In [2]:

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


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

    def Insert_at_beginning(self, data):
        node = Node(data, self.head)
        self.head = node

    def Insert_at_ending(self, data):
        if self.head is None:
            self.head = Node(data)
            return
        itr = self.head
        while itr.next:
            itr = itr.next
        itr.next = Node(data)

    def insert_values(self, value_list):
        self.head = None
        for value in value_list:
            self.Insert_at_ending(value)

    def get_length(self):
        count = 0
        itr = self.head
        while itr:
            count += 1
            itr = itr.next
        return count

    def insert_at(self, index, data):
        if index < 0 or index >= self.get_length():
            raise Exception('invalid index')
        if index == 0:
            self.Insert_at_ending(data)
            return
        itr = self.head
        count = 0
        while itr:
            if count == index-1:
                itr.next = Node(data, itr.next)
                break
            itr = itr.next
            count += 1

    def remove_at(self, index):

        if index < 0 or index >= self.get_length():
            raise Exception('invalid index')

        if index == 0:
            self.head = self.head.next
            return

        itr = self.head
        count = 0
        while itr:
            if count == index-1:
                itr.next = itr.next.next
                break
            itr = itr.next
            count += 1

    def insert_after_value(self, data_after, data_to_insert):
        if self.head is None:
            return
        if self.head.data == data_after:
            self.head.next = Node(data_to_insert, self.head.next)
            return
        itr = self.head
        while itr:
            if itr.data == data_after:
                itr.next = Node(data_to_insert, itr.next)
                break
            itr = itr.next

    def remove_by_value(self, data):
        if self.head is None:
            return
        if self.head.data == data:
            self.head = self.head.next
            return
        itr = self.head
        while itr.next:
            if itr.next.data == data:
                itr.next = itr.next.next
                break
            itr = itr.next

    def display(self):
        if self.head is None:
            print('list is empty')
            return
        itr = self.head
        llstr = ''
        while itr:
            llstr += str(itr.data) + '-->'
            itr = itr.next
        print(llstr[:-3])

    def search(self, data):
        itr = self.head
        while itr.next:
            if itr.data == data:
                return True
            itr = itr.next
        return False
```

```python


ll = LinkedList()
ll.insert_values(["banana", "mango", "grapes", "orange"])
ll.display()
ll.insert_after_value("mango", "apple")
ll.display()
ll.remove_by_value("orange")
ll.display()
ll.remove_by_value("figs")
ll.display()
ll.remove_by_value("banana")
ll.display()
print(ll.search("mango"))
print("Count: ",ll.get_length())
ll.remove_at(1)
ll.display()
ll.remove_by_value("mango")
ll.remove_by_value("apple")
ll.remove_by_value("grapes")
print("Count: ",ll.get_length())
```

```
banana-->mango-->grapes-->orange
banana-->mango-->apple-->grapes-->orange
banana-->mango-->apple-->grapes
banana-->mango-->apple-->grapes
mango-->apple-->grapes
True
Count:  3
mango-->grapes
Count:  0
```

In [4]:

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

    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        if self.head is not None:
            self.head.prev = new_node
        self.head = new_node

    def insertAfter(self, prev_node, new_data):
        if prev_node is None:
            print ("the given previous node cannot be NULL")
            return
        new_node = Node(new_data)
        new_node.next = prev_node.next
        prev_node.next = new_node
        new_node.prev = prev_node
        if new_node.next is not None:
            new_node.next.prev = new_node

    def append(self,new_data):
        new_node = Node(new_data)
        new_node.next = None
        if self.head is None:
            new_node.prev = None
            self.head = new_node
            return

        last = self.head
        while(last.next is not None):
            last = last.next
        last.next = new_node
        new_node.prev = last
        return

    def printList(self, node):
        print( "\nTraversal in forward direction")
        while(node is not None):
            print (" % d" %(node.data), )
            last = node
            node = node.next
        print ("\nTraversal in reverse direction")
        while(last is not None):
            print (" %d " %(last.data), )
            last = last.prev

    def countNodes(self):
        counter = 0
        current = self.head
        while(current != None):
            counter = counter + 1
            current = current.next
        return counter

    def search(self, x):
        current = self.head
        while current != None:
            if current.data == x:
                return True
            current = current.next
        return False


llist = DoublyLinkedList()
llist.append(1)
llist.push(2)
llist.push(3)
llist.append(4)
llist.insertAfter(llist.head.next, 5)
print( "Created DLL is: ", )
llist.printList(llist.head)
print("\nCount of nodes present in the list: " + str(llist.countNodes())
if llist.search(4):
    print("Yes")
else:
    print("No")
```

```
Created DLL is:

Traversal in forward direction
  3
  2
  5
  1
  4

Traversal in reverse direction
 4
 1
 5
 2
 3

Count of nodes present in the list: 5
Yes
```

In [ ]:

```

```