UNIVERSITY OF MUMBAI

CENTER FOR DISTANCE AND OPEN EDUCATION (CDOE)

CERTIFICATE

THE EXPERIMENTS DULY SIGNED IN THIS JOURNAL REPRESENT THE WORK DONE BY MST./MS Shilpa Bhosale APPLICATION ID.: 20672 & SEAT NO.: 8221054 RESPECTIVELY IN **SEMESTER I OF FIRST YEAR OF MASTER OF COMPUTER APPLICATION (NEP)** IN THE COMPUTER LABORATORY OF PCP CENTER Prahladrai Dalmia Lions College FOR SUBJECT Advance data structure DURING THE ACADEMIC YEAR 2025-2026.

SUBJECT INCHARGE	HEAD OF DEPARTMENT	EXTERNAL EXAMINER

Sr no.	Topic	Date	Remark
1	Implementation of different searching & sorting techniques.		
2	Perform various hashing techniques with Linear Probe as collision resolution		
3	Implementation of Stacks,Ordinary Queue & Circular queue (Using arrays)		
4	Implementation of Stack Applications like: o infix to postfix o Postfix evaluation o Balancing of Parenthesis		
5	Implementation of all types of linked lists.		
6	Demonstrate application of linked list (eg.Sparse matrix,Stack,Queue,Priority & Double ended Queue)		
7	Create and perform various operations on BST.		
8	Implementing Heap with different operations.		
9	Create a Graph storage structure (eg. Adjacency matrix)		
10	Implementation of Graph traversal. (DFS and BFS)		

Practical 1 Implementation of different searching & sorting techniques.

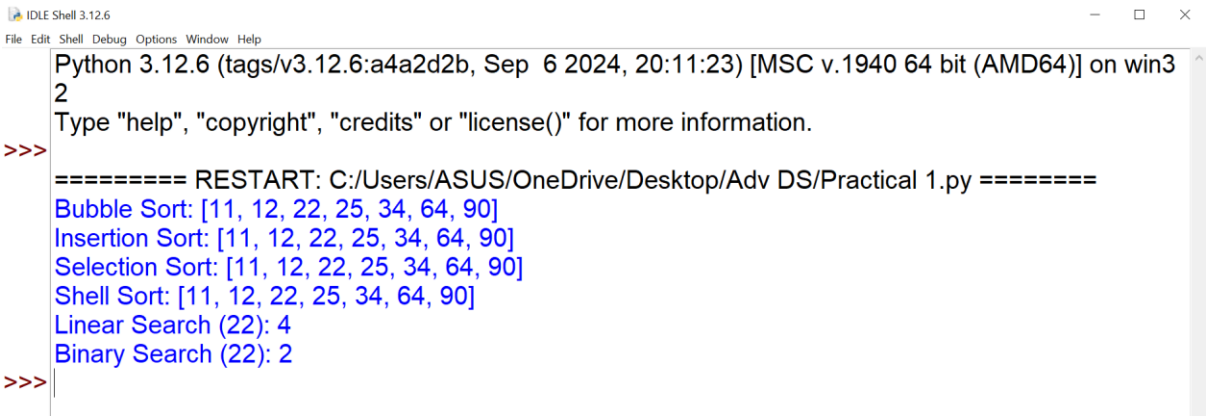
Code:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr


def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr


def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
```

```
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr


def shell_sort(arr):
    n = len(arr)
    gap = n // 2
    while gap > 0:
        for i in range(gap, n):
            temp = arr[i]
            j = i
            while j >= gap and arr[j - gap] > temp:
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = temp
        gap //= 2
    return arr


def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1
```

```
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1
    return -1



arr = [64, 34, 25, 12, 22, 11, 90]

print("Bubble Sort:", bubble_sort(arr.copy()))
print("Insertion Sort:", insertion_sort(arr.copy()))
print("Selection Sort:", selection_sort(arr.copy()))
print("Shell Sort:", shell_sort(arr.copy()))

print("Linear Search (22):", linear_search(arr, 22))
sorted_arr = sorted(arr)
print("Binary Search (22):", binary_search(sorted_arr, 22))
```

Output:


IDLE Shell 3.12.6
File Edit Shell Debug Options Window Help
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep 6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py ========
Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
Insertion Sort: [11, 12, 22, 25, 34, 64, 90]
Selection Sort: [11, 12, 22, 25, 34, 64, 90]
Shell Sort: [11, 12, 22, 25, 34, 64, 90]
Linear Search (22): 4
Binary Search (22): 2
>>>

Practical 2. Perform various hashing techniques with Linear Probe as collision resolution

Code:

