo se tiene referencia al tope (primer nodo). Cada nodo contiene un objeto Solicitud y un enlace hacia el siguiente nodo de la pila.

  Operaciones implementadas:
  - push(): Inserta una nueva solicitud urgente en el tope de la lista.
  - pop(): Elimina la solicitud ubicada en el tope, que es la última registrada.
  - buscar(criterio): Recorre los nodos desde el tope hasta la base buscando por nombre o materia.
  - mostrar(): Permite visualizar el contenido actual de la estructura sin usar iteradores ni listas nativas de Python, cumpliendo las restricciones del curso.
  - isEmpty(): Permite validar condiciones previas antes de operaciones de extracción o eliminación, previniendo errores de ejecución.
  - top(): retorna lo que se encuentra encima de la pila.

- eliminarPorAtributo(): Elimina una solicitud de la pila cuyo atributo coincide con el valor especificado.

Uso en el sistema:
Permite gestionar los casos urgentes de manera prioritaria, asegurando que las solicitudes más recientes se procesen primero, como ocurre en situaciones reales de atención inmediata.

CLASES Y ESTRUCTURA DEL SISTEMA

El sistema se organiza en un conjunto de clases interconectadas que modelan tanto los datos como las operaciones sobre las estructuras de tipo **lista simple enlazada**.
Cada solicitud, nodo y estructura de datos se representa como un objeto independiente, siguiendo principios de orientación a objetos y evitando el uso de estructuras nativas de Python para la gestión de los datos principales.

- **Clase Nodo**
  La clase **Nodo** representa la **unidad básica de una lista enlazada simple**, utilizada por las estructuras List, ColaSolicitudes y PilaSolicitudes. Cada nodo contiene un dato y una referencia al siguiente nodo, lo que permite enlazar los elementos entre sí y formar estructuras dinámicas.

  Atributos:
  - data: Contiene el valor u objeto almacenado en el nodo (por ejemplo, una solicitud).
  - next: Referencia al siguiente nodo en la lista. Si no hay más elementos, su valor es None.

  Metodos Principales:

  - getData/setData: Permiten acceder o modificar el contenido del nodo sin acceder directamente al atributo, garantizando encapsulación.
  - getNext/setNext: Permiten obtener o actualizar la referencia al siguiente nodo, asegurando la integridad de la estructura enlazada.

  Responsabilidad: El Nodo es la pieza fundamental de las estructuras enlazadas. Gracias a sus referencias (next), la lista puede crecer o reducirse dinámicamente, permitiendo que pilas y colas se construyan sobre ella sin límite de tamaño.

- **Clase Lista**

  La clase `List` implementa una **lista enlazada simple** que actúa como estructura base para las pilas y colas del sistema. Su diseño permite almacenar elementos en memoria de forma dinámica, sin depender de estructuras nativas de Python como listas o arrays. Cada elemento de la lista está representado por un objeto de tipo `Nodo`, que contiene un dato (`data`) y una referencia al siguiente nodo (`next`).

  Atributos:

  - head: permite acceder al primer nodo de la lista.
  - tail: permite acceder al último nodo de la lista.
  - _size: es el atributo que mantiene el tamaño o número de elementos almacenados en la lista

  Metodos Principales:

  - isEmpty(): Permite validar condiciones previas antes de operaciones de extracción o eliminación, previniendo errores de ejecución.
  - addFirst(data): Utilizado por la clase PilaUrgencias para aplicar el comportamiento LIFO (último en entrar, primero en salir). Facilita el registro de solicitudes urgentes.
  - addLast(data): Usado por ColaSolicitudes para mantener el comportamiento FIFO (primero en entrar, primero en salir) en las solicitudes normales.
  - removeFirst(): Permite extraer elementos tanto en pilas como en colas de manera ordenada según su política (LIFO o FIFO). Cumple el requisito de eliminación de elementos.}
  - removeLast(): Permite eliminar el último elemento de la lista. Se utiliza internamente en operaciones que requieren recorrer la lista completa.
  - eliminarPorAtributo(attr, value): Método clave en la implementación del sistema. Permite eliminar solicitudes específicas sin conocer su posición exacta.
  - mostrar(): Permite visualizar el contenido actual de la estructura sin usar iteradores ni listas nativas de Python, cumpliendo las restricciones del curso.

