



# FACULTAD DE INGENIERIA

Universidad de Buenos Aires

## TO DO LIST

☐☐☐☐☐

## TDA Lista

...y pila y cola

Nombre

Lautaro De Nobili

Padron

107394

Email

Idenobili@gmail.com

# Indice

- ❑ Teoria, explicacion de cada TDA
  - ❑ Lista? Que es eso?
  - ❑ Pila, claro, lo que necesita un control remoto
  - ❑ Cola, una fabrica de chocolate, Charly
- ❑ Lista
  - ❑ lista\_insertar
  - ❑ lista\_insertar en posicion
  - ❑ lista\_quitar
  - ❑ lista\_quitar en posicion
- ❑ Pila y cola

# Teoria

En esta ocasion, nos toca implementar un TDA de Lista, el cual luego reutilizamos para hacer los TDA de Pila y Cola.

Este TDA consiste en poder crear, utilizar y destruir una lista tal como lo haríamos con una lista de supermercado, pero en lugar de una birome, con funciones útiles...

Estas pueden ser:

- Simplemente enlazadas
- Doblemente enlazadas
- Circulares

Estos tipos se refieren al metodo de implementacion que utilizan, siendo este metodo el usar TDA de nodos

# Que es una lista?

Es un TDA con nombre autodescriptivo, por hacer justamente lo que esperarías de algo con ese nombre.

Basicamente, da a disposicion una lista en la que se pueden agregar y eliminar libremente elementos, siempre y cuando tengamos memoria.

Podemos insertar elementos en cualquier parte de la lista, al igual que eliminar de cualquier parte de ella algun elemento que querramos



# Que es una pila?

Una pila es una lista pero que la adaptamos para que cumpla con ciertas características de una pila, siendo esta algo comparable a una pila de platos.

Que significa esto? Agarramos una lista y le aplicamos el método de UEPS (ultimo entra, primero sale), tal cual haríamos si tuviésemos que apilar un montón de platos. Cada plato que ponemos, lo mandamos al tope de la pila, y cada plato que saquemos, lo sacamos del tope, no de algún otro lado (a menos que quieras romper todos los platos...)



# Que es una cola?

Con la cola vamos a hacer lo mismo que con pila, vamos a usar la lista y adaptarla segun un metodo especifico, solo que esta vez, usamos PEPS (primero entra, primero sale).

Esto es comparable a querer hacer una fila para pagar en el supermercado. La primer persona en llegar a la fila va a ser la primer persona en pagar, y cada persona nueva a la fila, ira al final de la fila. Hay que mantener el orden en una sociedad



# Implementacion de la lista

Ahora que quedo planteado que es cada cosa, toca hacerlo.

Para la lista, y consecuentemente para la pila y cola, vamos a usar nodos simplemente enlazados, pero que significa esto?

Básicamente, que nuestra lista son un montón de nodos, que cada uno apunta al elemento a agregar a la lista y a su nodo siguiente, por lo que nuestra lista no es como seria un vector ordenado, sino que el orden de nuestra lista se basa en que nodo apunte a cual antes, arrancando por nuestro nodo inicio, que esta en la estructura de lista.

En la lista disponemos de un puntero a nuestro primer nodo, que contendra la direccion del primer elemento, un puntero a nuestro ultimo nodo, y un conteo de la cantidad de elementos/nodos en la lista

# lista\_insertar()

Dato: comenzamos con una lista vacia. Recordar que cada nuevo elemento a insertar va a ser en forma de nodos, asi que tenemos que reservar la memoria para ese nodo y luego asignarle el elemento a insertar.

