PERIYAR UNIVERSITY

(NAAC 'A++' Grade with CGPA 3.61 (Cycle - 3)
State University - NIRF Rank 56 - State Public University Rank 25
SALEM - 636 011

CENTRE FOR DISTANCE AND ONLINE EDUCATION

(CDOE)

MASTER OF COMPUTER APPLICATION

SEMESTER - II

CORE V : DATA STRUCTURES & ALGORITHMS LAB

(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

M.C.A 2024 admission onwards

CORE – V
Data Structures & Algorithms Lab

Prepared by:

Centre for Distance and Online Education (CDOE)
Periyar University
Salem - 636011

TABLE OF CONTENTS

Recursion concepts- Linear recursion & Binary Recursion

A) **Linear recursion**

Step 1: Start the Process.

Step 2: Define Recursive-linear- reverse function and takes parameters 's' and ‘index’

Step 3: If Index is equal to the length of String 's' returns empty String

Step 4: Increment index by One to Move to the Next Character in the String and Concatenate It with the Character al Index in 's’

Step 5: Define the Main Function call reverse-linear-recursive function with the input string and Index the Input 0 to Start the recursion

Step 6: To execute the Program and call the Main function to Start the execution.

Step 7: Stop the Process

B) **Binary recursion**

Step 1: Start the Process

Step 2: Dezion the Reverse binary- recursive function, and take thru Parameters ''S', 'low" and "high"

Step 3: if low is less than Or Equal to high Continue Otherwise returns the String

Step 4: Find the Middle Index (mid) low' and 'high'

Step 5: Swap the characters at Louw and high Indices in the ‘S_list’

Step 6: Recursively Call the function and update the String 's Increment by low' by Ore and Decrement high by

Step 7: Define the main function and display the Menu to user to inter A String to be reversed.

Step 8: Stop the Process

SOURCE CODE:

i) Linear recursion:

```
def reverse_linear_recursive(s, index):
    """Reverse a string using linear recursion."""
    if index == len(s):
        return ""
    else:
        return reverse_linear_recursive(s, index + 1) + s[index]

def main():
    print("====================================")
    print("\t\t\tMENU")
    print("====================================")
    print("[1] Linear Recursion method for Reverse a String")

    input_string = input("Enter a string to reverse: ")

    print("\n*Linear Recursion method for Reverse a String*")
    result_linear = reverse_linear_recursive(input_string, 0)
    print(f"The reversed string using linear recursion is :{result_linear}")

if __name__ == "__main__":
    main()
```

OUTPUT (Linear recursion**):**

```
=====================================================
                        MENU
=====================================================
[1] Linear Recursion method for Reverse a String
Enter a string to reverse: data structures

*Linear Recursion method for Reverse a String*
The reversed string using linear recursion is: serutcurts atad
```

SOURCE CODE:

ii)Binary recursion:

```
def reverse_binary_recursive(s, low, high):
    if low <= high:
        mid = (low + high) // 2
        s_list = list(s)
        s_list[low], s_list[high] = s_list[high], s_list[low]
        s = ''.join(s_list)
        return reverse_binary_recursive(s, low + 1, high - 1)
    return s
def main():
    print("=========================================")
    print("\t\t\t MENU")
    print("=========================================")
    print("[1] Binary Recursion method for Reverse a String")
    input_string = input("Enter a string to reverse: ")
    print("\n*Binary Recursion method for Reverse a String*")
    result_binary = reverse_binary_recursive(input_string, 0,
                                len(input_string)- 1)
    print(f"The reversed string using binary recursion is: {result_binary}")
if __name__ == "__main__":
    main()
```

OUTPUT (Binary recursion**):**

```
================================================
                    MENU
================================================
[1] Binary Recursion method for Reverse a String
Enter a string to reverse: soft computing

*Binary Recursion method for Reverse a String*
The reversed string using binary recursion is: gnitupmoc tfos
```

RESULT

Thus the above program has been executed successfully.

