Lista doblemente enlazada

Es una estructura de datos útil y versátil en programación, se utiliza para almacenar elementos pero teniendo la particularidad de que nos podemos mover pa lante y pa atrás sobre los datos mediante un puntero.

Una lista doblemente enlazada al igual que la simplemente enlazada no es más que una colección de nodos donde cada uno contiene 3 elementos:

- Nodo anterior.
- dato.
- Nodo siguiente.

FIG 00: representación de un nodo al lado de su respectivo código.

Para poder utilizar esto necesitamos crear algo que se llama “lista doblemente enlazada”

Una lista doblemente enlazada en la colección de nodos que nos va a permitir guardar la información, dicha clase contiene un apuntador y varios datos… si observa la siguiente figura usted podrá notar cómo funciona:

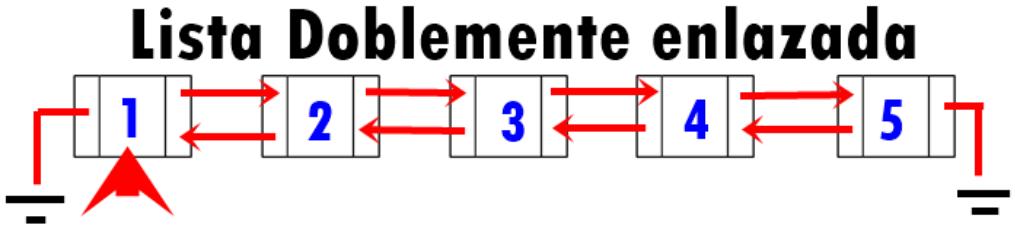

FIG 01: Ejemplo visual de una lista doblemente enlazada.

Las listas necesitan los siguientes atributos y métodos para poder funcionar:

- Se necesita un pivote el cual es el encargado de moverse para leer/escribir la información.
- Se necesita un método para agregar al inicio de la lista.
- Se necesita un método para agregar al final de la lista.

Advertencia: con esos pocos métodos nuestra lista es funcional.

Que es el pivote

El pivote no es más que una instancia del nodo, pero no podemos empezar la lista instanciando un nodo sin una data (o al menos no es recomendable) el pivote es la cabeza del nodo y se encarga de moverse para recorrer la lista y poder agregar elementos.

El pivote/cabezal es el encargado de ser el punto de acceso a los datos de la lista sin él no existiría una forma de recorrer la lista.

```
class DoubleLinkedList:
    def __init__(self) -> None:
        self.pivot = None
```

Advertencia: el pivote inicia en None dado a que cuando la lista es creada no posee datos.

Como se agregan elementos a la lista

Yo he decidido hacerlo de manera recursiva aunque también es posible hacerlo de manera iterativa. para agregar a la lista se tiene que partir de 2 casos base:

- Caso 1: no se ha creado la lista: lo primero que se hace es declarar el nodo con la respectiva información a almacenar y luego se declara como pivote ese nodo.

- Caso 2: la lista ya está creada: como antes ya se declaró un nodo lo con la información a almacenar lo que vamos a hacer es enviarla a un método recursivo para asociarla (linkearla) a su respectivo lugar.

Agregar elementos al inicio de la lista

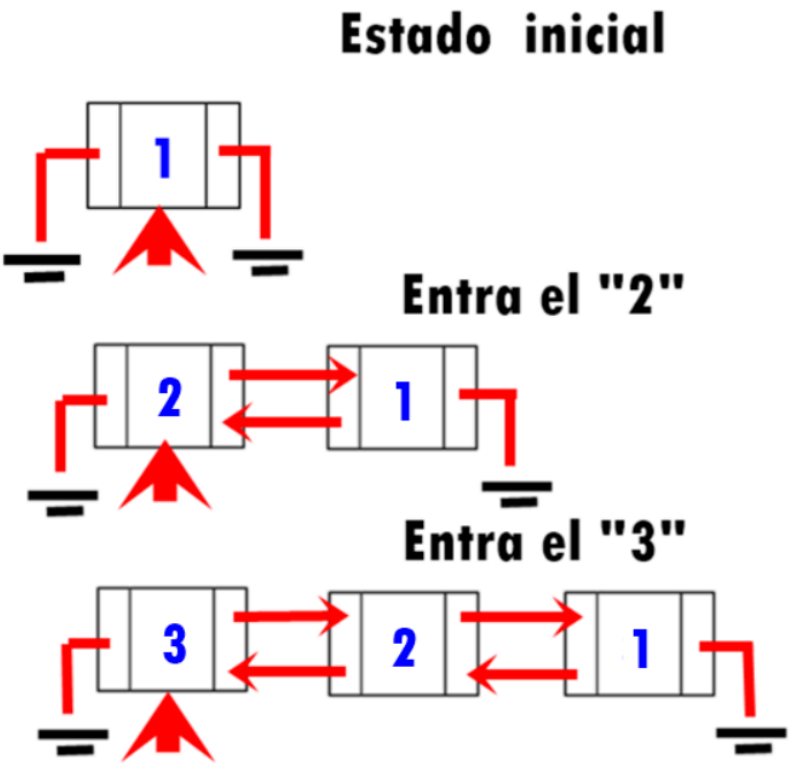

FIG 02: Ejemplo visual de cómo agregar elementos al inicio de la lista (Actualizando el pivote)

