Malla Reddy University

I B.Tech II Semester – CSE/AIML/CS/DS/IT/IOT

Question Bank

Data Structures and Its Applications

Course Code: MR22-1CS0105

DSA MID–2

QUESTIONS AND ANSWERS

UNIT - III

DONE AND
PRESENTED BY-

TEAM VISION

Contact:-
teamvision1729@gmail.com

UNIT-III

1. Explain the structure of a doubly linked list node and its key components with examples.
2. Write a python program to implement doubly linked list.
3. What is a circular linked list and how it is different from a single & double linked list? Discuss in detail its operations with examples.
4. Explain how to implement a stack using linked list with example. Describe the operations supported by stack using a singly linked list with an example program.
5. Explain how to implement a queue using linked list, also describe the operations supported by queue using a singly linked list with an example program.

UNIT-3

1. Explain the structure of a doubly linked list node and its key components with examples.

Ans:

A doubly linked list node is a data structure used in linked list implementations that allows traversal in both forward and backward directions. Each node in a doubly linked list contains three main components:

i)Data / Value

ii)Previous Pointer

iii)Next Pointer

- **Data/Value:** This component stores the actual data or value associated with the node. It can be of any data type depending on the application. For example, if you're implementing a linked list to store integers, the data component would be an integer value.
- **Previous Pointer:** This component holds a reference to the previous node in the linked list. It points to the node that comes before the current node. In the first node of the list, the previous pointer is usually set to null or some other sentinel value to indicate the start of the list.
- **Next Pointer:** This component holds a reference to the next node in the linked list. It points to the node that comes after the current node. In the last node of the list, the next pointer is typically set to null or some other sentinel value to indicate the end of the list.

Doubly Linked List Node Implementation –

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

The structure of a doubly linked list with multiple nodes –

```
# Create three nodes
node1 = DoublyLinkedListNode(10)
node2 = DoublyLinkedListNode(20)
node3 = DoublyLinkedListNode(30)

# Connect the nodes
node1.next = node2
node2.prev = node1
node2.next = node3
node3.prev = node2
```

2. Write a python program to implement doubly linked list.

Ans:

```
class DoublyLinkedListNode:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
class DoublyLinkedList:
    def __init__(self):
        self.head = None
    def append(self, data):
        new_node = DoublyLinkedListNode(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
            new_node.prev = current
    def prepend(self, data):
        new_node = DoublyLinkedListNode(data)
        if self.head is None:
            self.head = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
    def delete(self, data):
        if self.head is None:
            return
        current = self.head
        while current is not None:
            if current.data == data:
                if current.prev is not None:
                    current.prev.next = current.next
```

#Continue

```
                else:
                    self.head = current.next
                if current.next is not None:
                    current.next.prev = current.prev
                return
            current = current.next
    def display(self):
        if self.head is None:
            print("Doubly linked list is empty.")
            return

        current = self.head
        while current is not None:
            print(current.data, end=" ")
            current = current.next
        print()
dll = DoublyLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
dll.prepend(5)
dll.prepend(2)
dll.display()
dll.delete(10)
dll.display()
```

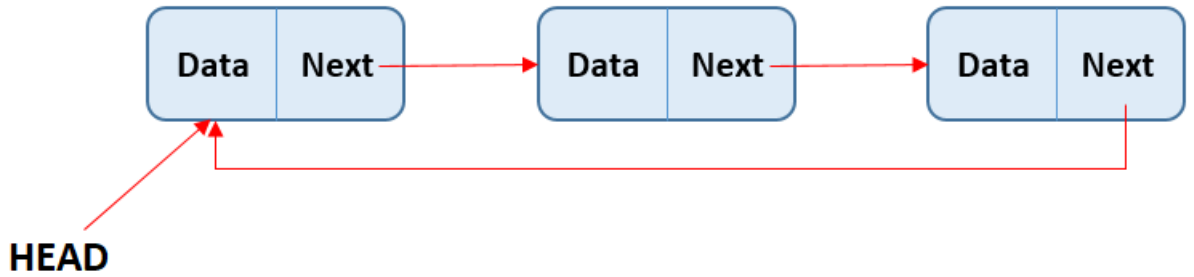
Output –

```
2 5 10 20 30
2 5 20 30
```

3. What is a circular linked list and how it is different from a single & double linked list? Discuss in detail its operations with examples.

Ans:

A circular linked list is a variation of a linked list in which the last node of the list points back to the first node, forming a circular loop. In other words, the "next" pointer of the last node points to the first node, creating a circular structure. Here's a representation of a circular linked list:



In a circular linked list, the traversal can start from any node, and it can loop through the entire list by following the "next" pointers until it reaches the starting node again.