Stack ADT

Step 1: Start the process

step 2: Define a class stack with 'arr' to store elements, capacity and top

step 3: using push operation to add an element at top of the stack

step 4: Using "pop' method to remove and return the top element from the stack.

Step 5: Using 'peek 'method to return the top element without removing it.

Step 6: using ‘size’ method to return the current size of stack.

Step 7: Using 'isEmpty' method to check if the stack is empty, return true if stack is and otherwise false

step 8: using ‘isfull' operation to check if the stack is full. returns true off the stack is equal to capacity Otherwise false.

Step-9: Define main stack Function and display menu and prompt

SOURCE CODE:

```
class Stack:
    def __init__(self, size):
        self.arr = [None] * size
        self.capacity = size
        self.top = -1
    def push(self, x):
        if self.isFull():
            print("Stack Overflow!! Calling exit()…")
            exit(1)
        print(f"Inserting {x} into the stack…")
        self.top += 1
        self.arr[self.top] = x
    def pop(self):
        if self.isEmpty():
            print("Stack Underflow!! Calling exit()…")
            exit(1)
        print(f"Removing {self.peek()} from the stack")
        top = self.arr[self.top]
        self.top -= 1
        return top
    def peek(self):
        if self.isEmpty():
            exit(1)
        return self.arr[self.top]
    def size(self):
        return self.top + 1
    def isEmpty(self):
        return self.size() == 0
    def isFull(self):
        return self.size() == self.capacity
def main_stack():
    stack_size = int(input("Enter the size of the stack: "))
    stack = Stack(stack_size)
    print("\nMenu:")
    print("1. Push element to the stack")
    print("2. Pop element from the stack")
    print("3. Peek at the top element")
```

```
        print("4. Check if the stack is empty")
        print("5. Check if the stack is full")
        print("6. Exit")
        while True:
            choice = int(input("Enter your choice: "))
            if choice == 1:
                element = int(input("Enter the element to push: "))
                stack.push(element)
            elif choice == 2:
                popped_element = stack.pop()
                print(f"Popped element: {popped_element}")
            elif choice == 3:
                print("Top element:", stack.peek())
            elif choice == 4:
                print("Is the stack empty?", stack.isEmpty())
            elif choice == 5:
                print("Is the stack full?", stack.isFull())
            elif choice == 6:
                print("Exiting the program.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")
if __name__ == '__main__':
    main_stack()
```

OUTPUT:

```
Enter the size of the stack: 3

Menu:
1. Push element to the stack
2. Pop element from the stack
3. Peek at the top element
4. Check if the stack is empty
5. Check if the stack is full
6. Exit
Enter your choice: 1
Enter the element to push: 4
Inserting 4 into the stack…
Enter your choice: 1
Enter the element to push: 5
Inserting 5 into the stack…
Enter your choice: 2
Removing 5 from the stack
Popped element: 5
Enter your choice: 3
Top element: 4
Enter your choice: 5
Is the stack full? False
Enter your choice: 4
Is the stack empty? False
Enter your choice: 6
Exiting the program.
```

RESULT

Thus the above program has been executed successfully.

QUEUE

Step1: Start the process.

Step2: Defining a class 'Queue' and define a constructor using –init—() method.

Step 3: Using 'enqueue (item)' method to appends the item to the end of the queue.

Step4: Using 'deque method its remove and return the first item from the queue.

Step5: Vising is empty()' method its check the queue is empty then returns True, otherwise false.

Step6: Using size() method its return the size of the queue.

Step 7: Vising 'front() method to returns the first element of the 1 otherwise queue is empty returns None.

Step 8: Using 'Rear() method its return the last element of the queue, otherwise is empty return None,

Step 9: 'display' method to Print all the elements of the queue.

Step 10: Define a function 'main- queue to inside this function create instance of class "Queue"

Step 11: Vising while loop a Meru and take user input. display

Step 12: stop the process.

