MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

LABORATORY MANUAL & RECORD

Name:.....

Roll No:.....Branch:.....

Year:.....Sem:.....

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

Certificate

Certified that this is the Bonafide Record of the Work Done by

Mr./Ms.....RoII.No.....of

B.Tech.....year..... Semester for Academic year.....

in.....Laboratory.

Date: Faculty Incharge HOD

Internal Examiner External Examiner

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

DATA STRUCTURES USING PYTHON LABMANUAL

B.TECH

(II YEAR – I SEM)
(2023-24)

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

(AIML)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGCACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally(Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Vision

To be a premier centre for academic excellence and research through innovative interdisciplinary collaborations and making significant contributions to the community, organizations, and society as a whole.

Mission

- To impart cutting-edge Artificial Intelligence technology in accordance with industry norms.
- To instill in students a desire to conduct research in order to tackle challenging technical problems for industry.
- To develop effective graduates who are responsible for their professional growth, leadership qualities and are committed to lifelong learning.

Quality Policy

- To provide sophisticated technical infrastructure and to inspire students to reach their full potential.
- To provide students with a solid academic and research environment for a comprehensive learning experience.
- To provide research development, consulting, testing, and customized training to satisfy specific industrial demands, thereby encouraging self-employment and entrepreneurship among students.

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Programme Educational Objectives (PEO):

PEO1: To possess knowledge and analytical abilities in areas such as maths, science, and fundamental engineering.

PEO2: To analyse, design, create products, and provide solutions to problems in Computer Science and Engineering.

PEO3: To leverage the professional expertise to enter the workforce, seek higher education, and conduct research on AI-based problem resolution.

PEO4: To be solution providers and business owners in the field of computer science and engineering with an emphasis on artificial intelligence and machine learning.

Programme Specific Outcomes (PSO):

After successful completion of the program a student is expected to have specific abilities to:

PSO1: To understand and examine the fundamental issues with AI and ML applications.

PSO2: To apply machine learning, deep learning, and artificial intelligence approaches to address issues in social computing, healthcare, vision, language processing, speech recognition, and other domains.

PSO3: Use cutting-edge AI and ML tools and technology to further your study and research.

PROGRAMOUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), thosewho come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab withthe synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm,Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any)needed in the lab.

c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer systemallotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observationnote book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which shouldbe utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours willbe treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

INDEX

S.No	Name of the program	Page No
1.	Write a Python program for class, Flower, that has three instance variables of type str, int, and float that respectively represent the name of the flower, its number of petals, and its price. Your class must include a constructor method that initializes each variable toan appropriate value, and your class should include methods forsetting the value of each type, and retrieving the value of each type.	1
2.	Develop an inheritance hierarchy based upon a Polygon class thathas abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon, that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods. Write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.	5
3.	Write a python program to implement Method Overloading andMethod Overriding.	8
4.	Write a program for Linear Search and Binary search	11
5.	Write a program to implement Bubble Sort and Selection Sort	14
6.	Write a program to implement Merge sort and Quick sort	16
7.	Write a program to implement Stacks and Queues	19
8.	Write a program to implement Singly Linked List	25
9.	Write a program to implement Doubly Linked List	33
10.	Write a python program to implement DFS & BFS graph Traversal Techniques.	41
11.	Write a program to implement Binary Search Tree	47
12.	Write a program to implement B+ Tree	53

1. Write a Python program for class, Flower, that has three instance variables of type str, int, and float, that respectively represent the name of the flower, its number of petals, and its price. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type, and retrieving the value of each type.

Program:

```
class Flower:
#Common base class for all Flowers
    def_init_(self, petalName, petalNumber, petalPrice):
        self.name = petalName
        self.petals = petalNumber
        self.price = petalPrice

    def setName(self, petalName):
        self.name = petalName

    def setPetals(self, petalNumber):
        self.petals = petalNumber

    def setPrice(self, petalPrice):
        self.price = petalPrice

    def getName(self):
        return self.name

    def getPetals(self):
        return self.petals

    def getPrice(self):
        return self.price


#This would create first object of Flower class
f1 = Flower("Sunflower", 2, 1000)
print ("Flower Details:")
print ("Name: ", f1.getName())
print ("Number of petals:", f1.getPetals())
print ("Price:",f1.getPrice())

print ("\n")

#This would create second object of Flower class
f2 = Flower("Rose", 5, 2000)
f2.setPrice(3333)
f2.setPetals(6)
print ("Flower Details:")
print ("Name: ", f2.getName())
print ("Number of petals:", f2.getPetals())
print ("Price:",f2.getPrice())
```

Output:

Signature of the Faculty

Exercise Programs:

2. Develop an inheritance hierarchy based upon a Polygon class that has abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon, that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods. Write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.

