Algoritmos TAD

Un ejemplo a tener en cuenta

TAD - Pila

Agregar (Push)

1. Recibir el nuevo dato.
2. Crear un nuevo nodo con el dato.
3. Apuntar el puntero "siguiente" del nuevo nodo hacia el actual **Tope**.
4. Actualizar el **Tope** para que sea el nuevo nodo.

Buscar

Nota: En una pila pura, no se busca sin "desapilar", pero lógicamente el algoritmo es:

1. Empezar desde el **Tope**.
2. Comparar el dato del nodo actual con el buscado.
3. Si no es igual, moverse al nodo "siguiente".
4. Repetir hasta encontrarlo o llegar al final (pila vacía).

TAD - Pila

Eliminar (Pop)

1. Verificar si la pila está vacía (Error: *Underflow*).
2. Guardar el nodo de la cabeza en una variable temporal.
3. Hacer que la cabeza ahora sea el nodo "siguiente" del tope actual.
4. Liberar la memoria o retornar el dato del nodo temporal.

Recorrer

1. Situarse en el **cabeza.**
2. Mientras el nodo actual no sea nulo:
    - Procesar/Imprimir el dato.
    - Moverse al nodo "siguiente".

TAD - Pila

Operación	Pila (Stack)
Acceso	Solo al Tope (LIFO).
Eficiencia	Muy rápida (O(1) para agregar/quitar).
Uso común	Deshacer (Ctrl+Z), recursividad.

TAD - lista (enlazada simple)

A diferencia de la pila, aquí podemos insertar o borrar en cualquier lugar (inicio, final o posición *n*). Usaremos el caso de **insertar al final** por ser el más común.

Agregar (Insertar al final)

1. Crear un nuevo nodo con el dato.
2. Si la lista está vacía, el nuevo nodo es la **Cabeza**.
3. Si no, recorrer la lista desde la **Cabeza** hasta llegar al último nodo (cuyo "siguiente" sea nulo).
4. Cambiar el puntero "siguiente" del último nodo para que apunte al nuevo nodo.

Buscar

1. Empezar en la **Cabeza**.
2. Iniciar un contador en 0 (opcional, para devolver el índice).
3. Mientras el nodo actual no sea nulo:
    - Si el dato coincide, retornar la posición o el nodo.
    - Si no, avanzar al "siguiente" nodo e incrementar el contador.
4. Si se llega al final, informar que no se encontró.

TAD - lista (enlazada simple)

Eliminar (Por valor)

1. Empezar en la **Cabeza**.
2. Si la **Cabeza** tiene el dato: hacer que la Cabeza apunte al segundo nodo y eliminar el primero.
3. Si no, recorrer buscando el nodo, manteniendo una referencia al **nodo anterior**.
4. Al encontrarlo: hacer que el "siguiente" del **anterior** se salte el actual y apunte al "siguiente" del **actual**.
5. Liberar la memoria del nodo actual.

Recorrer

1. Empezar en la **Cabeza**.
2. Mientras el nodo actual sea distinto de nulo:
    - Realizar la acción deseada (imprimir, sumar, etc.).
    - Pasar al nodo "siguiente".

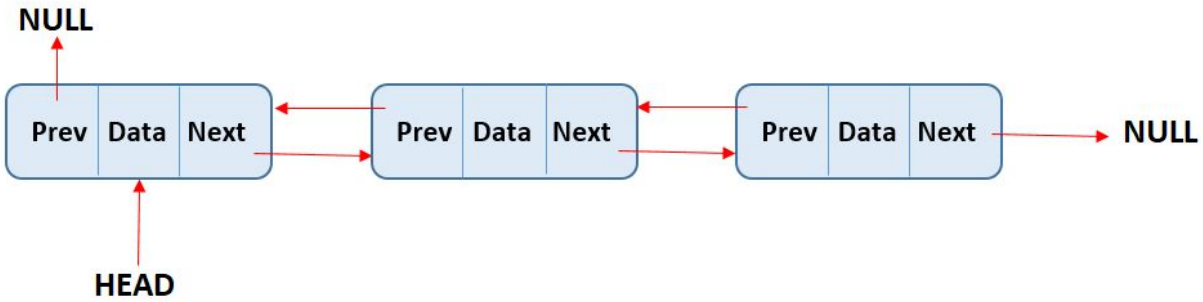
TAD - lista (enlazada simple)

Operación	Lista Enlazada
Acceso	A cualquier nodo.
Eficiencia	Variable (agregar al final suele ser O(n)).
Uso común	Bases de datos, listas de reproducción.

TAD - Lista doblemente enlazada

En el libro de Cormen una Lista Doblemente Enlazada consiste en una estructura de datos lineal en la que cada elemento (o nodo) es un objeto con un atributo clave (`key`) y dos punteros: `next` (siguiente) y `prev` (anterior).

Según el enfoque de Cormen, se suele utilizar un **centinela** (un nodo especial llamado `nil`) para simplificar las condiciones de borde, aunque a continuación se exponen los procedimientos estándar para una lista con puntero `head` (cabeza).