La lógica para agregar los elementos es que cada que entra un nuevo dato se establece como cabeza/pivote y luego al nuevo nodo se le establece el pivote como siguiente luego de ello procederemos a actualizar el pivote.

El código es el siguiente:

```
def addDataStart(self, data):
    new_node = Node(data)
    if self.pivot == Node:
        self.pivot = new_node
    else:
        new_node.next = self.pivot
        self.pivot.previous = new_node
        self.pivot = new_node
```

Para resolverlo tenemos 2 casos base:

- Caso 1: la lista no tiene información: pues entonces lo que se hace es establecer el nuevo nodo como pivote.

```
new_node = Node(data)
if self.pivot == None:
    self.pivot = new_node
```

- Caso 2: la lista ya tiene información: Pues entonces lo que vamos a hacer es que el nuevo nodo su siguiente apunte al pivote, luego hacemos que el anterior de nuestro pivote apunte al nuevo nodo y luego movemos el pivote al nuevo nodo.

```
new_node.next = self.pivot
self.pivot.previous = new_node
self.pivot = new_node
```

Para entender estos 3 pasos le aconsejamos mirar la siguiente imagen:

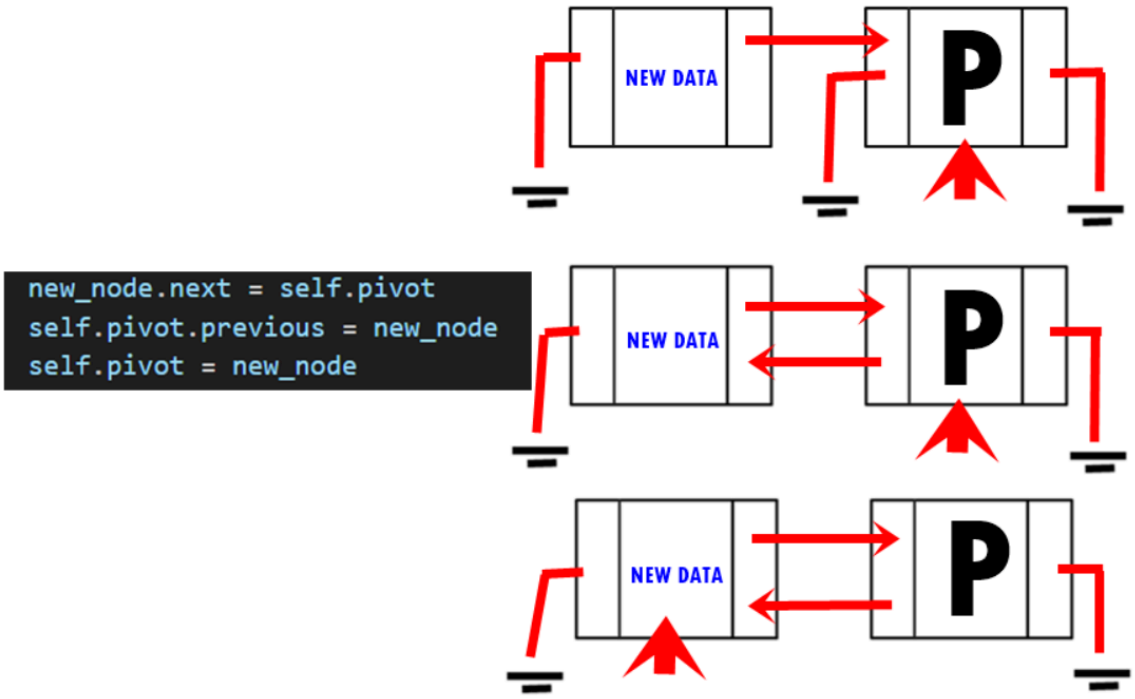

FIG 03: Ejemplo visual de cómo se agrega por la derecha cuando ya hay información (Actualización de pivote)