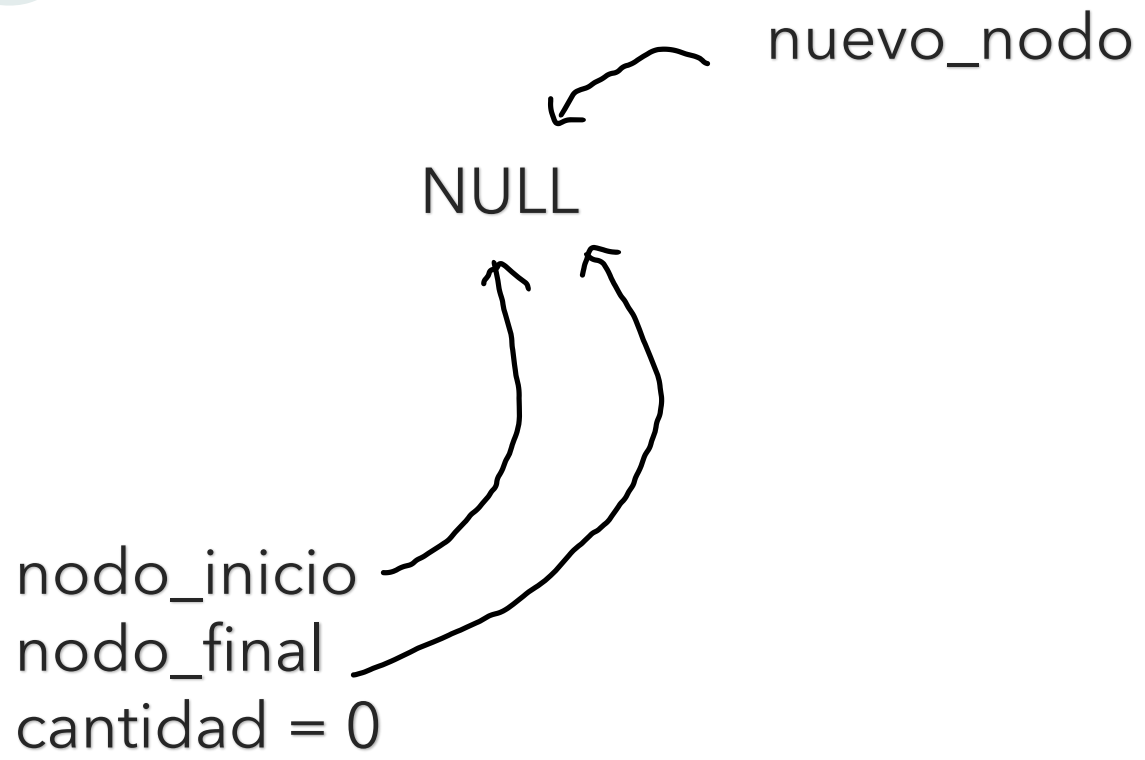
NULL

nodo\_inicio  
nodo\_final  
cantidad = 0





# lista\_insertar()

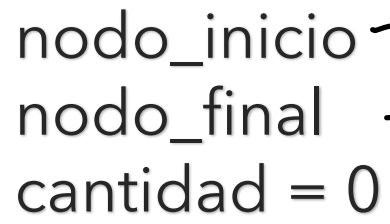


# lista\_insertar()

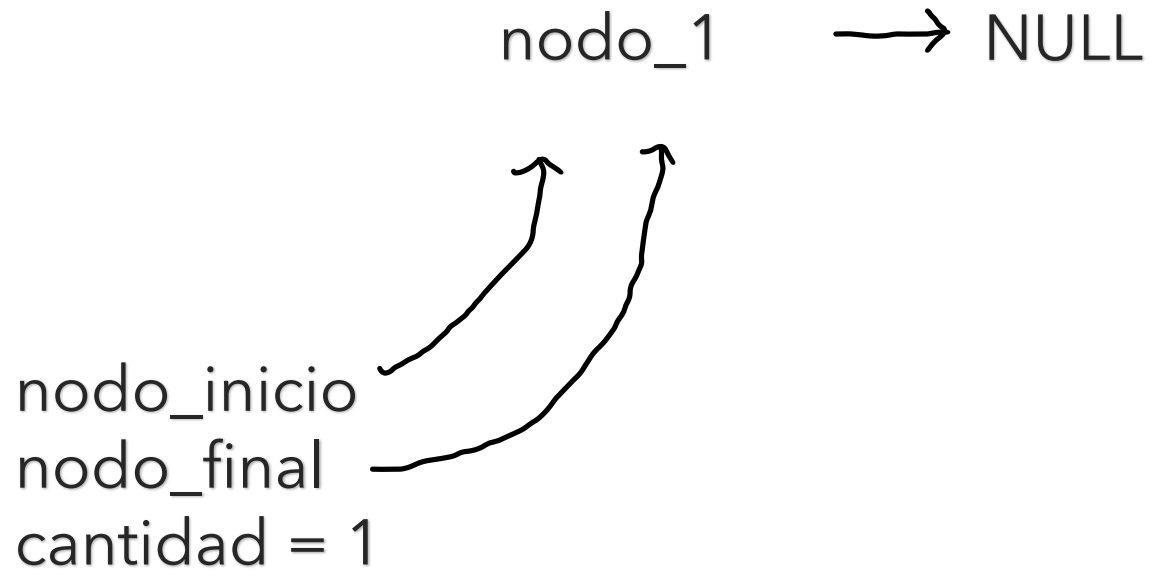
nuevo\_nodo → NULL



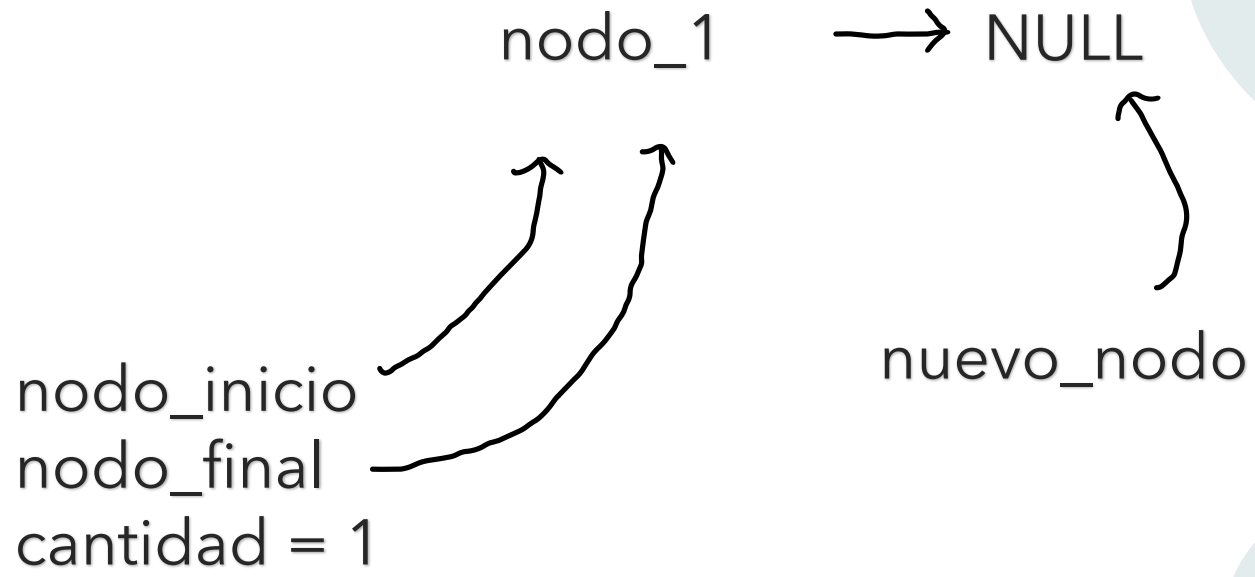
nodo\_inicio  
nodo\_final  
cantidad = 0



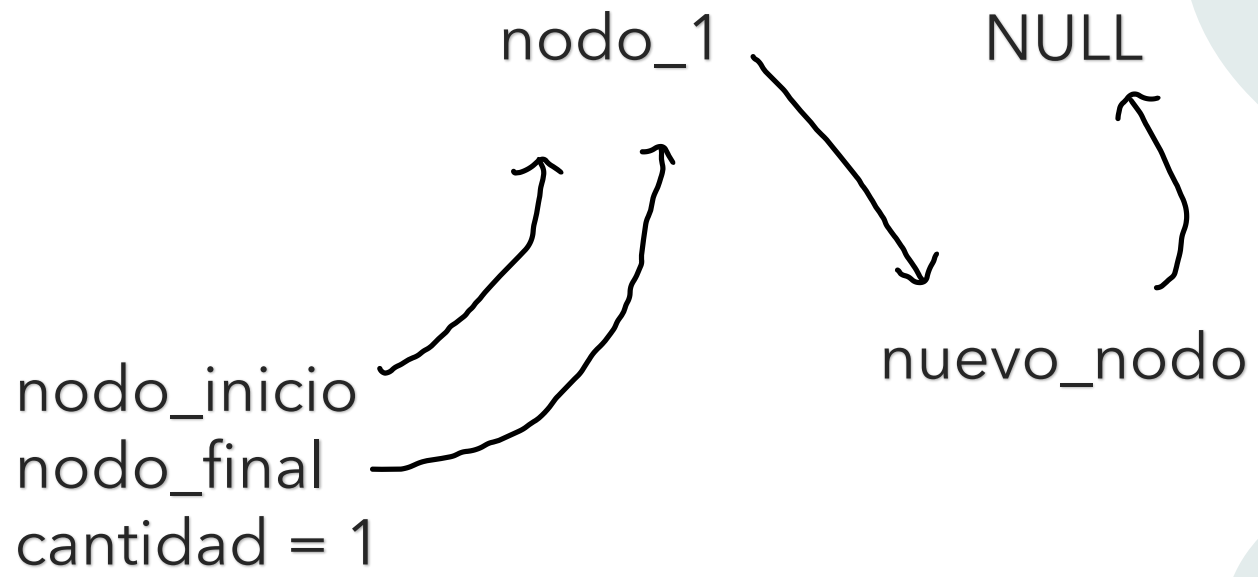
# lista\_insertar()



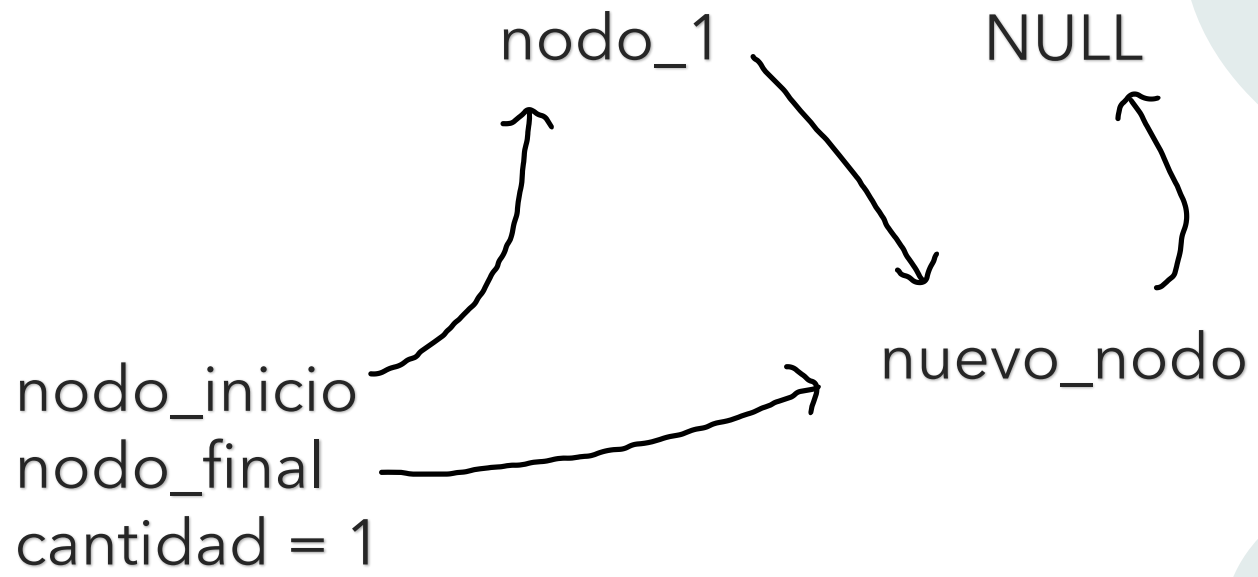
# lista\_insertar()



# lista\_insertar()



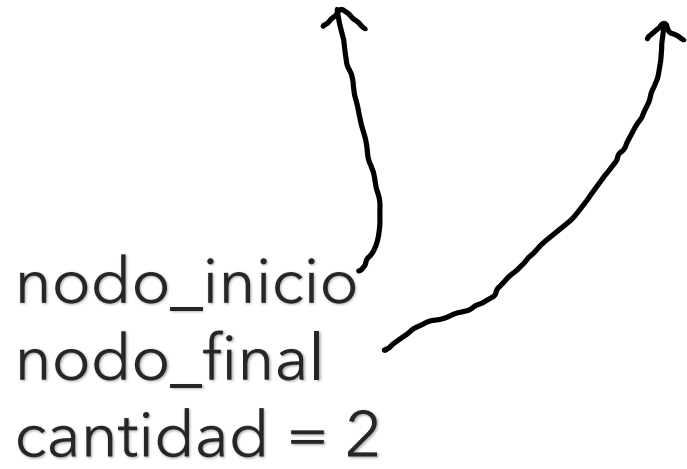
# lista\_insertar()



