Video11 (1)

Insertar al Final en Listas Enlazadas Simples

Conceptos Principales

1. a
2. b
3. c

Notas

Concepto General

- En este video se aborda el **algoritmo para insertar un nodo al final** de una **lista enlazada simple**.
- Se usan **diagramas de nodos** como herramienta visual (muy recomendados para tareas y comprensión).
- Reglas clave:
  - **No existe acceso a memoria sin un puntero que la referencie.**
  - Siempre se inicia con un puntero llamado `inicio`, que será la **cabeza de la lista**.

Situación Inicial

- `inicio` apunta a **NULL** (puntero nulo → `0x0000`).
- Cuando la lista está vacía:
  - Insertar el primer nodo implica que `inicio` apunte a ese nuevo nodo.

Estructura del Nodo

Cada nodo tiene **dos campos**:

1. **Valor (entero)** → identificador del nodo (ej. 11, 40, 31...)
2. **Siguiente (puntero)** → apunta al siguiente nodo o a `NULL` si es el último.

Ejemplo:

```
[ Valor | Siguiente ]
```

Algoritmo de Inserción al Final

1. **Crear nuevo nodo**
   - Llamarlo temporalmente `nuevo`.
   - Asignar:
     - `nuevo.valor = <dato>`
     - `nuevo.siguiente = NULL` (porque es el último).
2. **Caso especial**: Si la lista está vacía (`inicio == NULL`):
   - `inicio = nuevo`
3. **Caso normal** (lista NO vacía):
   - Crear un puntero auxiliar → `actual`
   - Inicializarlo: `actual = inicio`
   - Recorrer la lista hasta el último nodo:

```
while(actual->siguiente != NULL)
    actual = actual->siguiente;
```

   - Insertar:

```
actual->siguiente = nuevo;
```

Representación Visual

- **Antes** (con un solo nodo 11):

```
inicio → [ 11 | NULL ]
```

- **Después de insertar 40**:

```
inicio → [ 11 | * ] → [ 40 | NULL ]
```

- **Después de insertar 31**:

```
inicio → [ 11 | * ] → [ 40 | * ] → [ 31 | NULL ]
```

Ejemplo Paso a Paso

1. Insertar 11:
   - Crear `nuevo(11, NULL)`
   - Como `inicio == NULL` → `inicio = nuevo`
2. Insertar 40:
   - Crear `nuevo(40, NULL)`
   - `actual = inicio`
   - `actual->siguiente == NULL` → asignar `actual->siguiente = nuevo`
3. Insertar 31:
   - Crear `nuevo(31, NULL)`
   - Recorrer con `actual` hasta el último nodo
   - Insertar al final

Detalles Importantes

- **Nunca modificar `inicio`** directamente (salvo en el primer nodo).
- **El último nodo siempre apunta a NULL**.
- Si accidentalmente hacemos:

  ```
  actual->siguiente = NULL
  ```

  → Se **pierde la referencia** a los demás nodos (memoria queda reservada, pero inaccesible → **memory leak**).
- Cada nodo creado debe estar **dinámicamente en el heap** (usando `malloc` en C).

Pseudocódigo Simplificado

```
nuevo = crearNodo(valor)
nuevo->siguiente = NULL

if(inicio == NULL)
    inicio = nuevo
else
    actual = inicio
    while(actual->siguiente != NULL)
        actual = actual->siguiente
    actual->siguiente = nuevo
```

Concepto General

- En este video se aborda el **algoritmo para insertar un nodo al final** de una **lista enlazada simple**.
- Se usan **diagramas de nodos** como herramienta visual (muy recomendados para tareas y comprensión).
- Reglas clave:
  - **No existe acceso a memoria sin un puntero que la referencie.**
  - Siempre se inicia con un puntero llamado `inicio`, que será la **cabeza de la lista**.

Situación Inicial

- `inicio` apunta a **NULL** (puntero nulo → `0x0000`).
- Cuando la lista está vacía:
  - Insertar el primer nodo implica que `inicio` apunte a ese nuevo nodo. **Estructura del Nodo**

Cada nodo tiene **dos campos**:

1. **Valor (entero)** → identificador del nodo (ej. 11, 40, 31...)
2. **Siguiente (puntero)** → apunta al siguiente nodo o a `NULL` si es el último.

Ejemplo:

```
[ Valor | Siguiente ]
```

Algoritmo de Inserción al Final

1. **Crear nuevo nodo**
   - Llamarlo temporalmente `nuevo`.
   - Asignar:
     - `nuevo.valor = <dato>`
     - `nuevo.siguiente = NULL` (porque es el último).
2. **Caso especial**: Si la lista está vacía (`inicio == NULL`):
   - `inicio = nuevo`
3. **Caso normal** (lista NO vacía):
   - Crear un puntero auxiliar → `actual`
   - Inicializarlo: `actual = inicio`
   - Recorrer la lista hasta el último nodo:

```
while(actual->siguiente != NULL)
    actual = actual->siguiente;
```

- Insertar:

```
actual->siguiente = nuevo;
```

Representación Visual

- **Antes** (con un solo nodo 11):

```
inicio → [ 11 | NULL ]
```

- **Después de insertar 40**:

```
inicio → [ 11 | * ] → [ 40 | NULL ]
```

- **Después de insertar 31**:

```
inicio → [ 11 | * ] → [ 40 | * ] → [ 31 | NULL ]
```

Ejemplo Paso a Paso

1. Insertar 11:
   - Crear `nuevo(11, NULL)`
   - Como `inicio == NULL` → `inicio = nuevo`
2. Insertar 40:
   - Crear `nuevo(40, NULL)`
   - `actual = inicio`
   - `actual->siguiente == NULL` → asignar `actual->siguiente = nuevo`
3. Insertar 31:
   - Crear `nuevo(31, NULL)`
   - Recorrer con `actual` hasta el último nodo
   - Insertar al final

Detalles Importantes

- **Nunca modificar** `inicio` directamente (salvo en el primer nodo).

- **El último nodo siempre apunta a NULL**.
- Si accidentalmente hacemos:

```
actual->siguiente = NULL
```

  → Se **pierde la referencia** a los demás nodos (memoria queda reservada, pero inaccesible → **memory leak**).

- Cada nodo creado debe estar **dinámicamente en el heap** (usando `malloc` en C).

Pseudocódigo Simplificado

```
nuevo = crearNodo(valor)
nuevo->siguiente = NULL

if(inicio == NULL)
    inicio = nuevo
else
    actual = inicio
    while(actual->siguiente != NULL)
        actual = actual->siguiente
    actual->siguiente = nuevo
```