```
class LinearProbingHashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size

    def hash_function(self, key):
        return key % self.size

    def insert(self, key):
        index = self.hash_function(key)

        if self.table[index] is None:
            self.table[index] = key
        else:
            original_index = index
            while self.table[index] is not None:
                index = (index + 1) % self.size
                if index == original_index:
                    print("Hash table is full")
                    return
            self.table[index] = key

    def search(self, key):
        index = self.hash_function(key)
        original_index = index
```

```
        while self.table[index] is not None:
            if self.table[index] == key:
                return index
            index = (index + 1) % self.size
            if index == original_index:
                break
        return -1

    def display(self):
        for i in range(self.size):
            print(f"Index {i}: {self.table[i]}")


hash_table = LinearProbingHashTable(10)

elements = [23, 43, 13, 27, 33]
for key in elements:
    hash_table.insert(key)

print("Hash Table:")
hash_table.display()

key = 27
result = hash_table.search(key)
if result != -1:
    print(f"Element {key} found at index {result}")
```

```
else:
    print(f"Element {key} not found")
```

Output:

```
Hash Table:
Index 0: None
Index 1: None
Index 2: None
Index 3: 23
Index 4: 43
Index 5: 13
Index 6: 33
Index 7: 27
Index 8: None
Index 9: None
Element 27 found at index 7
```

Practical 3. Implementation of Stacks,Ordinary Queue & Circular queue (Using arrays)

Code Stack Implementation (Using Array):

```
class Stack:
    def __init__(self, size):
        self.size = size
        self.stack = [None] * size
        self.top = -1

    def push(self, item):
        if self.top == self.size - 1:
            print("Stack Overflow")
        else:
            self.top += 1
            self.stack[self.top] = item
            print(item, "pushed")

    def pop(self):
        if self.top == -1:
            print("Stack Underflow")
        else:
            print(self.stack[self.top], "popped")
            self.top -= 1

    def display(self):
        if self.top == -1:
            print("Stack is empty")
        else:
```

```
        print("Stack elements:")
        for i in range(self.top, -1, -1):
            print(self.stack[i])


s = Stack(5)
s.push(10)
s.push(20)
s.push(30)
s.display()
s.pop()
s.display()
```

Output:

```
========= RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py ========
10 pushed
20 pushed
30 pushed
Stack elements:
30
20
10
30 popped
Stack elements:
20
10
```

Code Ordinary Queue Implementation (Using Array):

```
class Queue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = -1
        self.rear = -1

    def enqueue(self, item):
        if self.rear == self.size - 1:
            print("Queue Overflow")
        else:
            if self.front == -1:
                self.front = 0
            self.rear += 1
            self.queue[self.rear] = item
            print(item, "enqueued")

    def dequeue(self):
        if self.front == -1 or self.front > self.rear:
            print("Queue Underflow")
        else:
            print(self.queue[self.front], "dequeued")
            self.front += 1

    def display(self):
        if self.front == -1 or self.front > self.rear:
```

```
            print("Queue is empty")
        else:
            print("Queue elements:")
            for i in range(self.front, self.rear + 1):
                print(self.queue[i])


q = Queue(5)
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
q.display()
q.dequeue()
q.display()
```

Output:

```
========= RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py ========
10 enqueued
20 enqueued
30 enqueued
Queue elements:
10
20
30
10 dequeued
Queue elements:
20
30
```

Code Circular Queue Implementation (Using Array):

```
class CircularQueue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = -1
        self.rear = -1

    def enqueue(self, item):
        if (self.rear + 1) % self.size == self.front:
            print("Circular Queue Overflow")
        else:
            if self.front == -1:
                self.front = 0
            self.rear = (self.rear + 1) % self.size
            self.queue[self.rear] = item
            print(item, "enqueued")

    def dequeue(self):
        if self.front == -1:
            print("Circular Queue Underflow")
        else:
            print(self.queue[self.front], "dequeued")
            if self.front == self.rear:
                self.front = -1
                self.rear = -1
            else:
```