# lista\_insertar()

nodo\_1 → nodo\_2 → NULL

nodo\_inicio  
nodo\_final  
cantidad = 2



```
graph LR; nodo_inicio --> nodo_1; nodo_final --> nodo_2; nodo_1 --> nodo_2; nodo_2 --> NULL;
```

# lista\_insertar\_en\_posicion()

Dato: queremos insertar en posicion 2 (donde esta el nodo 3 actualmente)

nodo\_aux

nuevo\_nodo

nodo\_1 → nodo\_2 → nodo\_3 → NULL

nodo\_inicio

nodo\_final

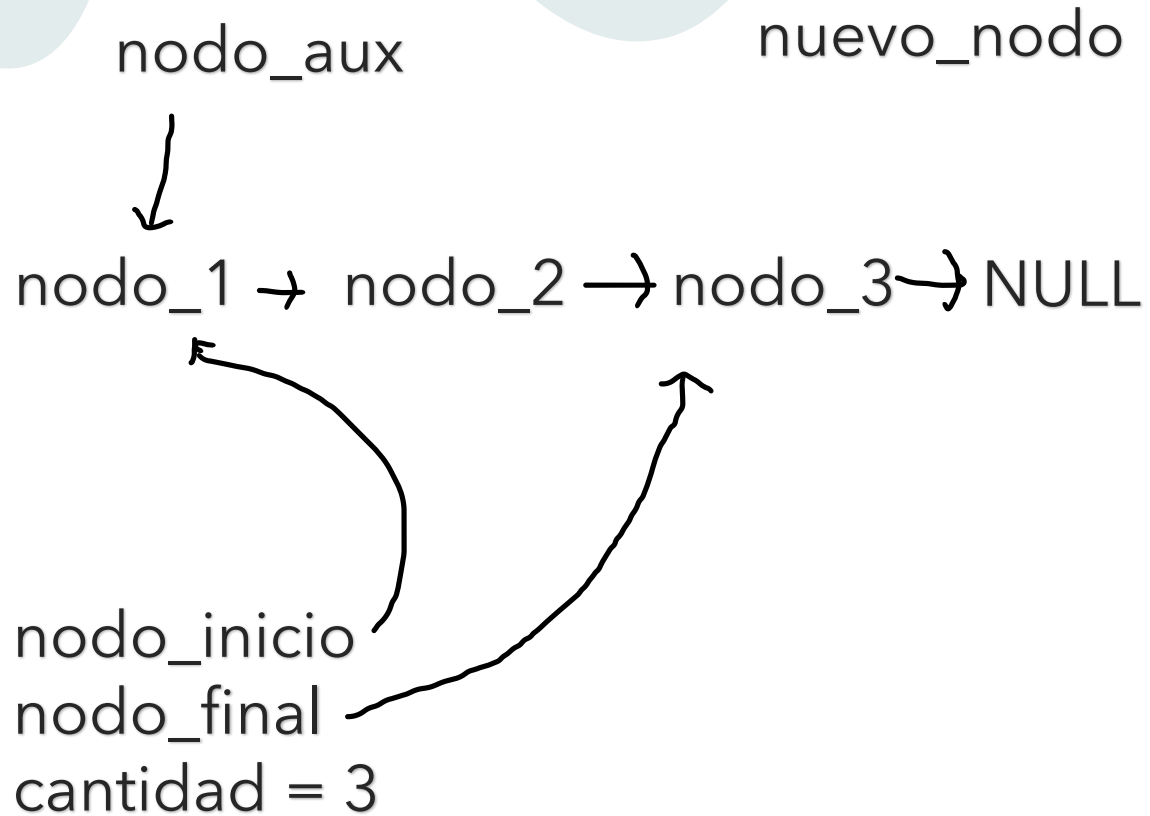
cantidad = 3



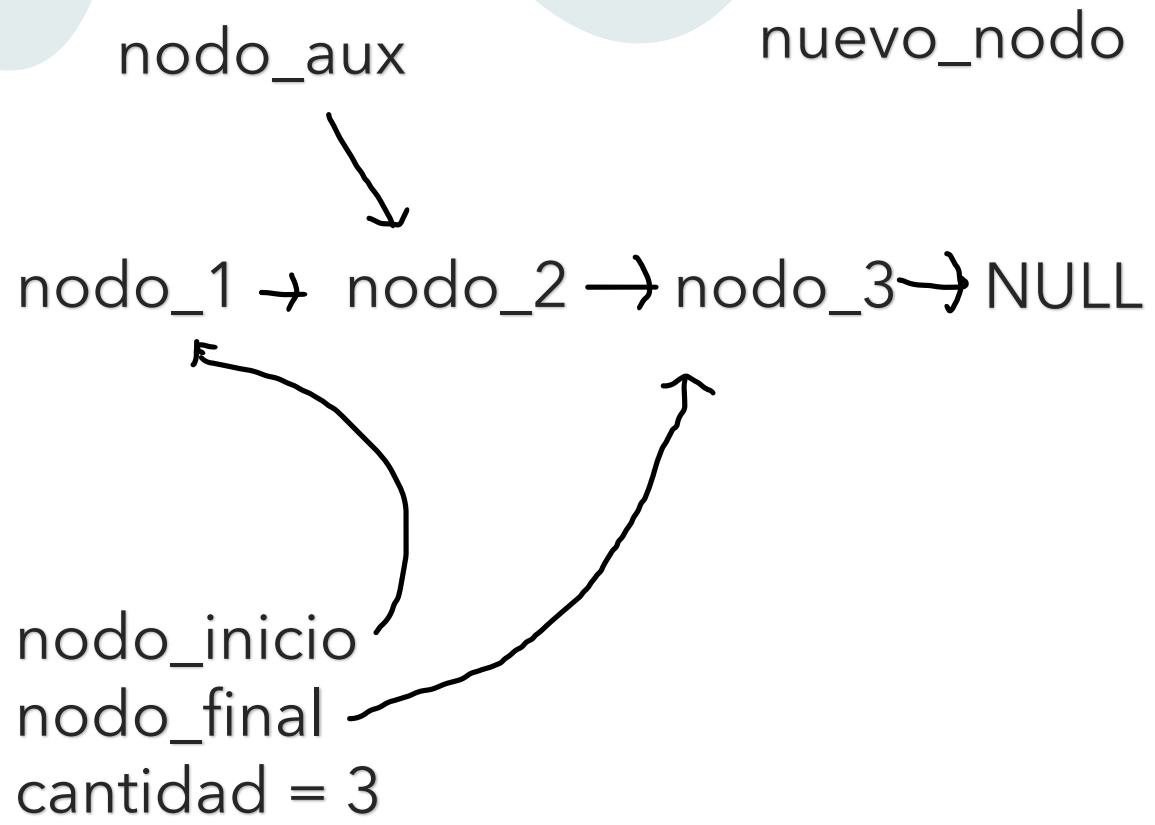


# lista\_insertar\_en\_posicion()

Dato: cambiamos nuestro nodo auxiliar hasta llegar al nodo en la posicion anterior al que buscamos.

