LISTA DOBLEMENTE ENLAZADA

TAREA 8 PRESENTACIÓN LISTAS DOBLEMENTE ENLAZADAS

ASIGNATURA: ALGORITMIA Y ESTRUCTURA DE DATOS

PROFESOR: D. SC. GERARDO GARCÍA GIL

2021-B

ALUMNO: JOSÉ RAFAEL RUIZ GUDIÑO

INGENIERÍA EN DESARROLLO DE SOFTWARE

CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL (CETI)

INTRODUCCIÓN

- En algunas aplicaciones podemos desear recorrer la lista hacia adelante y hacia atrás, o dado un elemento, podemos desear conocer rápidamente los elementos anterior y siguiente. En tales situaciones podríamos desear darle a cada celda sobre una lista un puntero a las celdas siguiente y anterior en la lista tal y como se muestra en la figura.

- Otra ventaja de las listas doblemente enlazadas es que podemos usar un puntero a la celda que contiene el i-ésimo elemento de una lista para representar la posición i, mejor que usar el puntero a la celda anterior aunque lógicamente, también es posible la implementación similar a la expuesta en las listas simples haciendo uso de la cabecera. El único precio que pagamos por estas características es la presencia de un puntero adicional en cada celda y consecuentemente procedimientos algo más largos para algunas de las operaciones básicas de listas.

DEFINICIÓN

- Una lista doblemente enlazada es una lista lineal en la que cada nodo tiene dos enlaces, uno al nodo siguiente, y otro al anterior.
- Las listas doblemente enlazadas no necesitan un nodo especial para acceder a ellas, pueden recorrerse en ambos sentidos a partir de cualquier nodo, esto es porque a partir de cualquier nodo, siempre es posible alcanzar cualquier nodo de la lista, hasta que se llega a uno de los extremos.
- El nodo típico es el mismo que para construir las listas que hemos visto, salvo que tienen otro puntero al nodo anterior:

```
struct nodo {
   int dato;
   struct nodo *siguiente;
   struct nodo *anterior;
};
```

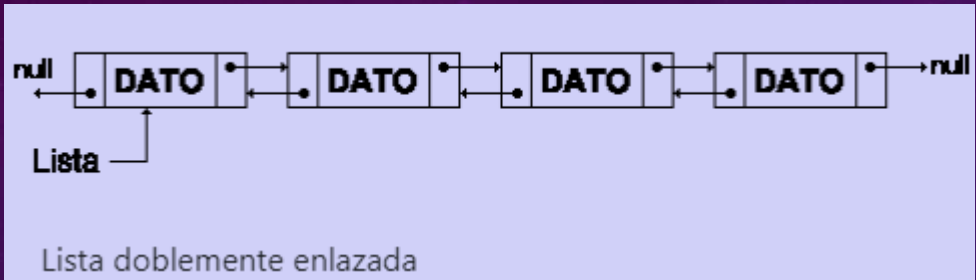
DECLARACIONES DE TIPOS PARA MANEJAR LISTAS DOBLEMENTE ENLAZADAS EN C

- Para C, y basándonos en la declaración de nodo que hemos visto más arriba, trabajaremos con los siguientes tipos:

```
typedef struct _nodo {
   int dato;
   struct _nodo *siguiente;
   struct _nodo *anterior;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;
```

- tipoNodo es el tipo para declarar nodos, evidentemente.
- pNodo es el tipo para declarar punteros a un nodo.
- Lista es el tipo para declarar listas abiertas doblemente enlazadas. También es posible, y potencialmente útil, crear listas doblemente enlazadas y circulares.



Lista doblemente enlazada

- El movimiento a través de listas doblemente enlazadas es más sencillo, y como veremos las operaciones de búsqueda, inserción y borrado, también tienen más ventajas.