SOURCE CODE:

```
class Queue:
    def __init__(self):
        self.items = []
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        else:
            print("Queue is empty.")
            return None
    def is_empty(self):
        return len(self.items) == 0
    def size(self):
        return len(self.items)
    def front(self):
        if not self.is_empty():
            return self.items[0]
        else:
            print("Queue is empty.")
            return None
    def rear(self):
        if not self.is_empty():
            return self.items[-1]
        else:
            print("Queue is empty.")
            return None

    def display(self):
        print("Queue:", self.items)

def main_queue():
    queue = Queue()
    print("\nMenu:")
    print("1. Enqueue element")
    print("2. Dequeue element")
    print("3. Check if the queue is empty")
    print("4. Display queue size")
    print("5. Display front element")
    print("6. Display rear element")
```

```
        print("7. Display queue")
        print("8. Exit")
        while True:
            choice = int(input("Enter your choice: "))
            if choice == 1:
                item = input("Enter the element to enqueue: ")
                queue.enqueue(item)
                print(f"{item} enqueued.")
            elif choice == 2:
                dequeued_item = queue.dequeue()
                if dequeued_item is not None:
                    print(f"{dequeued_item} dequeued.")
            elif choice == 3:
                print("Is the queue empty?", queue.is_empty())
            elif choice == 4:
                print("Queue size:", queue.size())
            elif choice == 5:
                front_element = queue.front()
                if front_element is not None:
                    print("Front element:", front_element)
            elif choice == 6:
                rear_element = queue.rear()
                if rear_element is not None:
                    print("Rear element:", rear_element
             elif choice == 7:
                queue.display()
            elif choice == 8:
                print("Exiting the program.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")
if __name__ == '__main__':
    main_queue()
```

OUTPUT:

```
Menu:
1. Enqueue element
2. Dequeue element
3. Check if the queue is empty
4. Display queue size
5. Display front element
6. Display rear element
7. Display queue
8. Exit
Enter your choice: 1
Enter the element to enqueue: 3
3 enqueued.
Enter your choice: 1
Enter the element to enqueue: 5
5 enqueued.
Enter your choice: 1
Enter the element to enqueue: 6
6 enqueued.
Enter your choice: 2
3 dequeued.
Enter your choice: 3
Is the queue empty? False
Enter your choice: 4
Queue size: 2
Enter your choice: 5
Front element: 5
Enter your choice: 6
Rear element: 6
Enter your choice: 7
Queue: ['5', '6']
Enter your choice: 8
Exiting the program.
```

RESULT

Thus the above program has been executed successfully.

DOUBLY LINKED LIST ADT

Step1: Start the process.

Step 2: Define a node class with three fields ‘_element’, ‘_ link', ‘_ prev’

Step 3: Define a 'Doubly LL' class its perform operations on Linked List.

Step 4: Using __len__ method its return the length of linked list.

Step 5: Using ‘isempty' method to check DLL is empty.

Step 6: Using 'Add Last’ method to add an element at end of DLL.

Step 7: Using 'Add First’ method to add an element at first of DLL

Step 8: Using 'Add Anywhere' method to add index. an element at the specified

Step 9: Using 'Remove First’ method to Remove an element from the beginning, the Uses to perform stack operation.

Step 10: Execute ‘main stack' function if the script is in as the main Program.

Step 11: Stop the process.

