D.Y. PATIL INTERNATIONAL UNIVERSITY
B.TECH CSE FY SEM-2
A.Y. 2022-2023
LAB ASSIGNMENT: 06

NAME: Suryakant Upadhyay
PRN: 20220802043
SUBJECT: DATA STRUCTURES
TOPIC: SEARCHING AND SINGLY LINKED LIST

SEARCHING

1) Write a python code to implement recursive binary search.

Answer:

```
def binary_search(my_list, low, high, elem):
       if high >= low:
      mid = (high + low) // 2
      if my_list[mid] == elem:
         return mid
      elif my_list[mid] > elem:
         return binary_search(my_list, low, mid - 1, elem)
      else:
         return binary_search(my_list, mid + 1, high, elem)
   else:
      return -1

my_list = [ 1, 9, 11, 21, 34, 54, 67, 90 ]
elem_to_search = 90
print("The list is:", my_list)

my_result = binary_search(my_list,0,len(my_list)-1,elem_to_search)

if my_result != -1:
   print("Element found at index: ", str(my_result))
else:
   print("Element not found!")
```

OUTPUT:

```
The list is: [1, 9, 11, 21, 34, 54, 67, 90]
Element found at index:  7
```

SINGLY LINKED LIST

1) Write a python code to create a single node.
   Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
node = Node(10)
print("Single Node has been created")
```

OUTPUT:

```
Single Node has been created
```

2) Write a python code to create an empty singly linked list.
   Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
linked_list = LinkedList()
print("The Empty singly linked list has been created")
```

OUTPUT:

```
The Empty singly linked list has been created
```

3) Write a python code to create singly linked list with single node.
   Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next is not None:
                current = current.next
            current.next = new_node

linked_list = LinkedList()
linked_list.add_node(10)
print("Singly Linked list with single node has been created")
```

OUTPUT:

```
Singly Linked list with single node has been created
```

4) Write a python code to create singly linked list with multiple nodes. (3 nodes).

Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next is not None:
                current = current.next
            current.next = new_node

linked_list = LinkedList()
linked_list.add_node(10)
linked_list.add_node(20)
linked_list.add_node(30)
print("singly linked list with three nodes has been created")
```

OUTPUT:

```
singly linked list with three nodes has been created
```

5) Write a python code to perform traversal operation on singly linked list having 3 nodes.

Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
```

```
            while current.next is not None:
                current = current.next
            current.next = new_node

    def print_list(self):
        current = self.head
        while current is not None:
            print(current.data)
            current = current.next

linked_list = LinkedList()
linked_list.add_node(10)
linked_list.add_node(20)
linked_list.add_node(30)

linked_list.print_list()
```

OUTPUT:

```
10
20
30
```

6) Write a python code to perform insertion operation on singly linked list having 3 nodes

Answer:

```
# define a Node class to creating a single node
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

# define a LinkedList class to create a linked list and perform operations on
it
class LinkedList:
    def __init__(self):
        self.head = None

    # function to insert a node at the beginning of the linked list
    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    # function to print the linked list
    def print_linked_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=' -> ')
            current_node = current_node.next
        print('NULL')

# create a linked list with 3 nodes
```

```
my_linked_list = LinkedList()
my_linked_list.head = Node(1)
second_node = Node(2)
third_node = Node(3)

# link the nodes together
my_linked_list.head.next = second_node
second_node.next = third_node

# print the original linked list
print('Original linked list:')
my_linked_list.print_linked_list()

# insert a new node at the beginning of the linked list
my_linked_list.insert_at_beginning(0)

# print the updated linked list
print('Updated linked list:')
my_linked_list.print_linked_list()
```

OUTPUT:

```
Original linked list:
1 -> 2 -> 3 -> NULL
Updated linked list:
0 -> 1 -> 2 -> 3 -> NULL
```

7) Write a python code to perform deletion operation on singly linked list having 3 nodes.

Answer:

```
# define a Node class to create a single node
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

# define a LinkedList class to create a linked list and perform operations on
it
class LinkedList:
    def __init__(self):
        self.head = None