OPERACIONES BÁSICAS CON LISTAS DOBLEMENTE ENLAZADAS

De nuevo tenemos el mismo repertorio de operaciones sobre este tipo listas:

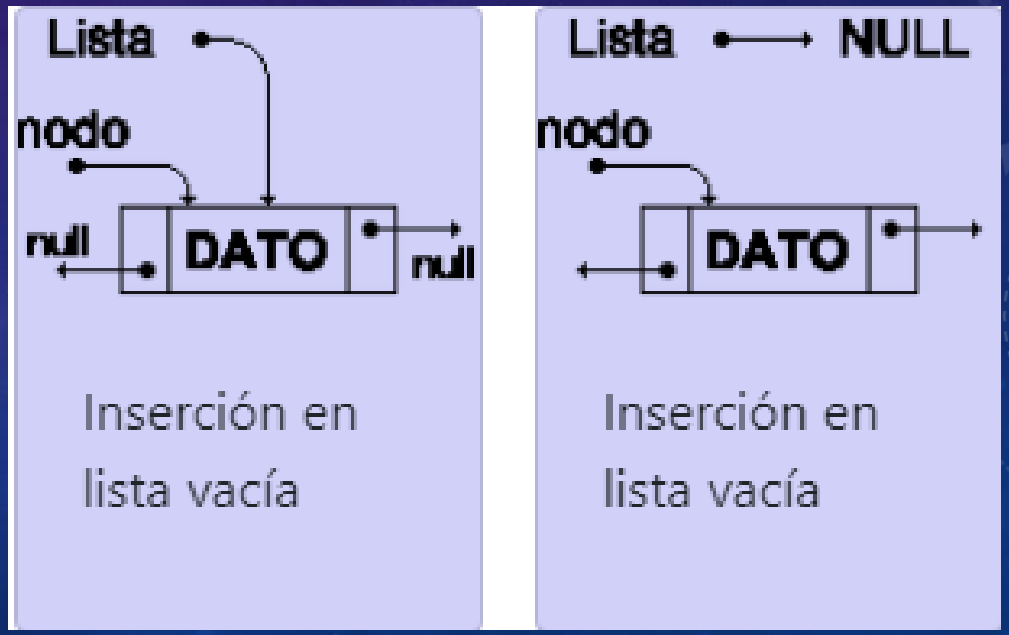
- Añadir o insertar elementos.
- Buscar o localizar elementos.
- Borrar elementos.
- Moverse a través de la lista, siguiente y anterior

AÑADIR UN ELEMENTO

Añadir elemento en una lista doblemente enlazada vacía

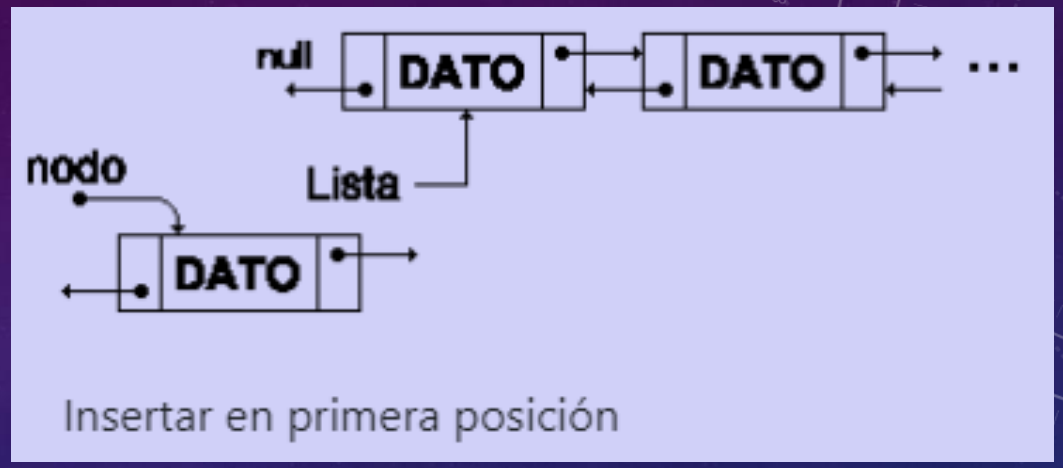
- Partiremos de que ya tenemos el nodo a insertar y, por supuesto un puntero que apunte a él, además el puntero que define la lista, que valdrá NULL:
- El proceso es muy simple, bastará con que:

1. lista apunta a nodo.
2. lista->siguiente y lista->anterior apunten a null.



