DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Análisis de Algoritmos y Estructura de Datos

TDA lista enlazada

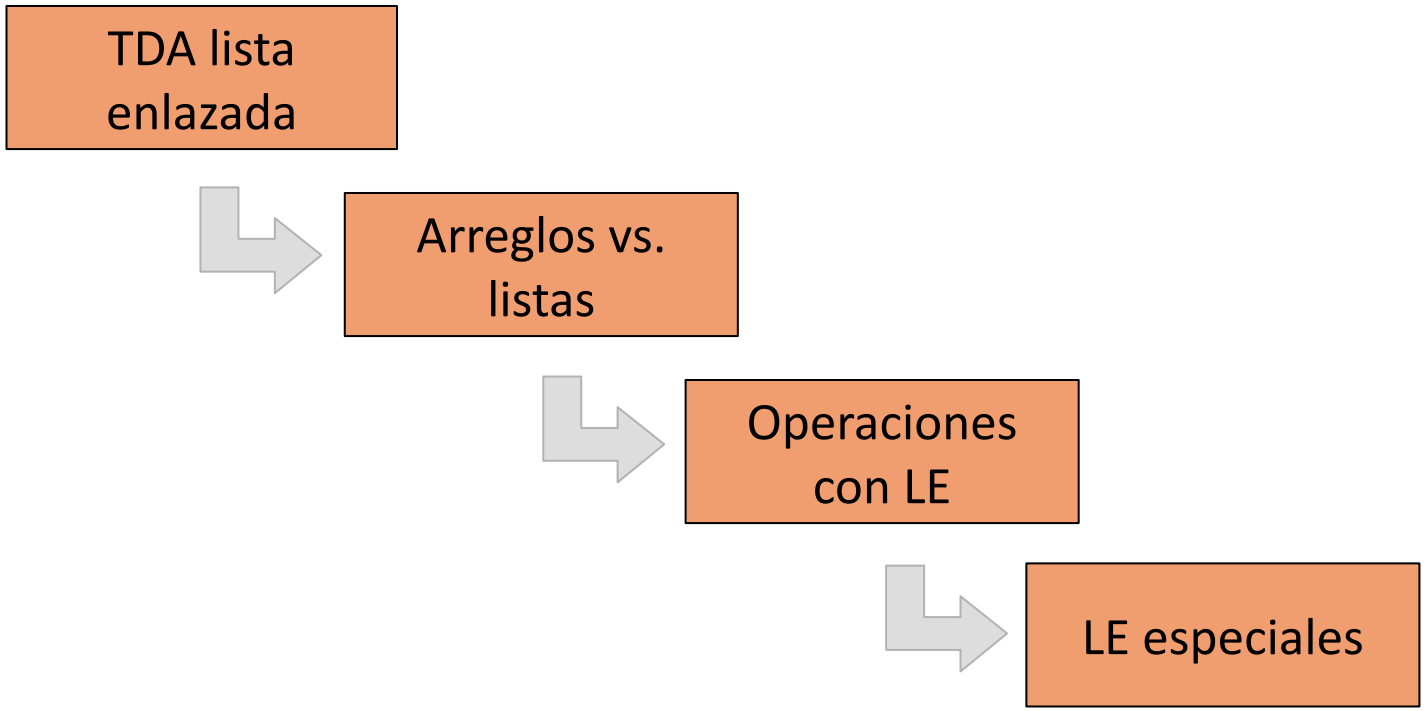
Prof. Violeta Chang C

Semestre 1 – 2022

TDA Lista Enlazada

- **Contenidos**:
  - Especificación de TDA lista enlazada
  - Operaciones con listas enlazadas
  - Listas enlazadas especiales
- **Objetivos**:
  - Entender y explicar estructura de datos de TDA lista enlazada
  - Diferenciar listas enlazadas y arreglos
  - Comprender funcionamiento de operaciones con listas enlazadas y determinar su complejidad
  - Conocer distintos tipos de listas enlazadas

Ruta de la sesión



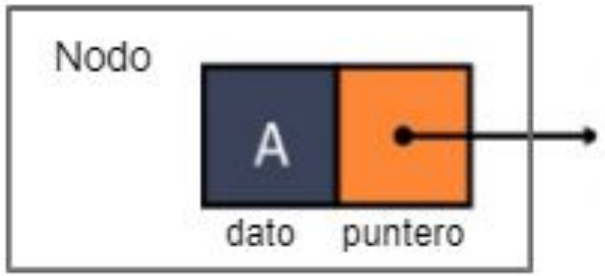
Especificación de TDA lista enlazada

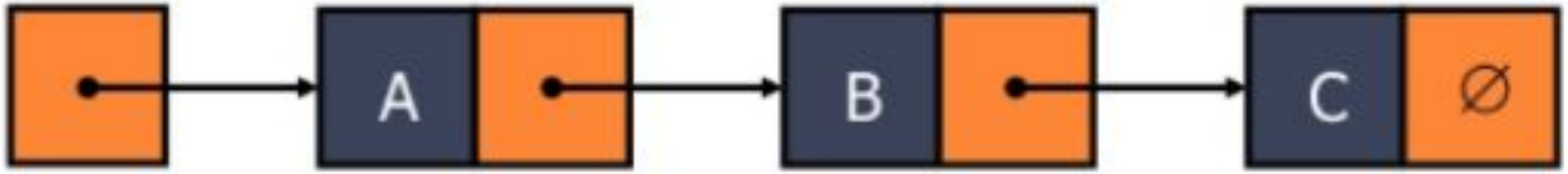
- **Estructura de datos**:
  - Una lista enlazada (LE) es una colección lineal de largo indeterminado con componentes homogéneos.
  - Homogéneo: Todos los componentes son del mismo tipo
  - Lineal: Componentes están ordenados en una línea por eso se conocen como listas enlazadas lineales

Khuram Shahzad, Data structures and algorithms

Especificación de TDA lista enlazada

- **Estructura de datos**:
  - Una lista enlazada (LE) es una secuencia de nodos conectados
  - A una lista con 0 nodos se le conoce como lista vacía
  - Cada nodo contiene:
    - Una parte de datos (cualquier tipo)
    - Un puntero al siguiente nodo de la lista
  - Cabeza: puntero al primer nodo
  - El último nodo apunta a nulo