Program:

```
from abc import abstractmethod, ABCMeta
import math

class Polygon(metaclass = ABCMeta):
    def___init__(self, side_lengths = [1,1,1],  num_sides = 3):
        self._side_lengths = side_lengths
        self._num_sizes = 3

    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

    def___repr__(self):
        return (str(self._side_lengths))



class Triangle(Polygon):
    def___init__(self, side_lengths):
        super()._init (side_lengths, 3)
        self._perimeter = self.perimeter()
        self._area = self.area()


    def perimeter(self):
        return(sum(self._side_lengths))

    def area(self):

        #Area of Triangle
        s = self._perimeter/2
        product = s
        for i in self._side_lengths:
            product*=(s-i)
        return product**0.5


class Quadrilateral(Polygon):
    def___init__(self, side_lengths):
        super()._init (side_lengths, 4)
        self._perimeter = self.perimeter()
```

```
        self._area = self.area()


    def perimeter(self):
        return(sum(self._side_lengths))

    def area(self):

        # Area of an irregular Quadrilateral
        semiperimeter = sum(self._side_lengths) / 2
        return math.sqrt((semiperimeter - self._side_lengths[0]) *
                    (semiperimeter - self._side_lengths[1]) *
                    (semiperimeter - self._side_lengths[2]) *
                    (semiperimeter - self._side_lengths[3]))

class Pentagon(Polygon):
    def __init__(self, side_lengths):
        super().__init__(side_lengths, 5)
        self._perimeter = self.perimeter()
        self._area = self.area()

    def perimeter(self):
        return((self._side_lengths) * 5)

    def area(self):

        # Area of a regular Pentagon
        a = self._side_lengths
        return (math.sqrt(5 * (5 + 2 * (math.sqrt(5)))) * a * a) /
        4

#object  of  Triangle
t1 = Triangle([1,2,2])
print(t1.perimeter(),
t1.area())

#object of Quadrilateral
q1 = Quadrilateral([1,1,1,1])
print(q1.perimeter(),
q1.area())
```

Output:

Signature of the Faculty

Exercise Programs:

3. Write a python program to implement method overloading and method overriding.

Method Overloading

Method overloading is an OOPS concept which provides ability to have several methods having the same name with in the class where the methods differ in types or number of arguments passed.

Method overloading in Python

Method overloading in its traditional sense (as defined above) as exists in other languages like method overloading in Java doesn't exist in Python.

In Python if you try to overload a function by having two or more functions having the same name but different number of arguments only the last defined function is recognized, calling any other overloaded function results in an error.

Achieving method overloading

Since using the same method name again to overload the method is not possible in Python, so achieving method overloading in Python is done by having a single method with several parameters. Then you need to check the actual number of arguments passed to the method and perform the operation accordingly.

Program:

```
class OverloadDemo:
  # sum method with default as None for parameters
  def sum(self, a=None, b=None, c=None):
    # When three params are passed
    if a!=None and b!=None and c!=None:
      s = a + b + c
      print('Sum = ', s)
    # When two params are passed
    elif a!=None and b!=None:
      s = a + b
      print('Sum = ', s)

od = OverloadDemo()
od.sum(7, 8)
od.sum(7, 8, 9)
```

Output:

Method overriding - Polymorphism through inheritance

Method overriding provides ability to change the implementation of a method in a child class which is already defined in one of its super class. If there is a method in a super class and method having the same name and same number of arguments in a child class then the child class method is said to be overriding the parent class method.

When the method is called with parent class object, method of the parent class is executed. When method is called with child class object, method of the child class is executed. So the appropriate overridden method is called based on the object type, which is an example of Polymorphism.