SOURCE CODE:

```
import os
class _Node:
    __slots__ = '_element', '_link', '_prev'

    def __init__(self, element, link, prev):
        self._element = element
        self._link = link
        self._prev = prev

class DoublyLL:
    def __init__(self):
        self._head = None
        self._tail = None
        self._size = 0

    def __len__(self):
        return self._size

    def isempty(self):
        return self._size == 0

    def addLast(self, e):
        newest = _Node(e, None, None)
        if self.isempty():
            self._head = newest
        else:
            self._tail._link = newest
            newest._prev = self._tail
        self._tail = newest
        self._size += 1

    def addFirst(self, e):
        newest = _Node(e, None, None)
        if self.isempty():
            self._head = newest
            self._tail = newest
        else:
            newest._link = self._head
            self._head._prev = newest
```

```
            self._head = newest
        self._size += 1

    def addAnywhere(self, e, index):
        if index > self._size:
            print(f'Index value out of range, it should be between 0 –
                    {self._size}')
        elif index == self._size:
            self.addLast(e)
        elif index == 0:
            self.addFirst(e)
        else:
            newest = _Node(e, None, None)
            p = self._head
            for _ in range(index - 1):
                p = p._link
            newest._link = p._link
            p._link._prev = newest
            newest._prev = p
            p._link = newest
            self._size += 1

    def removeFirst(self):
        if self.isempty():
            print('List is already empty')
            return

        e = self._head._element
        self._head = self._head._link
        self._size -= 1
        if self.isempty():
            self._tail = None
        else:
            self._head._prev = None
        return e

    def removeLast(self):
        if self.isempty():
            print("List is already empty")
            return
        e = self._tail._element
        self._tail = self._tail._prev
```

```
            self._size -= 1
            if self.isempty():
                self._head = None
            else:
                self._tail._link = None
            return e

        def removeAnywhere(self, index):
            if index >= self._size:
                print(f'Index value out of range, it should be between 0 –
                        {self._size - 1}')
            elif self.isempty():
                print("List is empty")
            elif index == 0:
                return self.removeFirst()
            elif index == self._size - 1:
                return self.removeLast()
            else:
                p = self._head
                for _ in range(index - 1):
                    p = p._link
                e = p._link._element
                p._link = p._link._link
                p._link._prev = p
                self._size -= 1
                return e

    def display(self):
        if self.isempty():
            print("List is Empty")
            return
        p = self._head
        print("NULL<-->", end='')
        while p:
            print(p._element, end="<-->")
            p = p._link
        print("NULL")
        print(f"\nHead : {self._head._element}, Tail : {self._tail._element}")

def options():
    options_list = ['Add Last', 'Add First', 'Add Anywhere', 'Remove First',
                    'Remove Last', 'Remove Anywhere', 'Display List', 'Exit']
```

```
        print("MENU")
        for i, option in enumerate(options_list):
            print(f'{i + 1}. {option}')
        choice = int(input("Enter choice: "))
        return choice
def switch_case(choice):
        os.system('cls')
        if choice == 1:
            elem = int(input("Enter Item: "))
            DL.addLast(elem)
            print("Added Item at Last!\n\n")
        elif choice == 2:
            elem = int(input("Enter Item: "))
            DL.addFirst(elem)
            print("Added Item at First!\n\n")
        elif choice == 3:
            elem = int(input("Enter Item: "))
            index = int(input("Enter Index: "))
            DL.addAnywhere(elem, index)
        elif choice == 4:
            print("Removed Element from First:", DL.removeFirst())
        elif choice == 5:
            print("Removed Element from last:", DL.removeLast())
        elif choice == 6:
            index = int(input("Enter Index: "))
            print(f"Removed Item: {DL.removeAnywhere(index)} !\n\n")
        elif choice == 7:
            print("List:")
            DL.display()
            print("\n")
        elif choice == 8:
            import sys
            sys.exit()
    if __name__ == '__main__':
        DL = DoublyLL()
        while True:
            try:
                choice = options()
                switch_case(choice)
            except SystemExit:
                print("Exiting the program.")
                break
```

OUTPUT:

```
MENU
1. Add Last
2. Add First
3. Add Anywhere
4. Remove First
5. Remove Last
6. Remove Anywhere
7. Display List
8. Exit
Enter choice: 2
Enter Item: 4
Added Item at First!


MENU
1. Add Last
2. Add First
3. Add Anywhere
4. Remove First
5. Remove Last
6. Remove Anywhere
7. Display List
8. Exit
Enter choice: 1
Enter Item: 2
Added Item at Last!



MENU
1. Add Last
2. Add First
3. Add Anywhere
4. Remove First
5. Remove Last
6. Remove Anywhere
7. Display List
8. Exit
Enter choice: 4
Removed Element from First: 4
MENU
1. Add Last
2. Add First
3. Add Anywhere
4. Remove First
5. Remove Last
6. Remove Anywhere
7. Display List
8. Exit
Enter choice: 7
List:
NULL<-->2<-->NULL

Head : 2, Tail : 2


MENU
1. Add Last
2. Add First
3. Add Anywhere
4. Remove First
5. Remove Last
6. Remove Anywhere
7. Display List
8. Exit
Enter choice: 8
Exiting the program.
```

RESULT

Thus the above program has been executed suc

HEAPS USING PRIORITY QUEUES

Step1: Start the process.