Insertar un elemento en la primera posición de la lista

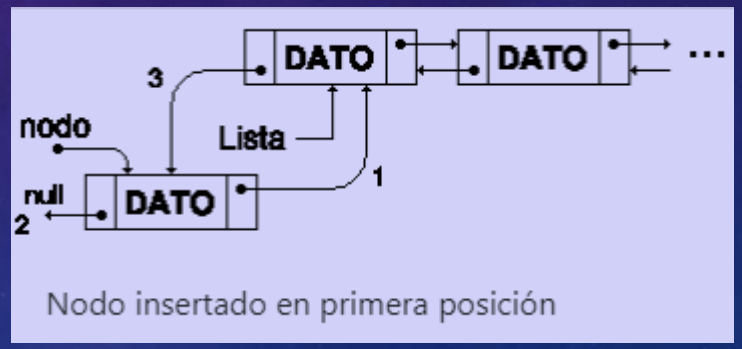
- Partimos de una lista no vacía. Para simplificar, consideraremos que lista apunta al primer elemento de la lista doblemente enlazada:



Insertar en primera posición

El proceso es el siguiente:

1. nodo->siguiente debe apuntar a Lista.
2. nodo->anterior apuntará a Lista->anterior.
3. Lista->anterior debe apuntar a nodo.

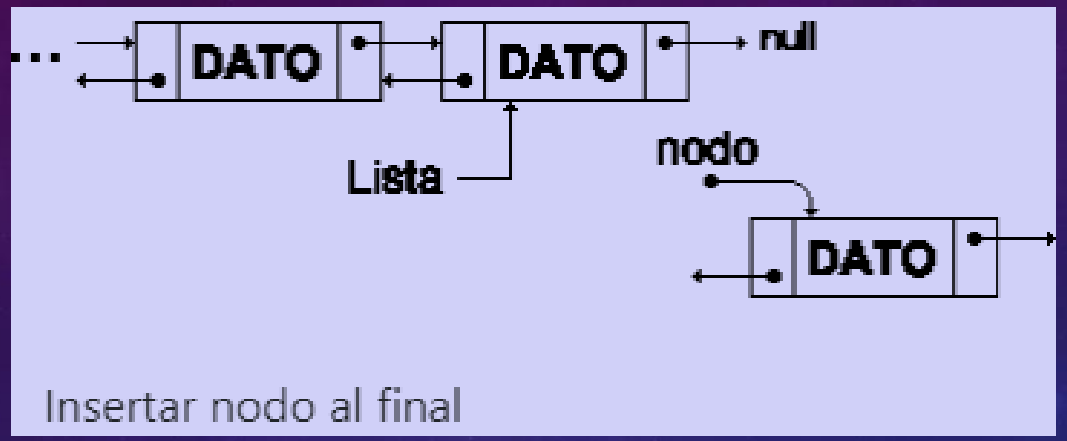


Nodo insertado en primera posición

Lista no tiene por qué apuntar a ningún miembro concreto de una lista doblemente enlazada, cualquier miembro es igualmente válido como referencia.

Insertar un elemento en la última posición de la lista

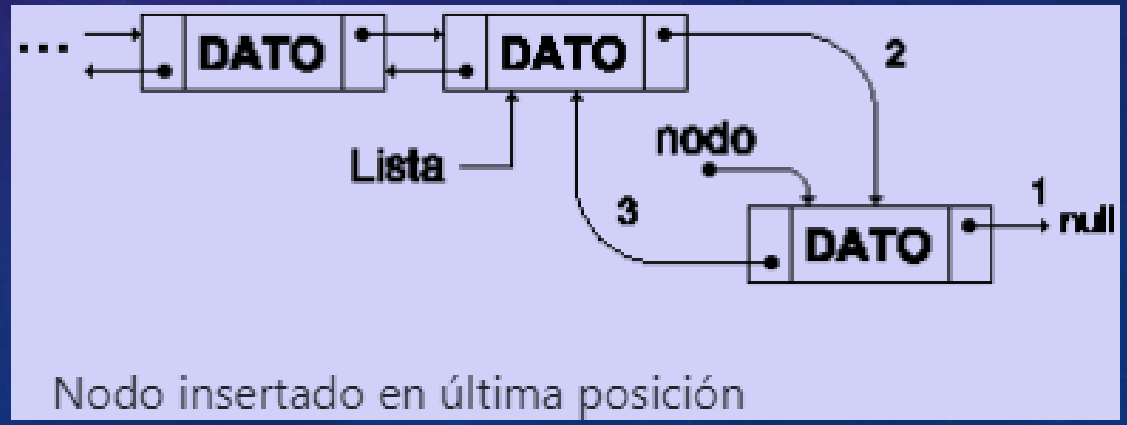
Igual que en el caso anterior, partiremos de una lista no vacía, y de nuevo para simplificar, que Lista está apuntando al último elemento de la lista:



Insertar nodo al final

El proceso es el siguiente:

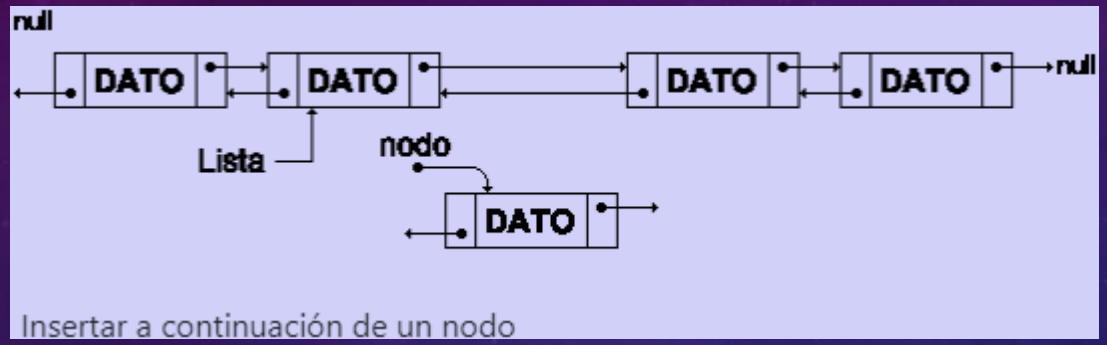
1. nodo->siguiente debe apuntar a Lista->siguiente (NULL).
2. Lista->siguiente debe apuntar a nodo.
3. nodo->anterior apuntará a Lista.



Nodo insertado en última posición

Insertar un elemento a continuación de un nodo cualquiera de una lista

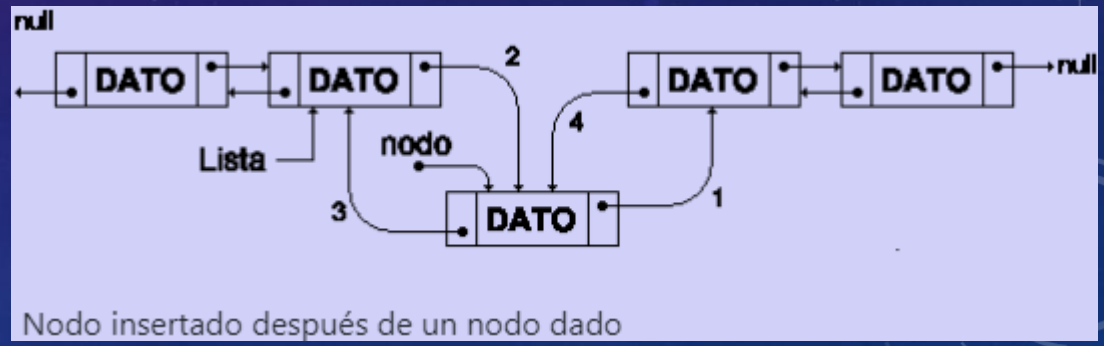
- Bien, este caso es más genérico, ahora partimos de una lista no vacía, e insertaremos un nodo a continuación de uno nodo cualquiera que no sea el último de la lista:



Insertar a continuación de un nodo

- El proceso sigue siendo muy sencillo:

1. Hacemos que nodo->siguiente apunte a lista->siguiente.
2. Hacemos que Lista->siguiente apunte a nodo.
3. Hacemos que nodo->anterior apunte a lista.
4. Hacemos que nodo->siguiente->anterior apunte a nodo.