    # function to insert a node at the beginning of the linked list
    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    # function to delete the first node of the linked list
    def delete_first_node(self):
        if self.head:
            self.head = self.head.next

    # function to print the linked list
```

```
    def print_linked_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=' -> ')
            current_node = current_node.next
        print('NULL')

# create a linked list with 3 nodes
my_linked_list = LinkedList()
my_linked_list.head = Node(1)
second_node = Node(2)
third_node = Node(3)

# link the nodes together
my_linked_list.head.next = second_node
second_node.next = third_node

# print the original linked list
print('Original linked list:')
my_linked_list.print_linked_list()

# delete the first node of the linked list
my_linked_list.delete_first_node()

# print the updated linked list
print('Updated linked list:')
my_linked_list.print_linked_list()
```

OUTPUT:

```
Original linked list:
1 -> 2 -> 3 -> NULL
Updated linked list:
2 -> 3 -> NULL
```

8) Write a python code to perform search operation on singly linked list having 3 nodes.

Answer:

```
# define a Node class to create a single node
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

# define a LinkedList class to create a linked list and perform operations on
it
class LinkedList:
    def __init__(self):
        self.head = None

    # function to insert a node at the beginning of the linked list
    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
```

```
        self.head = new_node

    # function to search for a node with a given data value
    def search_node(self, data):
        current_node = self.head
        while current_node:
            if current_node.data == data:
                return True
            current_node = current_node.next
        return False

    # function to print the linked list
    def print_linked_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=' -> ')
            current_node = current_node.next
        print('NULL')

# create a linked list with 3 nodes
my_linked_list = LinkedList()
my_linked_list.head = Node(1)
second_node = Node(2)
third_node = Node(3)

# link the nodes together
my_linked_list.head.next = second_node
second_node.next = third_node

# print the original linked list
print('Original linked list:')
my_linked_list.print_linked_list()

# search for a node with data value 2
if my_linked_list.search_node(2):
    print('Node with data value 2 found in the linked list')
else:
    print('Node with data value 2 not found in the linked list')

# search for a node with data value 4
if my_linked_list.search_node(4):
    print('Node with data value 4 found in the linked list')
else:
    print('Node with data value 4 not found in the linked list')
```

OUTPUT:

```
Original linked list:
1 -> 2 -> 3 -> NULL
Node with data value 2 found in the linked list
Node with data value 4 not found in the linked list
```

DOUBLY LINKED LIST

1) Write a python code to create a doubly linked list node.

Answer:

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
        self.prev = None
print("a doubly linked list node is created")
```

OUTPUT:

```
a doubly linked list node is created
```

2) Write a python code to create an empty a doubly linked list.

Answer:

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
        self.prev = None

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

    def is_empty(self):
        return self.head is None

    def print_linked_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=' ')
            current_node = current_node.next
        print()

# create an empty doubly linked list
my_doubly_linked_list = DoublyLinkedList()

# check if the list is empty
if my_doubly_linked_list.is_empty():
    print('The doubly linked list is empty')

# print the empty doubly linked list
print('The contents of the doubly linked list are:')
my_doubly_linked_list.print_linked_list()
```

OUTPUT:

```
The doubly linked list is empty
The contents of the doubly linked list are:
```

3) Write a python code to create doubly linked list with single node.

Answer:

```
class Node:
    def __init__(self, data=None):
```

```
        self.data = data
        self.next = None
        self.prev = None

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

    def is_empty(self):
        return self.head is None

    def print_linked_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=' ')
            current_node = current_node.next
        print()

    def insert_at_beginning(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.head = self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node

# create a doubly linked list with a single node
my_doubly_linked_list = DoublyLinkedList()
my_doubly_linked_list.insert_at_beginning(10)

# print the contents of the linked list
print('The contents of the doubly linked list are:')
my_doubly_linked_list.print_linked_list()
```

OUTPUT:

```
The contents of the doubly linked list are:
10
```

4) Write a python code to create doubly linked list with multiple node.

Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
```

```
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def print_list(self):
        current = self.head
        while current:
            print(current.data)
            current = current.next


doubly_linked_list = DoublyLinkedList()
doubly_linked_list.add_node(1)
doubly_linked_list.add_node(2)
doubly_linked_list.add_node(3)
doubly_linked_list.print_list()
```

OUTPUT:

```
1
2
3
```

5) Write a python code to perform traversal operation on doubly linked list having 3 nodes

Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def print_forward(self):
        current = self.head
        while current:
```

```
            print(current.data)
            current = current.next

    def print_backward(self):
        current = self.head
        while current.next:
            current = current.next
        while current:
            print(current.data)
            current = current.prev

# Example usage
doubly_linked_list = DoublyLinkedList()
doubly_linked_list.add_node(1)
doubly_linked_list.add_node(2)
doubly_linked_list.add_node(3)

print("Traversal in forward direction:")
doubly_linked_list.print_forward()

print("Traversal in backward direction:")
doubly_linked_list.print_backward()
```

OUTPUT:

```
Traversal in forward direction:
1
2
3
Traversal in backward direction:
3
2
1
```

6) Write a python code to perform insertion operation on doubly linked list having 3 nodes.

Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
```

```
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def insert_node(self, position, data):
        new_node = Node(data)
        if position == 0:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        else:
            current = self.head
            for i in range(position-1):
                current = current.next
            new_node.prev = current
            new_node.next = current.next
            current.next.prev = new_node
            current.next = new_node

    def print_list(self):
        current = self.head
        while current:
            print(current.data)
            current = current.next

# Example usage
doubly_linked_list = DoublyLinkedList()
doubly_linked_list.add_node(1)
doubly_linked_list.add_node(2)
doubly_linked_list.add_node(3)

print("Original list:")
doubly_linked_list.print_list()

print("Inserting node with data 4 at position 1:")
doubly_linked_list.insert_node(1, 4)
doubly_linked_list.print_list()

print("Inserting node with data 5 at position 3:")
doubly_linked_list.insert_node(3, 5)
doubly_linked_list.print_list()
```

OUTPUT:

```
Original list:
1
2
3
Inserting node with data 4 at position 1:
1
4
2
3
Inserting node with data 5 at position 3:
1
4
2
5
3
```

7) Write a python code to perform deletion operation on doubly linked list having 3 nodes.

Answer:

```
class Node:

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

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def delete_node(self, position):
        if position == 0:
            self.head = self.head.next
            self.head.prev = None
        else:
            current = self.head
```

```
            for i in range(position-1):
                current = current.next
                current.next = current.next.next
                current.next.next = current.next.prev
                current.next.prev= current

    def print_list(self):
        current = self.head
        while current:
            print(current.data)
            current = current.next


doubly_linked_list = DoublyLinkedList()
doubly_linked_list.add_node(1)
doubly_linked_list.add_node(2)
doubly_linked_list.add_node(3)

print("Original list:")
doubly_linked_list.print_list()

print("Deleting node at position 1:")
doubly_linked_list.delete_node(1)
doubly_linked_list.print_list()
```

OUTPUT:

```
Original list:
1
2
3
Deleting node at position 1:
1
2
3
```

8) Write a python code to perform search operation on doubly linked list having 3 nodes.

Answer:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
```

```
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def search_node(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def print_list(self):
        current = self.head
        while current:
            print(current.data)
            current = current.next

# Example usage
doubly_linked_list = DoublyLinkedList()
doubly_linked_list.add_node(1)
doubly_linked_list.add_node(2)
doubly_linked_list.add_node(3)

print("Original list:")
doubly_linked_list.print_list()

if doubly_linked_list.search_node(2):
    print("Node with data 2 found")
else:
    print("Node with data 2 not found")

if doubly_linked_list.search_node(4):
    print("Node with data 4 found")
else:
    print("Node with data 4 not found")
```

OUTPUT:

```
Original list:
1
2
3
Node with data 2 found
Node with data 4 not found
```