**UNIVERSIDAD NACIONAL MAYOR DE SAN**

**MARCOS FACULTAD DE CIENCIAS MATEMÁTICAS**

**ESCUELA PROFESIONAL DE COMPUTACIÓN CIENTÍFICA**



# ESTRUCTURADEDATOS

**TEMA: Listas Enlazadas**

**Profesor:** Benito Pacheco, Oscar

**Integrantes:** Velarde Pozo Luis Felipe

Huayta Huillcahuare Luis Enrique

**2024-I**

*Listo doble enlazadas*

consiste en una secuencia de nodos, donde cada nodo contiene un elemento de datos y dos enlaces que apuntan al nodo anterior y al siguiente en la secuencia. Esto permite el acceso tanto hacia adelante como hacia atrás en la lista.

***PSEUDOCÓDIGO:***

Se inicia el Programa

Definir la clase Nodo

Inicializar el método \_\_init\_\_ para asignar un dato al nodo

Definir la clase ListaDoblementeEnlazada

Inicializar el método \_\_init\_\_ para inicializar la lista con el primero y últimos elementos como None, y el tamaño de la lista en 0

Definir el método vacia para verificar si la lista está vacía

Definir el método agregar\_final para agregar un elemento al final de la lista

Definir el método agregar\_inicio para agregar un elemento al inicio de la lista

Definir el método recorrer\_inicio para recorrer la lista desde el principio

Definir el método recorrer\_final para recorrer la lista desde el final

Definir el método eliminar\_inicio para eliminar el primer elemento de la lista

Definir el método eliminar\_final para eliminar el último elemento de la lista

Iniciar el bucle principal

Mostrar el menú de opciones

Leer la opción ingresada por el usuario

Si la opción es 1, solicitar un número y agregarlo al final de la lista

Si la opción es 2, eliminar el último elemento de la lista

Si la opción es 3, imprimir "SI" si la lista está vacía, "NO" de lo contrario

Si la opción es 4, solicitar un número y agregarlo al inicio de la lista

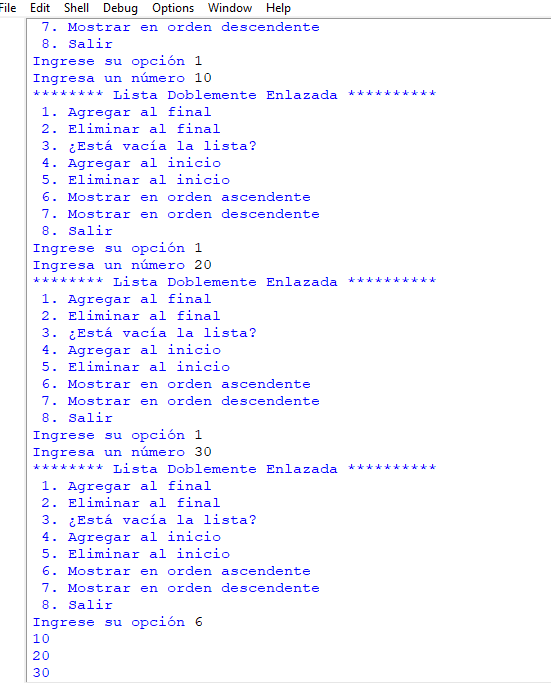
Si la opción es 5, eliminar el primer elemento de la lista

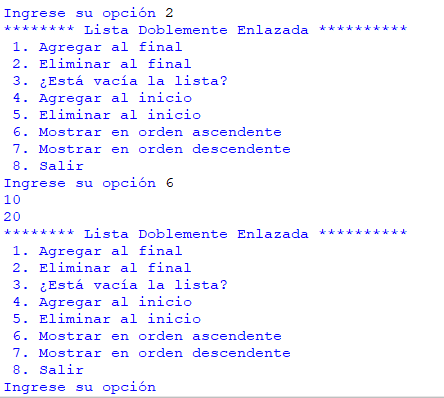
Si la opción es 6, mostrar los elementos de la lista en orden ascendente

Si la opción es 7, mostrar los elementos de la lista en orden descendente

Si la opción es 8, imprimir "sección finalizada" y salir del bucle principal

Fin del Programa

**PRUEBA DE ESCRITORIO:**



***Codigo Fuente:***

class Nodo:

def \_\_init\_\_(self, dato):

self.dato = dato

self.siguiente = None

self.anterior = None

class ListaDoblementeEnlazada:

def \_\_init\_\_(self):

self.primero = None

self.ultimo = None

self.tamañodelista = 0

def vacia(self):

return self.primero == None

def agregar\_final(self, dato):

if self.vacia():

self.primero = self.ultimo = Nodo(dato)

else:

encabezado = self.ultimo

self.ultimo = encabezado.siguiente = Nodo(dato)

self.ultimo.anterior = encabezado

self.tamañodelista += 1

def agregar\_inicio(self, dato):

if self.vacia():

self.primero = self.ultimo = Nodo(dato)

else:

encabezado = Nodo(dato)

encabezado.siguiente = self.primero

self.primero.anterior = None

self.primero = encabezado

self.tamañodelista += 1

def recorrer\_inicio(self):

encabezado = self.primero

while encabezado != None:

print(encabezado.dato)

encabezado = encabezado.siguiente

def recorrer\_final(self):

encabezado = self.ultimo

while encabezado:

print(encabezado.dato)

encabezado = encabezado.anterior

def eliminar\_inicio(self):

if self.vacia():

print("Lista vacía")

elif self.primero.siguiente == None:

self.primero = self.ultimo = None

self.tamañodelista = 0

else:

self.primero = self.primero.siguiente

self.primero.anterior = None

self.tamañodelista -= 1

def eliminar\_final(self):

