Assignment 2 CECS 274

1. It’s not efficient to implement add\_all(i, c) with repeated calls of add(i,x) because it takes up more time and uses more calls to change up the position of adding every time. For the length of the collection, you would have to call add(i,x) that amount of times to change the position for each call. Rather than repeatedly calling it, you can have an array list with resize so that if the number of elements is the same as the length of the array and increases it size by 2^n so that it is not a constant resize.

If n+m > len(a):

Resize()

1. import random

class RandomQueue:

   def \_\_init\_\_(self):

       Self.x = []

       self.count = 0

   def add(self, x):

       self.x.append(item)

       self.count += 1

   def remove(x):

       if x.count == 0:

           return None

       randIndex = random.randint(0, self.count-1)

       element = self.items[randIndex]

       self.x[randIndex], self.x[-1] = self.x[-1], self.x[randIndex]

       self.x.pop()

       self.count -= 1

       return element

   def isEmpty(self):

       return self.count == 0

q = RandomQueue()

for i in range(1,11):

   q.add(i)

while not q.isEmpty():

   print(q.remove())

1. import sys

from \_ctypes import resize

from sys import maxsize

import re

class ArrayQueue:

   def \_\_init\_\_(self):

       self.a = self.new\_array(1)

       self.n = 0

       self.j = 0

   def new\_array(self, m):

       self.stack = []

       for i in range(m):

           self.stack.append(0)

       return self.stack

   def add(self, xLab ):

       if (self.n+1) == len(self.a):

           resize()

           self.a[(self.j+self.n)%len(self.a)] = x

           self.n += 1

           return x

   def rotate(self, a, r):

b=[]

n=len(a)

for i in range(n-r,n):

b.append(a[i])

for i in range(0,n-r):

b.append(a[i])

print(b)

a=[]

a = list(map(int, input("Enter elements: ").split()))

r=int(input("Enter r: "))

n=len(a)

r=r%n

   def remove(self):

       x = self.a[self.j]

       j = (self.j + 1)%len(self.a)

       self.n -= 1

       if len(self.a) >= (3 \* self.n):

           resize()

       return x

   def resize(self):

       b = self.new\_array(max(1, 2 \* self.n))

       for k in range(0, self.n-1):

           b[k] = self.a[(self.j+k)%len(self.a)]

       a = b

       j = 0

   def isEmpty(self):

       return self.n == 0

1. class SLLIST():

   def \_\_init\_\_(self):

       self.head = None

       self.tail = None

       self.l = 0

       self.next = None

       self.value = None

   def add(self, x):

       y = self.node(x)

       if self.l == 0:

           self.head = y

           self.tail = self.head

       self.tail.setChild(y)

       self.tail = y

       self.l += 1

   def secLast(self):

       test = self.head

       found = False

       while found == False:

           if test.next.hasChild() == True:

               test = test.next

           else:

               found = True

               return test.value

   def node(self, x):

       self.value = x

   def setChild(self, y):

       self.next = y

   def hasChild(self):

       if (next != None):

           return True

       else:

           return False

1. def get(i):

Return a[i]

def set(i,x):

a[i] = x

return a

def add(i,x):

a.insert(i,x)

return a

def remove(i):

del at[i]

return a

1. def reverse()

  prev = None

  curr = head

  next = None

  while curr != None:

      next = curr.next

      curr.next=prev

      prev=curr

      curr = next

  head = prev

1. def is\_palindrome(l):

   mid = int(len(l) / 2)

   for i in range(mid):

    if l[i] != l[len(l) - i -1]:

    return False

   return True

1. initialize k = head;

move k forward (n - r ) times, k point to spliced out element

k -> last -> next = None

k -> last = None

temp\_head = k

now move k forward (r-1) times to get last element

k -> next = head

head -> last = k

head = temp\_head

1. import math

class Element:

   def Element(value, min):

       value = value

       min = min;

class MinStack:

   def \_\_init\_\_(self):

       self.top = None

   def push(self, x):

       if self.top == None:

           self.top = Element(x, x)

       else:

           e = Element(x, math.min(x, self.top.min))

           e.next = self.top

           self.top = e

   def pop(self):

       if (self.top == None):

           return

           temp = top.next

           top.next = null

           top = temp

   def top(self):

       if (self.top == None):

           return -1

       return self.top.value

   def findMin(self):

       if (self.top == None):

           return -1

       return self.top.min

1. X XOR Y does not make a good hashcode for your object because since two objects can have the same hashcode then x xor y would not be possible because x xor y makes it so that two objects cannot be equal. X XOR Y is only true if x and y are not the same.

|  |  |  |
| --- | --- | --- |
| x | y | a |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |