Contents

Clase 2: El Calendario de Eventos en la Simulación de Eventos Discretos

Objetivos de la Clase:

- Comprender en profundidad la estructura y el funcionamiento del calendario de eventos.
- Aprender a insertar, eliminar y procesar eventos en el calendario.
- Entender cómo el calendario de eventos impulsa el avance del tiempo en la simulación.
- Implementar un calendario de eventos básico utilizando una estructura de datos adecuada.

Contenido Teórico Detallado:

1. Introducción al Calendario de Eventos:

El calendario de eventos es la estructura de datos central en la simulación de eventos discretos. Mantiene una lista ordenada de eventos futuros, donde cada evento está asociado a un tiempo de ocurrencia específico. En lugar de avanzar el tiempo en incrementos fijos, la simulación salta directamente al tiempo del próximo evento en el calendario, procesa ese evento y actualiza el estado del sistema.

2. Estructura del Calendario de Eventos:

- **Eventos:** Cada evento en el calendario representa una acción que cambiará el estado del sistema. Un evento típicamente incluye:
  - **Tipo de Evento:** Identifica la acción que debe realizarse (e.g., llegada de un cliente, fin de servicio).
  - **Tiempo de Ocurrencia:** El momento específico en el tiempo de simulación en que el evento debe ser procesado.
  - **Entidad Asociada (opcional):** El objeto que está directamente involucrado en el evento (e.g., el cliente que llega).
  - **Otros Atributos:** Información adicional necesaria para procesar el evento.
- **Ordenamiento:** El calendario de eventos debe mantener los eventos ordenados por tiempo de ocurrencia, generalmente en orden ascendente (el evento con el tiempo más cercano al actual se encuentra al principio).
- **Implementación:** El calendario de eventos se puede implementar usando varias estructuras de datos, incluyendo:
  - **Listas Enlazadas:** Simples de implementar, pero la inserción de eventos puede ser costosa si la lista es larga.
  - **Árboles Binarios de Búsqueda:** Ofrecen un buen rendimiento para inserciones y eliminaciones.
  - **Montículos (Heaps):** Son la estructura de datos más común debido a su eficiencia para encontrar el evento con el tiempo de ocurrencia más temprano (O(1)) y para insertar y eliminar eventos (O(log n)).

3. Operaciones en el Calendario de Eventos:

- **Inserción de Eventos:** Cuando un nuevo evento necesita ser programado (e.g., un cliente llega a una cola), se crea un nuevo registro de evento y se inserta en el calendario de eventos en la posición correcta, manteniendo el orden temporal.
- **Eliminación del Próximo Evento:** En cada paso de la simulación, el evento con el tiempo de ocurrencia más temprano se elimina del calendario. Este es el evento que se va a procesar.
- **Procesamiento de Eventos:** Una vez que un evento se elimina del calendario, su lógica asociada se ejecuta. Esto puede implicar actualizar el estado del sistema, generar nuevos eventos y programarlos en el calendario.

4. Avanzando el Tiempo en la Simulación:

El tiempo de simulación no avanza de manera continua, sino a saltos. Después de procesar un evento, el tiempo de simulación se actualiza al tiempo de ocurrencia del evento que acaba de ser procesado. La simulación luego busca el siguiente evento en el calendario y repite el proceso.

Ejemplo o Caso de Estudio:

Consideremos una simulación de una barbería con un solo barbero.

