Topics

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

Homework

* develop an implementation of the equals method in the context of the SinglyLinkedList class.

def equals(self, other):

a, b = self.head, other.head

while a and b:

if a.data != b.data:

a = a.next

b = b.next

return a is None and b is None

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

* Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.

def secondToLast(self):

if self.head is None or self.head.next is None:

None return

current = self.head

while current.next and current.next.next:

current = current.next

return current.data

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

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

def size(self):

count = 0

current = self.head

while current:

count += 1

current = current.next

return count

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

* Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

def rotate(self):

if self.head is None or self.head.next is None:

return

old\_head = self.head

self.head = old\_head.next

current = self.head

while current.next:

current = current.next

current.next = old\_head

old\_head.next = None

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

* Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.

def concatenate(self, other):

if self.head is None:

head.other = head.self

else:

current = self.head

while current.next:

current = current.next

current.next = other.head

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

* Describe in detail an algorithm for reversing a singly linked
* list L using only a constant amount of additional space.

def reverse(self):

prev = None

current = self.head

while current:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

self.head = prev