Step2: Importing the heapq module, which Provides heap queue algorithm implementations

Step3: The class "ManHeap" is defined which represent map heap map structure

Step 4: The “insert" method is defined to insert an item into man heap.

Step 5: "Delete" method is defined to delete the highest priority element from man heap

Step 6: The "Peek" method is defined to View the highest priority element from man heap without removing it

Step 7 : The "Size method returns the size of heap which is the length of ‘heap' list

Step 8:An instance of priority Queue() clam is named “pq” is created.

Step9 : Taking the input from the User using while loop.

Step 10: stop the process

SOURCE CODE:

```
import heapq

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

    def insert(self, item):
        heapq.heappush(self._heap, -item)

    def delete(self):
        if len(self._heap) > 0:
            return -heapq.heappop(self._heap)
        else:
            raise Exception("Heap is empty")

    def peek(self):
        if len(self._heap) > 0:
            return -self._heap[0]
        else:
            raise Exception("Heap is empty")

    def size(self):
        return len(self._heap)


class PriorityQueue:
    def __init__(self):
        self.heap = MaxHeap()

    def push(self, item, priority):
        self.heap.insert(priority)

    def pop(self):
        if self.is_empty():
            raise IndexError("pop from an empty priority queue")
        return self.heap.delete()


    def is_empty(self):
        return self.heap.size() == 0


pq = PriorityQueue()

print("\nMENU:")
print("1. Add Task and Priority")
print("2. Remove Task with Highest Priority")
print("3. Exit")
```

```
while True:
    choice = input("Enter your choice: ")

    if choice == "1":
        task = input("Enter the task: ")
        priority = int(input("Enter the priority: "))
        pq.push(task, priority)
        print(f"Task '{task}' with priority {priority} added to the
              priority queue.")
    elif choice == "2":
        try:
            removed_task = pq.pop()
            print(f"Task '{removed_task}' with highest priority removed
                  from the priority queue.")
        except IndexError:
            print("Priority queue is empty. Cannot remove task.")
    elif choice == "3":
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please enter a valid option.")
```

OUTPUT:

```
MENU:
1. Add Task and Priority
2. Remove Task with Highest Priority
3. Exit
Enter your choice: 1
Enter the task: fun 1
Enter the priority: 3
Task 'fun 1' with priority 3 added to the priority queue.
Enter your choice: 1
Enter the task: fun 2
Enter the priority: 2
Task 'fun 2' with priority 2 added to the priority queue.
Enter your choice: 1
Enter the task: fun 3
Enter the priority: 1
Task 'fun 3' with priority 1 added to the priority queue.
Enter your choice: 2
Task '3' with highest priority removed from the priority queue.
Enter your choice: 3
Exiting the program.
```

RESULT

Thus the above program has been executed successfully.

MERGE SORT

Step1: Start the process

Step 2: Define the function "merge_sort"

Step3: Check if the array length is for less, if it is less, the array is already

Step 4: Calculate the mid. point of array. it divides array into two halves

Step 5: Split the array into left and right halves, using python slicing

Step 6: Recursively call merge sort on right & left halves until the base care is reached.

Step7: Define the "Merge "function"

Step8: Initialize variables i, & for iterating through both arrays compare elements of both arrays & merge

Step 9: Compare elements of both arrays & merge

Step 10: Append remaining elements and return the merged result array.

Step11:Stop the process.