- **Eventos:**
  - `LlegadaCliente`: Un cliente llega a la barbería.
  - `FinServicio`: El barbero termina de cortar el pelo a un cliente.
- **Simulación:**
  1. Inicialmente, se programa un evento `LlegadaCliente` en el calendario (con un tiempo generado aleatoriamente).
  2. La simulación extrae el evento `LlegadaCliente` del calendario.
  3. El tiempo de simulación se actualiza al tiempo de llegada del cliente.
  4. Se verifica si el barbero está libre.
     - Si está libre, se programa un evento `FinServicio` (con un tiempo basado en la duración del corte de pelo).
     - Si está ocupado, el cliente se une a la cola.
  5. Se programa el siguiente evento `LlegadaCliente` en el calendario.
  6. La simulación extrae el siguiente evento del calendario (ya sea `FinServicio` o `LlegadaCliente`).
  7. Se repiten los pasos 3-6 hasta que se alcance el tiempo de simulación deseado o se cumpla alguna condición de parada.

Problemas Prácticos o Ejercicios con Soluciones:

1. **Ejercicio:** Dada la siguiente secuencia de eventos y sus tiempos de ocurrencia:

   - Evento A: Tiempo 5
   - Evento B: Tiempo 2
   - Evento C: Tiempo 8
   - Evento D: Tiempo 1

   Muestra el orden en que estos eventos serían procesados por el calendario de eventos.

   Solución: D -> B -> A -> C

2. **Ejercicio:** Describe los pasos necesarios para insertar un nuevo evento en un calendario de eventos implementado con una lista enlazada ordenada por tiempo de ocurrencia. Considera los casos donde el nuevo evento tiene el tiempo de ocurrencia más temprano, más tardío, o intermedio.

   Solución:

   (a) Crear un nuevo nodo para el evento.
   (b) Recorrer la lista enlazada desde el principio.
   (c) Comparar el tiempo de ocurrencia del nuevo evento con el tiempo de ocurrencia de cada nodo de la lista.
   - Si el tiempo de ocurrencia del nuevo evento es menor que el tiempo de ocurrencia del nodo actual, insertar el nuevo nodo antes del nodo actual y terminar. (Caso: más temprano)
   - Si se llega al final de la lista sin encontrar un nodo con un tiempo de ocurrencia mayor, insertar el nuevo nodo al final de la lista. (Caso: más tardío)
   - Si se encuentra un nodo con el mismo tiempo de ocurrencia, insertar el nuevo nodo antes o después, dependiendo de la lógica de desempate deseada.

   (d) **Ejercicio:** Implementa en pseudocódigo la lógica para eliminar el evento más próximo del calendario de eventos implementado como un *min-heap*.

“‘ Función EliminarProximoEvento(Heap calendario): Si calendario.tamaño == 0: Retornar Nulo // El calendario está vacío

proximoEvento = calendario.elementos[0] // El evento más próximo está en la raíz

// Reemplazar la raíz con el último elemento del heap calendario.elementos[0] = calendario.elementos[calendario.tamaño - 1] calendario.tamaño = calendario.tamaño - 1

// Reorganizar el heap para mantener la propiedad min-heap Heapify(calendario, 0)

Retornar proximoEvento

Función Heapify(Heap calendario, índice i): menor = i izquierda = 2 * i + 1 derecha = 2 * i + 2

Si izquierda < calendario.tamaño y calendario.elementos[izquierda].tiempo < calendario.elementos[menor].tiempo: menor = izquierda

Si derecha < calendario.tamaño y calendario.elementos[derecha].tiempo < calendario.elementos[menor].tiempo: menor = derecha

Si menor != i: Intercambiar(calendario.elementos[i], calendario.elementos[menor]) Heapify(calendario, menor) // Recursivamente heapify el subárbol afectado "'

Materiales Complementarios Recomendados:

- **Libro:** "Discrete-Event System Simulation" de Jerry Banks, John S. Carson II, Barry L. Nelson, David M. Nicol. Capítulo sobre Calendario de Eventos.
- **Artículo:** "Efficient Event List Management for Discrete Event Simulation" de R. Fujimoto. (Buscar en IEEE Xplore o ACM Digital Library).
- **Simuladores:** Revisar la implementación del calendario de eventos en simuladores como Arena, Simio, o AnyLogic. (La documentación de estos simuladores puede proporcionar información adicional).