Contents

“‘

Módulo 4: Simulación de Eventos Discretos (SED) - Clase 3: Profundizando en el Calendario de Eventos y su Implementación

1. Objetivos de la Clase:

- Comprender a fondo las ventajas y desventajas de diferentes estructuras de datos para implementar el calendario de eventos.
- Analizar en detalle la complejidad computacional de las operaciones clave del calendario (inserción, eliminación, búsqueda del evento más próximo) para cada estructura de datos.
- Implementar un calendario de eventos básico utilizando una estructura de datos (lista enlazada o heap) en un lenguaje de programación de su elección.
- Comprender cómo la elección de la estructura de datos afecta el rendimiento general de la simulación.

2. Contenido Teórico Detallado:

- **Repaso del Calendario de Eventos:**
  - Función principal: Mantener una lista ordenada de eventos futuros, crucial para la simulación de eventos discretos.
  - Actualización continua: A medida que avanza la simulación, los eventos se eliminan del calendario (cuando se ejecutan) y se insertan nuevos eventos.
  - Relación con el reloj de simulación: El calendario determina el tiempo actual de la simulación, ya que el reloj avanza al tiempo del próximo evento en el calendario.
- **Análisis Comparativo de Estructuras de Datos:**
  - **Listas Enlazadas:**
    - Ventajas: Simples de implementar.
    - Desventajas: La búsqueda del evento más próximo (el primero en la lista ordenada) puede ser O(n) en el peor caso (requiere recorrer toda la lista). La inserción también es O(n) si se debe mantener la lista ordenada. La eliminación del primer elemento es O(1).
    - Adecuada para: Simulaciones con un número muy pequeño de eventos en el calendario, donde la simplicidad prima sobre el rendimiento.
  - **Árboles Binarios de Búsqueda:**
    - Ventajas: Mejor rendimiento que las listas enlazadas para la mayoría de las operaciones. La búsqueda, inserción y eliminación son, en promedio, O(log n).
    - Desventajas: El rendimiento puede degradarse a O(n) en el peor caso (árbol desbalanceado). Requiere una implementación más compleja que las listas enlazadas.
    - Adecuada para: Simulaciones con un número moderado de eventos y donde se necesite un buen rendimiento general.
  - **Heaps (Montículos):**
    - Ventajas: La estructura más eficiente para el calendario de eventos. La inserción y eliminación del elemento mínimo (el evento más próximo) son O(log n).
    - Desventajas: Más complejo de implementar que las listas enlazadas, aunque existen bibliotecas que facilitan su uso.

    * Adecuada para: Simulaciones con un gran número de eventos, donde el rendimiento es crítico. Los min-heaps son la opción más común porque la raíz siempre contiene el evento más próximo en el tiempo.

- **Complejidad Computacional Detallada:**

  | Operación | Lista Enlazada (Ordenada) | Árbol Binario de Búsqueda (Balanceado) | Heap (Montículo) | | ---------- | ---------- | ---------- | ---------- | | Búsqueda del más próximo | O(1) | O(log n) | O(1) | | Inserción | O(n) | O(log n) | O(log n) | | Eliminación del más próximo | O(1) | O(log n) | O(log n) |

  - **Nota:** La complejidad de los árboles binarios de búsqueda asume que el árbol está balanceado. En la práctica, se pueden utilizar árboles auto-balanceables (AVL, Rojo-Negro) para garantizar un rendimiento O(log n) en todas las operaciones.

- **Consideraciones de Memoria:**

  - Todas las estructuras requieren memoria para almacenar los eventos y punteros. La cantidad de memoria utilizada depende del número de eventos y la cantidad de datos asociados a cada evento. Los heaps generalmente requieren menos memoria que los árboles binarios de búsqueda auto-balanceables.

3. Ejemplos o Casos de Estudio:

- **Simulación de un Centro de Llamadas:**
  - Eventos: Llegada de llamadas, inicio de atención, fin de atención.
  - Escenario: Simular un centro de llamadas con un gran volumen de llamadas entrantes.
  - Análisis: Un heap es la mejor opción para el calendario de eventos debido al gran número de eventos y la necesidad de un alto rendimiento.
- **Simulación de una Red de Computadoras:**
  - Eventos: Envío de paquetes, recepción de paquetes, expiración de timeouts.
  - Escenario: Simular una red de computadoras con un número moderado de nodos y paquetes.
  - Análisis: Un árbol binario de búsqueda (balanceado) puede ser una buena opción para el calendario de eventos, ofreciendo un buen equilibrio entre rendimiento y complejidad de implementación.