Program:

```
class Person:
  def___init__(self, name, age):
    self.name = name
    self.age = age

  def displayData(self):
    print('In parent class displayData method')
    print(self.name)
    print(self.age)

class Employee(Person):
  def___init__(self, name, age, id):
    # calling constructor of super class
    super().__init__(name, age)
    self.empId = id

  def displayData(self):
    print('In child class displayData method')
    print(self.name)
    print(self.age)
    print(self.empId)

#Person class object
person = Person('Karthik Shaurya', 26)
person.displayData()
#Employee class object
emp = Employee(''Karthik Shaurya', 26, 'E317')
emp.displayData()
```

Output:

Signature of the Faculty

4.Write a program for Linear Search and Binary search

Linear Search Program:

```
def linearSearch(target, List):
    position = 0
    global iterations
    iterations = 0
    while position < len(List):
        iterations += 1
        if target == List[position]:
            return position
        position += 1
    return -1

if__name__== '__main__':
    List = [1, 2, 3, 4, 5, 6, 7, 8]
    target = 3
    answer = linearSearch(target, List)
    if answer != -1:
        print('Target found at index :', answer, 'in',
              iterations,'iterations')
    else:
        print('Target not found in the list')
```

Output:

Binary Search Program:

```
def binarySearch(target, List):

    left = 0
    right = len(List) - 1
    global iterations
    iterations = 0

    while left <= right:
        iterations += 1
        mid = (left + right) // 2
        if target == List[mid]:
            return mid
        elif target < List[mid]:
            right = mid - 1
        else:
            left = mid + 1
    return -1

if__name__== '__main__':
    List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14]
    target = 12
    answer = binarySearch(target, List)
    if(answer != -1):
        print('Target',target,'found at position', answer, 'in',
              iterations,'iterations')
    else:
        print('Target not found')
```

Output:

Signature of the Faculty

Exercise Programs:

5.Write a program to implement Bubble Sort and Selection Sort

Bubble Sort Program:

```
def bubble_sort(alist):
    for i in range(len(alist) - 1, 0, -1):
        no_swap = True
        for j in range(0, i):
            if alist[j + 1] < alist[j]:
                alist[j], alist[j + 1] = alist[j + 1], alist[j]
                no_swap = False
        if no_swap:
            return


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

Output:

Selection Sort Program:

```
def selection_sort(alist):
    for i in range(0, len(alist) - 1):
        smallest = i
        for j in range(i + 1, len(alist)):
            if alist[j] < alist[smallest]:
                smallest = j
        alist[i], alist[smallest] = alist[smallest], alist[i]


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

Output:

Signature of the Faculty

Exercise Programs:

6. Write a program to implement Merge sort and Quick sort

Merge Sort Program:

```
def merge_sort(alist, start, end):
    '''Sorts the list from indexes start to end - 1 inclusive.'''
    if end - start > 1:
        mid = (start + end)//2
        merge_sort(alist, start, mid)
        merge_sort(alist, mid, end)
        merge_list(alist, start, mid, end)

def merge_list(alist, start, mid, end):
    left = alist[start:mid]
    right = alist[mid:end]
    k = start
    i = 0
    j = 0
    while (start + i < mid and mid + j < end):
        if (left[i] <= right[j]):
            alist[k] = left[i]
            i = i + 1
        else:
            alist[k] = right[j]
            j = j + 1
        k = k + 1
    if start + i < mid:
        while k < end:
            alist[k] = left[i]
            i = i + 1
            k = k + 1
    else:
        while k < end:
            alist[k] = right[j]
            j = j + 1
            k = k + 1


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

Output:

Signature of the Faculty

Quick Sort Program:

```
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:

Signature of the Faculty

7. Write a program to implement Stacks and Queues

Stack Program:

```
# Custom stack implementation in Python
class Stack:

    # Constructor to initialize the stack
    def __init__(self, size):
        self.arr = [None] * size
        self.capacity = size
        self.top = -1

    # Function to add an element `x` to the stack
    def push(self, x):
        if self.isFull():
            print("Stack Overflow!! Calling exit()…")
            exit(1)

        print("Inserting", x, "into the stack…")
        self.top = self.top + 1
        self.arr[self.top] = x

    # Function to pop a top element from the stack
    def pop(self):
        # check for stack underflow
        if self.isEmpty():
            print("Stack Underflow!! Calling exit()…")
            exit(1)

        print("Removing", self.peek(), "from the stack")

        #decrease stack size by 1 and (optionally) return the popped element
        top = self.arr[self.top]
        self.top = self.top - 1
        return top

    # Function to return the top element of the stack
    def peek(self):
        if self.isEmpty():
            exit(1)
        return self.arr[self.top]

    # Function to return the size of the stack
    def size(self):
        return self.top + 1

    # Function to check if the stack is empty or not
    def isEmpty(self):
        return self.size() == 0
```

```
# Function to check if the stack is full or not
    def isFull(self):
        return self.size() == self.capacity


if__name__== '__main__':

    stack = Stack(3)

    stack.push(1)        # Inserting 1 in the stack
    stack.push(2)        # Inserting 2 in the stack

    stack.pop()          # removing the top element (2)
    stack.pop()          # removing the top element (1)

    stack.push(3)        # Inserting 3 in the stack

    print("Top element is", stack.peek())
    print("The stack size is", stack.size())

    stack.pop()          # removing the top element (3)

    # check if the stack is empty
    if stack.isEmpty():
        print("The stack is empty")
    else:
        print("The stack is not empty")
```

Output:

Queue Program:

```
# Custom queue implementation in Python
class Queue:

    # Initialize queue
    def__init__(self, size):
        self.q = [None] * size      # list to store queue elements
        self.capacity = size        # maximum capacity of the queue
        self.front = 0 # front points to the front element in the queue
        self.rear = -1 # rear points to the last element in the queue
        self.count = 0 # current size of the queue


    # Function to dequeue the front element
    def pop(self):
        # check for queue underflow
        if self.isEmpty():
            print("Queue Underflow!! Terminating process.")
            exit(1)

        print("Removing element…", self.q[self.front])

        self.front = (self.front + 1) % self.capacity
        self.count = self.count - 1


    # Function to add an element to the queue
    def append(self, value):
        # check for queue overflow
        if self.isFull():
            print("Overflow!! Terminating process.")
            exit(1)

        print("Inserting element…", value)

        self.rear = (self.rear + 1) % self.capacity
        self.q[self.rear] = value
        self.count = self.count + 1


    # Function to return the front element of the queue
    def peek(self):
        if self.isEmpty():
            print("Queue UnderFlow!! Terminating process.")
            exit(1)

        return self.q[self.front]


    # Function to return the size of the queue
    def size(self):
        return self.count
```

```
    # Function to check if the queue is empty or notdef
    isEmpty(self):
        return self.size() == 0


    # Function to check if the queue is full or notdef
    isFull(self):
        return self.size() == self.capacity


if__name__== '__main__':

    # create a queue of capacity 5q =
    Queue(5)

    q.append(1)
    q.append(2)
    q.append(3)

    print("The queue size is", q.size())
    print("The front element is", q.peek())
    q.pop()
    print("The front element is", q.peek())

    q.pop()
    q.pop()

    if q.isEmpty():
        print("The queue is empty")
    else:
        print("The queue is not empty")
```

Output:

Signature of the Faculty

8. Write a program to implement Singly Linked List

Program:

```
import os
from typing import NewType


class _Node:
    '''
    Creates a Node with two fields:
    1. element (accesed using ._element)
    2. link (accesed using ._link)
    '''
    __slots__= '_element', '_link'

    def___init__(self, element, link):
        '''
        Initialses _element and _link with element and link respectively.
        '''
        self._element = element
        self._link = link


class LinkedList:
    '''
    Consists of member funtions to perform different
    operations on the linked list.
    '''

    def___init__(self):
        '''
        Initialses head, tail and size with None, None and 0 respectively.
        '''
        self._head = None
        self._tail = None
        self._size = 0

    def___len__(self):
        '''
        Returns length of linked list
        '''
        return self._size

    def isempty(self):
        '''
        Returns True if linked list is empty, otherwise False.
        '''
        return self._size == 0

    def addLast(self, e):
        '''
```

```
        Adds the passed element at the end of the linked list.
        '''
        newest = _Node(e, None)

        if self.isempty():
            self._head = newest
        else:
            self._tail._link = newest

        self._tail = newest
        self._size += 1

    def addFirst(self, e):
        '''
        Adds the passed element at the beginning of the linked list.
        '''
        newest = _Node(e, None)

        if self.isempty():
            self._head = newest
            self._tail = newest
        else:
            newest._link = self._head
            self._head = newest
        self._size += 1

    def addAnywhere(self, e, index):
        '''
        Adds the passed element at the passed index position of the linked list.
        '''
        newest = _Node(e, None)

        i = index - 1
        p = self._head

        if self.isempty():
            self.addFirst(e)
        else:
            for i in range(i):
                p = p._link
            newest._link = p._link
            p._link = newest
            print(f"Added Item at index {index}!\n\n")
        self._size += 1


    def removeFirst(self):
        '''
        Removes element from the beginning of the linked list.
        Returns the removed element.
        '''
        if self.isempty():
            print("List is Empty. Cannot perform deletion
                   operation.")
            return
```

```
        e = self._head._element
        self._head = self._head._link
        self._size = self._size - 1

        if self.isempty():
            self._tail = None

        return e

    def removeLast(self):
        '''
        Removes element from the end of the linked list.
        Returns the removed element.
        '''
        if self.isempty():
            print("List is Empty. Cannot perform deletion
                    operation.")
            return

        p = self._head
        if p._link == None:
            e = p._element
            self._head = None
        else:
            while p._link._link != None:
                p = p._link
            e = p._link._element
            p._link = None
            self._tail = p

        self._size = self._size - 1
        return e

    def removeAnywhere(self, index):
        '''
        Removes element from the passed index position of the linked list.
        Returns the removed element.
        '''
        p = self._head
        i = index - 1

        if index == 0:
            return self.removeFirst()
        elif index == self._size - 1:
            return self.removeLast()
        else:
            for x in range(i):
                p = p._link
            e = p._link._element
            p._link = p._link._link

        self._size -= 1
        return e
```

```
    def display(self):

        Utility function to display the linked list.
        '''
        if self.isempty() == 0:
            p = self._head
            while p:
                print(p._element, end='-->')
                p = p._link
            print("NULL")
        else:
            print("Empty")

    def search(self, key):
        '''
        Searches for the passed element in the linked list.
        Returns the index position if found, else -1.
        '''
        p = self._head
        index = 0
        while p:
            if p._element == key:
                return index
            p = p._link
            index += 1
        return -1

######################################################################

def options():
    '''
    Prints Menu for operations
    '''
    options_list = ['Add Last', 'Add First', 'Add Anywhere',
                    'Remove First', 'Remove Last', 'Remove Anywhere',
                    'Display List', 'Print Size', 'Search', '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):
    '''
    Switch Case for operations
    '''
    if choice == 1:
        elem = int(input("Enter Item: "))
        L.addLast(elem)
        print("Added Item at Last!\n\n")

    elif choice == 2:
        elem = int(input("Enter Item: "))
        L.addFirst(elem)
        print("Added Item at First!\n\n")
```

```
    elif choice == 3:
        elem = int(input("Enter Item: "))
        index = int(input("Enter Index: "))
        L.addAnywhere(elem, index)

    elif choice == 4:
        print("Removed Element from First:", L.removeFirst())

    elif choice == 5:
        print("Removed Element from last:", L.removeLast())

    elif choice == 6:
        index = int(input("Enter Index: "))
        print(f"Removed Item: {L.removeAnywhere(index)} !\n\n")

    elif choice == 7:
        print("List: ", end='')
        L.display()
        print("\n")

    elif choice == 8:
        print("Size:", len(L))
        print("\n")

    elif choice == 9:
        key = int(input("Enter item to search: "))
        if L.search(key) >= 0:
            print(f"Item {key} found at index position
{L.search(key)}\n\n")
        else:
            print("Item not in the list\n\n")

    elif choice == 10:
        import sys
        sys.exit()

#####################################################################

if__name__== '__main__':
    L = LinkedList()
    while True:
        choice = options()
        switch_case(choice)
```

Output:

Signature of the Faculty

Exercise Programs:

9. Write a program to implement Doubly Linked list

Program:

```
import os

class _Node:
    '''
    Creates a Node with three fields:
    1. element (accessed using ._element)
    2. link (accessed using ._link)
    3. prev (accessed using ._prev)
    '''
    __slots__= '_element', '_link', '_prev'

    def__init__(self, element, link, prev):
        '''
        Initialses _element, _link and _prev with element, link and prev respectively.
        '''
        self._element = element
        self._link = link
        self._prev = prev

class DoublyLL:
    '''
    Consists of member funtions to perform different
    operations on the doubly linked list.
    '''

    def__init__(self):
        '''
        Initialises head, tail and size with None, None and 0 respectively.
        '''
        self._head = None
        self._tail = None
        self._size = 0

    def__len__(self):
        '''
        Returns length of linked list
        '''
        return self._size

    def isempty(self):
        '''
        Returns True if doubly linked list is empty, otherwise False.
        '''
        return self._size == 0

    def addLast(self, e):
        '''
        Adds the passed element at the end of the doubly linked list.
        '''
```

```
        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):
        '''
        Adds the passed element at the beginning of the doubly linked list.
        '''
        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):
        '''
        Adds the passed element at the passed index position of the
        doubly linked list.
        '''
        if index >= self._size:
            print(f'Index value out of range, it should be between
                  0 - {self._size - 1}')
        elif self.isempty():
            print("List was empty, item will be added at the end")
            self.addLast(e)
        elif index == 0:
            self.addFirst(e)
        elif index == self._size - 1:
            self.addLast(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):
        '''
        Removes element from the beginning of the doubly linked list.
        Returns the removed element.
        '''
```

```
        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):
        '''
        Removes element from the end of the doubly linked list.
        Returns the removed element.
        '''
        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):
        '''
        Removes element from the passed index position of the
        doubly linked list.
        Returns the removed element.
        '''
        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):
        '''
        Utility function to display the doubly linked list.
        '''
        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():
    '''
    Prints Menu for operations
    '''
    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):
    '''
    Switch Case for operations
    '''
    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:
        choice = options()
        switch_case(choice)
```

Output:

Signature of the Faculty

Exercise Programs:

10 . Write a python program to implement DFS & BFS graph traversal Techniques

BREADTH FIRST SEARCH TRAVERSAL

```
 # Python3 Program to print BFS traversal
# from a given source vertex. BFS(int s)
# traverses vertices reachable from s.
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

      # Constructor
      def __init__(self):

            # default dictionary to store graph
            self.graph = defaultdict(list)

      # function to add an edge to graph
# Make a list visited[] to check if a node is already visited or not
      def addEdge(self,u,v):
            self.graph[u].append(v)
            self.visited=[]

      # Function to print a BFS of graph
      def BFS(self, s):

            # Create a queue for BFS
            queue = []

            # Add the source node in
            # visited and enqueue it
            queue.append(s)
            self.visited.append(s)

            while queue:

                  # Dequeue a vertex from
                  # queue and print it
                  s = queue.pop(0)
                  print (s, end = " ")

                  # Get all adjacent vertices of the
                  # dequeued vertex s. If a adjacent
                  # has not been visited, then add it
                  # in visited and enqueue it
                  for i in self.graph[s]:
                        if i not in visited:
                              queue.append(i)
                              self.visited.append(s)
```

```
# Driver code

# Create a graph given in
# the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print ("Following is Breadth First Traversal"+
                       " (starting from vertex 2)")
g.BFS(2)
```

Output:

Signature of the Faculty

10. Write a program for BFS & DFS Graph traversal techniques

```
# from a given graph
from collections import defaultdict


# This class represents a directed graph using
# adjacency list representation
class Graph:

      # Constructor
      def __init__(self):

            # Default dictionary to store graph
            self.graph = defaultdict(list)


      # Function to add an edge to graph
      def addEdge(self, u, v):
            self.graph[u].append(v)


      # A function used by DFS
      def DFSUtil(self, v, visited):

            # Mark the current node as visited
            # and print it
            visited.add(v)
            print(v, end=' ')

            # Recur for all the vertices
            # adjacent to this vertex
            for neighbour in self.graph[v]:
                  if neighbour not in visited:
                        self.DFSUtil(neighbour, visited)


      # The function to do DFS traversal. It uses
      # recursive DFSUtil()
      def DFS(self, v):

            # Create a set to store visited vertices
            visited = set()

            # Call the recursive helper function
            # to print DFS traversal
            self.DFSUtil(v, visited)
```

```
# Driver's code
if __name__ == "__main__":
      g = Graph()
      g.addEdge(0, 1)
      g.addEdge(0, 2)
      g.addEdge(1, 2)
      g.addEdge(2, 0)
      g.addEdge(2, 3)
      g.addEdge(3, 3)

      print("Following is Depth First Traversal (starting from vertex
2)")

      # Function call
      g.DFS(2)
```