  Responsabilidad: Permitir la adaptación dinámica de las clases principales.

- **Clase Solicitud**
  Modela la información asociada a una solicitud de tutoría universitaria.

  Atributos:
  - estudiante: str
  - materia: str
  - fecha: str
  - tipo: str (normal / urgente)
  - prioridad: int(1 = normal, mayor valor = más urgente)

  Responsabilidad: Representar de forma completa cada solicitud de tutoría, incluyendo su nivel de prioridad y tipo. Esto permite ordenar, eliminar y mostrar la información dentro de las estructuras sin perder detalle.

- **Clase ColaSolicitudes (implementada mediante lista simple enlazada)**

  La clase ColaSolicitudes implementa una estructura de cola dinámica (FIFO) utilizando como base una lista simple enlazada, representada por la clase Lista. De esta forma, la gestión de los punteros al primer y último nodo se delega internamente a dicha lista, evitando el uso de estructuras nativas de Python y permitiendo un tamaño indefinido de elementos.

  Su propósito es simular una lista de espera de tutorías normales, donde las solicitudes se atienden estrictamente en el orden de llegada.

  Atributos:

  - data: Instancia de la clase Lista, encargada de gestionar los nodos enlazados que conforman la cola. Los punteros head y tail son manejados internamente dentro de esta estructura.

  Métodos principales:

  - enqueue(solicitud): Inserta una nueva solicitud al final de la cola delegando la operación en self.list.addLast(data). Representa el registro de una solicitud normal cuando el estudiante la envía; garantiza la semántica FIFO para la atención posterior.
  - dequeue(): Elimina y retorna la solicitud ubicada al frente de la cola (la más antigua).
  - isEmpty(): Indica si la cola está vacía.
  - mostrar(): Permite recorrer manualmente los nodos de la pila sin usar iteradores de Python.

  Responsabilidad: La clase ColaSolicitudes tiene como responsabilidad modelar y controlar la lista de espera de solicitudes de tutoría normales.

Gracias a su implementación con una lista enlazada (Lista), la estructura no requiere tamaño fijo y mantiene las referencias necesarias para realizar inserciones y eliminaciones eficientes sin estructuras nativas de Python.

- **Clase PilaSolicitudes (implementada mediante lista simple enlazada**

  La clase PilaSolicitudes implementa una pila dinámica (LIFO) basada en la clase Lista. Su función principal es gestionar solicitudes urgentes de tutorías, donde las operaciones más recientes se procesan primero, reproduciendo el comportamiento clásico de una pila.Atributos:

  - data: Instancia de la clase Lista que contiene los nodos enlazados que representan las solicitudes almacenadas en la pila.

  Métodos principales:

  - push(solicitud): Permite registrar solicitudes urgentes de último momento. Cumple con el principio LIFO.
  - pop(): Simula la atención o resolución de la solicitud más reciente.
  - mostrar(): Permite recorrer manualmente los nodos de la pila sin usar iteradores de Python.
  - eliminarPorAtributo(attr, value): Cumple el requisito del taller de incluir procedimientos de eliminación por atributo, aplicable a solicitudes urgentes específicas.

Responsabilidad: PilaSolicitudes modela la gestión de solicitudes urgentes dentro del sistema de tutorías. Se utiliza cuando se requiere prioridad inmediata, gracias a su implementación sobre la clase Lista, mantiene un tamaño dinámico y permite recorrer, eliminar o registrar elementos sin usar estructuras nativas.

- **Clase GestorTutorias**

  Coordina las operaciones entre la pila y la cola, actuando como el controlador principal del sistema.
  Se encarga de cargar datos desde archivos, procesar solicitudes, realizar búsquedas y exportar resultados.

  Atributos:

  - cola_normal: instancia de ColaSolicitudes.
  - pila_urgente: instancia de PilaSolicitudes.