```
            self.front = (self.front + 1) % self.size

    def display(self):
        if self.front == -1:
            print("Circular Queue is empty")
        else:
            print("Circular Queue elements:")
            i = self.front
            while True:
                print(self.queue[i])
                if i == self.rear:
                    break
                i = (i + 1) % self.size


cq = CircularQueue(5)
cq.enqueue(10)
cq.enqueue(20)
cq.enqueue(30)
cq.display()
cq.dequeue()
cq.display()
```

Output:

```
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
10 enqueued
20 enqueued
30 enqueued
Circular Queue elements:
10
20
30
10 dequeued
Circular Queue elements:
20
30
>
```

Practical 4. Implementation of Stack Applications like:

o infix to postfix

o Postfix evaluation

o Balancing of Parenthesis

Code (Infix to Postfix Conversion):

```
def precedence(op):
    if op == '+' or op == '-':
        return 1
    if op == '*' or op == '/':
        return 2
    if op == '^':
        return 3
    return 0

def infix_to_postfix(expression):
    stack = []
    postfix = ""

    for char in expression:
        if char.isalnum():
            postfix += char
        elif char == '(':
            stack.append(char)
        elif char == ')':
            while stack and stack[-1] != '(':
                postfix += stack.pop()
```

```
            stack.pop()
        else:
            while stack and precedence(stack[-1]) >= precedence(char):
                postfix += stack.pop()
            stack.append(char)

    while stack:
        postfix += stack.pop()

    return postfix


exp = "A+B*C"
print("Infix Expression :", exp)
print("Postfix Expression :", infix_to_postfix(exp))
```

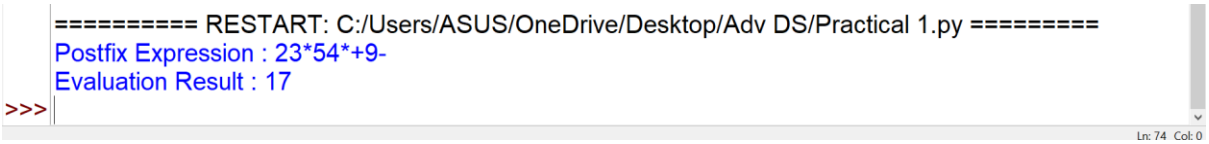
Output:

```
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
Infix Expression : A+B*C
Postfix Expression : ABC*+
```

Code (Postfix Expression Evaluation):

```
def postfix_evaluation(expression):
    stack = []

    for char in expression:
        if char.isdigit():
            stack.append(int(char))
        else:
            b = stack.pop()
            a = stack.pop()

            if char == '+':
                stack.append(a + b)
            elif char == '-':
                stack.append(a - b)
            elif char == '*':
                stack.append(a * b)
            elif char == '/':
                stack.append(a // b)

    return stack.pop()


exp = "23*54*+9-"
print("Postfix Expression :", exp)
print("Evaluation Result :", postfix_evaluation(exp))
```

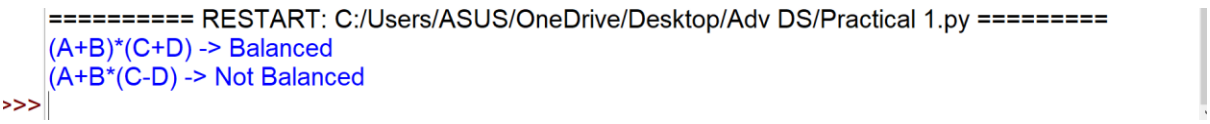
Output:



Code (Balancing of Parenthesis):

```
def is_balanced(expression):
    stack = []
    pairs = {')': '(', '}': '{', ']': '['}

    for char in expression:
        if char in "({[":
            stack.append(char)
        elif char in ")}]":
            if not stack or stack.pop() != pairs[char]:
                return False

    return len(stack) == 0



exp1 = "(A+B)*(C+D)"
exp2 = "(A+B*(C-D)"

print(exp1, "-> Balanced" if is_balanced(exp1) else "-> Not Balanced")
print(exp2, "-> Balanced" if is_balanced(exp2) else "-> Not Balanced")
```

Output:



Practical 5. Implementation of all types of linked lists.

Code (Singly Linked List):

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

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

    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

