## Topics

1. Implement Node Class

class Node:

def \_\_init\_\_(self, element, next\_node=None):

self.element = element

self.next = next\_node

1. Implement CircularlyLinkedList Class

class CircularlyLinkedList:

class \_Node:

def \_\_init\_\_(self, element, next\_node=None):

self.element = element

self.next = next\_node

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

self.\_tail = None

self.\_size = 0

def \_\_len\_\_(self):

return self.\_size

def is\_empty(self):

return self.\_size == 0

def first(self):

if self.is\_empty():

raise ValueError("The list is empty")

return self.\_tail.next.element

def last(self):

if self.is\_empty():

raise ValueError("The list is empty")

return self.\_tail.element

def add\_first(self, e):

newest = self.\_Node(e)

if self.is\_empty():

newest.next = newest

self.\_tail = newest

else:

newest.next = self.\_tail.next

self.\_tail.next = newest

self.\_size += 1

def add\_last(self, e):

self.add\_first(e)

self.\_tail = self.\_tail.next

def remove\_first(self):

if self.is\_empty():

raise ValueError("The list is empty")

head = self.\_tail.next

if self.\_size == 1:

self.\_tail = None

else:

self.\_tail.next = head.next

self.\_size -= 1

return head.element

def rotate(self):

if self.\_size > 0:

self.\_tail = self.\_tail.next

def \_\_iter\_\_(self):

current = self.\_tail.next

while current is not self.\_tail:

yield current.element

current = current.next

yield self.\_tail.element

1. Implement Basic Methods of CircularlyLinkedList

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()
* rotate()class CircularlyLinkedList:
* class \_Node:
* def \_\_init\_\_(self, element, next\_node=None):
* self.element = element
* self.next = next\_node
* def \_\_init\_\_(self):
* self.\_tail = None
* self.\_size = 0
* def is\_empty(self):
* return self.\_size == 0
* def size(self):
* return self.\_size
* def first(self):
* if self.is\_empty():
* raise ValueError("The list is empty")
* return self.\_tail.next.element
* def last(self):
* if self.is\_empty():
* raise ValueError("The list is empty")
* return self.\_tail.element
* def add\_first(self, e):
* newest = self.\_Node(e)
* if self.is\_empty():
* newest.next = newest
* self.\_tail = newest
* else:
* newest.next = self.\_tail.next
* self.\_tail.next = newest
* self.\_size += 1
* def add\_last(self, e):
* self.add\_first(e)
* self.\_tail = self.\_tail.next
* def remove\_first(self):
* if self.is\_empty():
* raise ValueError("The list is empty")
* head = self.\_tail.next
* if self.\_size == 1:
* self.\_tail = None
* else:
* self.\_tail.next = head.next
* self.\_size -= 1
* return head.element
* def rotate(self):
* if self.\_size > 0:
* self.\_tail = self.\_tail.next

## Homework

1. Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable.

def addFirst(self, e):

newest = self.\_Node(e, None) # إنشاء عقدة جديدة

if self.\_size == 0:

newest.next = newest # العقدة الوحيدة تشير إلى نفسها

self.\_tail = newest # تحديث العقدة الأخيرة

else:

newest.next = self.\_tail.next # العقدة الجديدة تشير إلى العقدة الأولى

self.\_tail.next = newest # تحديث العقدة الأخيرة

self.\_size += 1 # زيادة حجم القائم

1. Give an implementation of the size( ) method for the CircularlyLinkedList class, assuming that we did not maintain size as an instance variable.

def size(self):

count = 0

if self.\_tail is not None:

count = 1

current = self.\_tail.next

while current != self.\_tail:

count += 1

current = current.next

return count

1. Implement the equals( ) method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list.

def equals(self, other):

if self.\_size != other.\_size: # التحقق من أن القوائم لديها نفس الحجم

return False

if self.\_tail is None and other.\_tail is None: # التحقق من أن القوائم فارغة

return True

current\_self = self.\_tail.next

current\_other = other.\_tail.next

while current\_self != self.\_tail.next: # يتم التحقق من تطابق العناصر

if current\_self.element != current\_other.element:

return False

current\_self = current\_self.next

current\_other = current\_other.next

return True

1. Suppose you are given two circularly linked lists, L and M. Describe an algorithm for telling if L and M store the same sequence of elements (but perhaps with different starting points).

خوارزمية لمقارنة تسلسل العناصر في قوائم متجانستين (L و M):

يمكننا استخدام خوارزمية بسيطة للتحقق مما إذا كانت القوائم L و M تحتوي على نفس تسلسل العناصر بدءًا من نقاط البداية المختلفة. يمكن القيام بذلك عن طريق إجراء مقارنة تسلسلية بين العناصر في القوائم والتأكد من تطابقها. هنا خوارزمية تنفيذ هذا الفكر:

- قم بتحديد نقطة البداية في القائمة L.

- قم بتحديد نقطة البداية في القائمة M.

- قم بتنفيذ عملية المقارنة التسلسلية بين العناصر في القوائم L و M.

- إذا تطابقت جميع العناصر في القائمتين بنفس الترتيب، فإن القوائم تحتوي على نفس التسلسل.

5. طريقة لتقسيم قائمة متجانسة دائرية L إلى قائمتين متجانستين حجم كل منهما يساوي نصف حجم L:

1. Given a circularly linked list L containing an even number of nodes, describe how to split L into two circularly linked lists of half the size.

def splitList(self):

if self.\_size % 2 != 0:

return "قائمة L يجب أن تحتوي على عدد زوجي من العناصر"

half\_size = self.\_size // 2

current = self.\_tail.next

count = 0

new\_list = CircularlyLinkedList()

while count < half\_size:

new\_list.addLast(current.element)

current = current.next

count += 1

self.\_tail.next = current

self.\_size -= half\_size

new\_list.\_tail.next = current

new\_list.\_size = half\_size

return new\_list

1. Implement the clone( ) method for the CircularlyLinkedList class.

def clone(self):

if self.\_tail is None:

return CircularlyLinkedList()

current = self.\_tail.next

new\_list = CircularlyLinkedList()

while current != self.\_tail.next:

new\_list.addLast(current.element)

current = current.next

return new\_list