Output:

Signature of the Faculty

11. Write a program to implement Binary Search Tree

Program:

```
# # # Binary Search Tree
class binarySearchTree:
    def_init_(self,val=None):
        self.val  =  val
        self.left = None
        self.right = None

    def insert(self,val):
        # check if there is no root
        if (self.val == None):
            self.val = val
        # check where to insert
        else:
            # check for duplicate then stop and return
             if val == self.val: return 'no duplicates allowed in binary search tree'
            # check if value to be inserted < currentNode's value
            if (val < self.val):
                 # check if there is a left node to currentNode if true then recurse
                if(self.left):
                    self.left.insert(val)
                # insert where left of currentNode when currentNode.left=None
                else: self.left = binarySearchTree(val)

            # same steps as above here the condition we check is value to be
             # inserted > currentNode's value
            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

    # In order means first left child, then parent, at last right child
    def depthFirstSearch_INorder(self):
        return self.traverseInOrder([])
```

```
    # Pre order means first parent, then left child, at last right child
    def depthFirstSearch_PREorder(self):
        return self.traversePreOrder([])

    # Post order means first left child, then right child , at last parent
    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):
        # returning the node and its parent so we can delete the
        # node and reconstruct the tree from its parent
        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'

    # deleteing a node means we have to rearrange some part of the tree
    def delete(self,val):
        # check if the value we want to delete is in the tree
        if(self.findNodeAndItsParent(val)=='Not found'): return 'Node
        is not in tree'
        # we get the node we want to delete and its parent-node
        # from findNodeAndItsParent method
        deleteing_node, parent_node = self.findNodeAndItsParent(val)
        # check how many children nodes does the node we are going
        #to delete have by traversePreOrder from the deleteing_node
        nodes_effected = deleteing_node.traversePreOrder([])
```

```
# if len(nodes_effected)==1 means, the node to be deleted
doesn't# have any children
# so we can just check from its parent node the
position(left or# right) of node we want to delete
# and point the position to 'None' i.e node is deleted

if (len(nodes_effected)==1):
    if (parent_node.left.val == deleteing_node.val) :
    parent_node.left = None
    else: parent_node.right = None
    return 'Succesfully deleted'
# if len(nodes_effected) > 1 which means the node
we are# going to delete has 'children',
# so the tree must be rearranged from the deleteing_node
else:
    # if the node we want to delete doesn't have any
    parent# means the node to be deleted is 'root' node
    if (parent_node == None):
        nodes_effected.remove(deleteing_node.val)
        # make the 'root' nodee i.e self
        value,left,right to None,# this means we need
        to implement a new tree again without # the
        deleted node
        self.left = None
        self.right = None
        self.val = None

        #construction of new tree
        for node in nodes_effected:
            self.insert(node)
        return 'Succesfully deleted'

    # if the node we want to delete has a parent
    # traverse from parent_node
    nodes_effected = parent_node.traversePreOrder([])

    # deleting the node
    if (parent_node.left == deleteing_node) : parent_node.left
    = None
    else: parent_node.right = None

    # removeing the parent_node, deleteing_node and
    inserting# the nodes_effected in the tree
    nodes_effected.remove(deleteing_node.val)
    nodes_effected.remove(parent_node.val)
    for node in nodes_effected:
        self.insert(node)
```

```
        return 'Successfully deleted'


bst = binarySearchTree()
bst.insert(7)
bst.insert(4)
bst.insert(9)
bst.insert(0)
bst.insert(5)
bst.insert(8)
bst.insert(13)

#           7
#         /  \
#       /     \
#      4        9
#     / \      /  \
#    0   5    8    13

# IN order - useful in sorting the tree in ascending order
print('IN order: ',bst.depthFirstSearch_INorder())

# PRE order - useful in reconstructing a tree
print('PRE order:' ,bst.depthFirstSearch_PREorder())

# POST order - useful in finding the leaf nodes
print('POST order:', bst.depthFirstSearch_POSTorder())

print(bst.delete(5))
print(bst.delete(9))
print(bst.delete(7))

# after deleting
print('IN order: ',bst.depthFirstSearch_INorder())
print('PRE order:' ,bst.depthFirstSearch_PREorder())
print('POST order:', bst.depthFirstSearch_POSTorder())
```