Nodo insertado después de un nodo dado

Añadir elemento en una lista doblemente enlazada, caso general

Para generalizar todos los casos anteriores, sólo necesitamos añadir una operación:

1. Si lista está vacía hacemos que Lista apunte a nodo. Y nodo->anterior y nodo->siguiente a NULL.
2. Si lista no está vacía, hacemos que nodo->siguiente apunte a Lista->siguiente.
3. Después que Lista->siguiente apunte a nodo.
4. Hacemos que nodo->anterior apunte a Lista.
5. Si nodo->siguiente no es NULL, entonces hacemos que nodo->siguiente->anterior apunte a nodo.

El paso 1 es equivalente a insertar un nodo en una lista vacía.

Los pasos 2 y 3 equivalen a la inserción en una lista enlazada corriente.

Los pasos 4, 5 equivalen a insertar en una lista que recorre los nodos en sentido contrario.

Existen más casos, las listas doblemente enlazadas son mucho más versátiles, pero todos los casos pueden reducirse a uno de los que hemos explicado aquí.

ALGORITMO DE INSERCIÓN

1. El primer paso es crear un nodo para el dato que vamos a insertar.
2. Si Lista está vacía, o el valor del primer elemento de la lista es mayor que el del nuevo, insertaremos el nuevo nodo en la primera posición de la lista.
3. En caso contrario, buscaremos el lugar adecuado para la inserción, tenemos un puntero "anterior". Lo inicializamos con el valor de Lista, y avanzaremos mientras anterior->siguiente no sea NULL y el dato que contiene anterior->siguiente sea menor o igual que el dato que queremos insertar.
4. Ahora ya tenemos anterior señalando al nodo adecuado, así que insertamos el nuevo nodo a continuación de él.

BUSCAR O LOCALIZAR UN ELEMENTO DE UNA LISTA DOBLEMENTE ENLAZADA

Para recorrer una lista procederemos de un modo parecido al que usábamos con las listas abiertas, ahora no necesitamos un puntero auxiliar, pero tenemos que tener en cuenta que Lista no tiene por qué estar en uno de los extremos:

1. Retrocedemos hasta el comienzo de la lista, asignamos a lista el valor de lista->anterior mientras lista->anterior no sea NULL.
2. Abriremos un bucle que al menos debe tener una condición, que el índice no sea NULL.
3. Dentro del bucle asignaremos a lista el valor del nodo siguiente al actual.

```
typedef struct _nodo {
  int dato;
  struct _nodo *siguiente;
  struct _nodo *anterior;
  } tipoNodo;

  typedef tipoNodo *pNodo;
  typedef tipoNodo *Lista;
  ...
  pNodo = indice;
  ...
  indice = Lista;
  while(indice->anterior) indice = indice->anterior;
  while(indice) {
     printf("%d\n", indice->dato);
     indice = indice->siguiente;
  }
```

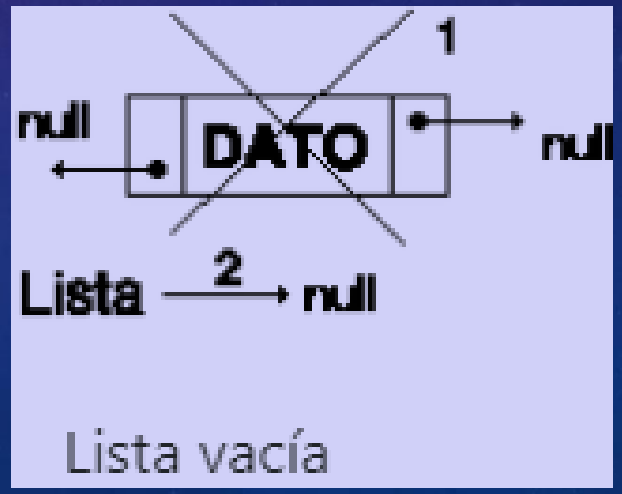
ELIMINAR UN ELEMENTO DE UNA LISTA DOBLEMENTE ENLAZADA

Eliminar el único nodo en una lista doblemente enlazada

En este caso, ese nodo será el apuntado por Lista.

