1. **Diferencia entre lista simplemente enlazada y doblemente enlazada**  
   En una lista **simplemente enlazada**, cada nodo tiene un puntero al siguiente nodo, permitiendo recorrerla en una sola dirección (de cabeza a cola). En una lista **doblemente enlazada**, cada nodo tiene dos punteros: uno al siguiente nodo y otro al anterior, lo que permite recorrerla en ambas direcciones (hacia adelante y hacia atrás).
2. **Ventajas y desventajas de una lista doblemente enlazada**

**Ventajas**:

* Permite recorrer la lista en ambas direcciones.
* Operaciones como eliminar un nodo o insertar antes de un nodo son más eficientes, ya que no se requiere recorrer toda la lista para acceder al nodo anterior.
* Facilita operaciones en ambos extremos (head y tail) en tiempo constante si existen referencias a ambos nodos.

**Desventajas**:

* Mayor consumo de memoria al necesitar un puntero adicional (*prev*).
* Operaciones de inserción/eliminación requieren actualizar dos punteros en lugar de uno, aumentando ligeramente la complejidad del código.

1. **Inserción de un nodo en una lista doblemente enlazada**  
   Se quiere insertar un nuevo nodo **N** después de un nodo ya existente **P**:
2. Establecer los punteros de **N**:
   * N->prev = P
   * N->next = P->next
3. Si **P->next** no es nulo, actualizar el prev del nodo siguiente:
   * P->next->prev = N
4. Actualizar el next de **P**:
   * P->next = N

**Casos aparte**:

* Inserción al inicio: Actualizar head y el puntero siguiente.
* Inserción al final: Actualizar tail y el punto anterior.

1. **Eliminación de un nodo en una lista doblemente enlazada**  
   Para eliminar un nodo **N**:
2. Si **N->prev** existe, actualizar su next:
   * N->prev->next = N->next
3. Si **N->next** existe, actualizar su prev:
   * N->next->prev = N->prev
4. Liberar la memoria de **N**.

**Casos aparte**:

* Si **N** es el nodo cabeza (head), actualizar head = N->next.
* Si **N** es el nodo cola (tail), actualizar tail = N->prev.

1. **Eficiencia de una lista doblemente enlazada frente a un array**  
   Una lista doblemente enlazada es más eficiente y útil cuando:

* Se requieren modificaciones frecuentes **en medio de la lista**, ya que no hay que desplazar elementos (costo O(1) si se tiene la referencia).
* Se necesitan operaciones en **ambos extremos** (ej: implementar una cola doble o deque), aprovechando acceso directo a head y tail.
* El tamaño es dinámico y no se conoce de antemano.
* No se requiere acceso aleatorio (índices), ya que en arrays esto es O(1), mientras que en listas es O(n).

1. **Ejemplos de uso**

* **Sistemas de "Undo/Redo" en editores de texto o gráficos**
  + Cada acción (como escribir, borrar o mover elementos) se almacena como un nodo.
  + Al usar **Undo**, se retrocede al nodo anterior (prev).
  + Al usar **Redo**, se avanza al nodo siguiente (next).
  + Ejemplo: Photoshop, Microsoft Word, o editores de código.
* **Navegación en reproductores multimedia**
  + En una lista de reproducción (playlist), los nodos representan canciones o videos.
  + Permite avanzar (next) a la siguiente canción o retroceder (prev) a la anterior.
  + Ejemplo: Spotify, VLC, o reproductores de música en consolas.
* **Aplicaciones de diseño CAD (Computer-Aided Design)**
  + Para manipular capas o componentes que pueden reordenarse hacia adelante o atrás en la jerarquía.
  + Ejemplo: AutoCAD, SolidWorks.

**1. Inserción de un nodo en una lista circular**  
La inserción depende de si la lista está vacía o no. Si está vacía, el nuevo nodo se apunta a sí mismo. Si no, se ajustan los punteros para mantener la circularidad:

* Al insertar al inicio, el nuevo nodo se enlaza al siguiente del último nodo existente.
* Al insertar al final, el nuevo nodo se enlaza al primer nodo, y el penúltimo nodo ahora apunta al nuevo.
* Siempre se garantiza que el último nodo apunte al primero.

**2. Eliminación de un nodo sin romper la estructura**  
Se requiere localizar el nodo a eliminar y reajustar los punteros de sus nodos adyacentes para "saltarlo". Si el nodo eliminado es el único, la lista se vacía. En caso contrario, el nodo anterior al eliminado debe apuntar al siguiente del eliminado, manteniendo la continuidad del ciclo.

**3. Recorrido sin bucle infinito**  
Se utiliza un nodo de referencia (punto de partida) y se avanza hasta que el recorrido regrese a ese nodo. Alternativamente, se puede contar el número de elementos y detenerse después de recorrer esa cantidad.

**4. Aplicaciones comunes**

* **Algoritmos de scheduling** (Round Robin en sistemas operativos).
* **Buffers circulares** para gestión de flujos de datos (ej: colas de audio/video).
* **Problemas cíclicos** como el problema de Josephus o rotación de turnos.
* Implementación de estructuras como colas o memorias caché con reutilización cíclica.

**5. Lista doblemente enlazada circular**  
Para convertir una lista doblemente enlazada en circular:

* El campo siguiente del último nodo apunta al primero.
* El campo anterior del primer nodo apunta al último.
* Esto permite navegar en ambas direcciones sin fin, manteniendo una estructura cerrada y coherente.

**6. Ejemplos de uso de listas circulares**

1. **Control de Tráfico Aéreo**
   * **Aplicación**: Gestionar el orden de aterrizaje/despegue de aviones en una pista única.
   * **Ejemplo**: Los aviones se añaden a una lista circular y se atienden en secuencia.
2. **Juegos de Mesa**
   * **Aplicación**: Determinar el orden de los jugadores en juegos como Monopoly o Parchís.
   * **Ejemplo**: Cada jugador se representa como un nodo, y el turno avanza al siguiente nodo cíclicamente.
3. **Carruseles de Contenido en Webs/Apps**
   * **Aplicación**: Mostrar banners, productos o anuncios en rotación automática.
   * **Ejemplo**: Un slider de imágenes que vuelve al inicio después de la última.
4. **Sistemas de Almacenamiento en Búfer**
   * **Aplicación**: Manejar flujos continuos de datos (audio, video, sensores).
   * **Ejemplo**: Un micrófono graba audio en un búfer circular, sobrescribiendo datos antiguos cuando se llena.

Las listas circulares son ideales cuando:

* Los elementos deben **recorrer un ciclo infinito**.
* Se necesita **reutilizar recursos** de manera eficiente.
* La **secuencia de procesamiento** debe reiniciarse automáticamente.