SOURCE CODE:

```
 def merge_sort(arr):if
len(arr)<=1: return arr
    mid=len(arr)//2
    left=arr[:mid]
    right=arr[mid:]

  left=merge_sort(left)
  right=merge_sort(right)
  return merge(left,right)

def merge(left,right):
  result=[]
  i,j=0,0
  while i<len(left) and j<len(right):if
    left[i]<right[j]:
      result.append(left[i])i+=1
    else:
      result.append(right[j])j+=1

  result+=left[i:]
  result+=right[j:]
  return result
arr=input("Enter the list of numbers separated by space:").split()arr=[int(num)
for num in arr]
print("Input Array:",arr)
sorted_arr=merge_sort(arr)
print("Sorted Array:",sorted_arr)
```

OUTPUT:

```
Enter the list of numbers separated by space: 43 45 23 35 65 66
Input Array: [43, 45, 23, 35, 65, 66]
Sorted Array: [23, 35, 43, 45, 65, 66]
```

RESULT

Thus the above program has been executed successfully.

QUICK SORT

Step 1: Start the process

Step 2: Accepts a list, start index and end index and check if the Sublist has more than one element.

Step 3: Select the first clement of the Sublist as the pivot

Step 4 : Initializes two pointers i and j at the start and end Sublist. of the

Step 5: Return the index; as the new Per Pivot position

Step 6: Prompts the user to enter a list of numbers

Step 7: Convert the input string of numbers into a list of integers.

Step 8: Call the quicksort function to sort the list and print the Sorted list

Step 9: Stop the process

SOURCE CODE:

```
def quicksort(alist, start, end):
    '''Sorts the list from indexes start to end - 1 inclusive.'''
    if end - start > 1:
        p = partition(alist, start, end)
        quicksort(alist, start, p)
        quicksort(alist, p + 1, end)

def partition(alist, start, end):
    pivot = alist[start]
    i = start + 1
    j = end - 1

    while True:
        while i <= j and alist[i] <= pivot:
            i = i + 1
        while i <= j and alist[j] >= pivot:
            j = j - 1

        if i <= j:
            alist[i], alist[j] = alist[j], alist[i]
        else:
            alist[start], alist[j] = alist[j], alist[start]
            return j

alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
quicksort(alist, 0, len(alist))
print('Sorted list:', alist)
```

OUTPUT:

```
Enter the list of numbers: 34 56 23 54 56 38
Sorted list: [23, 34, 38, 54, 56, 56]
```

RESULT

Thus the above program has been executed successfully.

BINARY SEARCH TREE

Step 1: Start the process

Step 2: Initializes a BST node with optional initial value.

Step 3: Inserts a new based node into the BST on its value.

Step 4: Traverses the tree level by level visiting all nodes at each level before moving to the next level

Step 5: Performs in-order, pre-order and post-order traversals recursively.

Step 6: Delete a node from the BST while maintaining the BST property.

Step 7: Displays a menu fox the user interact with the BST

Step 8 : Allows the user to insert elements, perform BFS or DFS, delete nodes, or exit the program.

