Topics

* Implement Node Class
* Implement DoublyLinkedList Class
* Implement Basic Methods of DoublyLinkedList
* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()
* removeLast()

Homework

* Describe a method for finding the middle node of a doubly linked list with header and trailer sentinels by “link hopping,” and without relying on explicit knowledge of the size of the list. In the case of an even number of nodes, report the node slightly left of center as the “middle.”

استخدم أسلوب التنقل عبر القفز (link hopping) بالخطوات التالية:

استخدم مؤشرين:

واحد يبدأ من البداية (header.next).

والآخر يبدأ من النهاية (trailer.prev).

استمر بتحريك المؤشرين خطوة واحدة في كل مرة:

الأول للأمام.

الثاني للخلف.

توقف عندما يتقاطع المؤشران أو يتقابلان عند نفس العقدة.

def findMiddle(self):

if self.isEmpty():

return None

start = self.header.next

end = self.trailer.prev

while start != end and start.next != end:

start = start.next

end = end.prev

return start

----------------------------------------------------------------------------------------

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

def size(self):

count = 0

current = self.header.next

while current != self.trailer:

count += 1

current = current.next

return count

------------------------------------------------------------------------------------

* Implement the equals( ) method for the DoublyLinkedList class.

def equals(self, other):

if self.size() != other.size():

return False

current1 = self.header.next

current2 = other.header.next

while current1 != self.trailer:

if current1.data != current2.data:

return False

current1 = current1.next

current2 = current2.next

return True

----------------------------------------------------------------------------------------

* Give an algorithm for concatenating two doubly linked lists L and M, with header and trailer sentinel nodes, into a single list L′.

def concatenate(L, M):

if L.isEmpty():

return M

if M.isEmpty():

return L

L.trailer.prev.next = M.header.next

M.header.next.prev = L.trailer.prev

L.trailer = M.trailer

return L

-----------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--------------------------------

* Our implementation of a doubly linked list relies on two sentinel nodes, header and trailer, but a single sentinel node that guards both ends of the list should suffice. Reimplement the DoublyLinkedList class using only one sentinel node.

class DoublyLinkedList:

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

self.sentinel = Node(None)

self.sentinel.next = self.sentinel

self.sentinel.prev = self.sentinel

----------------------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*----------------------

* Implement a circular version of a doubly linked list, without any sentinels, that supports all the public behaviors of the original as well as two new update methods, rotate( ) and rotateBackward.

class CircularDoublyLinkedList:

def rotate(self):

if not self.isEmpty():

self.tail = self.head

self.head = self.head.next

def rotateBackward(self):

if not self.isEmpty():

self.head = self.tail

self.tail = self.tail.prev

----------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-------------------------

* Implement the clone( ) method for the DoublyLinkedList class.

def clone(self):

cloned\_list = DoublyLinkedList()

current = self.header.next

while current != self.trailer:

cloned\_list.addLast(current.data)

current = current.next

return cloned\_list