Universidad Autónoma de Baja California

Facultad de Ciencias de la Ingeniería y Tecnología

Estructura de datos

Practica 3: Colas

Equipo: Uscanga Vizcarra Stefano Yahir

Reyes Morales Ángel Alejandro

Grupo: 544

Profesor: Muñoz Contreras José Manuel

Fecha: 01/10/2025

Tijuana, Baja California, México

Introducción

Con la práctica que se realizó sobre el uso e implementación de las colas en programación, se busca el identificar y aplicar las funciones que puede ofrecer una cola, para ello, se elaborara un programa que la implemente y se asemeje a una situación de la vida cotidiana la cual se le buscara dar solución tal es el caso como el de un gestor de turnos en una clínica.

Antecedentes

Una cola (queue) es una estructura de datos lineal que sigue el principio FIFO ("First In, First Out" — el primero en entrar es el primero en salir).

En el contexto de Python, una cola no es un tipo nativo especial, pero se puede implementar de diferentes maneras usando las estructuras del lenguaje o usando módulos de la biblioteca estándar. En particular, Python ofrece:

El módulo queue que define colas sincronizadas (FIFO), colas LIFO, colas con prioridad.

El contenedor collections.deque, que permite operaciones eficientes en ambos extremos, y puede usarse como cola FIFO.

Implementaciones más sencillas con listas (list), aunque con limitaciones de eficiencia.

¿Cómo se usa una cola en Python?

Las operaciones básicas de una cola son:

- enqueue (o poner / insertar): añadir un elemento al "final" (rear) de la cola.
- dequeue (o sacar / extraer): eliminar y retornar el elemento del "frente" (front).
- peek o front: ver cuál es el elemento del frente sin extraerlo.
- isEmpty: verificar si la cola está vacía.
- En colas con capacidad limitada: isFull, size o método equivalente.

¿Para qué se usa una cola?

- Las colas son útiles en muchas situaciones donde es importante mantener el orden de llegada. Algunos usos comunes:
- Planificación de tareas / scheduling: manejar trabajo en cola (jobs), por ejemplo en un sistema operativo o servidor de tareas.
- Buffer entre procesos o dispositivos: por ejemplo, cuando un productor genera datos más rápido que un consumidor los procesa.
- Procesamiento de eventos / colas de mensaje: por ejemplo, peticiones entrantes a un servidor web, colas en sistemas distribuidos.
- Algoritmos en grafos: por ejemplo, recorridos BFS (Breadth-First Search) usan una cola para visitar nodos en orden de nivel.
- Sistemas de impresión (spooling), atención de clientes, colas en interfaces de usuario.

Ventajas

- Orden garantizado (FIFO): Garantiza que los elementos sean procesados en el orden en que fueron agregados, lo cual es importante en muchos escenarios como colas de mensajes o pedidos.
- Operaciones eficientes: Con implementaciones apropiadas (por ejemplo deque o las clases del módulo queue), las operaciones de enqueue y dequeue pueden tener complejidad O(1).
- Sincronización entre productores/consumidores: En sistemas multihilo, las colas (por ejemplo queue.Queue) proporcionan mecanismos seguros para que varios hilos pongan o tomen elementos sin conflictos.
- Uso claro y modular: Separan la lógica de producción de datos de la lógica de consumo, lo que mejora la organización del código y facilita extensiones o cambios en el procesamiento. (Ventaja conceptual más que técnica).

Desventajas

- Inserción/eliminación en el medio es ineficiente: Las colas clásicas no permiten que se inserte o elimine un elemento en una posición arbitraria sin recorrer o desplazar.
- Búsqueda lenta (O(n)): Si necesitas buscar un elemento específico dentro de la cola, en general tendrás que recorrerla completamente.
- Tamaño fijo (en ciertas implementaciones): Si usas una implementación basada en array (o fijas), debes definir el tamaño máximo desde el inicio, lo que puede provocar "overflow" si la cola se llena.
- Memoria adicional o sobrecarga: En algunos casos, mantener los punteros front/rear, enlaces (en una lista enlazada) o administración interna implica costo extra en memoria y en operaciones internas de mantenimiento.

Desarrollo y análisis

1. Registrar Paciente en Cola

Proceso:

Captura de datos: Obtiene información del formulario (nombre, teléfono, fecha, hora, especialidad, tipo de turno)

Validación robusta:

Nombre: Solo letras, espacios y caracteres especiales del español

Teléfono: Solo números enteros

Especialidad: Selección obligatoria

Gestión de errores: Acumula todos los errores y los muestra en un solo mensaje

Registro en estructura: Utiliza encolar() para insertar un paciente al final de la cola

Confirmación: Mensaje de éxito y limpieza automática del formulario.

2. Llamar Siguiente Paciente

Implementación:

La función desencolar() implementa el patrón FIFO (First In, First Out) con priorización.

Características:

Eficiencia: O(1) - operación de tiempo constante

Priorización automática: Las emergencias ya están al frente

Información completa: Muestra datos del paciente llamado

Actualización automática: Refresca toda la interfaz

3. Ver cola de espera

Implementación:

La función obtener_lista_completa() convierte ambas colas (emergencia y general) en datos visualizables.

Funcionalidades:

Tabla interactiva: Muestra posición, datos del paciente y tiempo de espera

Código de colores:

Emergencias:

- Fondo rojo claro
- Turnos normales: Fondo gris claro

Información en tiempo real:

- Posición en cola
- Tiempo esperando (calculado dinámicamente)
- Tipo de turno claramente identificado
- Scrollbars: Para manejar listas largas

4. Turnos de Emergencia (Prioridad)

Implementación sofisticada:

El algoritmo de inserción prioriza emergencias manteniendo el orden FIFO dentro de cada categoría.

Ventajas del diseño:

- Prioridad absoluta: Emergencias siempre se atienden primero
- Orden justo: FIFO dentro de emergencias y turnos normales
- Flexibilidad: Puede manejar múltiples emergencias consecutivas
- Identificación visual: Checkbox destacado y colores diferenciados

5. Cancelar Turno

Implementación:

Dos métodos de cancelación: por selección en tabla o por búsqueda.

Características:

- Búsqueda insensible a mayúsculas: Más tolerante con el usuario
- Confirmación requerida: Previene eliminaciones accidentales
- Feedback inmediato: Mensaje de confirmación o error
- Actualización automática: La interfaz se refresca instantáneamente

6. Tiempo Estimado de Espera

Implementación multi-facética:

Funcionalidades:

- Tiempo actual de espera: Desde el momento del registro
- Tiempo estimado restante: Basado en posición × 15 minutos por consulta
- Consulta específica: Campo dedicado para buscar un paciente
- Estadísticas generales: Tiempo promedio de toda la cola

Información proporcionada:

- Tiempo ya esperado por el paciente
- Posición actual en la cola
- Estimación de tiempo restante
- Tipo de turno (afecta la prioridad)

Resultados

Todas las 6 funciones fueron implementadas exitosamente:

Registrar paciente: Validación robusta + interfaz intuitiva.

Llamar siguiente: Operación O(1) ultra eficiente.

Ver lista de espera: Tabla interactiva con colores y tiempo real.

Turnos de emergencia: Algoritmo sofisticado que prioriza correctamente.

Cancelar turno: Múltiples métodos con confirmación de seguridad.

Tiempo estimado: Cálculo preciso en tiempo real + estadísticas.

Fortalezas principales:

Estructura de datos inteligente: Cola de emergencia y simple ideales para el problema.

Algoritmo de priorización avanzado: Emergencias al frente, orden FIFO preservado.

Interfaz profesional: Colores, validaciones, retroalimentación constante.

Rendimiento excelente: Operaciones críticas en tiempo constante.

Sistema completamente funcional que cumple todos los requisitos y los supera. Demuestra comprensión sólida de estructuras de datos, algoritmos de cola con prioridad, y desarrollo de interfaces gráficas agradables a la vista.

Conclusión

El sistema implementa exitosamente todas las funciones requeridas con un diseño sólido que equilibra eficiencia técnica y usabilidad. La elección de dos colas como estructuras de datos principal es acertada para este tipo de aplicación, permitiendo operaciones eficientes de cola con priorización. La interfaz gráfica proporciona una experiencia de usuario completa con validaciones robustas y feedback constante.

Referencias

Zaczyński, B. (2023c, diciembre 1). *Python Stacks, Queues, and Priority Queues in Practice*. https://realpython.com/queue-in-python

Ghosh, A. K. (2025b, septiembre 23). *Queue in Data Structures - Types & Algorithm (With Example)*. ScholarHat. https://www.scholarhat.com/tutorial/datastructures/queue-data-structure-implementation

queue — A synchronized queue class. (s. f.-b). Python Documentation. https://docs.python.org/3/library/queue.html

5. Data structures. (s. f.-b). Python Documentation. https://docs.python.org/3/tutorial/datastructures.html

GeeksforGeeks. (2025f, septiembre 20). *Queue in Python*. GeeksforGeeks. https://www.geeksforgeeks.org/python/queue-in-python

W3Schools.com. (s. f.-d). https://www.w3schools.com/python/python_dsa_queues.asp