```
sll = SinglyLinkedList()
sll.insert(10)
sll.insert(20)
sll.insert(30)
print("Singly Linked List:")
sll.display()
```

Output:

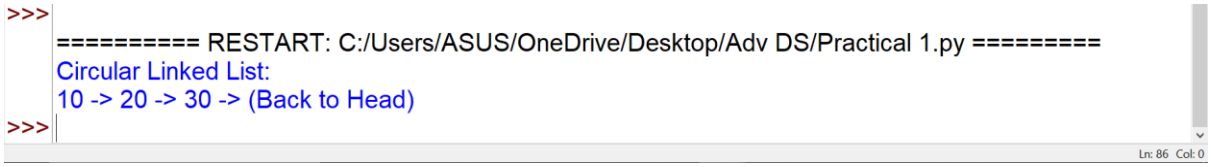
```
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
Singly Linked List:
10 -> 20 -> 30 -> None
```

Code (Circular Singly Linked List):

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

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

    def insert(self, data):
        new_node = CNode(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def display(self):
        if self.head is None:
            return
        temp = self.head
        while True:
```

```
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(Back to Head)")


cll = CircularLinkedList()
cll.insert(10)
cll.insert(20)
cll.insert(30)
print("Circular Linked List:")
cll.display()
```

Output:



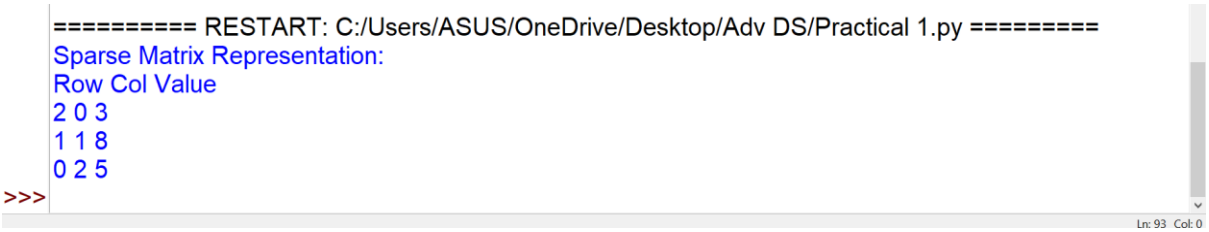
Practical 6. Demonstrate application of linked list (eg.Sparse matrix,Stack,Queue,Priority & Double ended Queue)

Code:

```
class SparseNode:
    def __init__(self, row, col, value):
        self.row = row
        self.col = col
        self.value = value
        self.next = None

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

    def insert(self, row, col, value):
        if value != 0:
            new_node = SparseNode(row, col, value)
            new_node.next = self.head
            self.head = new_node

    def display(self):
        temp = self.head
        print("Row Col Value")
        while temp:
            print(temp.row, temp.col, temp.value)
            temp = temp.next
```

```
sm = SparseMatrix()
sm.insert(0, 2, 5)
sm.insert(1, 1, 8)
sm.insert(2, 0, 3)

print("Sparse Matrix Representation:")
sm.display()
```

Output:



Practical 7. Create and perform various operations on BST.

Code:

```
class BSTNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None


class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert(self.root, data)

    def _insert(self, node, data):
        if node is None:
            return BSTNode(data)
        if data < node.data:
            node.left = self._insert(node.left, data)
        else:
            node.right = self._insert(node.right, data)
        return node

    def search(self, key):
```

```
        return self._search(self.root, key)

    def _search(self, node, key):
        if node is None:
            return False
        if node.data == key:
            return True
        if key < node.data:
            return self._search(node.left, key)
        return self._search(node.right, key)

    def inorder(self, node):
        if node:
            self.inorder(node.left)
            print(node.data, end=" ")
            self.inorder(node.right)

    def preorder(self, node):
        if node:
            print(node.data, end=" ")
            self.preorder(node.left)
            self.preorder(node.right)

    def postorder(self, node):
        if node:
            self.postorder(node.left)
            self.postorder(node.right)
```

```
            print(node.data, end=" ")



bst = BinarySearchTree()
elements = [50, 30, 70, 20, 40, 60, 80]

for ele in elements:
    bst.insert(ele)

print("Inorder Traversal:")
bst.inorder(bst.root)

print("\nPreorder Traversal:")
bst.preorder(bst.root)

print("\nPostorder Traversal:")
bst.postorder(bst.root)

key = 40
if bst.search(key):
    print("\nElement", key, "found in BST")
else:
    print("\nElement", key, "not found in BST")
```

Output:

```
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
Inorder Traversal:
20 30 40 50 60 70 80
Preorder Traversal:
50 30 20 40 70 60 80
Postorder Traversal:
20 40 30 60 80 70 50
Element 40 found in BST
>>>
```

Ln: 102 Col: 0

Practical 8. Implementing Heap with different operations.

