#Name: Shihab Muhtasim

#ID:21301610

#sec:01

class CircularArray:

def \_\_init\_\_(self, lin, st, sz):

# Initializing Variables

self.start = st

self.size = sz

self.cir = [None]\*len(lin)

k=self.start

for i in range(len(lin)):

self.cir[k]=lin[i]

k=(k+1)%len(self.cir)

# (TASK 1) Print from index 0 to len(cir) - 1

def printFullLinear(self): #Easy

for i in range(0, len(self.cir)):

if i!=len(self.cir)-1:

print(self.cir[i],end=", ")

else:

print(self.cir[i])

# (TASK 2) Print from start index and total size elements

def printForward(self): #Easy

k=self.start

for i in range(self.size):

if i!=self.size-1:

print(self.cir[k],end=", ")

else:

print(self.cir[k])

k=(k+1)%len(self.cir)

#TASK 3:

def printBackward(self): #Easy

k=(self.start+self.size-1)%len(self.cir)

for i in range(self.size):

if i!=self.size-1:

print(self.cir[k],end=", ")

else:

print(self.cir[k])

k=(k-1)

if k==-1:

k=len(self.cir)-1

# TASK 4 With no null cells

def linearize(self): #Medium

create\_lin=[None]\*self.size

k=self.start

for i in range(self.size):

create\_lin[i]=self.cir[k]

k=(k+1)%len(self.cir)

self.cir=create\_lin

# TASK 5: Do not change the Start index

def resizeStartUnchanged(self, newcapacity): #Medium #

create\_cir=[None]\*newcapacity

k=self.start

nst=self.start

for i in range(self.size):

create\_cir[nst]=self.cir[k]

k=(k+1)%len(self.cir)

nst=(nst+1)%len(create\_cir)

self.cir=create\_cir

# TASK 6: This method will check whether the array is palindrome or not

def palindromeCheck(self): #Hard

start= self.start

end=(self.start+self.size-1)%len(self.cir)

for i in range(self.size//2):

if self.cir[start]==self.cir[end]:

flag=True

else:

flag=False

break

start=(start+1)%len(self.cir)

end=(end-1)

if end==-1:

end=len(self.cir)-1

if flag==True:

print('This array is a palindrome')

else:

print("This array is NOT a palindrome")

# This method will sort the values by keeping the start unchanged

def sort(self):

st=self.start

for i in range(self.size-1):

check=(st+1)%len(self.cir)

for j in range(i+1,self.size):

if self.cir[st]>self.cir[check]:

self.cir[st],self.cir[check]=self.cir[check],self.cir[st]

check=(check+1)%len(self.cir)

st=(st+1)%len(self.cir)

return self.cir

# This method will check the given array across the base array and if they are equivalent interms of values return true, or else return false

def equivalent(self, cir\_arr):

temp=[None]\*len(self.cir)

s=self.start

for i in range(self.size):

temp[s]=self.cir[s]

s=(s+1)%len(self.cir)

sorted\_1= self.sort()

sorted\_2= cir\_arr.sort()

self.linearize()

cir\_arr.linearize()

if len(self.cir)!=len(cir\_arr.cir):

flag=False

else:

for i in range(len(self.cir)):

if self.cir[i]==cir\_arr.cir[i]:

flag=True

else:

flag=False

break

self.cir=temp

return flag

# the method take another circular array and returns a linear array containing the common elements between the two circular arrays.

def intersection(self,c2):

self.linearize()

c2.linearize()

arr\_1=self.cir

arr\_2=c2.cir

count=0

for i in range(len(arr\_1)):

for j in range(len(arr\_2)):

if arr\_1[i]==arr\_2[j]:

count+=1

new=[None]\*count

idx=0

for i in range(len(arr\_1)):

for j in range(len(arr\_2)):

if arr\_1[i]==arr\_2[j]:

new[idx]=arr\_1[i]

idx+=1

return new

#.................................................................

lin\_arr1 = [10, 20, 30, 40, None]

print("==========Test 1==========")

c1 = CircularArray(lin\_arr1, 2, 4)

c1.printFullLinear() # This should print: 40, None, 10, 20, 30

c1.printForward() # This should print: 10, 20, 30, 40

c1.printBackward() # This should print: 40, 30, 20, 10

print("==========Test 2==========")

c1.linearize()

c1.printFullLinear() # This should print: 10, 20, 30, 40

print("==========Test 3==========")

lin\_arr2 = [10, 20, 30, 40, 50]

c2 = CircularArray(lin\_arr2, 2, 5)

c2.printFullLinear() # This should print: 40, 50, 10, 20, 30

c2.resizeStartUnchanged(8) # parameter --> new Capacity

c2.printFullLinear() # This should print: None, None, 10, 20, 30, 40, 50, None

print("==========Test 4==========")

lin\_arr3 = [10, 20, 30, 20, 10, None, None]

c3 = CircularArray(lin\_arr3, 3, 5)

c3.printForward() # This should print: 10, 20, 30, 20, 10

c3.palindromeCheck() # This should print: This array is a palindrome

print("==========Test 5==========")

lin\_arr4 = [10, 20, 30, 20, None, None, None]

c4 = CircularArray(lin\_arr4, 3, 4)

c4.printForward() # This should print: 10, 20, 30, 20

c4.palindromeCheck() # This should print: This array is NOT a palindrome

print("==========Test 6==========")

lin\_arr5 = [10, 20, -30, 20, 50, 30, None]

c5 = CircularArray(lin\_arr5, 5, 6)

c5.printForward() # This should print: 10, 20, -30, 20, 50, 30

c5.sort()

c5.printForward() # This should print: -30, 10, 20, 20, 30, 50

print("==========Test 7==========")

lin\_arr6 = [10, 20, -30, 20, 50, 30, None]

c6 = CircularArray(lin\_arr6, 2, 6)

c7 = CircularArray(lin\_arr6, 5, 6)

c6.printForward() # This should print: 10, 20, -30, 20, 50, 30

c7.printForward() # This should print: 10, 20, -30, 20, 50, 30

print(c6.equivalent(c7)) # This should print: True

print("==========Test 8==========")

lin\_arr7 = [10, 20, -30, 20, 50, 30, None, None, None]

c8 = CircularArray(lin\_arr7, 8, 6)

c6.printForward() # This should print: 10, 20, -30, 20, 50, 30

c8.printForward() # This should print: 10, 20, -30, 20, 50, 30

print(c6.equivalent(c8)) # This should print: True

print("==========Test 9==========")

lin\_arr8 = [10, 20, 30, 40, 50, 60, None, None, None]

c9 = CircularArray(lin\_arr8, 8, 6)

c6.printForward() # This should print: 10, 20, -30, 20, 50, 30

c9.printForward() # This should print: 10, 20, 30, 40, 50, 60

print(c6.equivalent(c9)) # This should print: False

print("==========Test 10==========")

lin\_arr9 = [10, 20, 30, 40, 50, None, None, None]

c10 = CircularArray(lin\_arr9, 5, 5)

c10.printFullLinear() # This should print: 40, 50, None, None, None, 10, 20, 30

lin\_arr10 = [5, 40, 15, 25, 10, 20, 5, None, None, None, None, None]

c11 = CircularArray(lin\_arr10, 8, 7)

c11.printFullLinear() # This should print: 10, 20, 5, None, None, None, None, None, 5, 40, 15, 25

output = c10.intersection(c11)

print(output) # This should print: [10, 20, 40]