

EDA – Sección 7 – 2025-10
M1-HT-1

A continuación, se provee una implementación básica de una lista enlazada doble:

```
def new_double_node(element):
    """
    Crea un nodo para una lista doblemente enlazada.

    El nodo contiene:
    - info: Información almacenada en el nodo.
    - next: Referencia al siguiente nodo, inicializada en None.
    - prev: Referencia al nodo anterior, inicializada en None.

    :param element: Elemento del nodo.
    :type element: any

    :return: Nodo recién creado.
    :rtype: dict que representa a un nodo en una lista doblemente enlazada.
    """
    return {"info": element, "next": None, "prev": None}

def new_double_list():
    """
    Crea una lista doblemente enlazada vacía con nodos centinela.

    :return: Lista vacía recién creada.
    :rtype: dict que representa una lista doblemente enlazada.
    """
    header = {"info": None, "next": None, "prev": None} # Nodo centinela inicial
    trailer = {"info": None, "next": None, "prev": header} # Nodo centinela final
    header["next"] = trailer # Conexión mutua

    return {"header": header, "trailer": trailer, "size": 0}

def is_empty(my_list):
    """
    Verifica si la lista doblemente enlazada está vacía.
    """
    return my_list["size"] == 0

def size(my_list):
    """
    Retorna el tamaño de la lista doblemente enlazada.
    """
    return my_list["size"]
```

```

def add_first(my_list, element):
    """
    Agrega un elemento al inicio de la lista doblemente enlazada.

    :param my_list: Lista a la cual se agregará el elemento.
    :type my_list: dict que representa una lista doblemente enlazada.
    :param element: Elemento a agregar.
    :type element: any.

    :return: Lista con el elemento agregado.
    :rtype: dict que representa una lista doblemente enlazada.

    >>> my_list = new_double_list()
    >>> add_first(my_list, 10) is not None # Caso lista vacía
    True
    >>> my_list["header"]["next"]["info"] == 10
    True
    >>> my_list["trailer"]["prev"]["info"] == 10
    True
    >>> my_list["size"] == 1
    True

    >>> add_first(my_list, 30) is not None # Caso agregar un 2do elemento
    True
    >>> my_list["header"]["next"]["info"] == 30
    True
    >>> my_list["header"]["next"]["next"]["info"] == 10
    True
    >>> my_list["trailer"]["prev"]["info"] == 10
    True
    >>> my_list["size"] == 2
    True

    >>> add_first(my_list, 40) is not None # Caso agregar un 3er elemento
    True
    >>> my_list["header"]["next"]["info"] == 40
    True
    >>> my_list["header"]["next"]["next"]["info"] == 30
    True
    >>> my_list["header"]["next"]["next"]["next"]["info"] == 10
    True
    >>> my_list["trailer"]["prev"]["info"] == 10
    True
    >>> my_list["trailer"]["prev"]["prev"]["info"] == 30
    True
    >>> my_list["trailer"]["prev"]["prev"]["prev"]["info"] == 40
    True
    >>> my_list["size"] == 3
    True
    """
    new_node = new_double_node(element)

    new_node["next"] = my_list["header"]["next"]
    new_node["prev"] = my_list["header"]
    my_list["header"]["next"]["prev"] = new_node
    my_list["header"]["next"] = new_node

    if my_list["size"] == 0:
        my_list["trailer"]["prev"] = new_node

    my_list["size"] += 1

    return my_list

```

1. Implemente la función `add_last()`:

```
def add_last(my_list, element):
    """
    Agrega un elemento al final de la lista doblemente enlazada.

    :param my_list: Lista a la cual se agregará el elemento.
    :type my_list: dict que representa una lista doblemente enlazada.
    :param element: Elemento a agregar.
    :type element: any.

    :return: Lista con el elemento agregado.
    :rtype: dict que representa una lista doblemente enlazada.

    >>> my_list = new_double_list()
    >>> add_last(my_list, 10) is not None # Caso lista vacía
    True
    >>> my_list["trailer"]["prev"]["info"] == 10
    True
    >>> my_list["header"]["next"]["info"] == 10
    True
    >>> my_list["size"] == 1
    True

    >>> add_last(my_list, 30) is not None # Caso agregar un 2do elemento
    True
    >>> my_list["trailer"]["prev"]["info"] == 30
    True
    >>> my_list["trailer"]["prev"]["prev"]["info"] == 10
    True
    >>> my_list["header"]["next"]["info"] == 10
    True
    >>> my_list["size"] == 2
    True

    >>> add_last(my_list, 40) is not None # Caso agregar un 3er elemento
    True
    >>> my_list["trailer"]["prev"]["info"] == 40
    True
    >>> my_list["trailer"]["prev"]["prev"]["info"] == 30
    True
    >>> my_list["trailer"]["prev"]["prev"]["prev"]["info"] == 10
    True
    >>> my_list["header"]["next"]["info"] == 10
    True
    >>> my_list["size"] == 3
    True
    """
```