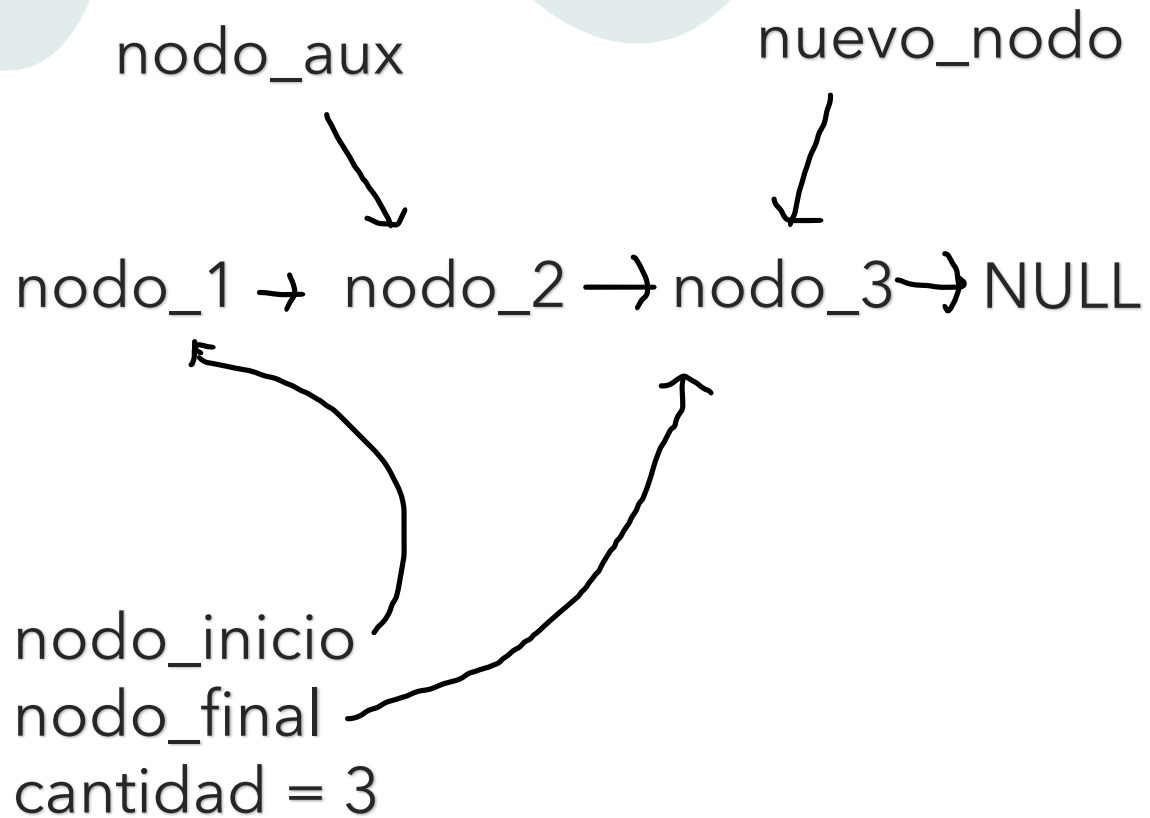


# lista\_insertar\_en\_posicion()



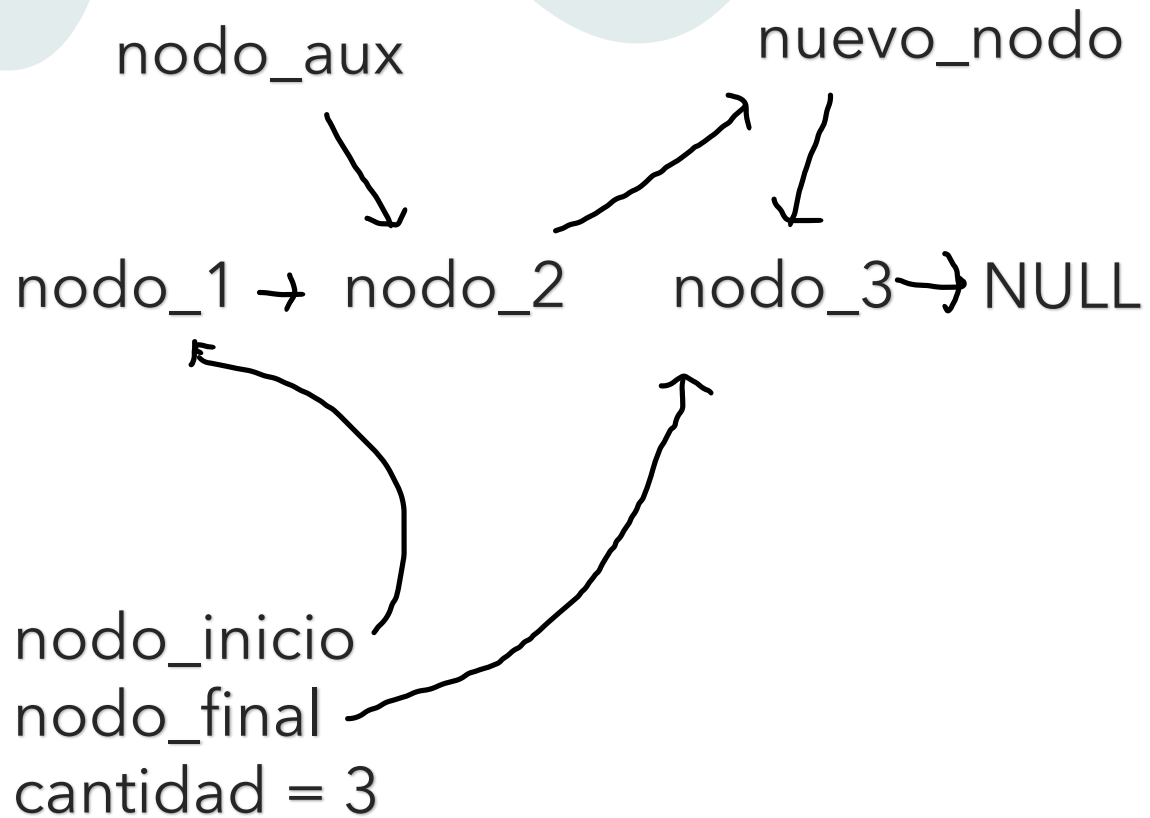
# lista\_insertar\_en\_posicion()

Dato: una vez llegado al nodo anterior, recordando que nuestro `nodo_aux` es igual a nuestro nodo anterior a la posicion, le cambiamos a donde apunta nuestro nuevo nodo, para que coincida con lo que apunta el anterior. De esta manera no perdemos la referencia al nodo al que queremos sustituir en posicion.



# lista\_insertar\_en\_posicion()

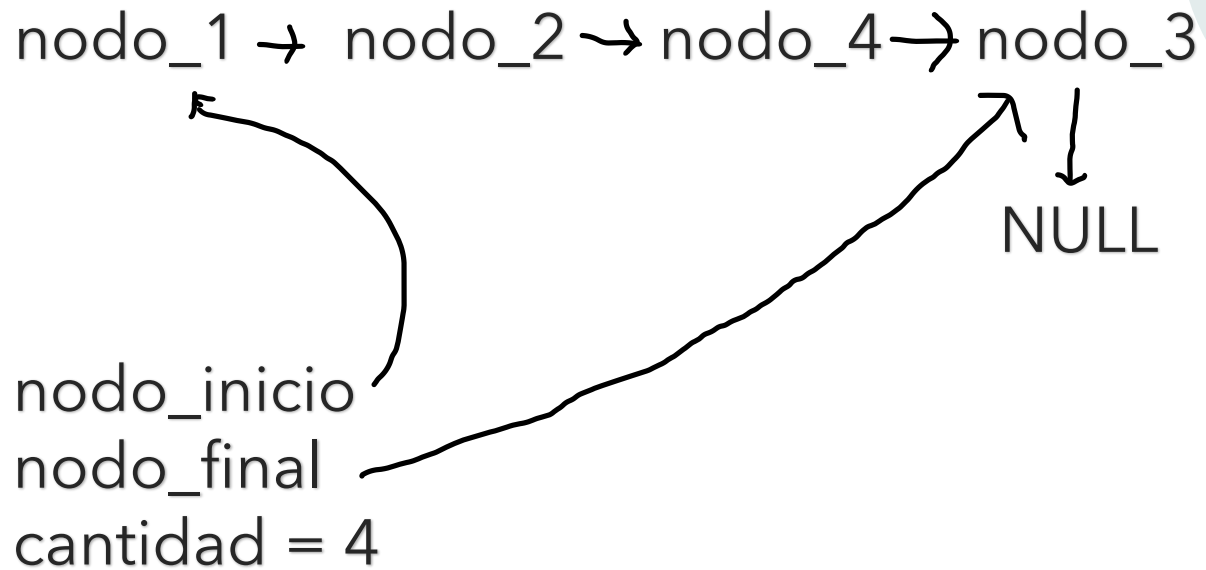
Dato: y ahora a donde apunta nuestro auxiliar