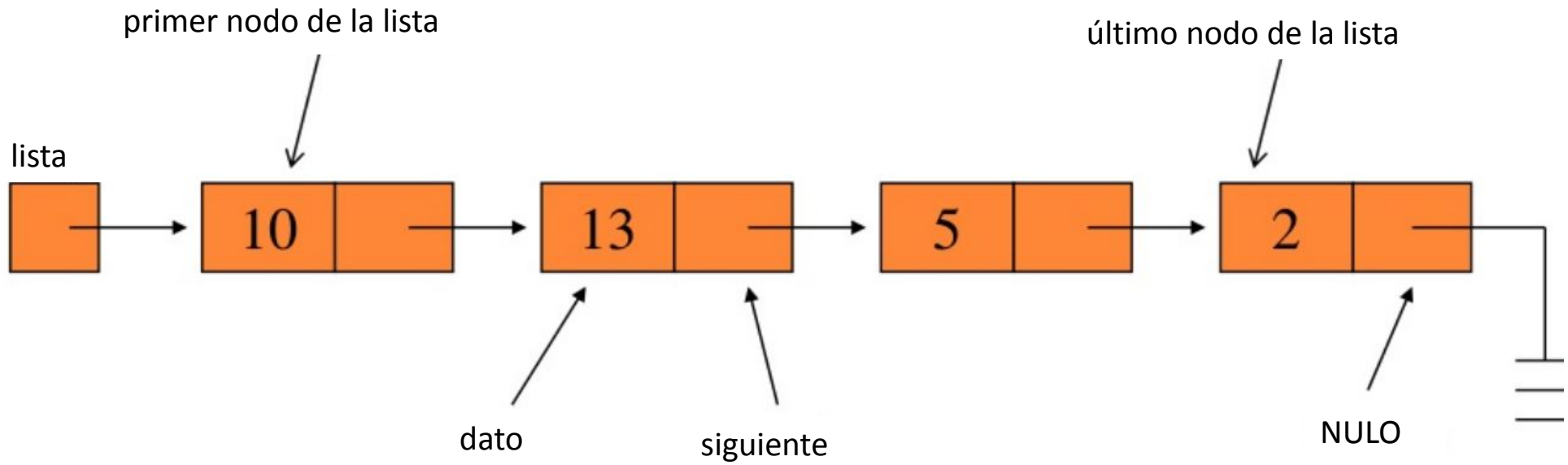


Khuram Shahzad, Data structures and algorithms

Especificación de TDA lista enlazada

- **Operaciones:**
  - **esListaVacía(L)**: determina si lista L está vacía o no
  - **insertarNodo(L,dato)**: inserta un nodo con dato en lista L
  - **eliminarNodo(L,dato)**: elimina nodo con dato de lista L
  - **buscarDato(L,dato)**: busca dato en lista L
  - **recorrerLista(L)**: muestra contenido de cada nodo de lista L

Especificación de TDA lista enlazada



Lista enlazada de enteros

Khuram Shahzad, Data structures and algorithms

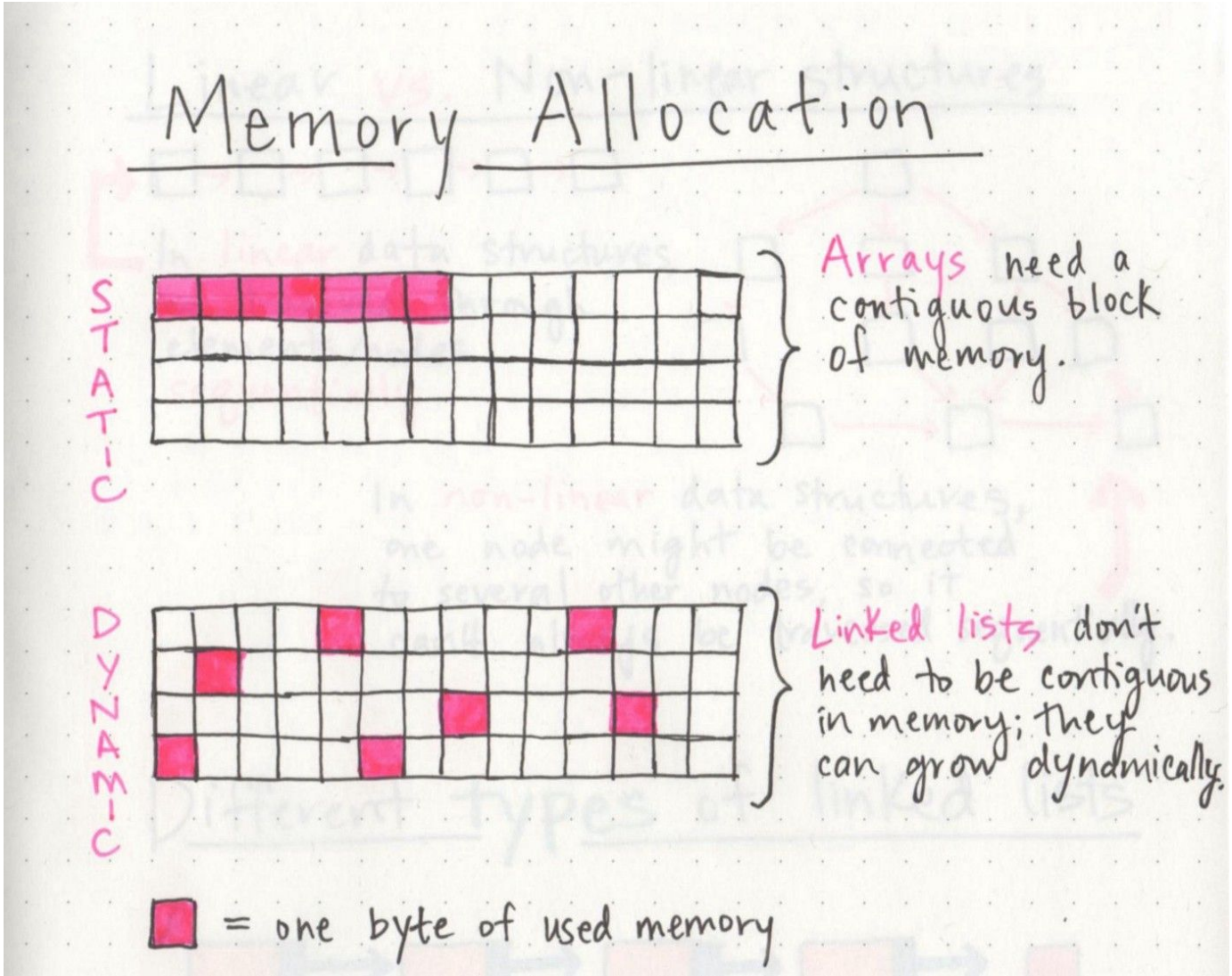
Arreglos vs. listas enlazadas

- Los arreglos son listas que tiene un tamaño fijo en memoria
- El programador debe estar pendiente del largo del arreglo
- Sin importar cuántos elementos de un arreglo sean usados en realidad, el arreglo tiene la misma cantidad de espacio de memoria ocupada
- Los elementos de un arreglo están almacenados en posiciones de memoria sucesivas. Además, el orden de los elementos almacenados en un arreglo es la misma tanto lógica como físicamente

Arreglos vs. listas enlazadas

- Una lista enlazada ocupa tanto espacio de memoria como sea necesario para almacenar el largo de la lista
- La lista se expande o contrae dependiendo si se insertan o eliminan nodos
- En una lista enlazada, los elementos no se almacenan en posiciones sucesivas de memoria
- Los elementos de una lista enlazada pueden ser insertados y eliminados tanto al inicio, al final o en una posición intermedia

Arreglos vs. listas enlazadas



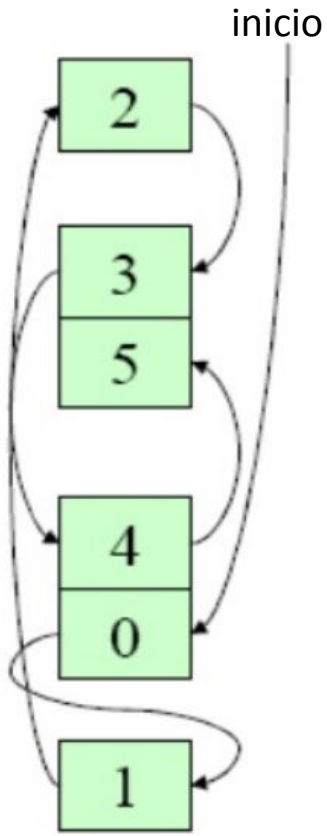
Daniel Ko, Singly Linked List in Ruby

Arreglos vs. listas enlazadas

- Trabajar con listas enlazadas es más complejo que con arreglos, pero tienen ventajas:
  - **Dinámica**: una LE puede crecer o contraerse fácilmente
    - No es necesario saber cuántos nodos habrán en la lista
    - Se va asignando memoria según se vaya necesitando
  - **Inserciones y eliminaciones fáciles y rápidas**
    - Para insertar o eliminar un elemento en un arreglo, es necesario copiar variables temporales para hacer espacio para nuevos elementos o cerrar espacios causados por elementos eliminados
    - Con una LE, no hay necesidad de mover otros nodos. Sólo se necesita actualizar algunos punteros