El proceso es simple:

1. Eliminamos el nodo.
2. Hacemos que Lista apunte a NULL

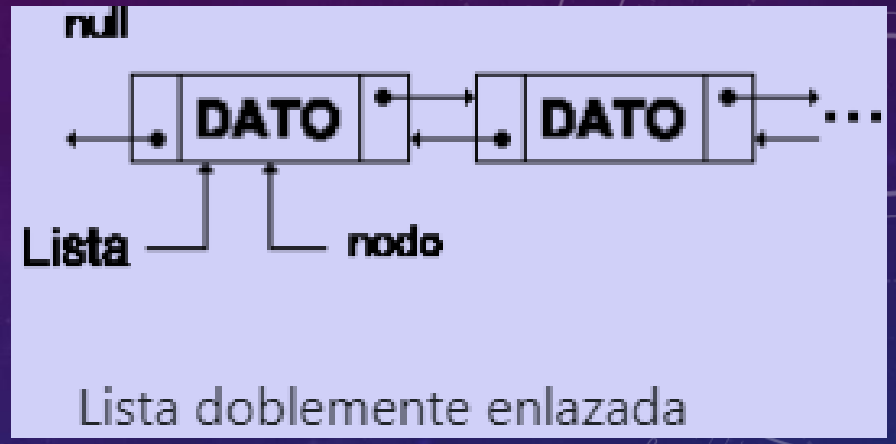


Eliminar el primer nodo de una lista doblemente enlazada

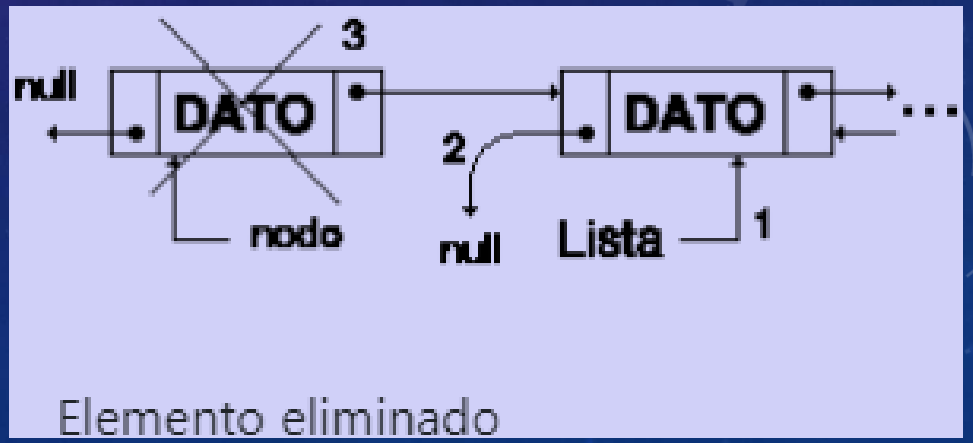
Tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->siguiente.

1. Si nodo apunta a Lista, hacemos que Lista apunte a Lista->siguiente.
2. Hacemos que nodo->siguiente->anterior apunte a NULL
3. Borramos el nodo apuntado por nodo.

- El paso 2 depara el nodo a borrar del resto de la lista, independientemente del nodo al que apunte Lista.