# lista\_insertar\_en\_posicion()

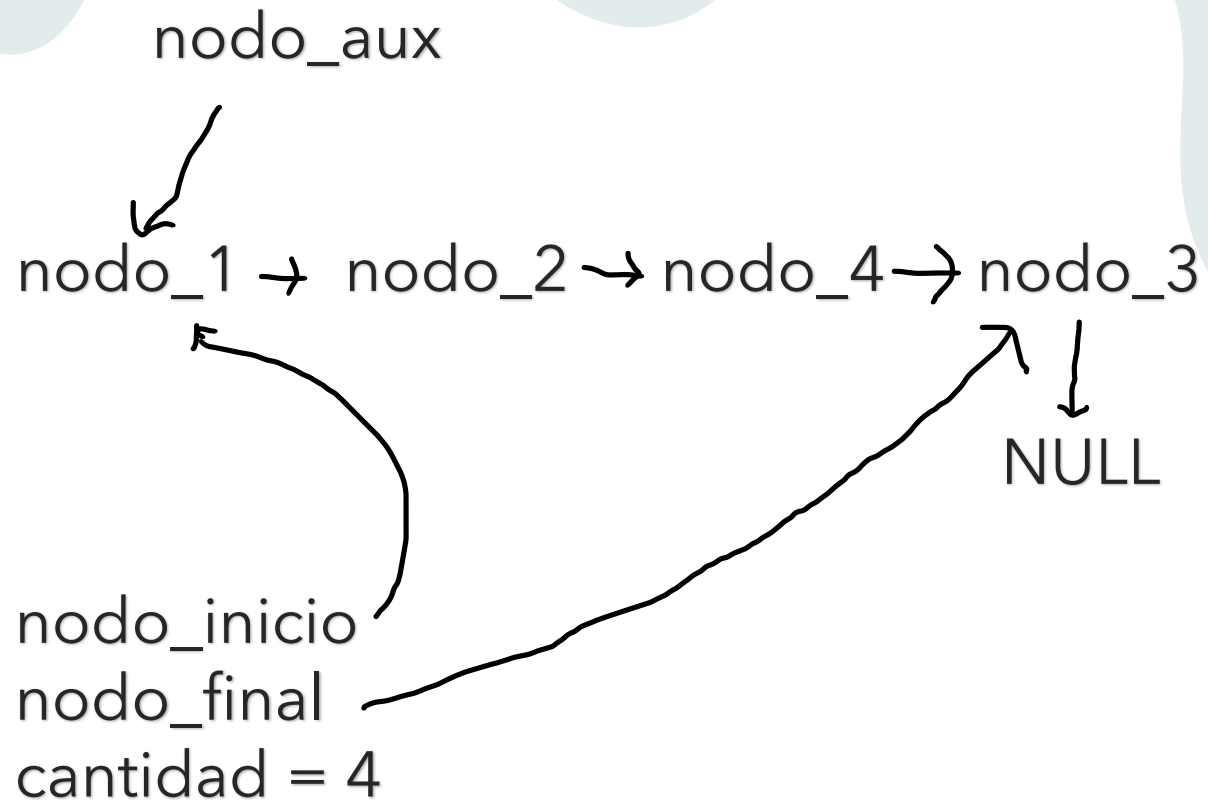
Dato: listo, insertado.

Dato 2: si quisiésemos insertar en posicion 0, basta con apuntar nuestro nuevo nodo al nodo inicio, y luego que nuestro nodo inicio apunte al nuevo nodo

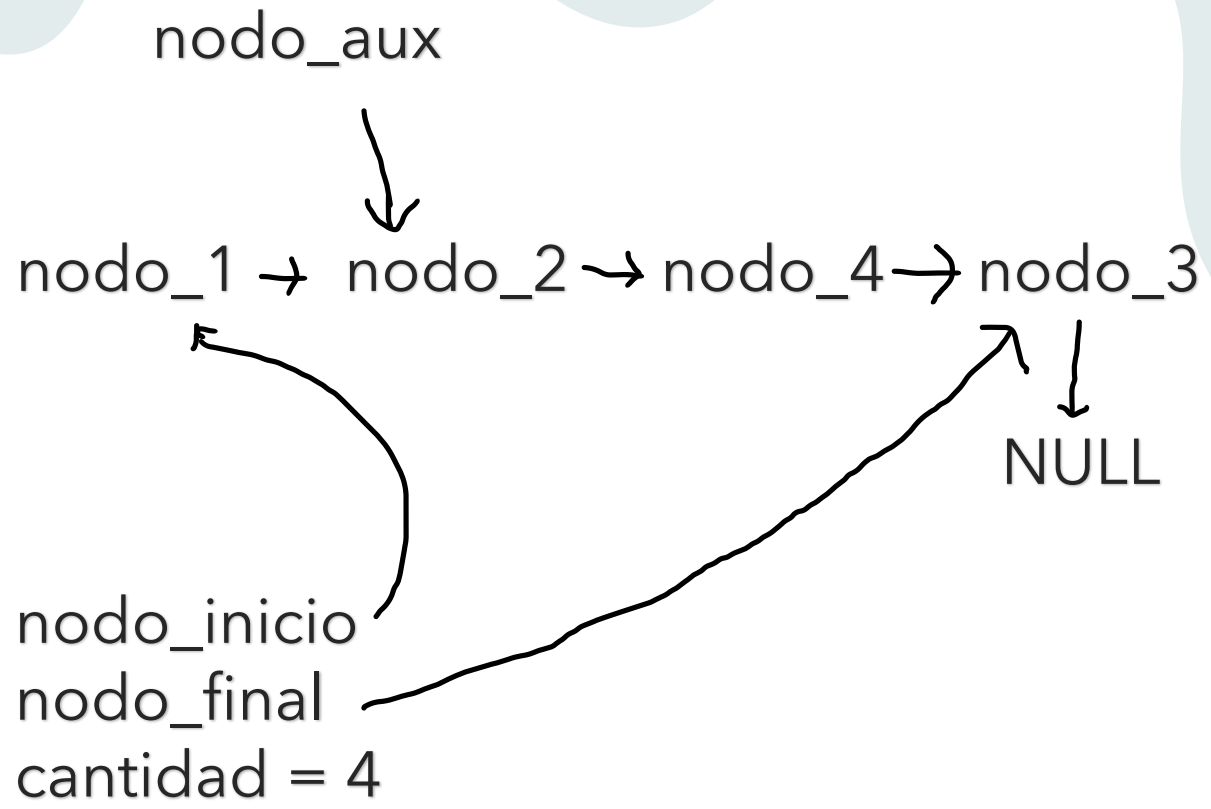


# lista\_quitar()

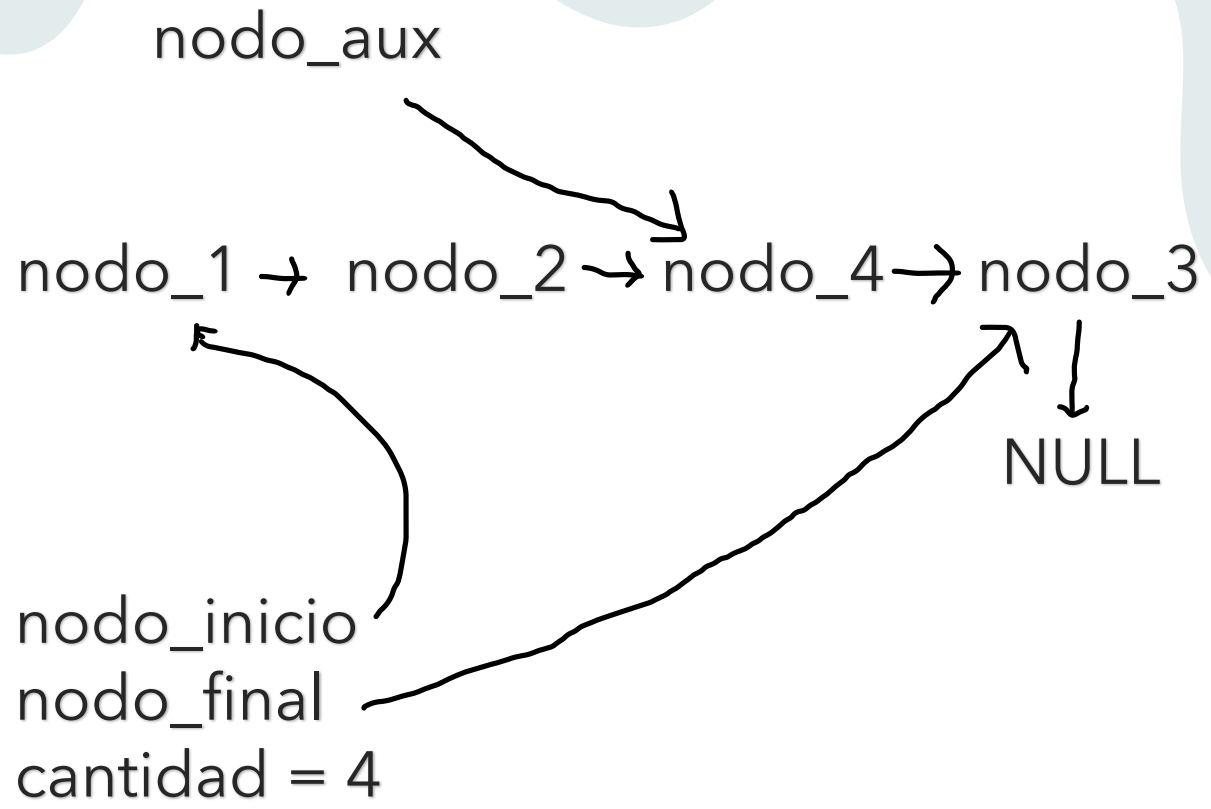
Dato: recorreremos con un auxiliar hasta el anterior al ultimo