Arreglos vs. listas enlazadas

Arreglo



Lista enlazada

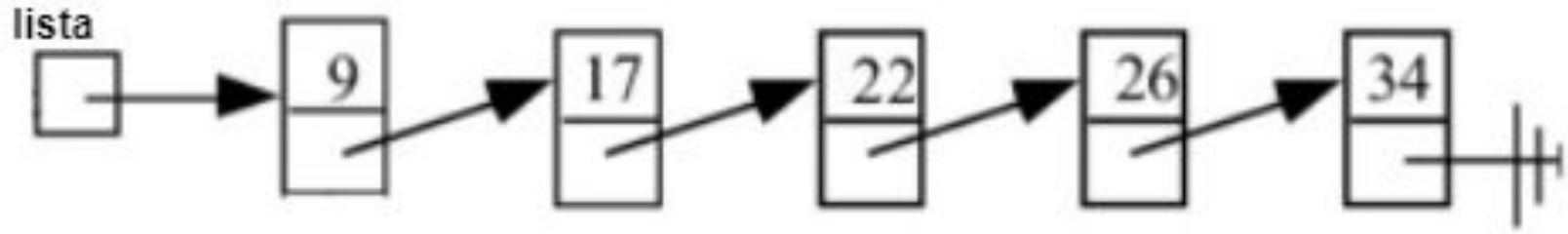
Khuram Shahzad, Data structures and algorithms

Operaciones con listas enlazadas

- Recorrido de lista
- Inserción de nodo
  - Inserción al inicio
  - Inserción al final
  - Inserción arbitraria
- Eliminación de nodo
  - Eliminación al inicio
  - Eliminación al final
  - Eliminación arbitraria

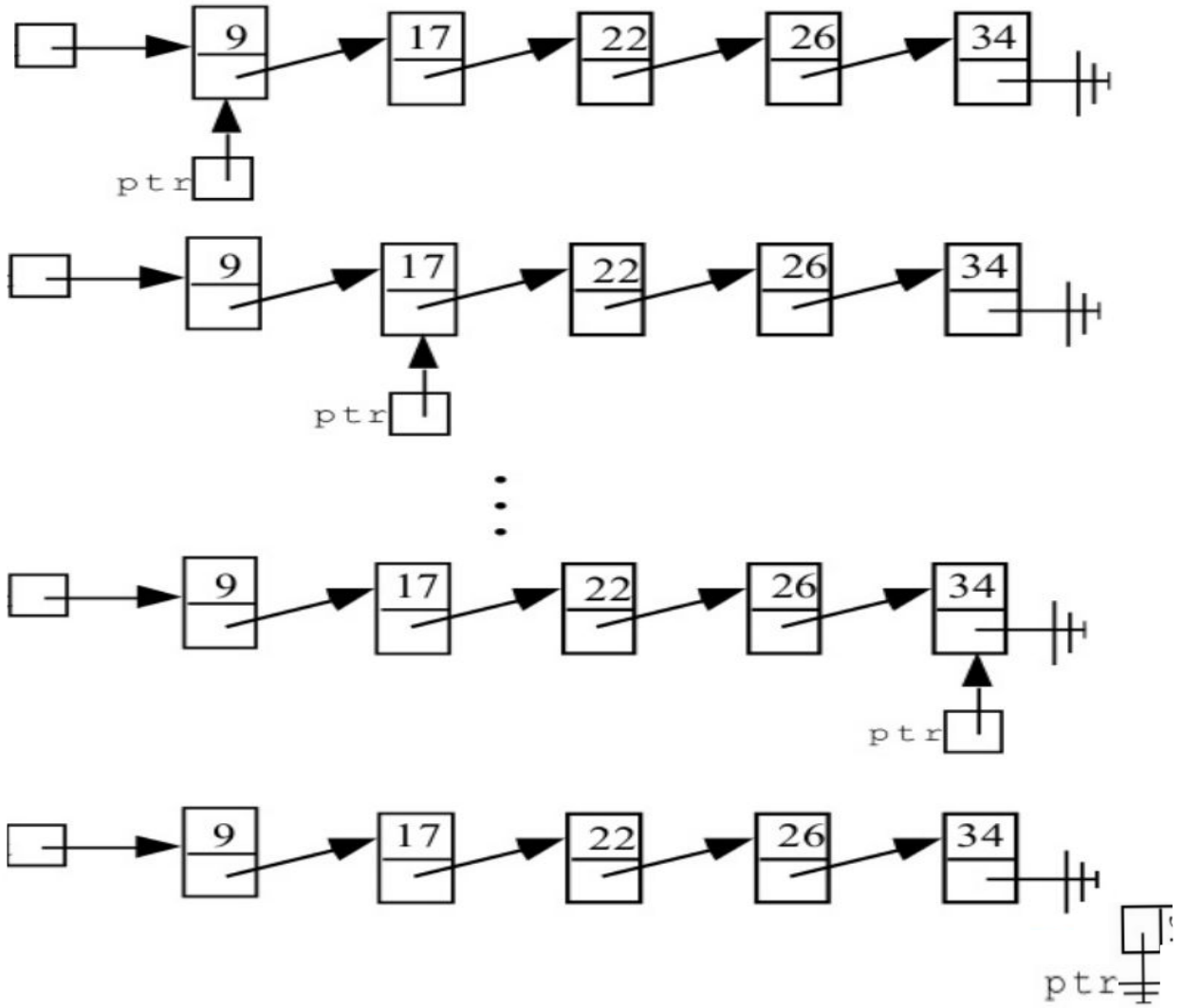
Recorrido de lista

- **Ejemplo:**
  - Considerar la siguiente lista enlazada de enteros:



  - Necesitamos recorrer la lista y visualizar cada uno de sus componentes

Recorrido de lista



Khuram Shahzad, Data structures and algorithms

Recorrido de lista

- Desde el puntero a la lista, y nodo por nodo, se recorre cada uno de los elementos para procesarlos de cierto modo hasta llegar al último nodo de la lista.

visualización, conteo, promedio,etc

```
recorrerLista(lista)
        ptr ← lista
        mientras ptr <> NULO hacer
                procesar(ptr→dato)
                ptr ← ptr→puntero
```

Recorrido de lista

- Desde el puntero a la lista, y nodo por nodo, se recorre cada uno de los elementos para procesarlos de cierto modo hasta llegar al último nodo de la lista.

visualización, conteo, promedio,etc

```
recorrerLista(lista)
        ptr ← lista
        mientras ptr <> NULO hacer
                procesar(ptr→dato)
                ptr ← ptr→puntero
```

O(1)

O(n)

