Algoritmos T.A.D Listas - Juan Cruz Ambrosini

class LinkedList:

    head = None

class Node:

    value = None

    nextNode = None

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(1)*

def add(lista, elemento):

    if lista.head == None:

        newHead = Node()

        newHead.value = elemento

        lista.head = newHead

    else:

        newHead = Node()

        newHead.value = elemento

        newHead.nextNode = lista.head

        lista.head = newHead

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(n)*

*#Orden de complejidad arreglo: O(n)*

def search(list, element):

    exist = False

    counter = 0

    current\_node = list.head

    while current\_node != None:

        if current\_node.value == element:

            exist = True

            return counter

        counter = counter + 1

        current\_node = current\_node.nextNode

    if exist == False:

        return None

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(n)*

*#Orden de complejidad arreglo: O(n)*

def insert(list, element, index):

    if lengthList(list) < index :

        return None

    else:

        current\_node = list.head

        new\_node = Node()

        new\_node.value = element

        if index == 0:

            add(list, element)

            return index

        else:

            for i in range (1,lengthList(list) + 1):

                    if i == index:

                        new\_node.nextNode = current\_node.nextNode

                        current\_node.nextNode = new\_node

                        break

                    current\_node = current\_node.nextNode

            return index

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad O(n)*

*#Orden de complejidad arreglo: O(n^2)*

def delete(list, element):

    element\_position = search(list, element)

    current\_node = list.head

    if element\_position == None:

        return None

    else:

        if (current\_node.value == element):

            list.head = list.head.nextNode

        else:

            while (current\_node.nextNode != None):

                if current\_node.nextNode.value == element :

                    current\_node.nextNode = current\_node.nextNode.nextNode

                    break

                current\_node = current\_node.nextNode

        return element\_position

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(n)*

def showList (lista):

    current\_node = lista.head

    while current\_node != None :

        print(current\_node.value," ", end="")

        current\_node = current\_node.nextNode

    print("")

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(n)*

def update (list, element, index):

    current\_node = list.head

    size = lengthList(list)

    value = None

    if index > size or index < 0 :

        return None

    for i in range(0, size):

        if i == index:

            current\_node.value = element

            return index

        current\_node = current\_node.nextNode

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(n)*

def access(list, index):

    current\_node = list.head

    size = lengthList(list) - 1

    value = None

    if index > size or index < 0 :

        return None

    for i in range(0, size + 1):

        if i == index:

            value = current\_node.value

            return value

        current\_node = current\_node.nextNode

"""---------------------------------------------------------------------------------------------"""

*#Orden de complejidad: O(n)*

*#Orden de complejidad arreglo: O(n)*

def lengthList(list):

    lenght = 0

    current\_node = list.head

    while current\_node != None:

        current\_node = current\_node.nextNode

        lenght = lenght + 1

    return lenght

"""---------------------------------------------------------------------------------------------"""

def invertList(list):

    prev = None

    current = list.head

    while current != None:

        next\_node = current.nextNode

        current.nextNode = prev

        prev = current

        current = next\_node

    list.head = prev

Unit Test verificando el funcionamiento del código.

Imagen que contiene Forma

El contenido generado por IA puede ser incorrecto.