Agregar elementos al final de la lista

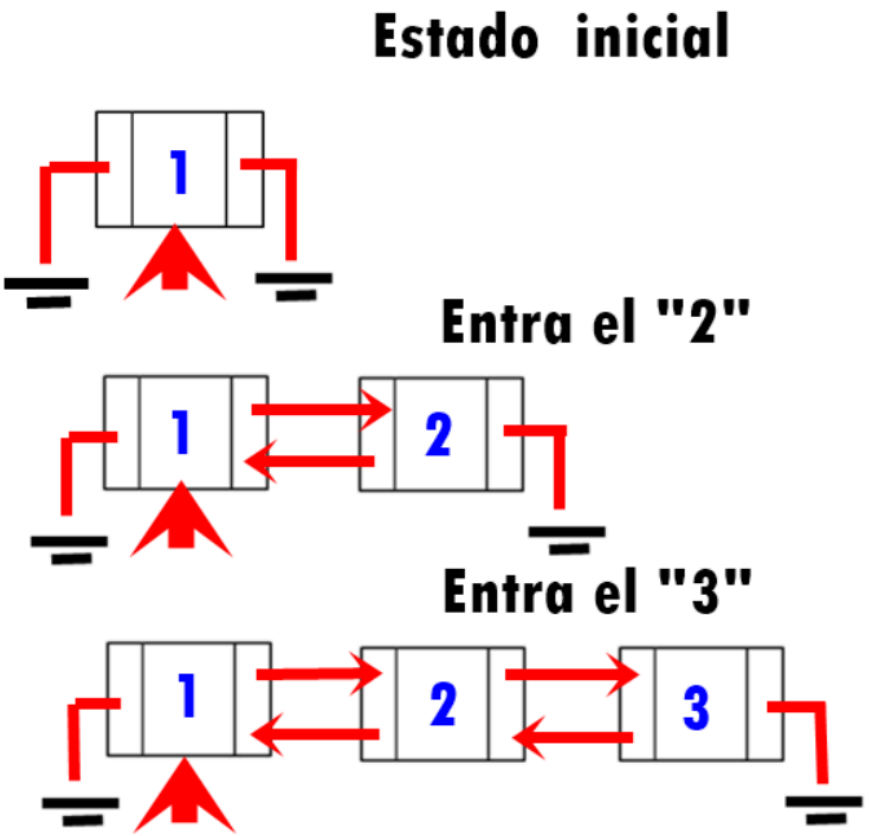

FIG 03: Ejemplo visual de cómo agregar elementos al final de la lista.

la lógica para lograr esto es buscar por la derecha un espacio disponible e insertar ahí el nuevo nodo, el lector debe de recordar que estamos tratando con listas doblemente enlazadas por ello se tiene que hacer la conexión del siguiente espacio disponible con el nuevo nodo y luego el nuevo nodo se interconecta por la izquierda con el espacio anterior al disponible.

El código para lograrlo es es el siguiente:

```
def addDataEnd(self, data):
    new_node = Node(data)
    if self.pivot == None:
        self.pivot = new_node
    else:
        _copy = self.pivot

        while _copy.next != None:
            _copy = _copy.next

        _copy.next = new_node
        new_node.previous = _copy
```

Para resolverlo tenemos 2 casos base:

- Caso 1: la lista está vacía: pues en ese caso lo único que tenemos que hacer es establecer el nuevo nodo como pivote.

```
new_node = Node(data)
if self.pivot == None:
    self.pivot = new_node
```

- Caso 2: la lista contiene información: en este caso lo que tenemos que hacer es recorrer con un while hasta encontrar un espacio disponible luego procederemos a realizar la doble conexión:

```
_copy = self.pivot

while _copy.next != None:
    _copy = _copy.next

_copy.next = new_node
new_node.previous = _copy
```

Métodos que le dan valor agregado a la lista:

- Se necesita un método para contar el total de elementos de la lista.
- Se necesita un método para verificar si existe un dato en la lista.
- Se necesita un método para actualizar un valor de la lista.
- Se necesita un método para eliminar un elemento.
- Se necesita un método para saber si la lista está vacía.
- Se necesita un método para obtener un dato en una posición x.

Contar los elementos de la lista

Lo único que tenemos que hacer es con un while recorrer todos los elementos de la lista.

```
def count(self):
    if self.pivot == None:
        return 0

    if self.pivot != None:
        _count = 1
        _copy = self.pivot.next

        while _copy != None:
            _count = _count + 1
            _copy = _copy.next

        return _count

    return 0
```

Verificar si existe un elemento en la lista

Lo que tenemos que hacer es recorrer con un while hasta encontrar el dato buscado

```
def isDataInList(self, data):
    if self.pivot == None:
        return False

    _copy = self.pivot
    while _copy != None:
        if _copy.data == data:
            return True

        _copy = _copy.next

    return False
```

Actualizar un valor de la lista

```
def updateValue(self, index, data):
    _count = self.count()

    if index >= 0 and index < _count:
        _copy = self.pivot
        _counter = 0
        while _counter < index:
            _copy = _copy.next
            _counter = _counter + 1

        # UPDATE
        _copy.data =  data
```