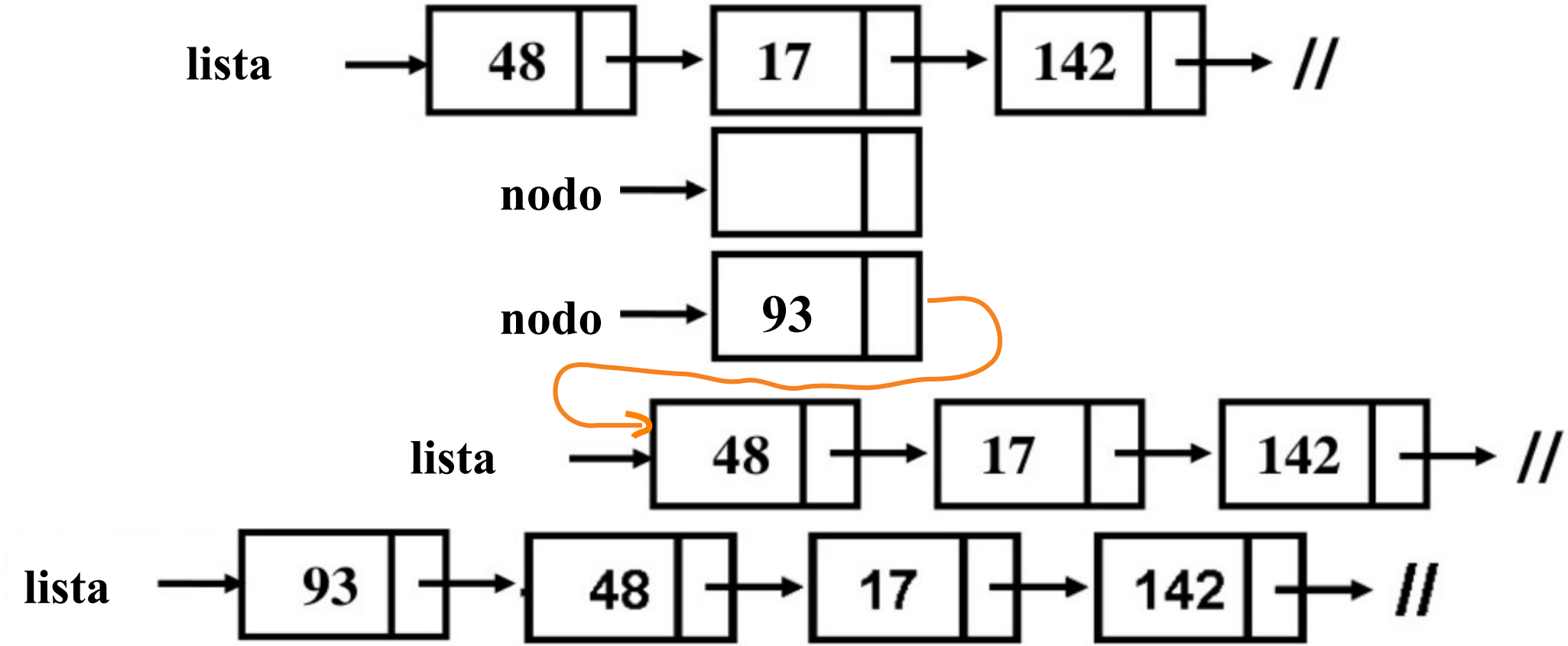
Orden de complejidad: O(n)

Inserción de nodo

- Un nodo se puede insertar:
  - Al inicio de la lista (como primer nodo)
  - Al final de la lista (como último nodo)
  - En una posición arbitraria en la lista
- Todos los casos implican:
  - Creación de un nodo
  - Asignación de valor de dato al nodo
  - Conexión de punteros

Inserción de nodo

- Inserción de nodo al inicio de la lista: Ejemplo



Inserción de nodo

- Inserción de nodo al inicio de la lista

```
insertarNodoInicio(lista, valor)
        nodo ← crearNodoVacio()
        nodo→dato ← valor
        nodo→puntero ← lista
        lista ← nodo
```

Inserción de nodo

- Inserción de nodo al inicio de la lista

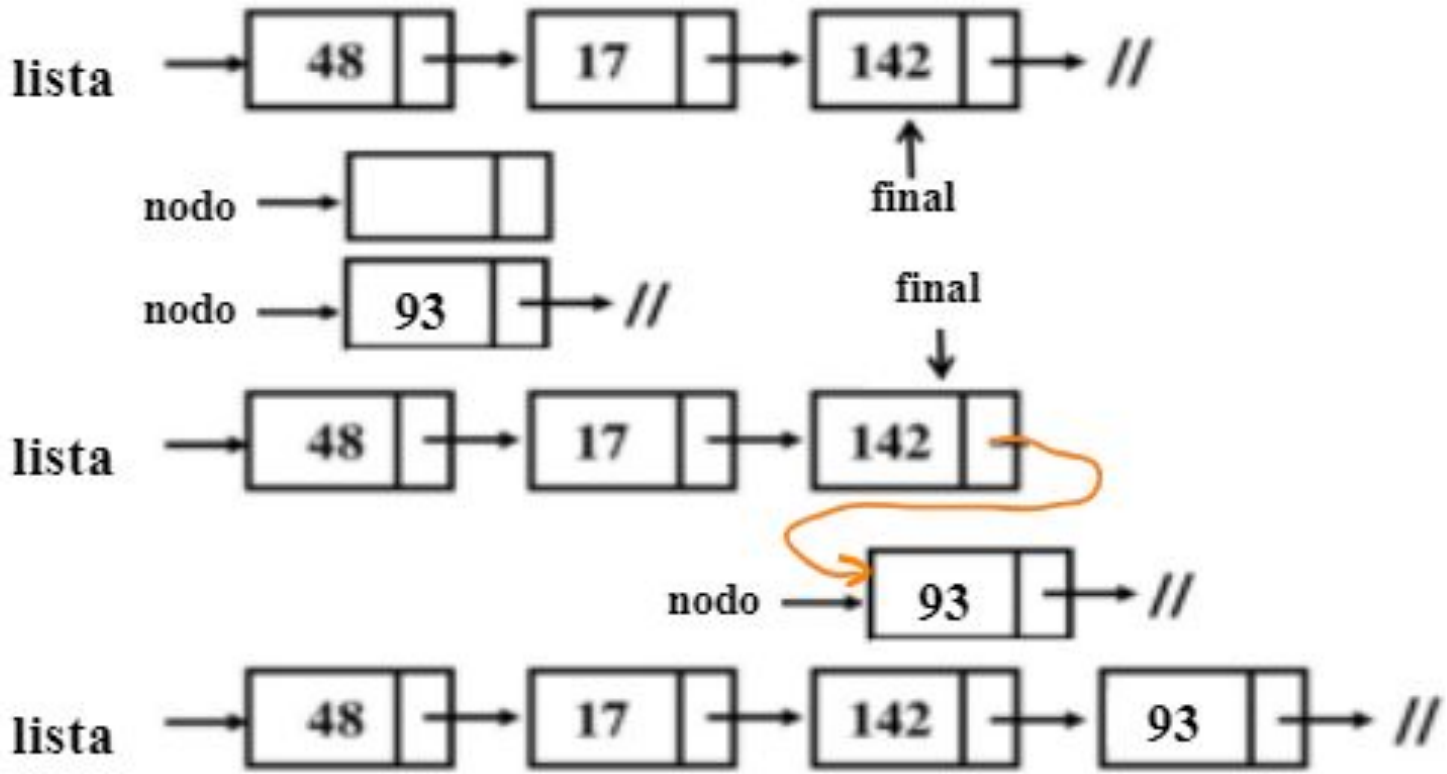
```
insertarNodoInicio(lista, valor)
        nodo ← crearNodoVacio()
        nodo→dato ← valor
        nodo→puntero ← lista
        lista ← nodo
```