Now let's discuss the differences between a circular linked list, a singly linked list, and a doubly linked list:

- Singly Linked List:
  - In a singly linked list, the last node's "next" pointer is set to null, indicating the end of the list.
  - The traversal can only be done in a forward direction, starting from the head node.
  - It does not form a circular loop.

- Doubly Linked List:
  - In a doubly linked list, each node has a "next" pointer that points to the next node and a "previous" pointer that points to the previous node.
  - The traversal can be done in both forward and backward directions.
  - It does not form a circular loop by default.
- Circular Linked List:
  - In a circular linked list, the last node's "next" pointer points back to the first node, forming a circular loop.
  - The traversal can start from any node, and it can loop through the entire list by following the "next" pointers until it reaches the starting node again.
  - It does not have a distinct end or beginning since the last node connects to the first node, creating a continuous loop.

Now let's discuss some common operations on a circular linked list:

1. Traversal:
   - Since a circular linked list forms a loop, we can start traversing from any node and continue until we reach the starting node again.
2. Insertion:
   - Insertion in a circular linked list can be done at different positions: at the beginning, at the end, or at a specific position.
3. Deletion:
   - Deletion in a circular linked list can also be done at different positions: at the beginning, at the end, or at a specific position.
4. Searching:
   - Searching in a circular linked list is similar to singly or doubly linked lists. We can traverse through the list until we find the desired element or reach the starting node again.

4. Explain how to implement a stack using linked list with example. Describe the operations supported by the stack using a singly linked list with an example program.

Ans:

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

    def display(self):
        if self.top is None:
            print('The stack is empty')
        else:
            temp=self.top
            print('The elements of stack are:')
            while temp is not None:
                print('-->',temp.data,end='')
                temp=temp.next
            print()

    def push(self,data):
        new_node=Node(data)
        new_node.next=self.top
        self.top=new_node
```

#Continue

```
    def pop(self):
        if self.top is None:
            print('The Stack is empty')
        else:
            temp=self.top
            print('The popped element is',self.top.data)
            self.top=temp.next
            temp.next=None

S=Stack()
S.push(1)
S.push(2)
S.push(3)
S.display()
S.pop()
S.pop()
S.pop()
S.display()
```

#Output

```
The elements of stack are:
--> 3--> 2--> 1
The popped element is 3
The popped element is 2
The popped element is 1
The stack is empty
```

- The Node class represents a node in the linked list, while the Stack class represents the stack and contains methods for pushing, popping, and displaying the elements.
- The push method adds an element to the top of the stack by creating a new node and updating the top reference to point to the new node.
- The pop method removes and returns the element from the top of the stack by updating the top reference and detaching the original top node.
- The display method prints the elements of the stack by traversing the linked list from the top to the bottom and printing the data of each node.
- An example usage creates a stack, pushes three elements, displays the elements, pops three elements, and displays the stack after popping.

5. Explain how to implement a queue using a linked list, also describe the operations supported by the queue using a singly linked list with an example program.

Ans:

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

class Queue:
    def __init__(self):
        self.front=None
        self.rear=None

    def display(self):
        if self.front is None:
            print('The Queue is empty')
        else:
            temp=self.front
            print('The elements of Queue are :')
            while temp is not None:
                print(temp.data,end=' ')
                temp=temp.next
            print()

    def enqueue(self,data):
        new_node=Node(data)
        if self.front is None:
            self.rear=self.front=new_node
        else:
            self.rear.next=new_node
            self.rear=new_node
```

#Continue

```
    def dequeue(self):
        if self.front is None:
            print('Queue is empty')
        elif self.front==self.rear:
            self.front=self.rear=None
        else:
            temp=self.front
            self.front=temp.next
            temp.next=None

q=Queue()
q.enqueue(0)
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
q.dequeue()
q.display()
```

#Output

```
The elements of Queue are :
1 2 3
```

- The Node class represents a node in the linked list, while the Queue class represents the queue and contains methods for enqueueing, dequeuing, and displaying the elements.
- The enqueue method adds an element to the rear of the queue by creating a new node and updating the rear reference.
- The dequeue method removes and returns the element from the front of the queue by updating the front reference and detaching the original front node.
- The display method prints the elements of the queue by traversing the linked list from the front to the rear and printing the data of each node.
- An example usage creates a queue, enqueues four elements, dequeues the front element, and displays the remaining elements.