นายปณต ศรีนัครินทร์ 645020096-3

Assignment 2

1.ตัวอย่างการสร้าง Stack

```python
# Python program to demonstrate
# stack implementation using a linked list.
# node class
class Node:
  def __init__(self, value):
    self.value = value
    self.next = None


class Stack:

  # Initializing a stack.
  # Use a dummy node, which is
  # easier for handling edge cases.
  def __init__(self):
    self.head = Node("head")
    self.size = 0

  # String representation of the stack
  def __str__(self):
    cur = self.head.next
    out = ""
    while cur:
      out += str(cur.value) + "->"
      cur = cur.next
    return out[:-3]

  # Get the current size of the stack
  def getSize(self):
    return self.size
```

```
def getSize(self):
  return self.size

# Check if the stack is empty
def isEmpty(self):
  return self.size == 0

# Get the top item of the stack
def peek(self):

  # Sanitary check to see if we
  # are peeking an empty stack.
  if self.isEmpty():
    raise Exception("Peeking from an empty stack")
  return self.head.next.value

# Push a value into the stack.
def push(self, value):
  node = Node(value)
  node.next = self.head.next
  self.head.next = node
  self.size += 1

# Remove a value from the stack and return.
def pop(self):
  if self.isEmpty():
    raise Exception("Popping from an empty stack")
  remove = self.head.next
  self.head.next = self.head.next.next
  self.size -= 1
  return remove.value
```

```
# Driver Code
if __name__ == "__main__":
  stack = Stack()
  for i in range(1, 11):
    stack.push(i)
  print(f"Stack: {stack}")

  for _ in range(1, 6):
    remove = stack.pop()
    print(f"Pop: {remove}")
  print(f"Stack: {stack}")
```

ผลที่ได้จากการ Driver Code

```
Stack: 10->9->8->7->6->5->4->3->2->
Pop: 10
Pop: 9
Pop: 8
Pop: 7
Pop: 6
Stack: 5->4->3->2->
```

อีกหนึ่งตัวอย่างการสร้าง Stack

```
class Empty(Exception):
  pass
class ArrayStack:
  def __init__(self):
      self.data=[]
  def __len__(self):
      return len(self.data)
  def is_empty(self):
      return len(self.data)==0
  def push(self,e):
      self.data.append(e)
  def top(self):
      if self.is_empty():
        raise Empty('Stack is empty')
      return self.data[-1]
  def pop(self):
      if self.is_empty():
        raise Empty('Stack is empty')
      return self.data.pop()
```

นายปณต ศรีนัครินทร์ 645020096-3

```python
#Driver Code
if __name__ == "__main__":
  A=ArrayStack()
  A.push('e')
  A.push(1)
  A.push(2)
  A.push(3)
  print("ลบ element ตัวแรกออกจาก Stack")
  print(A.pop())
  print("\nprint stack")
  print(A.data)
  print("\nความยาวStack")
  print(A.__len__())
  print("\nเช็ควาStackว่างไหม")
  print(A.is_empty())
  print("\print บนสุดของ Stack")
  print(A.top())
```

ผลจากการ Driver Code

```
ลบ element ตัวแรกออกจาก Stack
3

print stack
['e', 1, 2]

ความยาวStack
3

เช็ควาStackว่างไหม
False

print บนสุดของ Stack
2
```

2.ตัวอย่างของการสร้าง Queue

```
class ArrayQueue:
  Capacity=10
  def __init__(self):
    self.data=[None]*ArrayQueue.Capacity
    self.size=0
    self.front=0

  def __len__(self):
    return self.size

  def empty(self):
    return self.size==0

  def first(self):
      if self.empty():
         raise NameError('Queue is empty')
      return self.data[self.front]
  def dequeue(self):
      if self.empty():
          raise NameError('Queue is empty')
      answer=self.data[self.front]
      self.data[self.front]=None
      self.front=(self.front+1)%len(self.data)
      self.size-=1
      return answer
```

```
  def enqueue(self,e):
      if self.size==len(self.data):
          self.resize(2*len(self.data))
      avail=(self.front+self.size)%len(self.data)
      self.data[avail]=e
      self.size+=1

  def resize(self,cap):
      old=self.data
      self.data=[None]*cap
      walk=self.front
      for k in range(self.size):
          self.data[k]=old[walk]
          walk=(1+walk)%len(old)
      self.front=0
```