  Métodos principales:

  • cargar_desde_archivo(ruta): Carga datos desde un archivo CSV, validando errores de archivo inexistente, formato incorrecto o

columnas faltantes.
• registrar_solicitud(solicitud): Inserta una solicitud en la estructura correspondiente según su tipo (normal o urgente).
• procesar_solicitud(): Atiende solicitudes pendientes, priorizando las urgentes.
• eliminar_solicitud(nombre): Elimina solicitudes específicas por atributo.
• guardar_resultados(): Exporta las solicitudes atendidas o canceladas a un archivo CSV.

Responsabilidad: Actuar como interfaz entre el usuario y las estructuras de datos, manteniendo la lógica de negocio del sistema.

RESTRICCIONES Y VALIDACIONES

Se manejan excepciones controladas en el método cargar_desde_archivo() para capturar errores de archivo inexistente, columnas faltantes o formato inválido, mostrando mensajes de advertencia en consola sin interrumpir la ejecución.

- No se permite el uso de estructuras nativas como listas de Python para almacenar datos (aclaración).
- Los métodos push, pop, enqueue, y dequeue deben operar directamente sobre nodos enlazados.
- La prioridad debe ser un valor numérico entre 1 y 5.
- No se pueden registrar dos solicitudes simultáneas del mismo estudiante para la misma materia y fecha.
- Si se intenta eliminar o buscar una solicitud inexistente, el sistema debe mostrar un mensaje de error controlado (considerando que nuestro único front end es la consola, viene bien por lo menos tipificar o capturar errores).

FUNCIONALIDADES CLAVE DEL SISTEMA

- Registrar solicitud normal (enqueue)
- Registrar solicitud urgente (push)
- Atender solicitud (dequeue o pop)
- Buscar solicitud (por nombre o materia)
- Eliminar solicitud (por nombre o materia)
- Ordenar solicitudes (por prioridad o fecha)
- Cargar solicitudes desde archivo
- Guardar solicitudes atendidas a archivo
- Mostrar en consola las solicitudes pendientes o procesadas

EXPLICACION

totalmente válido y pertinente para el desarrollo del sistema de gestión de solicitudes de tutorías universitarias, ya que permiten simular de manera fiel y organizada el flujo real con el que se atienden las peticiones académicas.

En primer lugar, la Cola (Queue) representa de forma natural la atención secuencial de las solicitudes normales, aplicando el principio FIFO (First In, First Out). En un contexto universitario, los estudiantes que solicitan tutorías lo hacen en orden de llegada, y este orden debe respetarse para garantizar equidad y organización. Así, las colas permiten que el sistema gestione una lista de espera dinámica, donde la primera solicitud registrada es también la primera en ser atendida.

Por otro lado, la Pila (Stack) se emplea para manejar las solicitudes urgentes, aplicando el principio LIFO (Last In, First Out). En la vida académica real, pueden surgir casos que requieran atención inmediata —por ejemplo, exámenes próximos o situaciones excepcionales—. Al usar una pila, el sistema puede priorizar las solicitudes más recientes y críticas, garantizando una respuesta rápida ante emergencias académicas sin alterar el flujo de las solicitudes normales.

Además, estas estructuras favorecen una gestión eficiente y dinámica de la memoria, ya que las pilas y colas implementadas con listas enlazadas no requieren un tamaño fijo y pueden crecer o reducirse conforme se registran o atienden solicitudes. Esto permite que el sistema funcione de manera flexible, sin depender de estructuras nativas de Python ni de límites predefinidos, cumpliendo con los principios de la programación orientada a objetos y del manejo estructurado de datos.

En conclusión, el uso de Pilas y Colas en este ejercicio no solo es técnicamente válido, sino también conceptualmente coherente con la realidad que se busca modelar: un proceso de atención a tutorías que distingue entre solicitudes regulares y urgentes, manteniendo orden, prioridad y trazabilidad en todas las operaciones. Estas estructuras permiten reflejar el comportamiento real del sistema de atención académica y facilitan la implementación de un flujo de trabajo justo, ordenado y eficiente.