4. Problemas Prácticos o Ejercicios con Soluciones:

- **Ejercicio 1: Implementación de un Calendario de Eventos con Lista Enlazada:**
  - Implemente un calendario de eventos utilizando una lista enlazada ordenada.
  - Incluya las funciones para insertar un evento, eliminar el evento más próximo y mostrar el contenido del calendario.
  - Pruebe su implementación con un conjunto de eventos de prueba.
  - **Solución (Pseudo-código):**

  “‘python class Evento: def **init**(self, tiempo, tipo, datos): self.tiempo = tiempo self.tipo = tipo self.datos = datos

  class Calendario: def **init**(self): self.lista_eventos = [] # Lista enlazada simple

```
def insertar_evento(self, evento):
    # Insertar el evento en la posición correcta para mantener la lista ordenada
    if not self.lista_eventos:
        self.lista_eventos.append(evento)
    else:
        i = 0
        while i < len(self.lista_eventos) and evento.tiempo > self.lista_eventos[i].tiempo:
```

```
            i += 1
        self.lista_eventos.insert(i, evento) # Insertar en la posición i

def eliminar_evento_mas_proximo(self):
    if self.lista_eventos:
        return self.lista_eventos.pop(0) # Eliminar el primer elemento
    else:
        return None

def mostrar_calendario(self):
    for evento in self.lista_eventos:
        print(f"Tiempo: {evento.tiempo}, Tipo: {evento.tipo}, Datos: {evento.datos}")
```

Ejemplo de uso

calendario = Calendario() evento1 = Evento(5, ”Llegada”, {”cliente”: ”A”}) evento2 = Evento(2, ”Salida”, {”cliente”: ”B”}) evento3 = Evento(8, ”Llegada”, {”cliente”: ”C”})

calendario.insertar_evento(evento1) calendario.insertar_evento(evento2) calendario.insertar_evento(evento3)

print(”Calendario inicial:”) calendario.mostrar_calendario()

evento_proximo = calendario.eliminar_evento_mas_proximo() print(f”\nEvento próximo eliminado: Tiempo: {evento_proximo.tiempo}, Tipo: {evento_proximo.tipo}”)

print(”\nCalendario después de la eliminación:”) calendario.mostrar_calendario() “‘

- **Ejercicio 2: Comparación Empírica:**
  - Implemente un calendario de eventos utilizando una lista enlazada y un heap.
  - Genere un conjunto de eventos aleatorios con tiempos de ocurrencia aleatorios.
  - Mida el tiempo que tarda en insertar y eliminar un número grande de eventos utilizando cada implementación.
  - Compare los resultados y analice las diferencias en el rendimiento.
  - Discute por qué la diferencia se hace más notable al aumentar el número de eventos.
- **Ejercicio 3: Análisis de Escenarios:**
  - Considere los siguientes escenarios de simulación:
    - Un sistema de inventario con un número pequeño de productos.
    - Un sistema de control de tráfico aéreo con un gran número de vuelos.
    - Un sistema de colas en un banco con un número moderado de clientes.
  - ¿Qué estructura de datos recomendaría para el calendario de eventos en cada escenario? Justifique su respuesta.

5. Materiales Complementarios Recomendados:

- **Artículo Científico:** ”Discrete-Event Simulation Software: Survey and Analysis” - Este artículo ofrece una visión general de las diferentes herramientas y técnicas utilizadas en la simulación de eventos discretos, incluyendo una discusión sobre las estructuras de datos para el calendario de eventos.
- **Libro:** ”Simulation with Arena” por David Kelton, Randall Sadowski y Nancy Zupick - Este libro es una referencia estándar en la simulación de eventos discretos y cubre el calendario de eventos en detalle. (Los capítulos sobre manejo de eventos y estructuras de datos son los más relevantes).
- **Tutoriales en Línea:** Buscar tutoriales sobre implementaciones de heaps (min-heaps) en el lenguaje de programación de su elección. Sitios como GeeksforGeeks o tutorialspoint son excelentes recursos.
- **Visualizaciones:** Buscar animaciones o visualizaciones de heaps y árboles binarios de búsqueda. Entender visualmente cómo funcionan estas estructuras puede ayudar a comprender mejor su rendimiento.