# lista\_quitar()



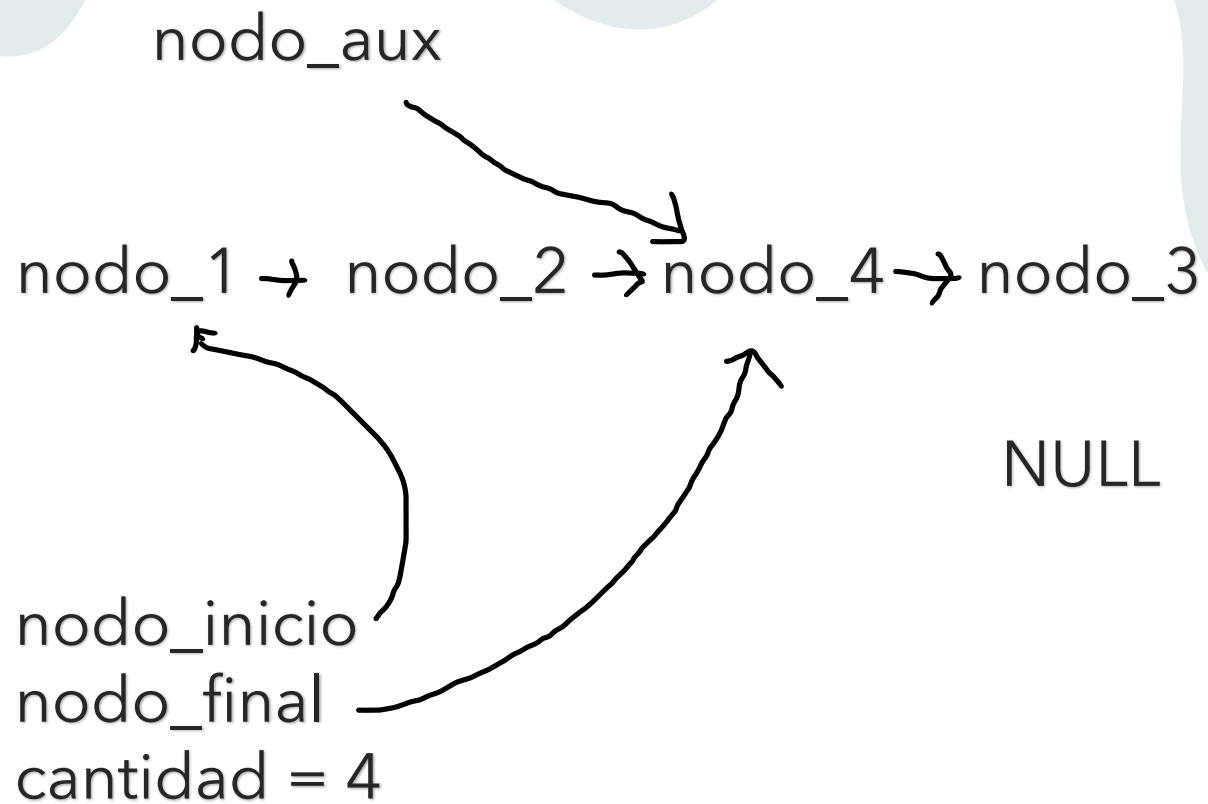
# lista\_quitar()





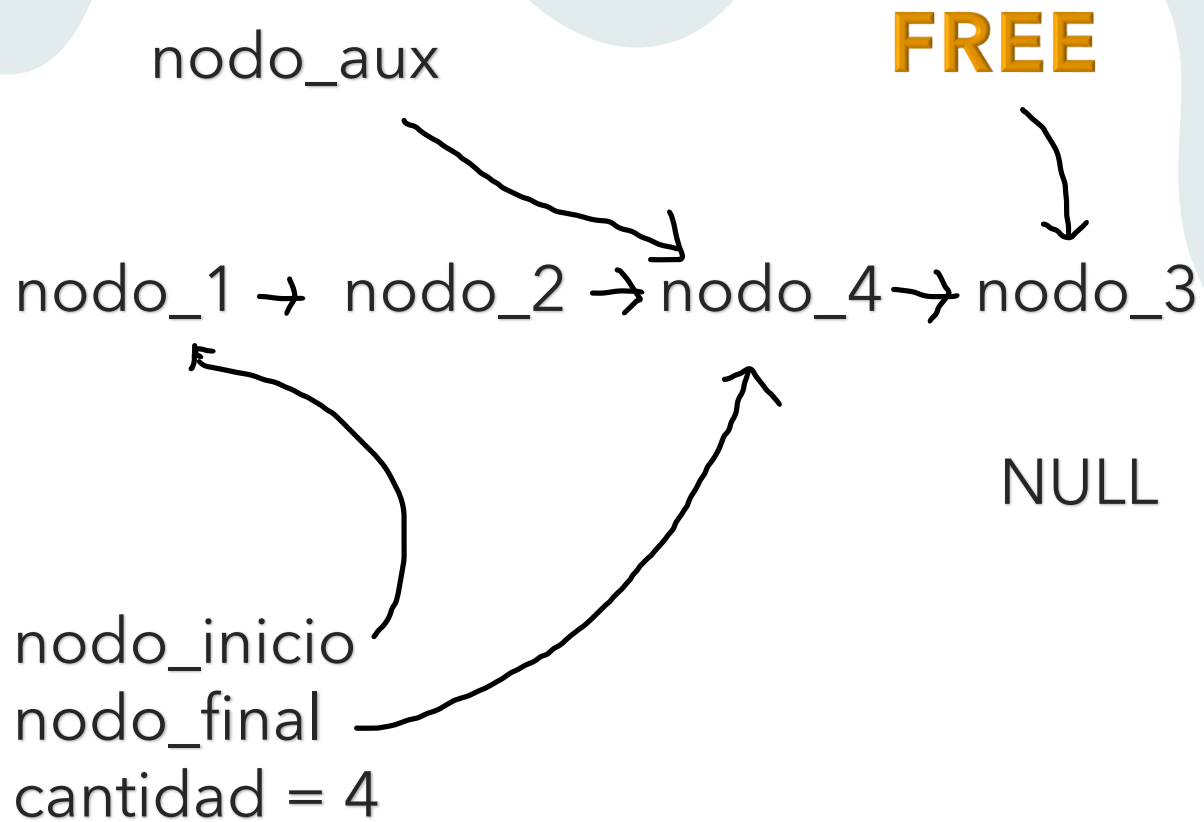
# lista\_quitar()

Dato: cambiamos nuestro fin para que apunte al auxiliar



# lista\_quitar()

Dato: liberamos la memoria del nodo al que apunta nuestro nodo final. Recordar que antes de eso, debemos almacenar en otro puntero la referencia a nuestro elemento, así podemos devolverlo.



# lista\_quitar()

Dato: ahora nuestro final apuntaria a basura, asi que lo cambiamos para que apunte a NULL. Y listo el pollo, pelada la gallina.

nodo\_1 → nodo\_2 → nodo\_4 → NULL

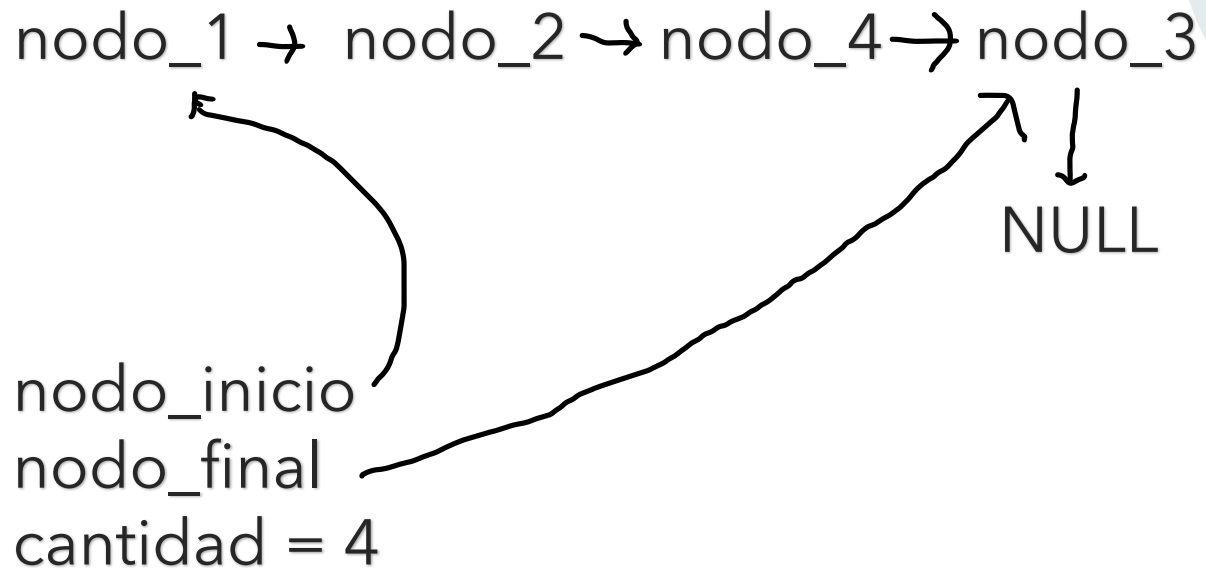
nodo\_inicio  
nodo\_final  
cantidad = 3



