ANDERSSON DAVID SÁNCHEZ MÉNDEZ

**CONTROL DE LECTURA LINKED LIST**

Considerando una cola S, vacía en un principio. Ilustre el código de las siguientes operaciones:

[2] [2, 8] [2, 8, 11] 11 [2, 8, -3] [2, 8, -3, 7] 7 -3

S.push(2) → S.push(8) → S.push(11) → S. pop () → S. push (-3) → S. push (7) → S.pop() → S.pop()

S = [2, 8]

Considere una cola Q, vacía en un principio. Ilustre el código de las siguientes operaciones:

[4] [4, 17] [4, 17, 20] [4, 17, 20, 6] 4

Q.enqueue(4) → Q.enqueue(17) → Q.enqueue(20) → Q.enqueue(6) → Q.dequeue() →

17 [20, 6, -5] 20 Q = [6, -5]

Q.dequeue() → Q.enqueue(-5) → Q.dequeue()

```python
"""Considere una pila S y una cola Q, ambas vacías en un principio.Ilustre el paso a paso"""
from collections import deque

#a)

    #Ambas cadenas vacías
S = []
Q = deque([])

print("SECUENCIA S STACK")

    #S.push(2)
S.append(2)

    #S.push(8)
S.append(8)

    #S.push(11)
S.append(11)

    #S.pop()
print(S.pop())

    #S.push(-3)
S.append(-3)

    #S.push(7)
S.append(7)

    #S.pop()
print(S.pop())

    #S.pop()
print(S.pop())

print(S)

#b)

print("SECUENCIA Q QUEUE")

    #Q.enqueue(4)
Q.append(4)

    #Q.enqueue(17)
Q.append(17)

    #Q.enqueue(20)
Q.append(20)

    #Q.enqueue(6)
Q.append(6)

    #Q.enqueue()
print(Q.popleft())

    #Q.enqueue()
print(Q.popleft())

    #Q.enqueue(-5)
Q.append(-5)

    #Q.enqueue()
print(Q.popleft())

print(Q)
```

Desarrolle un algoritmo que identifique si una cadena de texto contiene una lista de paréntesis correctamente anidados y balanceados, por ejemplo:

- Correcto: [{(((())())()}]{[]}
- Correcto: (({}))[]
- Incorrecto: )([])(
- Incorrecto: (){[]([]

**Bonus**: identifique la posición del primer paréntesis mal ubicado en caso de que los paréntesis no estén correctamente anidados y balanceados.

```python
"""Desarrolle un algoritmo que identifique si una cadena de texto contiene una lista de paréntesi s
correctamente anidados y balanceados, por ejemplo:"""
# Importa la biblioteca requerida
from sys import stdin

def check_parentheses(s):
    # Define los pares de paréntesis
    combinaciones= {'(': ')', '[': ']', '{': '}'}

    # Inicializa una pila para llevar un registro de los paréntesis de apertura
    stack = []

    # Itera sobre los caracteres en la cadena
    for i, char in enumerate(s):
        # Si el carácter es un paréntesis de apertura, lo agrega a la pila
        if char in combinaciones:
            stack.append(char)
        # Si el carácter es un paréntesis de cierre
        elif char in combinaciones.values():
            # Si la pila está vacía o el elemento en la cima de la pila no coincide con el paréntesis de cierre
            if not stack or combinaciones[stack[-1]] != char:
                # Imprime "Incorrecto" y la posición del paréntesis mal ubicado
                print("Incorrecto, el primer paréntesis mal ubicado está en la posición",i,)
                return ""
            # Si el elemento en la cima de la pila coincide con el paréntesis de cierre, saca el elemento de la pila
            else:
                stack.pop()

    # Si la pila no está vacía después de iterar sobre la cadena, imprime "Incorrecto" y la posición del primer paréntesis de apertura que no se cerró
    if stack:
        print("Incorrecto, el primer paréntesis mal ubicado está en la posición" , s.index(stack[0]))
        return ""
    else:
        # Si la pila está vacía, imprime "Correcto"
        print("Correcto")
    return ""

# Lee múltiples líneas desde la consola
for line in stdin:
    # Elimina el carácter de nueva línea al final de la línea
    line = line.rstrip('\n')
    # Verifica los paréntesis en la línea
    check_parentheses(line)
```