if self.vacia():

print("Lista vacía")

elif self.primero.siguiente == None:

self.primero = self.ultimo = None

self.tamañodelista = 0

else:

self.ultimo = self.ultimo.anterior

self.ultimo.siguiente = None

self.tamañodelista -= 1

try:

if \_\_name\_\_ == "\_\_main\_\_":

opcion = 0

lista = ListaDoblementeEnlazada()

while opcion != 8:

print("\*\*\*\*\*\*\*\* Lista Doblemente Enlazada \*\*\*\*\*\*\*\*\*\*\n 1. Agregar al final\n 2. Eliminar al final\n 3. ¿Está vacía la lista?\n 4. Agregar al inicio\n 5. Eliminar al inicio\n 6. Mostrar en orden ascendente\n 7. Mostrar en orden descendente\n 8. Salir")

opcion = int(input("Ingrese su opción "))

if opcion == 1:

dato = input("Ingresa un número ")

lista.agregar\_final(dato)

elif opcion == 2:

lista.eliminar\_final()

elif opcion == 3:

print("SI" if lista.vacia() else "NO")

elif opcion == 4:

dato = input("Ingresa un número ")

lista.agregar\_inicio(dato)

elif opcion == 5:

lista.eliminar\_inicio()

elif opcion == 6:

lista.recorrer\_inicio()

elif opcion == 7:

lista.recorrer\_final()

elif opcion == 8:

print("sección finalizada")

else:

print("Ingresaste una opción errónea, vuelve a intentarlo.")

except Exception as e:

print(e)

**Listas Circulares**

Una lista circular es una lista lineal en la que el último nodo apunta al primero.

**PSEUDOCÓDIGO:**

**Datos de entrada:**

Rango: 1 al 14

Inserción: 111 y 999

Eliminación: Primer nodo y ultimo nodo

**Variables**

# Definir la estructura del nodo

Estructura Nodo:

dato

siguiente

# Inicializar la lista enlazada circular con un nodo inicial

lista = CircularLinkedList(crearNodo(1))

# Función para insertar un nodo en una posición específica

Procedimiento InsertarNodo(nodo, posicion):

Si posicion == 0:

ultimoElemento = BuscarUltimoElemento()

ultimoElemento.siguiente = nodo

nodo.siguiente = lista.cabeza

lista.cabeza = nodo

Sino Si posicion < 0:

ultimoElemento = BuscarUltimoElemento()

ultimoElemento.siguiente = nodo

nodo.siguiente = lista.cabeza

Sino:

elemento = BuscarElementoEnPosicion(posicion)

nodo.siguiente = elemento.siguiente

elemento.siguiente = nodo

# Función para eliminar un nodo en una posición específica

Procedimiento EliminarNodo(posicion):

Si posicion > 1:

anterior = BuscarElementoEnPosicion(posicion - 1)

Sino Si posicion < 0:

anterior = BuscarElementoEnPosicion(-1)[0]

Sino:

anterior = BuscarUltimoElemento()

lista.cabeza = anterior.siguiente.siguiente

anterior.siguiente = anterior.siguiente.siguiente

# Función para imprimir la lista enlazada circular

Procedimiento ImprimirListaCircular():

Imprimir(lista.cabeza.dato)

actual = lista.cabeza.siguiente

Mientras actual no es igual a lista.cabeza:

Imprimir(actual.dato)

actual = actual.siguiente

# Ejemplo de uso

Para i desde 2 hasta 14:

InsertarNodo(crearNodo(i), i)

EliminarNodo(1)

InsertarNodo(crearNodo(111), 0)

EliminarNodo(-1)

InsertarNodo(crearNodo(999), -1)

EliminarNodo(3)

ImprimirListaCircular()

**PRUEBA DE ESCRITORIO:**

Texto

Descripción generada automáticamente

CÓDIGO FUENTE:

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class CircularLinkedList:

def \_\_init\_\_(self, head):

self.head = head

self.head.next = head

def length(self):

current = self.head.next

count = 1

if self.head is not None:

while current != self.head:

count += 1

current = current.next

return count

else:

return 0

def search(self, position, prev=False):

anterior.

if position == 0:

if prev is False:

return self.head

else:

return [self.search(-1), self.head]

elif position < 0:

current = self.head

while current.next !=self.head:

previous = current

current = current.next

return current if prev is False else [previous, current]

else:

k = 1

current = self.head

while k < position and current.next !=self.head:

k += 1

previous = current

current = current.next

return current if prev is False else [previous, current]

def insert(self, node, position):

# Inserta el nodo después de la posición indicada. Si se indica una posición negativa, o mayor que

# el tamaño de la lista, se inserta al final

if position == 0:

last\_element = self.search(-1)

last\_element.next = node

node.next = self.head

self.head = node

elif position < 0:

last\_element = self.search(-1)

last\_element.next = node

node.next = self.head

else:

element = self.search(position)

node.next = element.next

element.next = node

def delete(self, position):

# Para posiciones con valor negativo, borra el último nodo

if position > 1:

previous = self.search(position-1)

elif position < 0:

previous = self.search(-1, True)[0]

else:

previous = self.search(-1)

self.head = previous.next.next

previous.next = previous.next.next

c\_list = CircularLinkedList(Node(1))

for i in range(2,15):

c\_list.insert(Node(i), i)

c\_list.delete(1)

c\_list.insert(Node(111), 0)

c\_list.delete(-1)

c\_list.insert(Node(999), -1)

c\_list.delete(3)

print(c\_list.head.data)

current = c\_list.head.next

while current is not c\_list.head:

print(current.data)

current = current.next