Lista doblemente enlazada

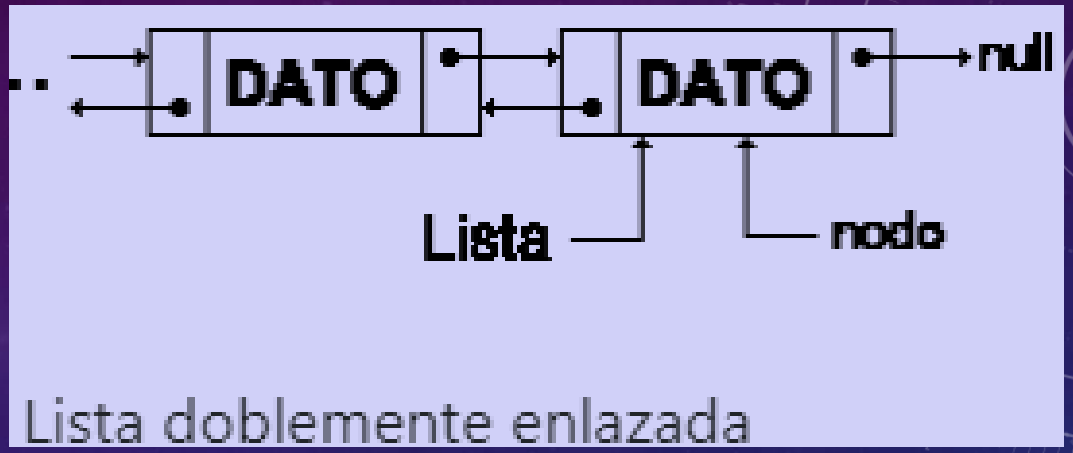


Elemento eliminado

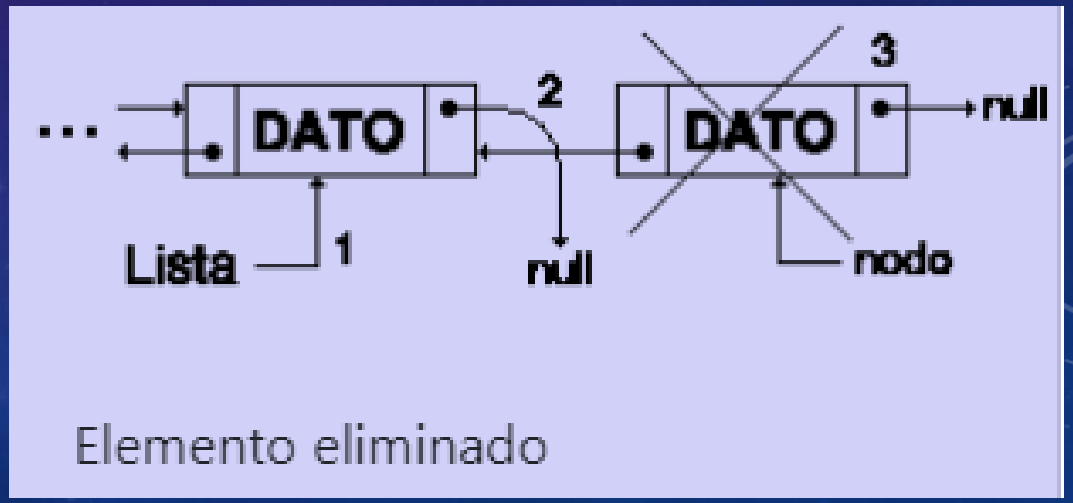
Eliminar el último nodo de una lista doblemente enlazada

De nuevo tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->anterior.

1. Si nodo apunta a Lista, hacemos que Lista apunte a Lista->anterior.
2. Hacemos que nodo->anterior->siguiente apunte a NULL
3. Borramos el nodo apuntado por nodo.



