Lista simplemente enlazada

Son una estructura de datos fundamental en la programación, se utiliza para almacenar elementos de manera secuencial, almacena una lista de elementos y la manera en que podemos movernos es mediante un puntero.

Su funcionamiento es mediante una colección de nodos y cada nodo contiene la data y un apuntador hacia el siguiente nodo:

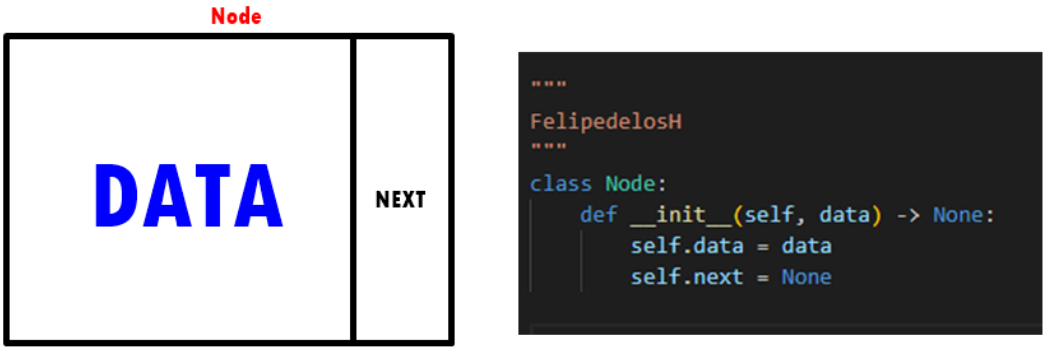

FIG 00: representación de un nodo al lado de su respectivo código.

Para poder utilizar esto necesitamos crear algo que se llama "lista simplemente enlazada"

Una lista simplemente 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:

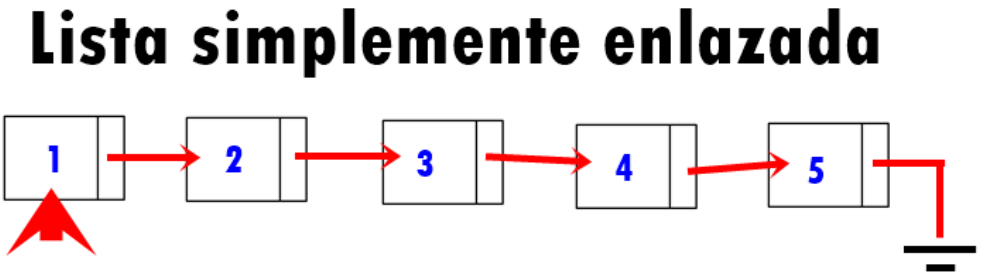

FIG 01: Ejemplo visual de una lista simplemente enlazada.

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.
- Se necesita un método para mostrar toda 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 SinglyList:
    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: la lista está vacía: La solución es simple se declara un nuevo nodo con la información a almacenar y luego el pivote se asigna a el nuevo nodo:

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

- Caso 2: La lista no está vacía en ese caso lo que hacemos es crear un nuevo nodo donde va a estar contenida el nuevo dato y con un while empezamos a recorrer los nodos de siguiente en siguiente hasta encontrar el que se encuentre vacío, luego lo guardamos en la siguiente posición vacía.

```
_copy = self.pivot
while _copy.next != None:
    _copy = _copy.next

_copy.next = new_node
```

Advertencia: se saca copia del pivote por precaución, para evitar borrar el original

Como se agregan elementos a la lista de manera recursiva

Advertencia

UTILIZAR MÉTODOS RECURSIVOS
podría causar un error de desbordamiento de pila si la lista es muy larga.

- Caso 1: la lista está vacía: en ese caso la solución es muy simple entonces el pivote se declara como nodo y se le ingresa el dato.

```
def addData(self, data):
    if self.pivot == None:
        self.pivot = Node(data)
```

- Caso 2: La lista no está vacía en ese caso lo que hacemos es crear un nuevo nodo donde va a estar contenida el nuevo dato y luego recursivamente empezamos a recorrer los nodos de siguiente en siguiente hasta encontrar el siguiente que se encuentre vacío y guardamos el nuevo nodo en la siguiente posición vacía.

```python
def _addData(self, pivot, new_node):
    if pivot != None:
        if pivot.next == None:
            pivot.next = new_node
        else:
            self._addData(pivot.next, new_node)
```

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 el total de elementos de una lista

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

    _copy = self.pivot
    _counter = 1
    while _copy.next != None:
        _counter = _counter + 1
        _copy = _copy.next

    return _counter
```

Lo que hacemos es partir de 2 casos base:

- Caso 1: la lista está vacía: en ese caso retornaremos cero.

- caso 2: la lista no está vacía: en ese caso hay 2 posibilidades que solo contenga un elemento o que contenga más. Para el caso donde contiene 1 solo elemento pues entonces no tiene un siguiente por lo que retornamos el uno que le fue asignado al contador; para el caso en donde contiene más de 1 elemento la lista no llegará hasta el nulo ya que está protegida por la condición del while por lo que faltaría un elemento por contar ello implica que nuestro contador aprovechará la limitante de no poder llegar hasta el final brindándonos la cuenta correcta.

Verificar si un elemento existe en la lista

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

    if self.pivot.data == data:
        return True

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

        if _copy.data == data:
            return True

        _copy = _copy.next

    return False
```

Para ello partimos de 3 casos base:

- caso 1: la lista está vacía: en ese caso lo que hacemos es retornar falso debido a que es imposible que exista un dato en esa lista.

- caso 2: la lista contiene información pero solamente de un elemento: en caso de que la lista contenga un solo elemento no le será posible entrar en el bucle por ello necesitamos una segunda condición que nos diga si el elemento que estamos buscando se encuentra en el pivote.

- Caso 3: El elemento no está en el pivote: en caso de que el elemento no esté en la cabeza/pivote se pasa a buscar en los nodos siguientes. Para este punto el lector estará pensando en el escenario 2. La buena noticia es: “si solo existe un elemento pues ya en el caso 2 se demostré que lo que se está buscando no existe por ello el while no se ejecuta y se retorna el falso de la parte final” por el otro lado “si existen varios elementos pues entonces se buscará iterativamente hasta encontrarlo y si no existe pues entonces se retornará el falso del final del método”

Eliminar un elemento

```python
def deleteValue(self, data):
    if self.pivot != None:
        # Case 01: is head
        if self.pivot.data == data:
            self.pivot = self.pivot.next
        # Case 02: Element it's not in head
        else:
            _copy = self.pivot

            while _copy.next != None:
                if _copy.next.data == data:
                    break
                _copy = _copy.next

            # Delete
            if _copy.next != None  and _copy.next.data == data:
                _copy.next = _copy.next.next
```

Para ello partimos de 2 casos base:

- Caso 1: lo que se quiere eliminar está en la cabeza: entonces para eliminar lo que hacemos es mover el pivote a la derecha y hay 2 posibilidades que sea otro nodo por lo cual ya no tendríamos cómo acceder a la antigua cabeza y en caso de que la lista solo contiene un elemento pues entonces la cabeza va a ser nula.

- Caso 2: lo que queremos eliminar no está en la cabeza: para ese caso lo que hacemos es con un while movernos hasta un paso antes de donde está el elemento a eliminar en caso de que exista se rompe el ciclo. Y por último preguntamos si en verdad lo encontramos y procederemos a eliminarlo como se hizo en el caso 1 (Asignando al siguiente).