## **¿Qué es una lista enlazada?**

Una **lista enlazada** (*linked list*) es una estructura de datos lineal compuesta por nodos. A diferencia de los arrays, las listas enlazadas no almacenan los elementos en ubicaciones contiguas de memoria. En su lugar, cada nodo contiene:

* **Un dato o valor**
* **Una referencia (o enlace) al siguiente nodo** en la lista

## **Tipos de listas enlazadas**

### **1. Lista Enlazada Simple (Singly Linked List)**

Cada nodo tiene dos componentes:

* El **valor del nodo**
* El **puntero al siguiente nodo**

plaintext

CopiarEditar

[Valor | Siguiente] → [Valor | Siguiente] → NULL

### **2. Lista Doblemente Enlazada (Doubly Linked List)**

Cada nodo contiene tres partes:

* Un **puntero al nodo anterior**
* El **valor del nodo**
* Un **puntero al nodo siguiente**

plaintext

CopiarEditar

NULL ← [Anterior | Valor | Siguiente] ↔ [Anterior | Valor | Siguiente] → NULL

### **3. Lista Enlazada Circular**

En una lista enlazada circular, el último nodo apunta al primero:

* **Simple circular**: el último nodo apunta al primero
* **Doble circular**: el primero y el último nodo se enlazan entre sí por ambos lados

plaintext

CopiarEditar

[Valor | Siguiente] → ... → [Valor | Siguiente] ↘

↑\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_↑

## **Ventajas**

* **Inserción y eliminación eficientes**: Las operaciones de inserción y eliminación pueden realizarse en tiempo constante (O(1)) si se tiene acceso al nodo anterior, ya que solo es necesario ajustar los punteros .
* **Tamaño dinámico**: La longitud de una lista puede incrementarse o reducirse según sea necesario, lo que la hace adecuada cuando el número de elementos es impredecible .
* **Uso eficiente de memoria**: No hay necesidad de grandes cantidades de memoria contigua, y el uso de memoria se adapta dinámicamente .

## **Desventajas**

* **Acceso secuencial**: Sin índices, se debe recorrer nodo por nodo para encontrar un elemento específico, lo que puede resultar en tiempos de acceso más lentos en comparación con los arrays .
* **Mayor uso de memoria**: Cada nodo requiere espacio para almacenar tanto el valor como el puntero, lo que puede ser ineficiente para listas de pequeños datos como caracteres o valores booleanos .

## **Operaciones básicas**

### **1. Recorrer una lista**

Se parte desde la cabeza (primer nodo) y se sigue el enlace al siguiente nodo hasta llegar a NULL.

### **2. Insertar un nodo**

* **Al inicio**: Se crea un nuevo nodo y se enlaza como nuevo primer nodo.
* **En medio o final**: Se recorre hasta la posición deseada y se modifica el puntero del nodo anterior.

## **¿Qué es una lista enlazada?**

### **Una lista enlazada (*linked list*) es una estructura de datos lineal compuesta por nodos. A diferencia de los arrays, las listas enlazadas no almacenan los elementos en ubicaciones contiguas de memoria. En su lugar, cada nodo contiene:**

### **Un dato o valor**

### **Una referencia (o enlace) al siguiente nodo en la lista**

## **Tipos de listas enlazadas**

### **1. Lista Enlazada Simple (Singly Linked List)**

### **Cada nodo tiene dos componentes:**

### **El valor del nodo**

### **El puntero al siguiente nodo**

### **plaintext**

### **CopiarEditar**

### **[Valor | Siguiente] → [Valor | Siguiente] → NULL**

### 

### **2. Lista Doblemente Enlazada (Doubly Linked List)**

### **Cada nodo contiene tres partes:**

### **Un puntero al nodo anterior**

### **El valor del nodo**

### **Un puntero al nodo siguiente**

### **plaintext**

### **CopiarEditar**

### **NULL ← [Anterior | Valor | Siguiente] ↔ [Anterior | Valor | Siguiente] → NULL**

### 

### **3. Lista Enlazada Circular**

### **En una lista enlazada circular, el último nodo apunta al primero:**

### **Simple circular: el último nodo apunta al primero**

### **Doble circular: el primero y el último nodo se enlazan entre sí por ambos lados**

### **plaintext**

### **CopiarEditar**

### **[Valor | Siguiente] → ... → [Valor | Siguiente] ↘**

### **↑\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_↑**

### 

## **Ventajas**

### **Inserción y eliminación eficientes: Las operaciones de inserción y eliminación pueden realizarse en tiempo constante (O(1)) si se tiene acceso al nodo anterior, ya que solo es necesario ajustar los punteros .**

### **Tamaño dinámico: La longitud de una lista puede incrementarse o reducirse según sea necesario, lo que la hace adecuada cuando el número de elementos es impredecible .**

### **Uso eficiente de memoria: No hay necesidad de grandes cantidades de memoria contigua, y el uso de memoria se adapta dinámicamente .**

## **Desventajas**

### **Acceso secuencial: Sin índices, se debe recorrer nodo por nodo para encontrar un elemento específico, lo que puede resultar en tiempos de acceso más lentos en comparación con los arrays .**

### **Mayor uso de memoria: Cada nodo requiere espacio para almacenar tanto el valor como el puntero, lo que puede ser ineficiente para listas de pequeños datos como caracteres o valores booleanos .**

## **Operaciones básicas**

### **1. Recorrer una lista**

### **Se parte desde la cabeza (primer nodo) y se sigue el enlace al siguiente nodo hasta llegar a NULL.**

### **2. Insertar un nodo**

### **Al inicio: Se crea un nuevo nodo y se enlaza como nuevo primer nodo.**

### **En medio o final: Se recorre hasta la posición deseada y se modifica el puntero del nodo anterior.**

### **3. Eliminar un nodo**

### **Se ajustan los punteros del nodo anterior y posterior para excluir el nodo actual.**

## **Ejemplo en Python (Lista Enlazada Simple)**

### **python**

### **CopiarEditar**

### **class Nodo:**

### **def \_\_init\_\_(self, dato):**

### **self.dato = dato**

### **self.siguiente = None**

### 

### **class ListaEnlazada:**

### **def \_\_init\_\_(self):**

### **self.cabeza = None**

### 

### **def insertar\_al\_final(self, dato):**

### **nuevo = Nodo(dato)**

### **if not self.cabeza:**

### **self.cabeza = nuevo**

### **return**

### **actual = self.cabeza**

### **while actual.siguiente:**

### **actual = actual.siguiente**

### **actual.siguiente = nuevo**

### 

### **def mostrar(self):**

### **actual = self.cabeza**

### **while actual:**

### **print(actual.dato, end=" -> ")**

### **actual = actual.siguiente**

### **print("NULL")**

### **3. Eliminar un nodo**

Se ajustan los punteros del nodo anterior y posterior para excluir el nodo actual.

## **Ejemplo en Python (Lista Enlazada Simple)**

python

CopiarEditar

class Nodo:

def \_\_init\_\_(self, dato):

self.dato = dato

self.siguiente = None

class ListaEnlazada:

def \_\_init\_\_(self):

self.cabeza = None

def insertar\_al\_final(self, dato):

nuevo = Nodo(dato)

if not self.cabeza:

self.cabeza = nuevo

return

actual = self.cabeza

while actual.siguiente:

actual = actual.siguiente

actual.siguiente = nuevo

def mostrar(self):

actual = self.cabeza

while actual:

print(actual.dato, end=" -> ")

actual = actual.siguiente

print("NULL")