O(1)

Orden de complejidad: O(1)

Inserción de nodo

- Inserción de nodo al final de la lista: Ejemplo



Inserción de nodo

- Inserción de nodo al final de la lista

```
insertarNodoFinal(lista, valor)
        final ← lista
        mientras final→puntero <> NULO hacer
                final← final→puntero
        nodo ← crearNodoVacio()
        nodo→dato ← valor
        nodo→puntero ← NULO
        final→puntero ← nodo
```

Inserción de nodo

- Inserción de nodo al final de la lista

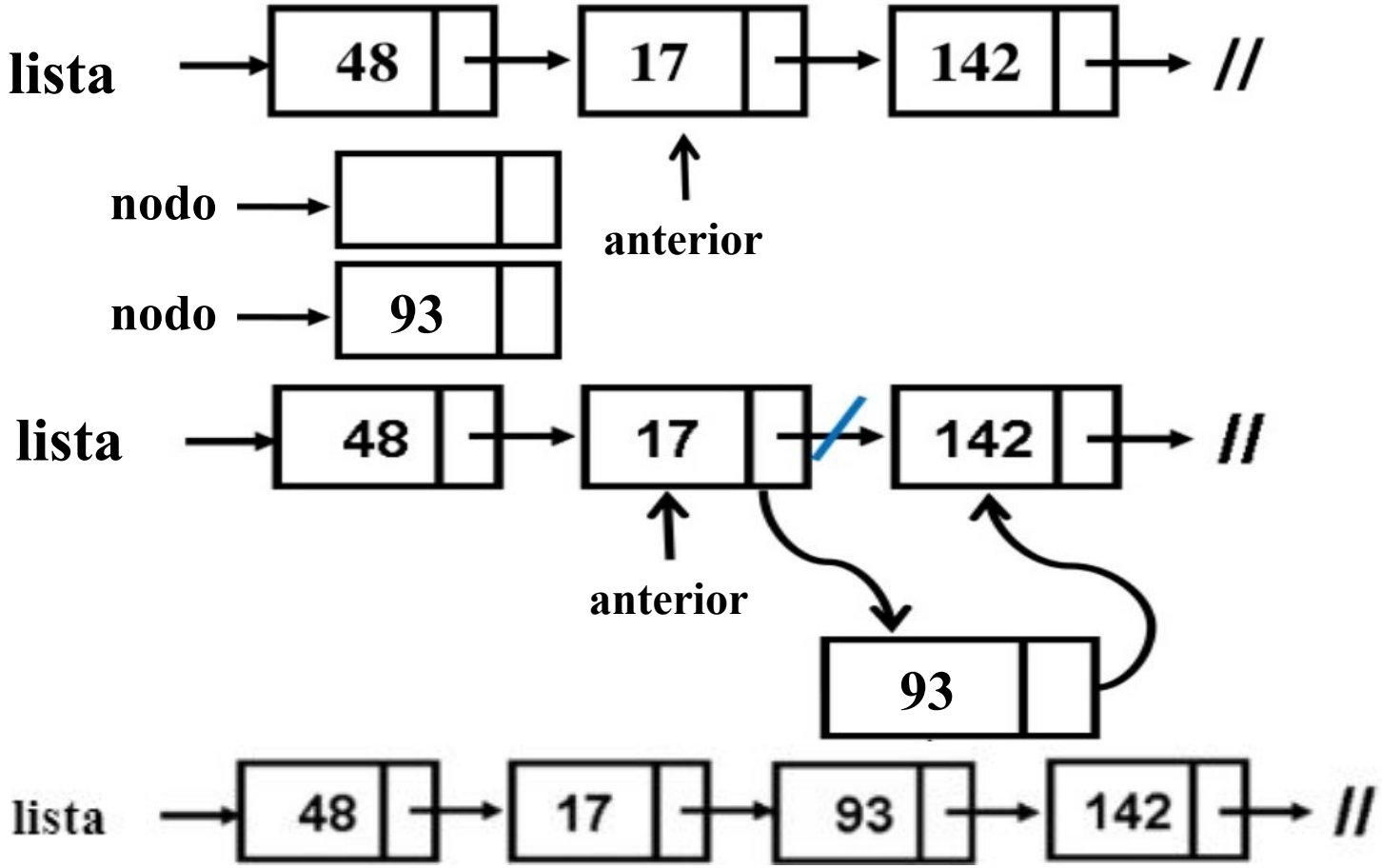
```
insertarNodoFinal(lista, valor)
        final ← lista
        mientras final→puntero <> NULO hacer
                final← final→puntero
        nodo ← crearNodoVacio()
        nodo→dato ← valor
        nodo→puntero ← NULO
        final→puntero ← nodo
```

O(1)
O(n)
O(1)

Orden de complejidad: O(n)

Inserción de nodo

- Inserción de nodo en posición arbitraria de la lista (después de un cierto valor): Ejemplo



Inserción de nodo

- Inserción de nodo en posición arbitraria de la lista (después de un cierto valor)

```
insertarNodoDespues(lista,valor,datoAnterior)
        anterior ← lista
        mientras anterior→dato <> datoAnterior hacer
                anterior← anterior→puntero
        nodo ← crearNodoVacio()
        nodo→dato ← valor
        nodo→puntero ← anterior→puntero
        anterior→puntero ← nodo
```

Inserción de nodo

- Inserción de nodo en posición arbitraria de la lista (después de un cierto valor)

```
insertarNodoDespues(lista,valor,datoAnterior)
        anterior ← lista
        mientras anterior→dato <> datoAnterior hacer
                anterior← anterior→puntero
        nodo ← crearNodoVacio()
        nodo→dato ← valor
        nodo→puntero ← anterior→puntero
        anterior→puntero ← nodo
```

O(1)
O(n)
O(1)