Step 9: Stop the process

SOURCE CODE:

```
class binarySearchTree:
    def __init__(self, val=None):
        self.val = val
        self.left = None
        self.right = None

    def insert(self, val):
        if self.val is None:
            self.val = val
        else:
            if val == self.val:
                return 'No duplicates allowed in the binary search tree'
            if val < self.val:
                if self.left:
                    self.left.insert(val)
                else:
                    self.left = binarySearchTree(val)
            else:
                if self.right:
                    self.right.insert(val)
                else:
                    self.right = binarySearchTree(val)

    def breadthFirstSearch(self):
        currentNode = self
        bfs_list = []
        queue = []
        queue.insert(0, currentNode)
        while len(queue) > 0:
            currentNode = queue.pop()
            bfs_list.append(currentNode.val)
            if currentNode.left:
                queue.insert(0, currentNode.left)
            if currentNode.right:
                queue.insert(0, currentNode.right)
        return bfs_list
```

```
def depthFirstSearch_INorder(self):
    return self.traverseInOrder([])

def depthFirstSearch_PREorder(self):
    return self.traversePreOrder([])

def depthFirstSearch_POSTorder(self):
    return self.traversePostOrder([])

def traverseInOrder(self, lst):
    if self.left:
        self.left.traverseInOrder(lst)
    lst.append(self.val)
    if self.right:
        self.right.traverseInOrder(lst)
    return lst

def traversePreOrder(self, lst):
    lst.append(self.val)
    if self.left:
        self.left.traversePreOrder(lst)
    if self.right:
        self.right.traversePreOrder(lst)
    return lst

def traversePostOrder(self, lst):
    if self.left:
        self.left.traversePostOrder(lst)
    if self.right:
        self.right.traversePostOrder(lst)
    lst.append(self.val)
    return lst

def findNodeAndItsParent(self, val, parent=None):
    if val == self.val:
        return self, parent
```

```
        if val < self.val:
            if self.left:
                return self.left.findNodeAndItsParent(val, self)
            else:
                return 'Not found'
        else:
            if self.right:
                return self.right.findNodeAndItsParent(val, self)
            else:
                return 'Not found'

    def delete(self, val):
        if self.findNodeAndItsParent(val) == 'Not found':
            return 'Node is not in tree'

        deleting_node, parent_node = self.findNodeAndItsParent(val)
        nodes_effected = deleting_node.traversePreOrder([])

        if len(nodes_effected) == 1:
            if parent_node is not None:
                if parent_node.left == deleting_node:
                    parent_node.left = None
                else:
                    parent_node.right = None
            else:
                self.val = None
                self.left = None
                self.right = None
        else:
            if parent_node is None:
                nodes_effected.remove(deleting_node.val)
                self.left = None
                self.right = None
                self.val = None
                for node in nodes_effected:
                    self.insert(node)
            else:
                nodes_effected.remove(deleting_node.val)
```

```
                if parent_node.left == deleting_node:
                    parent_node.left = None
                else:
                    parent_node.right = None
                if parent_node is not None:
                    nodes_effected.remove(parent_node.val)
                for node in nodes_effected:
                    self.insert(node)
                    return 'Successfully deleted'
    bst = binarySearchTree()
    print("\nMENU:")
    print("1. Insert Element")
print("2. Breadth-First Search")
print("3. Depth-First Search (In-Order)")
print("4. Depth-First Search (Pre-Order)")
print("5. Depth-First Search (Post-Order)")
print("6. Delete Node")
print("7. Exit")
    while True:
        choice = input("Enter your choice: ")
        if choice == "1":
            element = int(input("Enter the element to insert: "))
            print(bst.insert(element))
        elif choice == "2":
            print("Breadth-First Search:", bst.breadthFirstSearch())
        elif choice == "3":
            print("Depth-First Search (In-Order):", bst.depthFirstSearch_INorder())
        elif choice == "4":
            print("Depth-First Search (Pre-Order):", bst.depthFirstSearch_PREorder())
        elif choice == "5":
            print("Depth-First Search (Post-Order):", bst.depthFirstSearch_POSTorder())
        elif choice == "6":
            node_to_delete = int(input("Enter the node value to delete: "))
            print(bst.delete(node_to_delete))
        elif choice == "7":
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please enter a valid option.")
```

OUTPUT:

```
MENU:
1. Insert Element
2. Breadth-First Search
3. Depth-First Search (In-Order)
4. Depth-First Search (Pre-Order)
5. Depth-First Search (Post-Order)
6. Delete Node
7. Exit
Enter your choice: 1
Enter the element to insert: 3
None
Enter your choice: 1
Enter the element to insert: 5
None
Enter your choice: 1
Enter the element to insert: 6
None
Enter your choice: 2
Breadth-First Search: [3, 5, 6]
Enter your choice: 3
Depth-First Search (In-Order): [3, 5, 6]
Enter your choice: 4
Depth-First Search (Pre-Order): [3, 5, 6]
Enter your choice: 5
Depth-First Search (Post-Order): [6, 5, 3]
Enter your choice: 6
Enter the node value to delete: 6
Successfully deleted
Enter your choice: 7
Exiting the program.
```

RESULT

Thus the above program has been executed successfully.