2. Implemente la función `first_element()`:

```
def first_element(my_list):
    """
    Retorna la información del primer elemento de la lista doblemente enlazada.

    :param my_list: Lista doblemente enlazada.
    :type my_list: dict
    :return: Primer elemento de la lista o None si está vacía.
    :rtype: any

    >>> my_list = new_double_list()
    >>> first_element(my_list) is None # Lista vacía
    True

    >>> add_first(my_list, 10) is not None
    True
    >>> first_element(my_list) == 10 # Lista con un solo elemento
    True

    >>> add_last(my_list, 20) is not None
    True
    >>> first_element(my_list) == 10 # Lista con varios elementos
    True
    """
```

3. Implemente la función `last_element()`:

```
def last_element(my_list):
    """
    Retorna la información del último elemento de la lista doblemente enlazada.

    :param my_list: Lista doblemente enlazada.
    :type my_list: dict
    :return: Último elemento de la lista o None si está vacía.
    :rtype: any

    >>> my_list = new_double_list()
    >>> last_element(my_list) is None # Lista vacía
    True

    >>> add_first(my_list, 10) is not None
    True
    >>> last_element(my_list) == 10 # Lista con un solo elemento
    True

    >>> add_last(my_list, 20) is not None
    True
    >>> last_element(my_list) == 20 # Lista con varios elementos
    True
    """
```

4. Implemente la función `get_element()`:

```
def get_element(my_list, pos):
    """
    Retorna la información del elemento en la posición especificada de la lista
    doblemente enlazada.

    :param my_list: Lista doblemente enlazada.
    :type my_list: dict
    :param pos: Posición del elemento a obtener (0-indexado).
    :type pos: int
    :return: Elemento en la posición especificada o None si la posición es inválida.
    :rtype: any

    >>> my_list = new_double_list()
    >>> get_element(my_list, 0) is None # Lista vacía
    True

    >>> add_last(my_list, 10) is not None
    True
    >>> add_last(my_list, 20) is not None
    True
    >>> add_last(my_list, 30) is not None
    True

    >>> get_element(my_list, 0) == 10 # Primer elemento
    True
    >>> get_element(my_list, 1) == 20 # Segundo elemento
    True
    >>> get_element(my_list, 2) == 30 # Tercer elemento
    True
    >>> get_element(my_list, 3) is None # Índice fuera de rango
    True
    >>> get_element(my_list, -1) is None # Índice negativo
    True
    """
```

5. Implemente la función `remove_first()`:

```
def remove_first(my_list):
    """
    Elimina el primer elemento de la lista doblemente enlazada.

    :param my_list: Lista doblemente enlazada.
    :type my_list: dict
    :return: Información del elemento eliminado o None si la lista estaba vacía.
    :rtype: any

    >>> my_list = new_double_list()
    >>> remove_first(my_list) is None # Lista vacía
    True

    >>> add_last(my_list, 10) is not None
    True
    >>> add_last(my_list, 20) is not None
    True
    >>> add_last(my_list, 30) is not None
    True

    >>> remove_first(my_list) == 10 # Eliminando el primer elemento
    True
    >>> my_list["size"] == 2
    True
    >>> my_list["header"]["next"]["info"] == 20
    True

    >>> remove_first(my_list) == 20 # Eliminando el siguiente elemento
    True
    >>> my_list["size"] == 1
    True
    >>> my_list["header"]["next"]["info"] == 30
    True

    >>> remove_first(my_list) == 30 # Último elemento eliminado
    True
    >>> my_list["size"] == 0
    True
    >>> my_list["header"]["next"] == my_list["trailer"] # Lista vacía correctamente enlazada
    True
    >>> my_list["trailer"]["prev"] == my_list["header"]
    True
    """
```