# lista\_quitar\_en\_posicion()

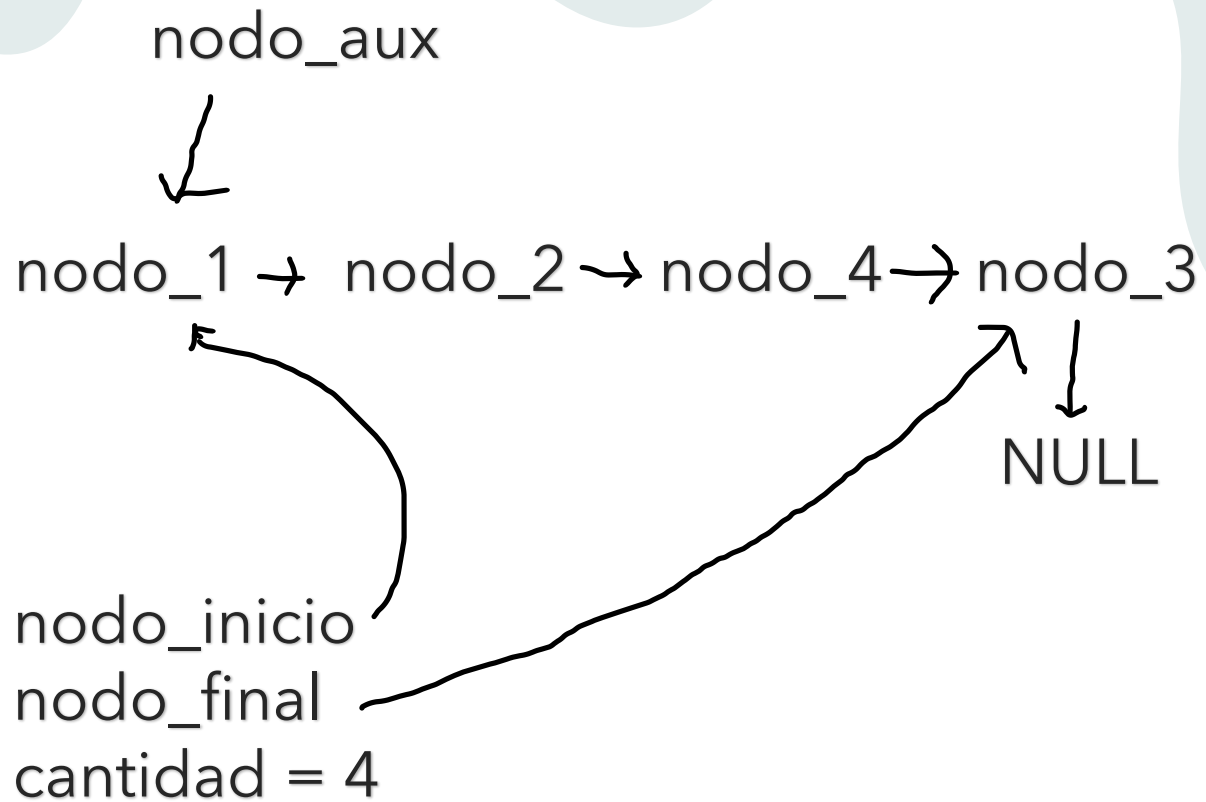
Dato: vamos a querer quitar el elemento en posicion 2

Dato 2: si quisiésemos quitar de la posicion 0, solo cambiamos nuestro nodo inicio, luego de almacenarnos la referencia al elemento y liberar el nodo.



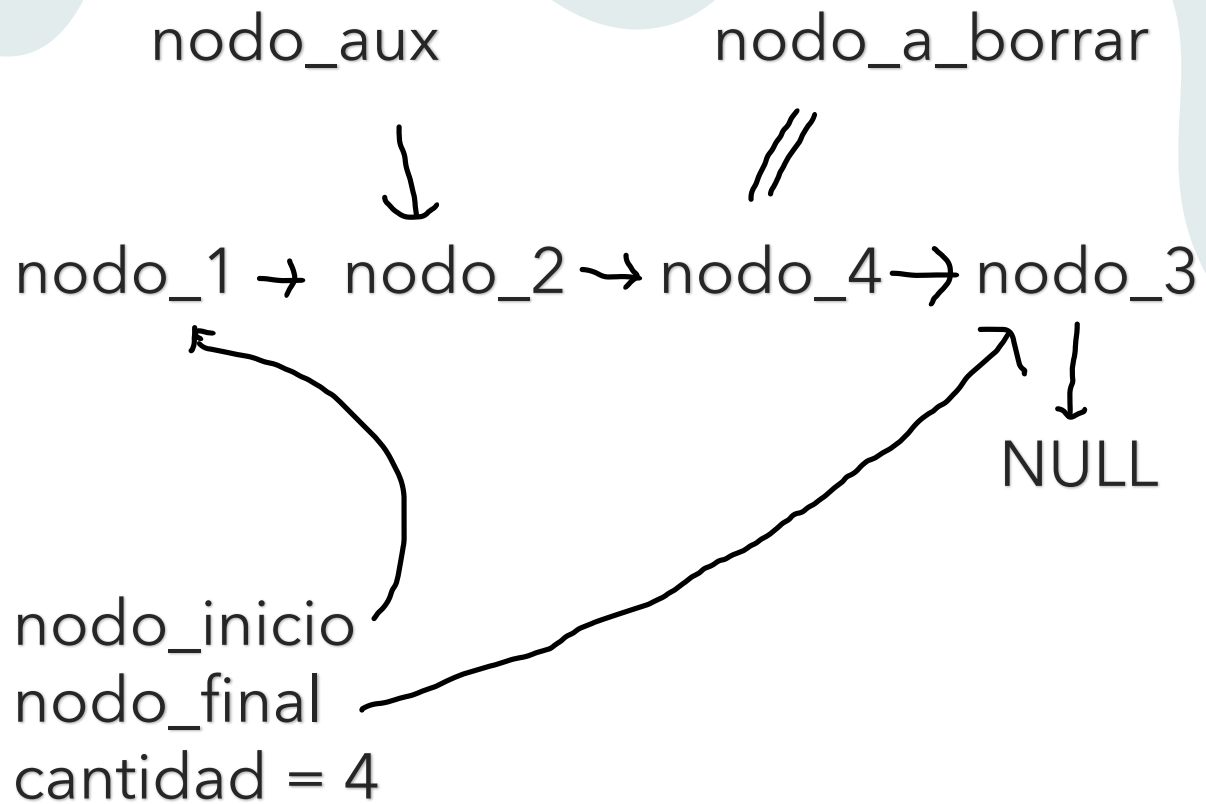
# lista\_quitar\_en\_posicion()

Dato: nuevamente, cazamos un nodo auxiliar y recorremos hasta llegar al nodo anterior al que queremos sacar de la lista



