**Experiment 6:**

**Aim:** To implement a Doubly Linked List (DLL) using Python

**Practical Learning Objectives:**

The objective of this practical is to learn

1. To implement a Doubly Linked List (DLL)
2. To display the contents of the DLL
3. To add nodes to the DLL
4. To remove nodes from the DLL

**Software Required:**

Python 3

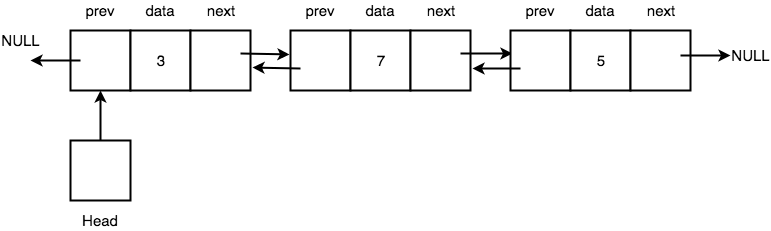
**Theory:**

A Doubly Linked List is an advanced version of Singly Linked List which has an additional pointer **prev** to store the location of the previous node.

A Doubly Linked list is also made up of nodes, but as compared to the node used in Singly linked list, node in case of doubly linked list has 3 parts:

1. **Data Part**: Holds the data
2. **Prev Address Part**: Holds the address of the previous node.
3. **Next Address Part**: Holds the address of the next node.

Below we have a simple pictorial representation of Nodes connected to form a Doubly Linked List.



As you can see, every node has some data stored in it, and it also stores the location of the next node and the previous node. It is not necessary that all the nodes get saved in contiguous memory locations. Also, the first node's **prev** pointer and the last node's next pointer points to NULL because there is no node before the first node and after the last node. **Head** is a pointer which always points to the first node of the list, if **Head** points to nothing, it means the linked list is empty.

Advantages of a Doubly Linked List over a Singly Linked List.

1. We can traverse a Doubly Linked List in both forward and backward directions.
2. Also, when we must delete any node in a Singly Linked List, we must declare an additional pointer to keep track of the previous node, which is not required in case of Doubly Linked List.

The major disadvantage is maintaining an extra **prev** pointer

1. Demands extra memory space
2. Adds extra steps to carry out basic operations on the Doubly Linked list
   * update two pointers every time we insert a new node or delete a node.

**Algorithm to display contents of the DLL**

1. Define a new node 'current' that will point to the head.
2. Print current.data till current points to null.
3. Current will point to the next node in the list in each iteration.

* **Algorithm to add a Node to the DLL**

1. It first checks whether the head is null, then it will insert the node as the head.
2. Both head and tail will point to a newly added node.
3. Head's previous pointer will point to null and tail's next pointer will point to null.
4. If the head is not null, the new node will be inserted at the end of the list such that new node's previous pointer will point to tail.
5. The new node will become the new tail. Tail's next pointer will point to null.

* **Algorithm to remove a Node from the DLL**

1. Take the position of the node to be removed as the input
2. Retrieve the node from the DLL using the index
3. If the node doesn’t exist, display appropriate error message
4. If the retrieved node is the **head** node, then reassign the node after the retrieved node as the **head** node.
5. Otherwise link the address part of the previous node to the node after the retrieved node.
6. If the retrieved node is the **tail** node, then assign the previous node as the tail node else link the address part of the next node to the previous node.

**Code:**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

self.prev = None

class DoublyLinkedList:

def \_\_init\_\_(self):

self.head = None

self.tail = None

def display(self):

current = self.head

if self.head is None:

print("List is empty")

return

print("Nodes of doubly linked list:")

while current is not None:

print(current.data, end=" <-> ")

current = current.next

print("None")

def addNode(self, data):

newNode = Node(data)

if self.head is None:

self.head = self.tail = newNode

self.head.prev = None

self.tail.next = None

else:

self.tail.next = newNode

newNode.prev = self.tail

self.tail = newNode

self.tail.next = None

def push(self, data):

newNode = Node(data)

if self.head is None:

self.head = self.tail = newNode

self.head.prev = None

self.tail.next = None

else:

newNode.next = self.head

newNode.prev = None

if self.head is not None:

self.head.prev = newNode

self.head = newNode

dList = DoublyLinkedList()

dList.addNode(1)

dList.addNode(2)

dList.addNode(3)

dList.addNode(4)

dList.addNode(5)

dList.display()

**Output**:

Nodes of doubly linked list:

1 <-> 2 <-> 3 <-> 4 <-> 5 <-> None

**Conclusion:**

**Practical Learning Outcomes:**

|  |  |
| --- | --- |
| After performing the practical, the learner is able to: | Marked |
| 1. To implement a Doubly Linked List (DLL) 2. To display the contents of the DLL 3. To add nodes to the DLL 4. To remove nodes from the DLL | ✓  ✓  ✓  ✓ |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Outcome** | **PLO 1** | **PLO 2** | **PLO 3** | **PLO 4** | **Performance** | **Attendance** | **Total Score** | **E&TC DEPARTMENT- TCET** |
| **Weight** | **15** | **15** | **15** | **15** | 20 | 20 | **100** | Date of  Performance:  Date of Correction:  Roll No:  Marks: \_\_\_\_\_\_/100  Signature of Faculty: |
| **Score** |  |  |  |  |  |  |  |