Tomada de https://www.alphacodingskills.com/ds/doubly-linked-list.php

TAD - Lista doblemente enlazada

Agregar (Insertar al inicio)

Este procedimiento coloca un nuevo nodo `x` al principio de la lista.

1. `x.next = L.head` (El siguiente del nuevo nodo apunta a lo que antes era el primer elemento).
2. **Si** `L.head != NULL`:
    - `L.head.prev = x` (El puntero anterior de la antigua cabeza ahora apunta al nuevo nodo).
3. `L.head = x` (La cabeza de la lista ahora es oficialmente el nuevo nodo).
4. `x.prev = NULL` (Como es el primero, no tiene a nadie antes).

Buscar

Busca la primera aparición de una clave `k`.

1. `x = L.head`
2. **Mientras** `x != NULL` y `x.key != k`:
    - `x = x.next`
3. **Retornar** `x` (Si no se encontró, devolverá `NULL`).

TAD - Lista doblemente enlazada

Eliminar

Para eliminar un nodo específico `x` ya localizado:

1. **Si** `x.prev != NULL`:
   - `x.prev.next = x.next` (El nodo anterior se conecta con el siguiente).
2. **Si no** (significa que `x` era la cabeza):
   - `L.head = x.next`
3. **Si** `x.next != NULL`:
   - `x.next.prev = x.prev` (El nodo siguiente se conecta con el anterior).

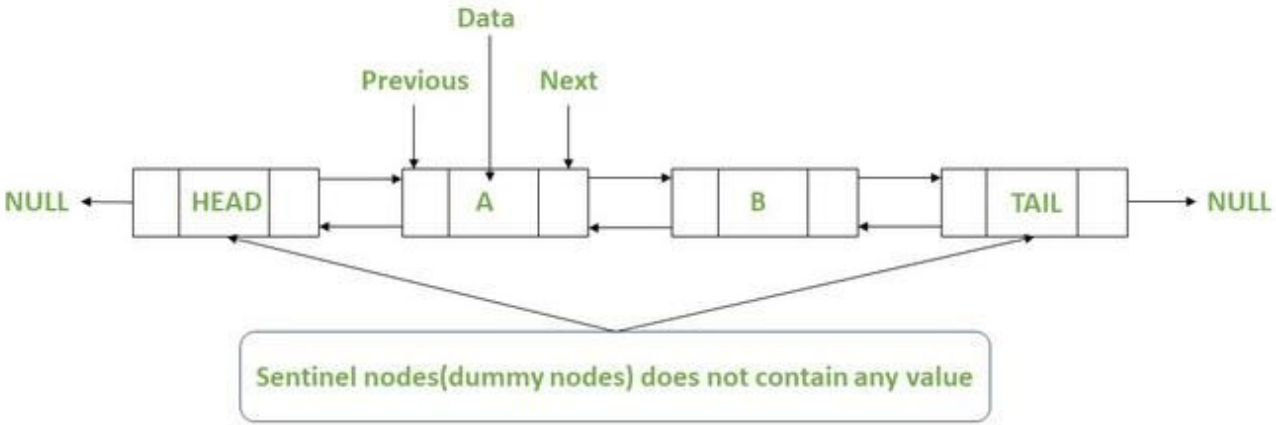
Recorrer

1. `actual = L.head`
2. **Mientras** `actual != NULL`:
   - Leer/Procesar el dato de `actual`.
   - `actual = actual.next`

TAD - Lista doblemente enlazada con centinela

What is a sentinel node?

*Sentinel nodes are specially designed nodes that **do not hold or refer to any data** of the doubly linked list (that data structure).*



Tomado de: https://www.geeksforgeeks.org/dsa/doubly-linked-list-using-sentinel-nodes/

TAD - Lista doblemente enlazada con centinela

Agregar (Insertar al inicio)

La ventaja aquí es que no necesitamos evaluar si la lista está vacía.

1. `x.next = L.nil.next`
2. `L.nil.next.prev = x`
3. `L.nil.next = x`
4. `x.prev = L.nil`

Buscar

1. `x = L.nil.next` (Empezamos en el primer elemento real).
2. **Mientras** `x != L.nil` y `x.key != k`:
    - `x = x.next`
3. **Retornar** `x` (Si el resultado es `L.nil`, significa que no se encontró).

TAD - Lista doblemente enlazada con centinela

Eliminar

Es el algoritmo más limpio de todos porque el centinela garantiza que `x.prev` y `x.next` siempre existan.

1. `x.prev.next = x.next`
2. `x.next.prev = x.prev`

Recorrer

1. `actual = L.nil.next`
2. **Mientras** `actual != L.nil`:
    - Leer/Procesar el dato de `actual`.
    - `actual = actual.next`

TAD - Lista doblemente enlazada con centinela

Característica	Sin Centinela (NULL)	Con Centinela (L.nil)
Puntero de Inicio	L.head	L.nil.next
Puntero de Fin	Un nodo que apunta a NULL	Un nodo que apunta a L.nil
Lista Vacía	L.head == NULL	L.nil.next == L.nil
Ventaja	Menos uso de memoria.	Código más simple, sin condicionales if.