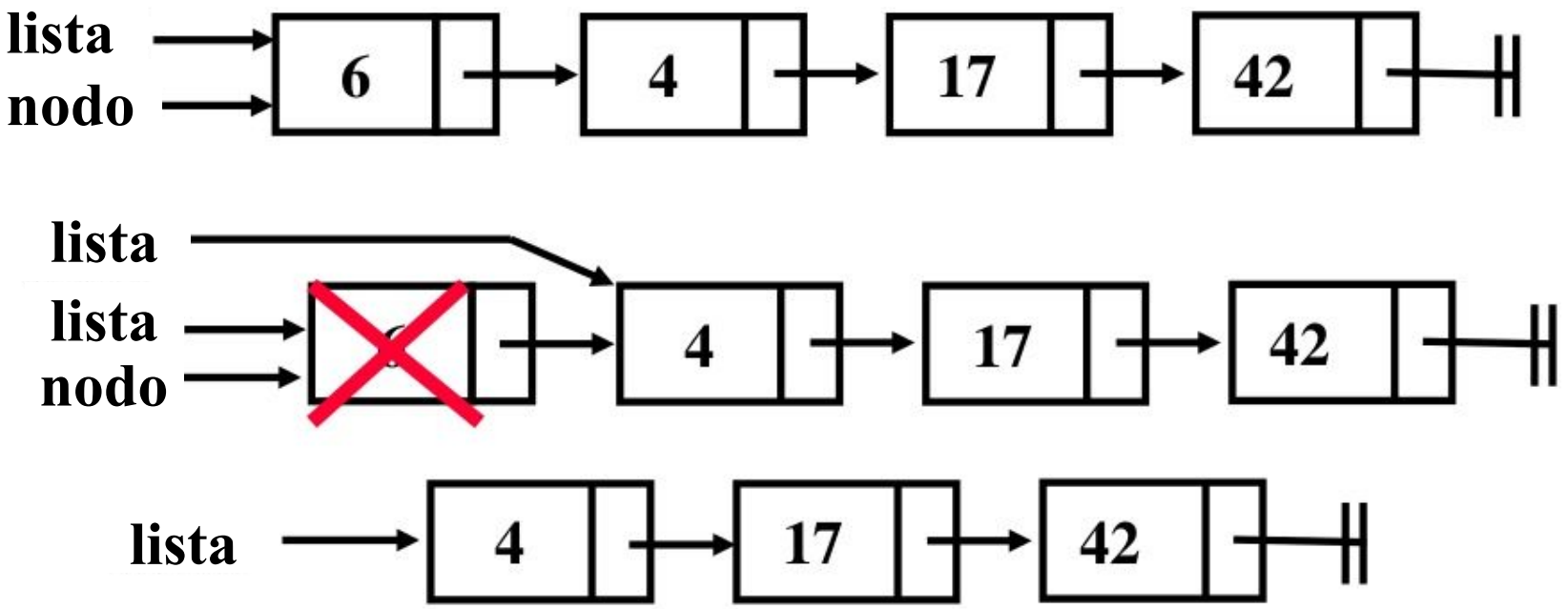
Orden de complejidad: O(n)

Eliminación de nodo

- Un nodo se puede eliminar:
  - Al inicio de la lista (era el primer nodo)
  - Al final de la lista (era el último nodo)
  - En una posición arbitraria en la lista
- Todos los casos implican:
  - Anulación de conexión de punteros
  - Reconexión de nodos
  - Eliminar nodo - Liberar

Eliminación de nodo

- Eliminación de nodo al inicio de la lista: Ejemplo



Eliminación de nodo

- Eliminación de nodo al inicio de la lista

```
eliminarNodoInicio(lista)
        nodo ← lista
        lista ← lista→puntero
        liberar(nodo)
```

Eliminación de nodo

- Eliminación de nodo al inicio de la lista

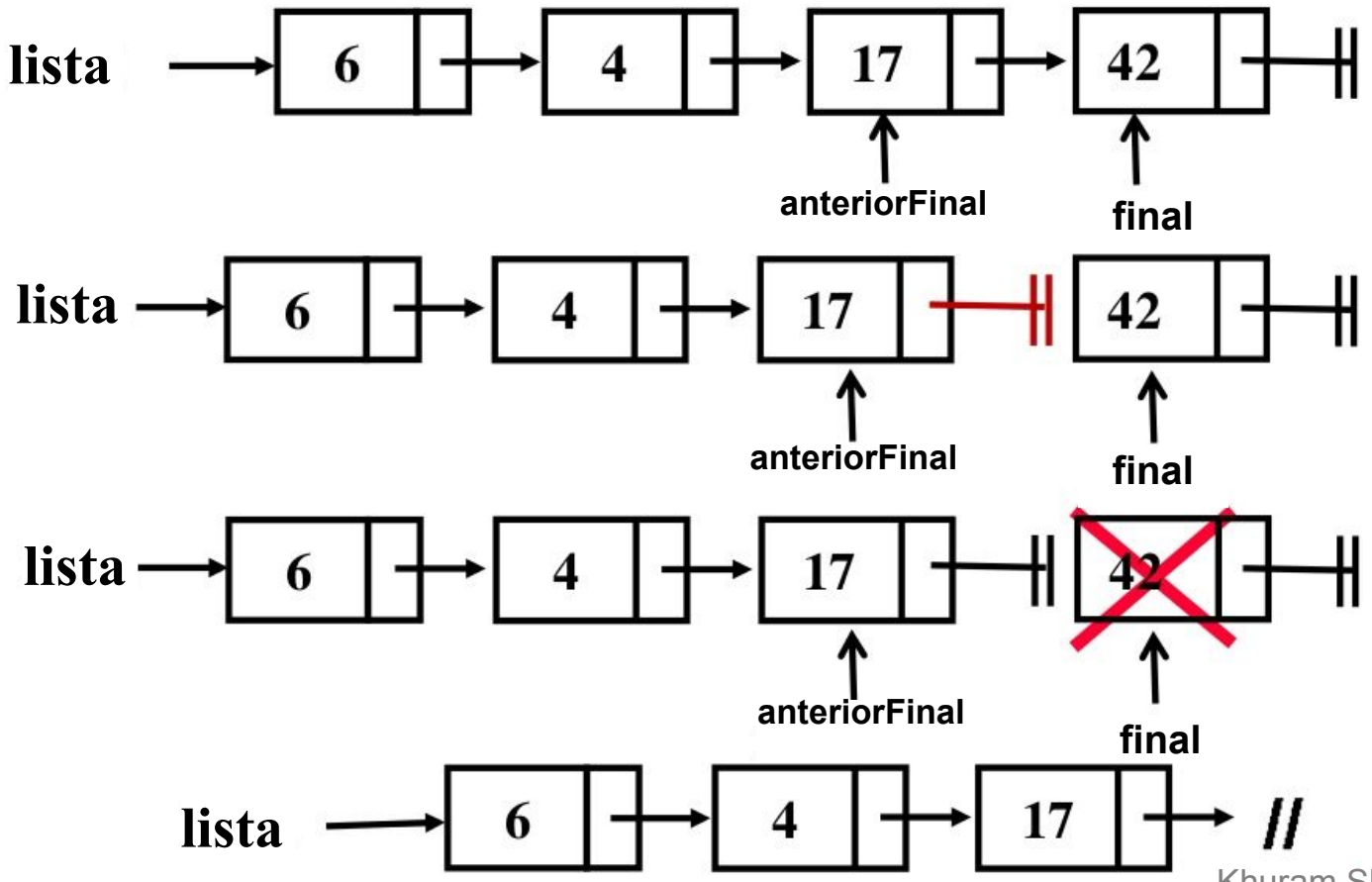
```
eliminarNodoInicio(lista)
        nodo ← lista
        lista ← lista→puntero
        liberar(nodo)
```

O(1)

Orden de complejidad: O(1)

Eliminación de nodo

- Eliminación de nodo al final de la lista: Ejemplo



Eliminación de nodo

- Eliminación de nodo al final de la lista

```
eliminarNodoFinal(lista)
        final ← lista
        anteriorFinal ← NULO
        mientras final→puntero <> NULO hacer
                anteriorFinal ← final
                final← final→puntero
        anteriorFinal→puntero ← NULO
        liberar(final)
```