6. Implemente la función `remove_last()`:

```
def remove_last(my_list):
    """
    Elimina el último elemento de la lista doblemente enlazada.

    :param my_list: Doubly linked list.
    :type my_list: dict
    :return: Information of the removed element or None if the list was empty.
    :rtype: any

    >>> my_list = new_double_list()
    >>> remove_last(my_list) is None # Lista vacía
    True

    >>> add_last(my_list, 10) is not None
    True
    >>> add_last(my_list, 20) is not None
    True
    >>> add_last(my_list, 30) is not None
    True

    >>> remove_last(my_list) == 30 # Eliminando el último elemento
    True
    >>> my_list["size"] == 2
    True
    >>> my_list["trailer"]["prev"]["info"] == 20
    True

    >>> remove_last(my_list) == 20 # Eliminando el siguiente elemento
    True
    >>> my_list["size"] == 1
    True
    >>> my_list["trailer"]["prev"]["info"] == 10
    True

    >>> remove_last(my_list) == 10 # Último elemento eliminado
    True
    >>> my_list["size"] == 0
    True
    >>> my_list["header"]["next"] == my_list["trailer"] # Lista vacía correctamente enlazada
    True
    >>> my_list["trailer"]["prev"] == my_list["header"]
    True
    """
```

7. Implemente la función delete_element():

```
def delete_element(my_list, pos):
    """
    Elimina el elemento en la posición especificada de la lista doblemente enlazada.

    :param my_list: Lista doblemente enlazada.
    :type my_list: dict
    :param pos: Posición del elemento a eliminar (0-indexado).
    :type pos: int
    :return: Información del elemento eliminado o None si la posición es inválida.
    :rtype: any

    >>> my_list = new_double_list()
    >>> delete_element(my_list, 0) is None # Lista vacía
    True

    >>> add_last(my_list, 10) is not None
    True
    >>> add_last(my_list, 20) is not None
    True
    >>> add_last(my_list, 30) is not None
    True
    >>> add_last(my_list, 40) is not None
    True

    >>> delete_element(my_list, 1) == 20 # Eliminando el segundo elemento
    True
    >>> my_list["size"] == 3
    True
    >>> get_element(my_list, 1) == 30 # Verificando que el nuevo segundo elemento es 30
    True

    >>> delete_element(my_list, 0) == 10 # Eliminando el primer elemento
    True
    >>> my_list["size"] == 2
    True
    >>> get_element(my_list, 0) == 30
    True

    >>> delete_element(my_list, 1) == 40 # Eliminando el último elemento
    True
    >>> my_list["size"] == 1
    True
    >>> get_element(my_list, 0) == 30
    True

    >>> delete_element(my_list, 0) == 30 # Eliminando el único elemento restante
    True
    >>> my_list["size"] == 0
    True
    >>> my_list["header"]["next"] == my_list["trailer"] # Lista correctamente vacía
    True
    >>> my_list["trailer"]["prev"] == my_list["header"]
    True

    >>> delete_element(my_list, 0) is None # Intentar eliminar en lista vacía
    True
    """
```


8. Implemente la función `insert_element()`:

```
def insert_element(my_list, pos, element):
    """
    Inserta un elemento en la posición especificada de la lista doblemente enlazada.

    :param my_list: Lista doblemente enlazada.
    :type my_list: dict
    :param pos: Posición donde se insertará el elemento (0-indexado).
    :type pos: int
    :param element: Elemento a insertar.
    :type element: any
    :return: True si la inserción fue exitosa, False si la posición es inválida.
    :rtype: bool

    >>> my_list = new_double_list()
    >>> insert_element(my_list, 0, 10) # Insertar en lista vacía (equivalente a add_first)
    True
    >>> my_list["size"] == 1
    True
    >>> get_element(my_list, 0) == 10
    True

    >>> insert_element(my_list, 1, 30) # Insertar al final (equivalente a add_last)
    True
    >>> my_list["size"] == 2
    True
    >>> get_element(my_list, 1) == 30
    True

    >>> insert_element(my_list, 1, 20) # Insertar en medio
    True
    >>> my_list["size"] == 3
    True
    >>> get_element(my_list, 1) == 20
    True
    >>> get_element(my_list, 2) == 30
    True

    >>> insert_element(my_list, 4, 40) # Posición inválida
    False
    """
```