MINIMUM SPANNING TREE

Step1: Define graph class: create a class named graph, initializing it with attributes for vertices v and empty graph list.

Step 2: Implement add-edge method in graph class to include edges with their weights.

Step 3 Develop search (parent,i) to find the parent of a node i in union-find structure.

Step4: Implement union operators to merge subsets based on rank

Step5: implement kruskal's algorithm: initialize list for minimum spanning tree, sort edges by weight.

Step6: Instantiate graph, apply kruskal's algorithm and print resulting minimum spanning tree.

Step 7: Verify code functionality, ensuring each step is correctly implemented and produces the expected output.

Step 8: Stop the process

SOURCE CODE:

```
class Graph:
    def __init__(self, vertex):
        self.V = vertex
        self.graph = []

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

    def search(self, parent, i):
        if parent[i] == i:
            return i
        return self.search(parent, parent[i])

    def apply_union(self, parent, rank, x, y):
        xroot = self.search(parent, x)
        yroot = self.search(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    def kruskal(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = [i for i in range(self.V)]
        rank = [0] * self.V
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.search(parent, u)
            y = self.search(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.apply_union(parent, rank, x, y)
```

```
            for u, v, weight in result:
                print("Edge:", u, v, "-", weight)
def main():
    num_vertices = int(input("Enter the number of vertices: "))
    g = Graph(num_vertices)
    num_edges = int(input("Enter the number of edges: "))
    print("Enter edges in the format 'source destination weight':")
    for _ in range(num_edges):
        u, v, w = map(int, input().split())
        g.add_edge(u, v, w)

    print("Minimum Spanning Tree (Kruskal's Algorithm):")
    g.kruskal()


if __name__ == "__main__":
    main()
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter edges in the format 'source destination weight':
1 3 5
2 4 6
3 5 7
4 6 8
5 7 9
Minimum Spanning Tree (Kruskal's Algorithm):
Edge: 1 3 - 5
Edge: 2 4 - 6
Edge: 3 5 - 7
```

RESULT

Thus the above program has been executed successfully.

DEPTH FIRST SEARCH

Step1: Start the process

Step2: Define a class named graph, implement (u, v) method to add edge.

Step3: Implement start vertex method within the graph class to initialize DFS traversal.

Step 4: Initialize a visited list to keep track of visited vertices.

Step5: prompt the user for the number of vertices and edges ,store the input value for further processsing.

Step6: Instantiate the graph class to work with and performing DFS.

Step7: Get the input of starting vertex for DFS traversal from the user.

Step 8: call the dfs method with the starting vertex to perform dfs and begin traversing the graph from the specified starting vertex.

Step9: Display the result of dfs traversal, showing the order of vertices visited starting from the specified vertex.
Step10: stop the process.

SOURCE CODE:

```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
    def add_edge(self, u, v):
        self.graph[u].append(v)
    def dfs_util(self, v, visited):
        visited[v] = True
        print(v, end=' ')
        for i in self.graph[v]:
            if not visited[i]:
                self.dfs_util(i, visited)
    def dfs(self, start_vertex):
        max_vertex=max(max(self.graph.keys()),
                       max(max(self.graph.values())))
        visited = [False] * (max_vertex + 1)
        self.dfs_util(start_vertex, visited)

num_vertices = int(input("Enter the number of vertices: "))
g=Graph()

num_edges = int(input("Enter the number of edges: "))
for _ in range(num_edges):
  u, v = map(int, input("Enter edge (u v): ").split())
  g.add_edge(u, v)

start_vertex = int(input("Enter the starting vertex for DFS: "))
print("Depth First Search (DFS) starting from vertex", start_vertex, ":")
g.dfs(start_vertex)
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edge (u v): 4 5
Enter edge (u v): 5 6
Enter edge (u v): 6 7
Enter edge (u v): 7 8
Enter the starting vertex for DFS: 6
Depth First Search (DFS) starting from vertex 6 :
6 7 8
```

RESULT

Thus the above program has been executed successfully.