**Data Structures**

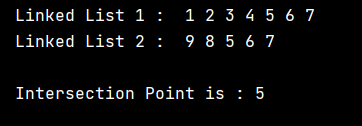
**S Abhishek**

**AM.EN.U4CSE19147**

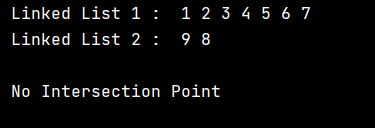
1. Find the merge point of two linked lists.

class Node:  
 def \_\_init\_\_(self**,** data):  
 self.data = data  
 self.next = None  
  
  
def display(n):  
 temp = n  
  
 while temp:  
 print(temp.data**,** end=" ")  
 temp = temp.next  
 print()  
  
  
def intersection(n1**,** n2):  
 temp1 = n1  
  
 while temp1 is not None:  
 temp2 = n2  
 while temp2 is not None:  
 if temp1 == temp2:  
 print("\nIntersection Point is : {}".format(temp1.data))  
 return  
 else:  
 temp2 = temp2.next  
  
 temp1 = temp1.next  
  
 print("\nNo Intersection Point")  
  
  
head1 = Node(**1**)  
  
nod1 = Node(**2**)  
nod2 = Node(**3**)  
nod3 = Node(**4**)  
nod4 = Node(**5**)  
nod5 = Node(**6**)  
nod6 = Node(**7**)  
nod7 = Node(**8**)  
  
head1.next = nod1  
head1.next.next = nod2  
head1.next.next.next = nod3  
head1.next.next.next.next = nod4  
head1.next.next.next.next.next = nod5  
head1.next.next.next.next.next.next = nod6  
  
head2 = Node(**9**)  
head2.next = nod7  
head2.next.next = nod4  
head2.next.next.next = nod5  
head2.next.next.next.next = nod6

print("Linked List 1 : "**,** end=" ")  
display(head1)  
print("Linked List 2 : "**,** end=" ")  
display(head2)  
  
intersection(head1**,** head2)

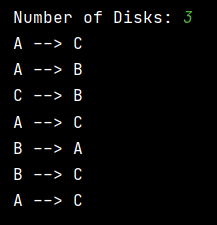


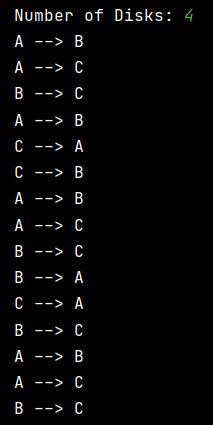
head2 = Node(**9**)  
head2.next = nod7  
#head2.next.next = nod4  
#head2.next.next.next = nod5  
#head2.next.next.next.next = nod6



2. Create an n-disc towers of Hanoi. Move all the discs from tower A to tower C.

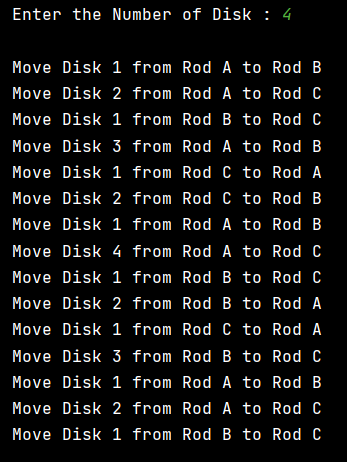
class Hanoi:  
  
 def \_\_init\_\_(self**,** label):  
 self.data = list()  
 self.label = label  
  
 def is\_empty(self):  
 return len(self.data) == **0** def push(self**,** e):  
 self.data.append(e)  
  
 def top(self):  
 return self.data[len(self.data) - **1**]  
  
 def pop(self):  
 return self.data.pop()  
  
  
def traversal(rod\_A**,** rod\_B):  
 if rod\_A.is\_empty():  
 rod\_A.push(rod\_B.pop())  
 print(rod\_B.label**,** "-->"**,** rod\_A.label)  
 elif rod\_B.is\_empty():  
 rod\_B.push(rod\_A.pop())  
 print(rod\_A.label**,** "-->"**,** rod\_B.label)  
 elif rod\_B.top() < rod\_A.top():  
 rod\_A.push(rod\_B.pop())  
 print(rod\_B.label**,** "-->"**,** rod\_A.label)  
 else:  
 rod\_B.push(rod\_A.pop())  
 print(rod\_A.label**,** "-->"**,** rod\_B.label)  
  
  
def HanoiTower(n):  
 rod\_A = Hanoi("A")  
 rod\_B = Hanoi("B")  
 rod\_C = Hanoi("C")  
  
 min = (**2** \*\* n) - **1** for obj in range(n**, 0,** -**1**):  
 rod\_A.push(obj)  
  
 if n % **2** == **0**:  
 rod\_B**,** rod\_C = rod\_C**,** rod\_B  
  
 for i in range(min):  
 if i % **3** == **0**:  
 traversal(rod\_A**,** rod\_C)  
 if i % **3** == **1**:  
 traversal(rod\_A**,** rod\_B)  
 if i % **3** == **2**:  
 traversal(rod\_B**,** rod\_C)  
  
  
temp = int(input("Number of Disks: "))  
if temp == **0**:  
 print("No Disk")  
else:  
 HanoiTower(temp)

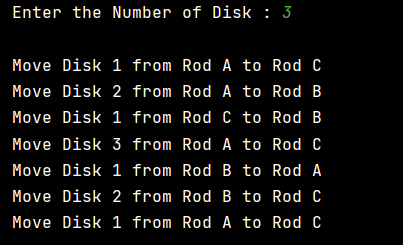




3. Solve towers of hanoi using recursion

def hanoi(n**,** source**,** destination**,** dummy):  
 if n == **1**:  
 print("Move Disk 1 from Rod"**,** source**,** "to Rod"**,** destination)  
 return  
 hanoi(n - **1,** source**,** dummy**,** destination)  
 print("Move Disk"**,** n**,** "from Rod"**,** source**,** "to Rod"**,** destination)  
 hanoi(n - **1,** dummy**,** destination**,** source)  
  
  
n\_rod = int(input("Enter the Number of Disk : "))  
print()  
hanoi(n\_rod**,** 'A'**,** 'C'**,** 'B')





4. Reverse a stack using recursion.

def push(s**,** item):  
 s.append(item)  
 return s  
  
  
def display(s1):  
 print("\nStack is "**,** end="")  
 for i in range(len(s1) - **1,** -**1,** -**1**):  
 print(s1[i]**,** end=' ')  
 print()  
  
  
def isempty(s2):  
 return len(s2) == **0**def pop(s3):  
 if isempty(s3):  
 print("Stack Underflow")  
 return  
 return s3.pop()  
  
  
def insertAtBottom(s5**,** item):  
 if isempty(s5):  
 push(s5**,** item)  
 else:  
 temp = pop(s5)  
 insertAtBottom(s5**,** item)  
 push(s5**,** temp)  
  
  
def reverse(s4):  
 if not isempty(s4):  
 temp = pop(s4)  
 reverse(s4)  
 insertAtBottom(s4**,** temp)  
  
  
stack = []  
  
n = int(input("Enter the Number of Elements : "))  
  
for j in range(n):  
 m = int(input("Enter the Element {} : ".format(j + **1**)))  
 push(stack**,** m)  
  
display(stack)  
  
# pop(stack)  
# pop(stack)  
  
reverse(stack)  
  
display(stack)

