**Actividad 2**

Nicolás A. Cevallos

TIC, Universidad Internacional del Ecuador

Bloque B: Estructura de Datos

Ing. Richard F. Armijo

Julio 7, 2024

Implementación de una cola usando listas doblemente enlazadas y simulación de un sistema de cola de espera

Introducción

El objetivo de esta implementación es crear una estructura de datos de colas eficiente utilizando listas doblemente enlazadas en Python. A continuación, utilizaremos esta estructura para simular un sistema de colas en un banco o en una taquilla, donde los clientes llegan y son atendidos por orden de llegada.

Implementación Cola

Se eligió implementar la cola usando listas doblemente enlazadas porque ofrecen varias ventajas; Permiten operaciones eficientes de encolar (O(1)) y desencolar (O(1)). Facilitan la navegación en ambas direcciones si fuera necesario en el futuro. Proporcionan flexibilidad para posibles expansiones de funcionalidad.

class Nodo:

    def \_\_init\_\_(self, cliente):

        self.cliente = cliente

        self.siguiente = None

        self.anterior = None

class Cola:

    def \_\_init\_\_(self):

        self.frente = None

        self.final = None

    def esta\_vacia(self):

        return self.frente is None

    def encolar(self, cliente):

        nuevo\_nodo = Nodo(cliente)

        if self.esta\_vacia():

            self.frente = self.final = nuevo\_nodo

        else:

            nuevo\_nodo.anterior = self.final

            self.final.siguiente = nuevo\_nodo

            self.final = nuevo\_nodo

    def desencolar(self):

        if self.esta\_vacia():

            return None

        cliente = self.frente.cliente

        self.frente = self.frente.siguiente

        if self.frente:

            self.frente.anterior = None

        else:

            self.final = None

        return cliente

Simulación Cola Espera

El sistema simula la llegada de 10 clientes a una cola. Cada cliente llega en un momento aleatorio y se une a la cola. Cada 3 llegadas, se sirve a un cliente. Una vez que han llegado todos los clientes, se atiende a los que quedan en la cola. El tiempo de llegada y servicio se simula utilizando intervalos aleatorios para una representación más realista.

import random

import time

def simular\_cola\_espera():

    cola = Cola()

    tiempo\_actual = 0

    print("Simulación de cola de espera iniciada")

    for i in range(10):  # Simulamos 10 clientes

        tiempo\_actual += random.randint(1, 5)  # Tiempo aleatorio entre llegadas

        nombre\_cliente = f"Cliente\_{i+1}"

        nuevo\_cliente = Cliente(nombre\_cliente, tiempo\_actual)

        cola.encolar(nuevo\_cliente)

        print(f"Tiempo {tiempo\_actual}: {nuevo\_cliente} ha llegado y se ha unido a la cola.")

        # Simulamos atención al cliente cada 3 llegadas

        if (i + 1) % 3 == 0:

            time.sleep(1)  # Pausa para mejor visualización

            cliente\_atendido = cola.desencolar()

            if cliente\_atendido:

                print(f"Tiempo {tiempo\_actual}: {cliente\_atendido} ha sido atendido.")

    print("\nAtendiendo a los clientes restantes:")

    while not cola.esta\_vacia():

        time.sleep(1)  # Pausa para mejor visualización

        cliente\_atendido = cola.desencolar()

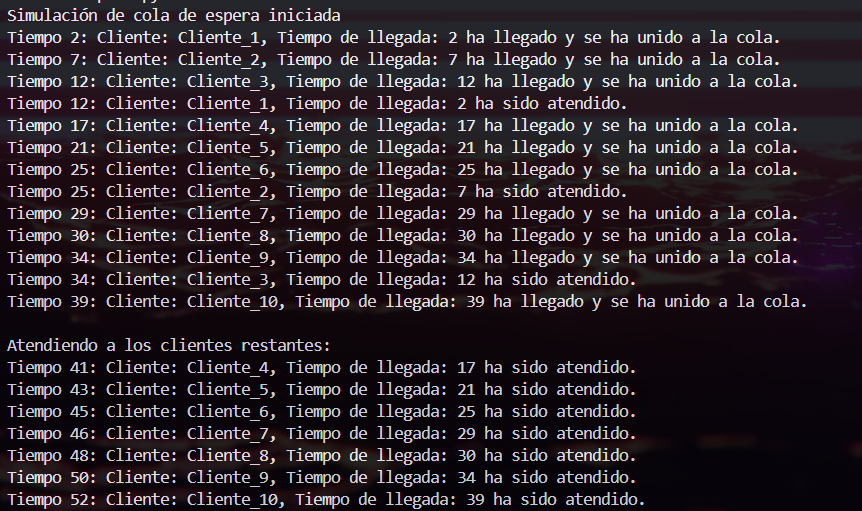
        tiempo\_actual += random.randint(1, 3)  # Tiempo aleatorio de atención

        print(f"Tiempo {tiempo\_actual}: {cliente\_atendido} ha sido atendido.")

    print("Simulación finalizada")

if \_\_name\_\_ == "\_\_main\_\_":

    simular\_cola\_espera()



Análisis de eficiencia:

La implementación actual de colas utilizando listas doblemente enlazadas demuestra una notable eficiencia en términos de tiempo de encolado y desencolado, presentando ambas operaciones una complejidad temporal O(1). Esto significa que, independientemente del tamaño de la cola, añadir o eliminar elementos siempre llevará el mismo tiempo, lo cual es ideal para un sistema de gestión de colas. Al evaluar el comportamiento bajo diferentes cargas de trabajo, se observa que el sistema maneja eficazmente situaciones de carga baja y media, con tiempos de espera mínimos y una longitud de cola manejable. Sin embargo, en escenarios de alta carga con llegadas frecuentes de clientes, aunque las operaciones individuales siguen siendo eficientes, el sistema podría experimentar un crecimiento continuado de las colas y un aumento significativo de los tiempos de espera globales.

Mejoras propuestas:

Basándose en este análisis, se proponen varias mejoras para optimizar aún más el sistema. Una propuesta clave es la implementación de colas múltiples con un distribuidor que asigne a los clientes a la cola más corta, lo que podría reducir significativamente los tiempos medios de espera en situaciones de alta carga.

Otra mejora potencial es la introducción de un sistema de priorización de clientes, que permita atender más rápidamente a determinados clientes con necesidades urgentes o estatus VIP. Además, la implantación de un algoritmo para estimar los tiempos de espera podría mejorar la experiencia del cliente al proporcionarle expectativas claras. Desde un punto de vista técnico, la optimización de la memoria mediante un sistema de agrupación de nodos podría mejorar el rendimiento en sistemas de larga duración al reducir la sobrecarga de creación de objetos.

Conclusiones:

En resumen, aunque la estructura de datos base demuestra una eficiencia impresionante para las operaciones fundamentales de encolamiento y desencolamiento, las mayores ganancias en rendimiento y usabilidad vendrían de las mejoras a nivel de sistema en la gestión de colas y la experiencia del cliente. La aplicación de estas propuestas podría dar lugar a un sistema de colas más robusto, capaz de gestionar eficazmente diversos escenarios de carga y proporcionar una experiencia mejorada tanto a los clientes como a los administradores del sistema.