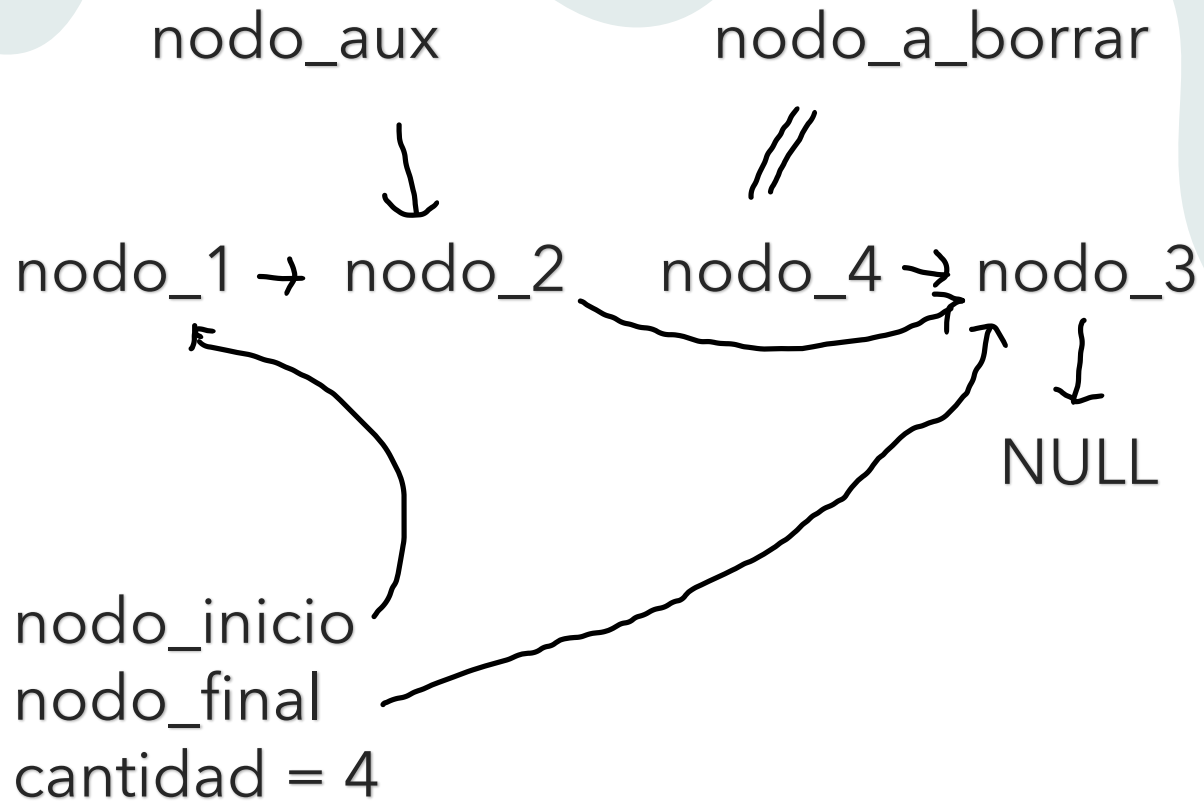
# lista\_quitar\_en\_posicion()

Dato: una vez llegado al nodo anterior, referenciamos nuestro nodo a eliminar por separado



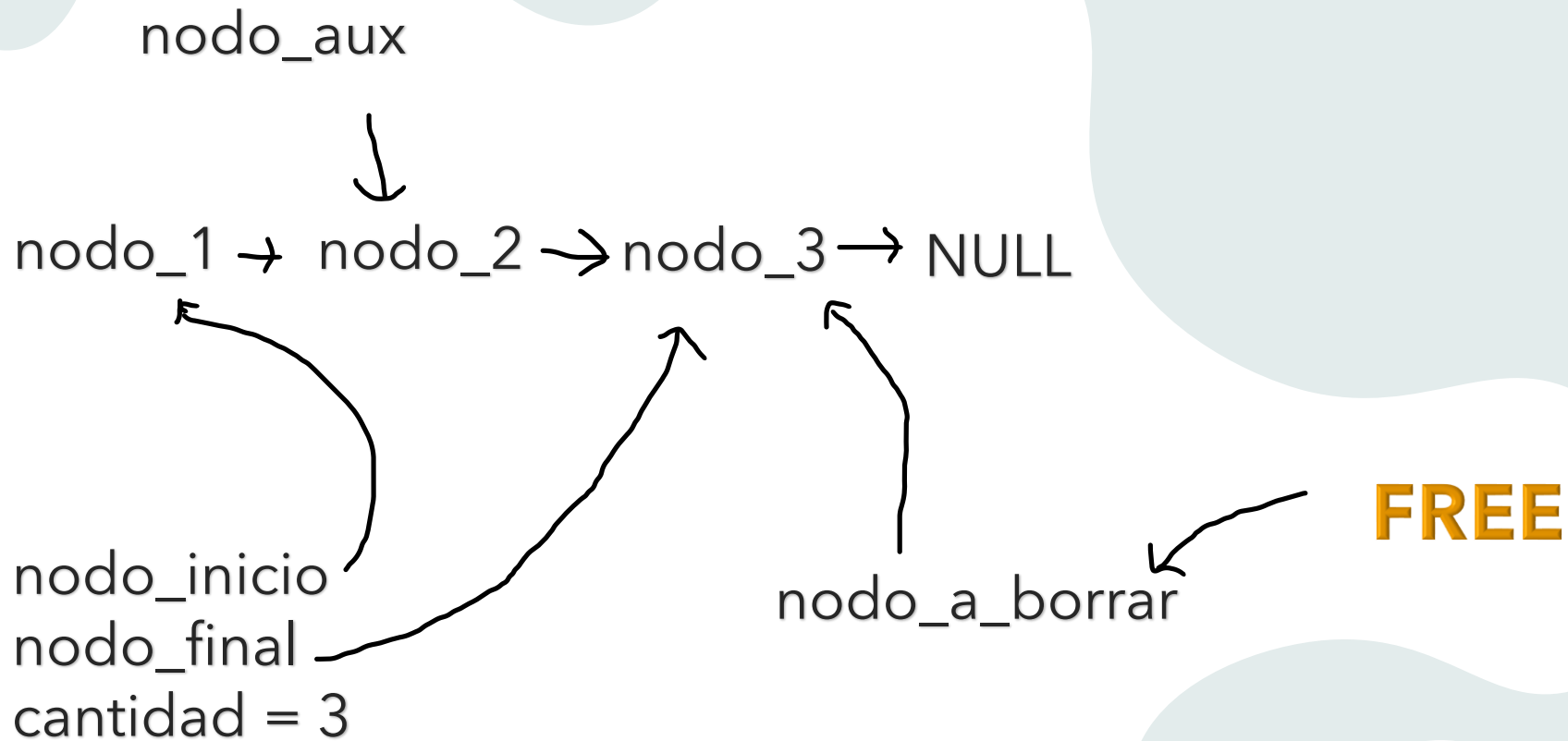
# lista\_quitar\_en\_posicion()

Dato: ahora nuestro nodo anterior va a apuntar a nuestro nodo siguiente al que queremos borrar.  
Dato 2: lastima por nuestro nodo\_3, se debe sentir juzgado de lo mucho que lo apuntan



# lista\_quitar\_en\_posicion()

Dato: nos guardamos por separado el elemento a devolver, liberamos el nodo a borrar



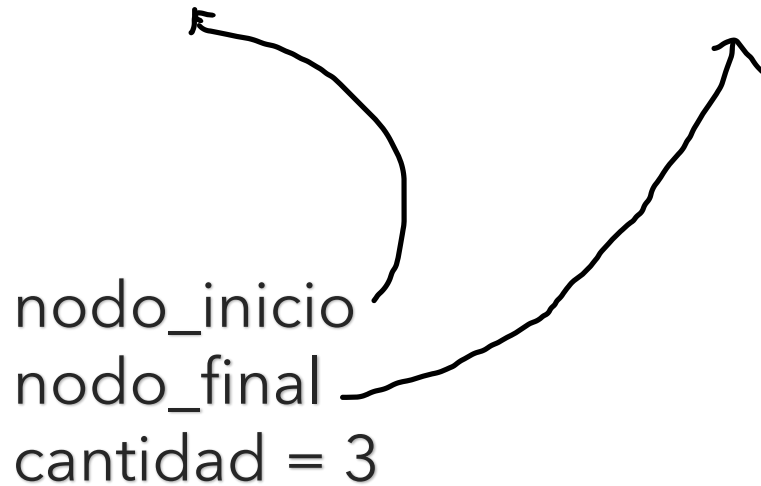


# lista\_quitar\_en\_posicion()

Dato: devolvemos el puntero al elemento y listo todo

nodo\_1 → nodo\_2 → nodo\_3 → NULL

nodo\_inicio  
nodo\_final  
cantidad = 3



The diagram illustrates the state of a linked list before a removal operation. A sequence of nodes is shown: 'nodo\_1 → nodo\_2 → nodo\_3 → NULL'. Below this, three variables are listed: 'nodo\_inicio', 'nodo\_final', and 'cantidad = 3'. A curved arrow originates from 'nodo\_inicio' and points to 'nodo\_1'. A longer, more complex arrow originates from 'nodo\_final' and also points to 'nodo\_1', indicating that both pointers currently reference the first node of the list.

# Implementacion de pila y cola

## Insertar:

- Pila: usamos lista\_insertar porque cada elemento nuevo se va al final
- Cola: lo mismo, nadie se puede colar en una cola

## Quitar:

- Pila: usamos lista\_quitar porque, como es el ultimo que entra el primero que sale, siempre sacamos el ultimo elemento
- Cola: usamos lista\_quitar\_en\_posicion 0, ya que el primer elemento en ser quitado es aquel que primero fue puesto (nuestro nodo inicio)

Para ambas implementaciones, reutilizamos la lista para que funcionen como funcionarían una pila y una cola, con UEPS y PEPS, respectivamente.

Esto quiere decir que tendremos que usar muchos casteos...

Dato: para el resto de funciones usamos las mismas para los 3 TDAs, exceptuando...

- lista\_ultimo para el tope de la pila, y...
- lista\_primero para el frente de la cola