Diseñe una función para invertir la dirección de una lista enlazada simple, es decir, una función que invierta todos los punteros entre los elementos de la lista. El algoritmo debe tener complejidad lineal O(n)

```python
from random import randint
from time import time
from uuid import uuid1

SIZE = int(2e6)

class Node:
    def __init__(self, value = None):
        self.value = None
        self.next = None
        self.setValue(value)

    def setValue(self, value):
        self.value = value

    def getValue(self):
        return self.value

    def getNext(self):
        return self.next

    def setNext(self, next):
        if not( isinstance(next, Node) or next is None):
            raise Exception("El tipo del atributo next no es del tipo de dato esperado.")
        self.next = next

    def __str__(self):
        return "Node({}) --> {}".format(self.value, self.next if self.next else "X")

class LinkedList:
    def __init__(self, elements = []):
        self.head = None
        self.tail = None
        self.len = 0
        for e in elements:
            self.append(e)

    def __len__(self):
        return self.len

    def setHead(self, node):
        if not isinstance(node, Node):
            raise Exception("El tipo del atributo head no es del tipo de dato esperado.")
        self.head = node

    def setTail(self, node):
        if not isinstance(node, Node):
            raise Exception("El tipo del atributo tail no es del tipo de dato esperado.")
        self.tail = node
        if self.tail is not None:
            self.tail.setNext(None)

    def append(self, value):
        new_node = Node(value)
        if self.isEmpty():
            self.setHead(new_node)
            self.setTail(new_node)
        else:
            tail = self.tail
            tail.setNext(new_node)
            self.setTail(new_node)
        self.len += 1

    def find(self, value):
        node_result = self.head
        while node_result is not None and node_result.getValue() != value:
            node_result = node_result.getNext()
        return node_result

    def update(self, old_value, new_value):
        node_to_update = self.find(old_value)
        if node_to_update is not None:
            node_to_update.setValue(new_value)

    def isEmpty(self):
        return self.head is None

    def __str__(self):
        return str(self.head)

    def join(self, ll2):
        if not isinstance(ll2, LinkedList):
            raise Exception("El tipo del parametro ll2 no es del tipo de dato esperado.")
        if self.isEmpty():
            return ll2
        if not ll2.isEmpty():
            self.tail.setNext(ll2.head)
            self.tail = ll2.tail
            self.len = self.len + len(ll2)
        return self

    def invertir(self):
        previo = None
        actual = self.head

        while actual is not None:
            siguiente = actual.getNext()
            actual.setNext(previo)
            previo = actual
            actual = siguiente

        self.head, self.tail = self.tail, self.head

def main():
    ll = LinkedList([1,2,3])
    print("Lista original:", ll)
    ll.invertir()
    print("Lista invertida:", ll)

    arr1 = [ uuid1() for i in range(SIZE)]
    arr2 = [ uuid1() for i in range(SIZE)]
    ll1 = LinkedList(arr1)
    ll2 = LinkedList(arr2)
    t0 = time()
    arr3 = arr1 + arr2
    t1 = time()
    print("Classic Join {}".format(t1-t0))
    t0 = time()
    ll3 = ll1.join(ll2)
    t1 = time()
    print("New Join {}".format(t1-t0))

main()
```

Modifique el código de la lista enlazada para que sea una doble lista enlazada, e implemente las siguientes funciones:

- Insertar un nuevo elemento.
- Eliminar un elemento dado su valor (considere el caso de borrar la cabeza de la lista)
- Eliminar los elementos duplicados
- Unir dos listas

```
class Node:
    def __init__(self, value=None):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, value):
        new_node = Node(value)
        if self.head is None:
            self.head = self.tail = new_node
        else:
            new_node.prev = self.tail
            self.tail.next = new_node
            self.tail = new_node

    def pop(self):
        if self.head is None:
            return None
        value = self.tail.value
        self.tail = self.tail.prev
        if self.tail is None:
            self.head = None
        else:
            self.tail.next = None
        return value

    def peek(self):
        return self.tail.value if self.tail else None

    def popleft(self):
        if self.head is None:
            return None
        value = self.head.value
        self.head = self.head.next
        if self.head is None:
            self.tail = None
        else:
            self.head.prev = None
        return value

class Stack:
    def __init__(self):
        self.list = DoublyLinkedList()

    def push(self, value):
        self.list.append(value)

    def pop(self):
        return self.list.pop()

    def peek(self):
        return self.list.peek()

class Queue:
    def __init__(self):
        self.list = DoublyLinkedList()

    def enqueue(self, value):
        self.list.append(value)

    def dequeue(self):
        return self.list.popleft()

    def peek(self):
        return self.list.head.value if self.list.head else None
def main():
    # Prueba de la pila
    print("Stack's test:")
    stack = Stack()
    stack.push(1)
    stack.push(2)
    stack.push(3)
    print(stack.pop())
    print(stack.peek())

    # Prueba de la cola
    print("\nQueue's test:")
    queue = Queue()
    queue.enqueue(1)
    queue.enqueue(2)
    queue.enqueue(3)
    print(queue.dequeue())
    print(queue.peek())

main()
```

Fuentes

/stackQueue.py

/parenthesis.py /inputParenthesis.txt /salidaParenthesis.txt

/reverseLinkedList.py

/doubleLinkedList.py

/myProduct #Caso de negocio.