3.การสร้าง Singly Linked List

```python
class Node:

    def __init__(self, data):
        self.data = data
        self.next_node = None


class LinkedList:

    def __init__(self):
        self.head = None
        self.num_of_nodes = 0

    # O(1)
    def insert_start(self, data):

        self.num_of_nodes = self.num_of_nodes + 1
        new_node = Node(data)

        # the head is NULL (so the data structure is empty)
        if not self.head:
            self.head = new_node
        # there is at lest one item in the linked list
        else:
            new_node.next_node = self.head
            self.head = new_node
```

```python
# O(N)
def insert_end(self, data):

    self.num_of_nodes = self.num_of_nodes + 1
    new_node = Node(data)

    actual_node = self.head

    # we have to find the end of the linked list in O(N) linear running time
    while actual_node.next_node is not None:
        actual_node = actual_node.next_node

    # actual_node is the last node: so we insert the new_node right after the actual_node
    actual_node.next_node = new_node

# O(1)
def size_of_list(self):
    return self.num_of_nodes

# have to consider all the items in O(N) linear running time
def traverse(self):

    actual_node = self.head

    while actual_node is not None:
        print(actual_node.data)
        actual_node = actual_node.next_node
```

นายปณต ศรีนัครินทร์ 645020096-3

```python
# O(N) linear running time for finding arbitrary item
def remove(self, data):

    # list is empty
    if self.head is None:
        return

    actual_node = self.head
    # we have to track the previous node for future pointer updates
    # this is why doubly linked lists are better - we can get the previous
    # node (here with linked lists it is impossible)
    previous_node = None

    # search for the item we want to remove (data)
    while actual_node is not None and actual_node.data != data:
        previous_node = actual_node
        actual_node = actual_node.next_node

    # search miss
    if actual_node is None:
        return

    # the head node is the one we want to remove
    if previous_node is None:
        self.head = actual_node.next_node
    else:
        # remove an internal node by updating the pointers
        # NO NEED TO del THE NODE BECAUSE THE GARBAGE COLLECTOR WILL DO THAT
        previous_node.next_node = actual_node.next_node
```

4.การสร้าง Doubly Linked List

```python
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None
        self.previous = None


class DoublyLinkedList:

    def __init__(self):
        self.head = None
        self.tail = None

    # this operation inserts items at the end of the linked list
    # so we have to manipulate the tail node in O(1) running time
    def insert(self, data):

        new_node = Node(data)

        # when the list is empty
        if self.head is None:
            self.head = new_node
            self.tail = new_node
        # there is at least 1 item in the data structure
        # we keep inserting items at the end of the linked list
        else:
            new_node.previous = self.tail
            self.tail.next = new_node
            self.tail = new_node
```

```python
  # we can traverse a doubly linked list in both directions
  def traverse_forward(self):

      actual_node = self.head

      while actual_node is not None:
          print("%d" % actual_node.data)
          actual_node = actual_node.next

  def traverse_backward(self):

      actual_node = self.tail

      while actual_node is not None:
          print("%d" % actual_node.data)
          actual_node = actual_node.previous
```

นายปณต ศรีนัครินทร์ 645020096-3

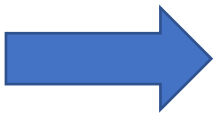
5.การสร้าง Tree และ การ Traverse Tree

```
class Tree:
    def __init__(self, val):
        self.value=val
        self.left = None
        self.right = None

def PreorderTra(Tree):
    if Tree:
      print(Tree.value)
      PreorderTra(Tree.left)
      PreorderTra(Tree.right)
def Postordertra(Tree):
    if Tree:
      PreorderTra(Tree.left)
      PreorderTra(Tree.right)
      print(Tree.value)
def Inordertra(Tree):
    if Tree:
      PreorderTra(Tree.left)
      print(Tree.value)
      PreorderTra(Tree.right)
```

ผลจากการ Driver Code

```
T=Tree(0)
T.left=Tree(1)
T.right=Tree(2)
T.left.left=Tree(3)
T.left.right=Tree(4)
T.right.left=Tree(5)
T.right.right=Tree(6)
print("Preorder")
PreorderTra(T)
print("Postorder")
Postordertra(T)
print("Inorder")
Inordertra(T)
```



```
Preorder
0
1
3
4
2
5
6
Postorder
1
3
4
2
5
6
0
Inorder
1
3
4
0
2
5
6
```

นายปณต ศรีนัครินทร์ 645020096-3