Eliminación de nodo

- Eliminación de nodo al final de la lista

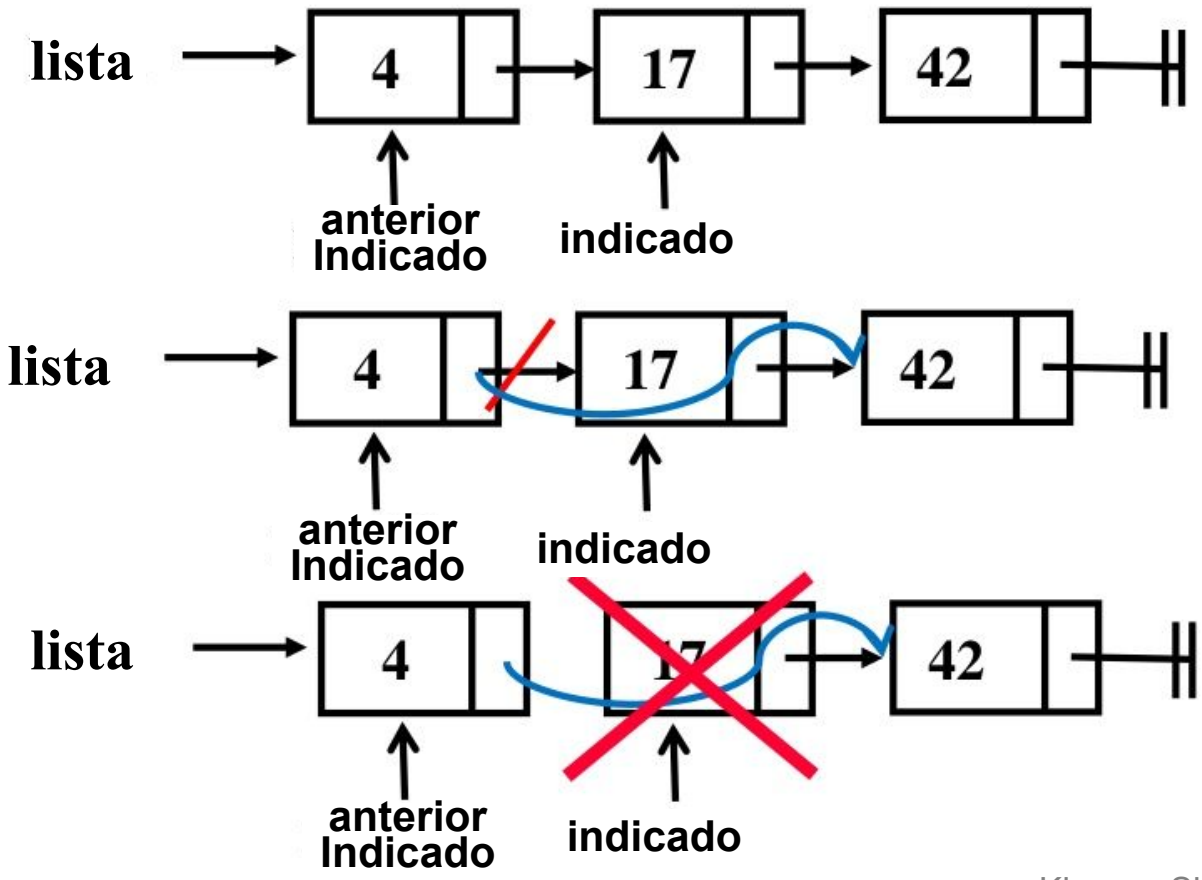
```
eliminarNodoFinal(lista)
        final ← lista
        anteriorFinal ← NULO
        mientras final→puntero <> NULO hacer
                anteriorFinal ← final
                final← final→puntero
        anteriorFinal→puntero ← NULO
        liberar(final)
```

O(1)
O(n)
O(1)

Orden de complejidad: O(n)

Eliminación de nodo

- Eliminación en posición arbitraria de la lista (nodo con un cierto valor): Ejemplo



Eliminación de nodo

- Eliminación de nodo en posición arbitraria de la lista (nodo con un cierto valor)

```
eliminarNodoDato(lista,dato)
        indicado ← lista
        anteriorIndicado ← NULO
        mientras indicado→dato <> dato hacer
                anteriorIndicado ← indicado
                indicado ← indicado→puntero
        anteriorIndicado→puntero ← indicado→puntero
        liberar(indicado)
```

Eliminación de nodo

- Eliminación de nodo en posición arbitraria de la lista (nodo con un cierto valor)

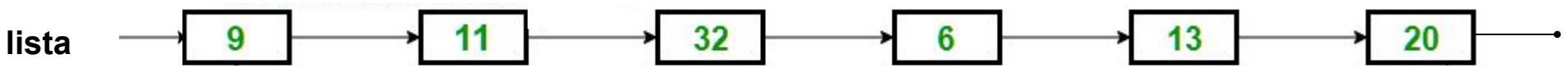
```
eliminarNodoDato(lista,dato)
        indicado ← lista
        anteriorIndicado ← NULO
        mientras indicado→dato <> dato hacer
                anteriorIndicado ← indicado
                indicado ← indicado→puntero
        anteriorIndicado→puntero ← indicado→puntero
        liberar(indicado)
```

O(1)
O(n)
O(1)

Orden de complejidad: O(n)

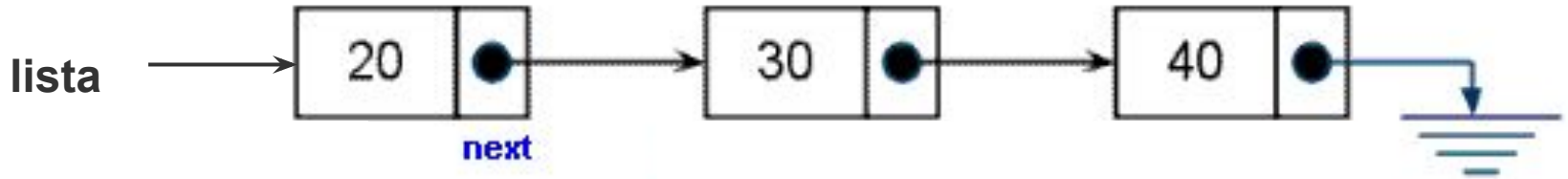
Ejercicio propuesto

- Ejercicio:
  - Escribir un algoritmo para buscar un cierto dato en una lista enlazada
  - Hacer una traza del algoritmo para buscar el dato 6 usando la lista:

