Islamic University of Technology

CSE 4810: Algorithm Engineering Lab

Lab 1 Report

Submitted by:
Name: Sidratul Tanzila Tasmi
ID: 190041138
Group: 1B
Department of Computer Science and Engineering
Islamic University of Technology
Board Bazar, Gazipur
Email: sidratultanzila@iut-dhaka.edu

February 12, 2024

1 Task 1

1.1 Implementing 2 stacks in one array

```
    class TwoStackOneArray:
  def __init__(self,no_of_elements):
    self.n=no_of_elements
    self.array=[0]*self.n
    self.mid_point=self.n//2
    self.top1= self.mid_point
    self.top2= self.mid_point-1
  def push1(self, element):
    if self.top1<0:
      print("Cannot Push Any Element")
      return
    print("Previous Index of Top 1 is: {}".format(self.top1))
    self.top1= self.top1-1
    self.array[self.top1]=element
    print("New Array is:\n")
    print(self.array)
    print("New Index of Top 1 is: {}\n".format(self.top1))
  def pop1(self):
    if self.top1 > self.mid_point:
      print("Cannot Pop!")
      return
    print("Previous Index of Top 1 is: {}".format(self.top1))
    print(self.array[self.top1])
    self.array[self.top1]=0
    self.top1=self.top1+1
    print("New Array After Popping:\n")
    print(self.array)
    print("New Index of Top 1 is: {}\n".format(self.top1))
  def push2(self, element):
    if self.top2< self.mid_point-1:
      print("Cannot Push Any Element")
      return
    print("Previous Index of Top 2 is: {}".format(self.top2))
    self.top2= self.top2+1
    self.array[self.top2]=element
    print("New Array is:\n")
    print(self.array)
    print("New Index of Top 2 is: {}\n".format(self.top2))
  def pop2(self):
    if self.top2 < (self.n-1)//2:
      print("Cannot Pop!")
      return
    print("Previous Index of Top 2 is: {}".format(self.top2))
    print(self.array[self.top2])
    self.array[self.top2]=0
    print("New Array After Popping:\n")
    print(self.array)
    self.top2=self.top2-1
    print("New Index of Top 2 is: {}\n".format(self.top2))

object1=TwoStackOneArray(9)
object1.push1(96)
```

```
object1.push2(138)
object1.push1(105)
object1.push2(116)
object1.push1(108)
object1.pop1()
object1.pop2()
object1.pop1()
```

Output is:

```
    Previous Index of Top 1 is: 4
New Array is:

[0, 0, 0, 96, 0, 0, 0, 0, 0]
New Index of Top 1 is: 3

Previous Index of Top 2 is: 3
New Array is:

[0, 0, 0, 96, 138, 0, 0, 0, 0]
New Index of Top 2 is: 4

Previous Index of Top 1 is: 3
New Array is:

[0, 0, 105, 96, 138, 0, 0, 0, 0]
New Index of Top 1 is: 2

Previous Index of Top 2 is: 4
New Array is:

[0, 0, 105, 96, 138, 116, 0, 0, 0]
New Index of Top 2 is: 5

Previous Index of Top 1 is: 2
New Array is:

[0, 108, 105, 96, 138, 116, 0, 0, 0]
New Index of Top 1 is: 1

Previous Index of Top 1 is: 1
108
New Array After Popping:

[0, 0, 105, 96, 138, 116, 0, 0, 0]
New Index of Top 1 is: 2

Previous Index of Top 2 is: 5
116
New Array After Popping:

[0, 0, 105, 96, 138, 0, 0, 0, 0]
New Index of Top 2 is: 4

Previous Index of Top 1 is: 2
105
New Array After Popping:
```

```
[0, 0, 0, 96, 138, 0, 0, 0, 0]
New Index of Top 1 is: 3
```

Time Complexity: O(1), constant because all the operations took constant time

1.2 Implement Stack Using Queue

```
    from queue import Queue
class StackUsingQueue:
  def __init__(self):
    self.q=[]
    self.idx=-1
  def push(self, element):
    q1=self.q
    q2=[]
    q2.append(element)
    for i in range(0, len(q1)):
      q2.append(q1[i])
    self.q=q2
    print("New Array after pushing {} is:".format(element))
    print(self.q)
    self.idx= self.idx+1
  def empty(self):
    print("Checking if the stack is empty: True/False")
    if self.q:
      print("Stack is not empty!")
    else:
      print("Stack is empty!")
  def pop(self):
    if self.idx<0:
      print("Nothing to pop!")
    #print(self.q[self.idx])
    print("Based on LIFO, last element popped is: {}".format(self.q[0]))
    del self.q[0]
    self.idx=self.idx-1

object2= StackUsingQueue()
object2.push(3)
object2.push(4)
object2.push(5)
object2.push(6)
object2.empty()
object2.pop()
object2.push(7)
object2.pop()
```

Output:

```
    New Array after pushing 3 is:
[3]
New Array after pushing 4 is:
[4, 3]
New Array after pushing 5 is:
[5, 4, 3]
New Array after pushing 6 is:
[6, 5, 4, 3]
```

```
Checking if the stack is empty: True/False
Stack is not empty!
Based on LIFO, last element popped is: 6
New Array after pushing 7 is:
[7, 5, 4, 3]
Based on LIFO, last element popped is: 7
```

Time Complexity: O(n) because we are iterating through a loop to append value from queue q1 to q2.