Code:

```
class MaxHeap:
    def __init__(self):
        self.heap = []

    def insert(self, value):
        self.heap.append(value)
        self._heapify_up(len(self.heap) - 1)
        print(value, "inserted")

    def _heapify_up(self, index):
        parent = (index - 1) // 2
        while index > 0 and self.heap[index] > self.heap[parent]:
            self.heap[index], self.heap[parent] = self.heap[parent], self.heap[index]
            index = parent
            parent = (index - 1) // 2

    def extract_max(self):
        if len(self.heap) == 0:
            print("Heap is empty")
            return
        max_value = self.heap[0]
        self.heap[0] = self.heap[-1]
        self.heap.pop()
        self._heapify_down(0)
        print(max_value, "deleted")
```

```
    def _heapify_down(self, index):
        size = len(self.heap)
        largest = index
        left = 2 * index + 1
        right = 2 * index + 2

        if left < size and self.heap[left] > self.heap[largest]:
            largest = left
        if right < size and self.heap[right] > self.heap[largest]:
            largest = right

        if largest != index:
            self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]
            self._heapify_down(largest)

    def display(self):
        print("Heap elements:", self.heap)


h = MaxHeap()
h.insert(50)
h.insert(30)
h.insert(40)
h.insert(10)
h.insert(60)
```

```
h.display()
h.extract_max()
h.display()
```

Output:

```
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
50 inserted
30 inserted
40 inserted
10 inserted
60 inserted
Heap elements: [60, 50, 40, 10, 30]
60 deleted
Heap elements: [50, 30, 40, 10]
>>>
```

Practical 9. Create a Graph storage structure (eg. Adjacency matrix)

Code:

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adj_matrix = [[0 for _ in range(vertices)] for _ in range(vertices)]

    def add_edge(self, v1, v2):
        self.adj_matrix[v1][v2] = 1
        self.adj_matrix[v2][v1] = 1

    def display(self):
        print("Adjacency Matrix:")
        for row in self.adj_matrix:
            for value in row:
                print(value, end=" ")
            print()


g = Graph(4)

g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 2)
g.add_edge(2, 3)

g.display()
```

Output:

```
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
Adjacency Matrix:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
>>>
```

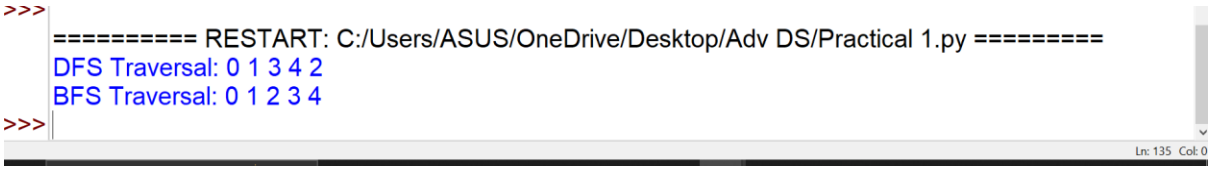
Practical 10. Implementation of Graph traversal. (DFS and BFS)

Code:

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = {i: [] for i in range(vertices)}

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)

    def dfs(self, start):
        visited = [False] * self.vertices
        print("DFS Traversal:", end=" ")
        self._dfs_util(start, visited)
        print()

    def _dfs_util(self, v, visited):
        visited[v] = True
        print(v, end=" ")
        for neighbour in self.graph[v]:
            if not visited[neighbour]:
                self._dfs_util(neighbour, visited)

    def bfs(self, start):
        visited = [False] * self.vertices
        queue = []
```

```
        visited[start] = True
        queue.append(start)

        print("BFS Traversal:", end=" ")

        while queue:
            v = queue.pop(0)
            print(v, end=" ")

            for neighbour in self.graph[v]:
                if not visited[neighbour]:
                    visited[neighbour] = True
                    queue.append(neighbour)
        print()


g = Graph(5)

g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 3)
g.add_edge(1, 4)

g.dfs(0)
g.bfs(0)
```

Output:


>>>
========== RESTART: C:/Users/ASUS/OneDrive/Desktop/Adv DS/Practical 1.py =========
DFS Traversal: 0 1 3 4 2
BFS Traversal: 0 1 2 3 4
>>>
Ln: 135 Col: 0