Lista doblemente enlazada



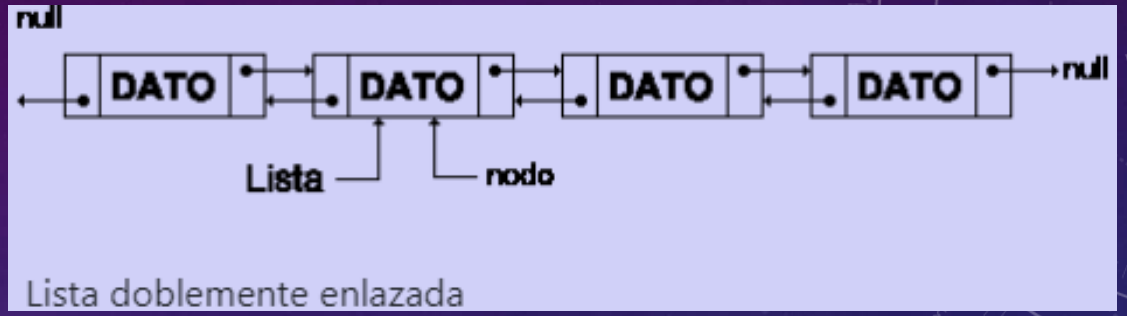
Elemento eliminado

ELIMINAR UN NODO INTERMEDIO DE UNA LISTA DOBLEMENTE ENLAZADA

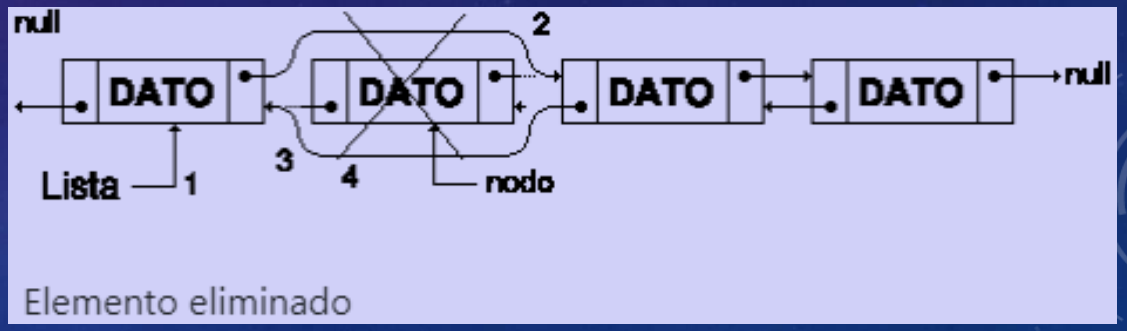
De nuevo tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->anterior o Lista->siguiente

Se trata de un caso más general de los dos casos anteriores.

1. Si nodo apunta a Lista, hacemos que Lista apunte a Lista->anterior (o Lista->siguiente).
2. Hacemos que nodo->anterior->siguiente apunte a nodo->siguiente.
3. Hacemos que nodo->siguiente->anterior apunte a nodo->anterior.
4. Borramos el nodo apuntado por nodo



Lista doblemente enlazada



Elemento eliminado

ELIMINAR UN NODO DE UNA LISTA DOBLEMENTE ENLAZADA, CASO GENERAL

De nuevo tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->anterior, si no es NULL o Lista->siguiente en caso contrario.

1. Si nodo apunta a Lista,
    - Si Lista->anterior no es NULL hacemos que Lista apunte a Lista->anterior.
    - Si Lista->siguiente no es NULL hacemos que Lista apunte a Lista->siguiente.
    - Si ambos son NULL, hacemos que Lista sea NULL.
2. Si nodo->anterior no es NULL, hacemos que nodo->anterior->siguiente apunte a nodo->siguiente.
3. Si nodo->siguiente no es NULL, hacemos que nodo->siguiente->anterior apunte a nodo->anterior.
4. Borramos el nodo apuntado por nodo.

ALGORITMO DE LA FUNCIÓN "BORRAR"

1. Localizamos el nodo de valor v
2. ¿Existe?
    - **SI**:
        - ¿Es el nodo apuntado por lista?
            - **SI**: Hacer que lista apunte a otro sitio.
        - ¿Es el primer nodo de la lista?
            - **NO**: nodo->anterior->siguiente = nodo->siguiente
        - ¿Es el último nodo de la lista?
            - **NO**: nodo->siguiente->anterior = nodo->anterior
        - Borrar nodo

REFERENCIAS

- Con Clase. (s. f.). *C Con Clase | Estructuras de datos (cap5)*. Recuperado 24 de octubre de 2021, de http://conclase.net/c/edd/cap5
- Universidad de Granada. (s. f.). *LISTAS DOBLEMENTE-ENLAZADAS*. Recuperado 24 de octubre de 2021, de https://ccia.ugr.es/%7Ejfv/ed1/tedi/cdrom/docs/ldoble.html