1.3 To reverse a linkedlist using stack

```
    class Node:
  def __init__(self, value):
    self.value = value
    self.next = None
class LinkedList:
  def __init__(self):
    self.head=Node(None)
  def insertNode(self, value):
    new= Node(value)
    print("Newly inserted element in the linked list is: {}".format(value))
    if self.head.value==None:
      self.head= new
      return
    iterate=self.head
    while iterate.next!=None:
      iterate= iterate.next
    iterate.next=new
  def printLinkedList(self):
    iterate=self.head
    print("Elements of the linked list are:")
    while iterate!=None:
      print(iterate.value)
      iterate=iterate.next
  def getLinkedListinStack(self):
    arr=[]
    iterate=self.head
    while iterate:
      arr.append(iterate.value)
      iterate=iterate.next
    return arr
  def clearLinkedList(self):
    iterate=self.head
    while iterate:
      iterate.value=None
      iterate=iterate.next


object3=LinkedList()
object3.insertNode(3)
object3.insertNode(4)
object3.insertNode(5)
#object3.printLinkedList()
arr= object3.getLinkedListinStack()
print("\nElements of the Linked List Added to the stack")
```

```
print(arr)
print("\nPopping elements from the stack:")
object3.clearLinkedList()
while arr:
  object3.insertNode(arr.pop())
object3.printLinkedList()
```

Output:

```
    Newly inserted element in the linked list is: 3
Newly inserted element in the linked list is: 4
Newly inserted element in the linked list is: 5

Elements of the Linked List Added to the stack
[3, 4, 5]

Popping elements from the stack:
Newly inserted element in the linked list is: 5
Newly inserted element in the linked list is: 4
Newly inserted element in the linked list is: 3
Elements of the linked list are:
5
4
3
```

Time Complexity: O(n) because while inserting an element in the tail, we had to iterate through a loop of nodes.
So In conclusion, the time complexity of the 3 subtasks are O(1), O(n), and O(n)

2 Task 2

```
    stack1=[34, 3, 31, 40, 98, 92, 23]
print("Test Case 1:")
print(stack1)
stack2=[]
stack3=[]
i=0
while stack1:
  stack2.append(stack1.pop())
  i=i+1
mid=0
if i%2==0:
  mid= i/2
else:
  mid= i//2
for i in range(0, mid):
  stack3.append(stack2.pop())
stack2.pop()
while stack2:
  stack3.append(stack2.pop())
print("\nNew Array After deleting the middle element:")
print(stack3)
```

Output:

```
    Test Case 1:
[34, 3, 31, 40, 98, 92, 23]

New Array After deleting the middle element:
[34, 3, 31, 98, 92, 23]

    Test Case 1:
[3, 5, 1, 4, 2, 8]

New Array After deleting the middle element:
[3, 5, 4, 2, 8]
```

Time Complexity: O(n) because we are iterating through a list to append element from one stack that is popped from the other.

3 Task 3

```
    import numpy as np
input= [1, 2, 3, 1, 4, 5, 2, 3, 6]
k = 3
idx=0
length=len(input)
output=[]
for i in range(0,length):
  stack=[]
  if i+k<= length:
    for j in range(i, i+k):
      stack.append(input[j])
  else:
    continue
  element=float('-inf')
  while stack:
    p=stack.pop()
    if p> element:
      element=p
  output.append(element)
print(output)
```

Time Complexity is:
$O(n^2)$

for iterating twice inside nested loop. One loop is each of the element in the input array and the second loop is for the sub array of k length that is generated from each index of input from which we would find the maximum element.

4 Task 4

```
    arr1=[1,2,3,4,6]
arr2=[2,4,6,8]
#This elements of the lists will be inserted into the linked list first

class Node:
  def __init__(self, value):
    self.value = value
    self.next = None
class LinkedList:
  def __init__(self):
    self.head=Node(None)
  def insertNode(self, value):
    new= Node(value)
    print("Newly inserted element in the linked list is: {}".format(value))
    if self.head.value==None:
      self.head= new
      return
    iterate=self.head
    while iterate.next!=None:
      iterate= iterate.next
    iterate.next=new
  def printLinkedList(self):
    iterate=self.head
    print("Elements of the linked list are:")
    while iterate!=None:
      print(iterate.value)
      iterate=iterate.next
  def getLinkedListinStack(self):
    arr=[]
    iterate=self.head
    while iterate:
      arr.append(iterate.value)
      iterate=iterate.next
    return arr
  def clearLinkedList(self):
    iterate=self.head
    while iterate:
      iterate.value=None
      iterate=iterate.next
linkedlist1= LinkedList()
print("For Linked List 1")
for i in range(0, len(arr1)):
  linkedlist1.insertNode(arr1[i])
print("For Linked List 2")
linkedlist2=LinkedList()
for i in range(0, len(arr2)):
  linkedlist2.insertNode(arr2[i])

#iteration
common=[]
head1=linkedlist1.head
while head1:
  head2=linkedlist2.head
  while head2:
```

```
        if head2.value==head1.value:
          common.append(head2.value)
        head2=head2.next
      head1=head1.next
print(common)
```

Output:

```
    For Linked List 1
Newly inserted element in the linked list is: 1
Newly inserted element in the linked list is: 2
Newly inserted element in the linked list is: 3
Newly inserted element in the linked list is: 4
Newly inserted element in the linked list is: 6
For Linked List 2
Newly inserted element in the linked list is: 2
Newly inserted element in the linked list is: 4
Newly inserted element in the linked list is: 6
Newly inserted element in the linked list is: 8
[2, 4, 6]
```

Time COmplexity is: $O(n^2)$

because we iterated through 2 linked list and tried to find the common elements between them that was appended to a list called common.