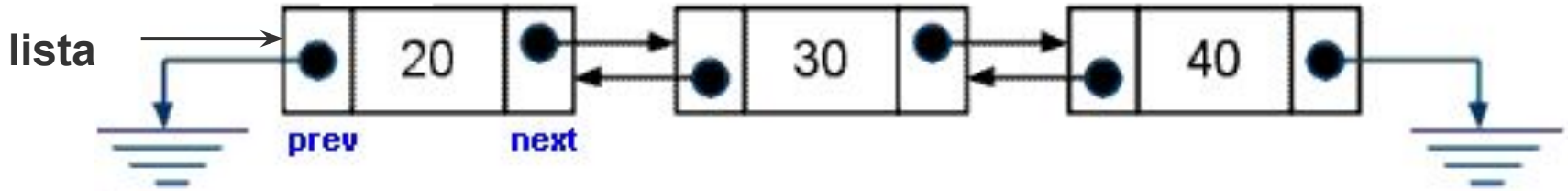


Tipos de listas enlazadas

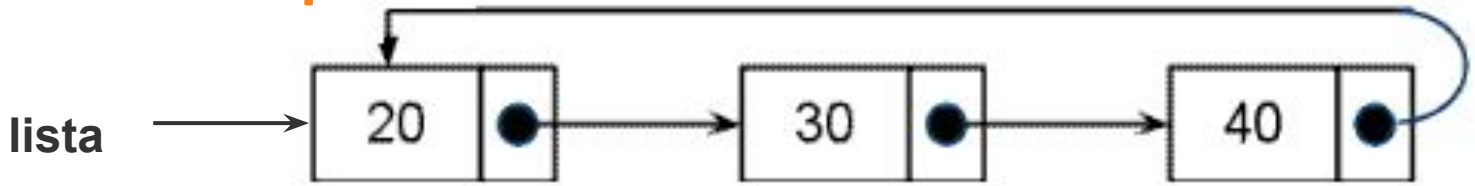
- **Lista enlazada simple**



- **Lista doblemente enlazada**

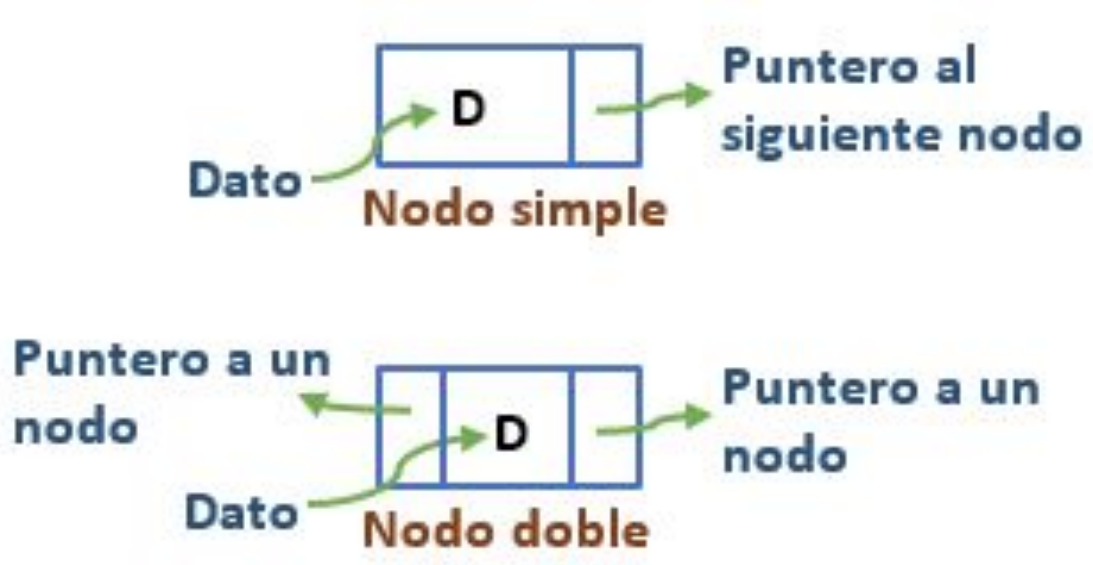


- **Lista enlazada simple circular**



Listas doblemente enlazadas

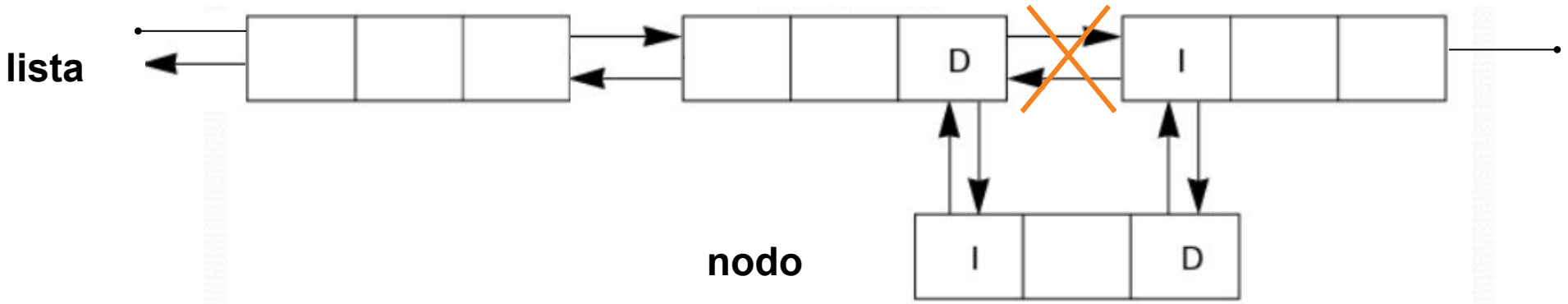
- Estructura de nodo doble:



https://www.glosarioit.com/Lista

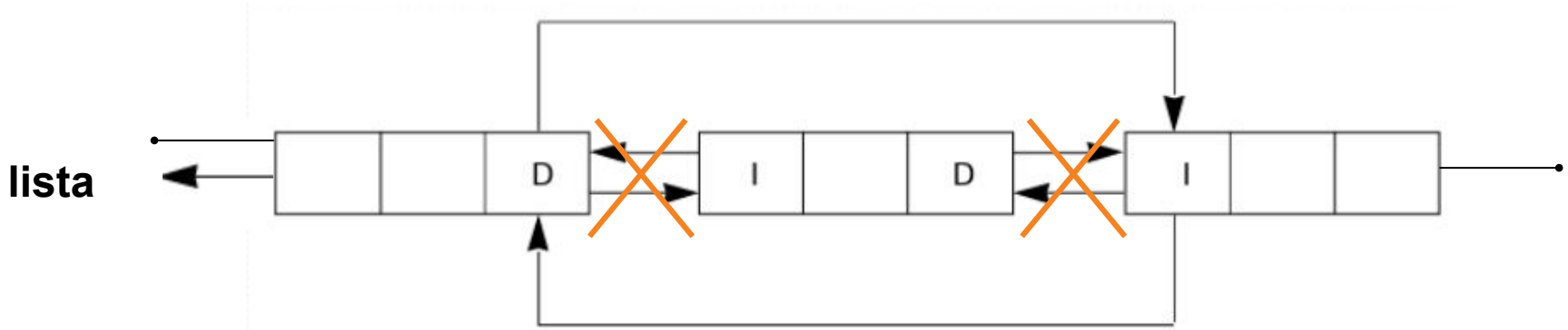
Listas doblemente enlazadas

- Inserción de nodo:



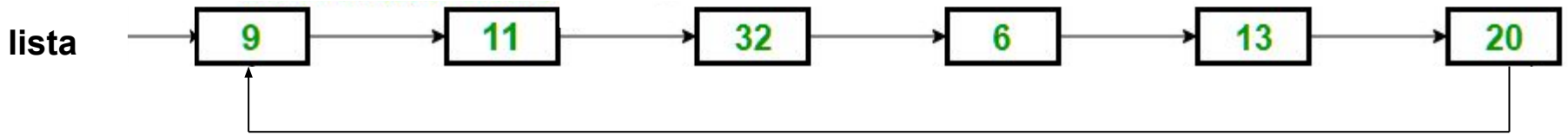
Listas doblemente enlazadas

- Eliminación de nodo:



Ejercicio propuesto

- Ejercicio:
  - Escribir un algoritmo para eliminar un cierto dato en una lista enlazada circular
  - Hacer una traza del algoritmo para eliminar el nodo con dato 6 usando la lista:



Próximas fechas...

- Resumen de la semana:
  - *TDA lista enlazada*
  - *Listas enlazadas especiales*

- **Incentivo 3**: miércoles 20/abril

- Próxima semana:
  - *TDA pila*
  - *TDA cola*

U2 - S6