Output:

Signature of the Faculty

12. Write a program for implementing B+ Tree

```
import math
# Node creation
class Node:
    def __init__(self, order):
       self.order = order
       self.values = []
       self.keys = []
       self.nextKey = None
       self.parent = None
       self.check_leaf = False

    # Insert at the leaf
    def insert_at_leaf(self, leaf, value, key):
       if (self.values):
            temp1 = self.values
            for i in range(len(temp1)):
                 if (value == temp1[i]):
                      self.keys[i].append(key)
                      break
                 elif (value < temp1[i]):
                      self.values = self.values[:i] + [value] +
    self.values[i:]
                      self.keys = self.keys[:i] + [[key]] +
    self.keys[i:]
                      break
                 elif (i + 1 == len(temp1)):
                      self.values.append(value)
                      self.keys.append([key])
                      break
       else:
            self.values = [value]
            self.keys = [[key]]

# B plus tree
class BplusTree:
    def __init__(self, order):
       self.root = Node(order)
       self.root.check_leaf = True

    # Insert operation
    def insert(self, value, key):
       value = str(value)
       old_node = self.search(value)
       old_node.insert_at_leaf(old_node, value, key)

       if (len(old_node.values) == old_node.order):
            node1 = Node(old_node.order)
```

```
            node1.check_leaf = True
            node1.parent = old_node.parent
            mid = int(math.ceil(old_node.order / 2)) - 1
            node1.values = old_node.values[mid + 1:]
            node1.keys = old_node.keys[mid + 1:]
            node1.nextKey = old_node.nextKey
            old_node.values = old_node.values[:mid + 1]
            old_node.keys = old_node.keys[:mid + 1]
            old_node.nextKey = node1
            self.insert_in_parent(old_node, node1.values[0], node1)

    # Search operation for different operations
    def search(self, value):
        current_node = self.root
        while(current_node.check_leaf == False):
            temp2 = current_node.values
            for i in range(len(temp2)):
                if (value == temp2[i]):
                    current_node = current_node.keys[i + 1]
                    break
                elif (value < temp2[i]):
                    current_node = current_node.keys[i]
                    break
                elif (i + 1 == len(current_node.values)):
                    current_node = current_node.keys[i + 1]
                    break
        return current_node

    # Find the node
    def find(self, value, key):
        l = self.search(value)
        for i, item in enumerate(l.values):
            if item == value:
                if key in l.keys[i]:
                    return True
                else:
                    return False
        return False

    # Inserting at the parent
    def insert_in_parent(self, n, value, ndash):
        if (self.root == n):
            rootNode = Node(n.order)
            rootNode.values = [value]
            rootNode.keys = [n, ndash]
            self.root = rootNode
            n.parent = rootNode
            ndash.parent = rootNode
            return
```

```
        parentNode = n.parent
        temp3 = parentNode.keys
        for i in range(len(temp3)):
              if (temp3[i] == n):
                    parentNode.values = parentNode.values[:i] + \
                          [value] + parentNode.values[i:]
                    parentNode.keys = parentNode.keys[:i +
                                                      1] + [ndash]
    + parentNode.keys[i + 1:]
                    if (len(parentNode.keys) > parentNode.order):
                          parentdash = Node(parentNode.order)
                          parentdash.parent = parentNode.parent
                          mid = int(math.ceil(parentNode.order / 2)) - 1
                          parentdash.values = parentNode.values[mid + 1:]
                          parentdash.keys = parentNode.keys[mid + 1:]
                          value_ = parentNode.values[mid]
                          if (mid == 0):
                                parentNode.values = parentNode.values[:mid +
    1]
                          else:
                                parentNode.values = parentNode.values[:mid]
                          parentNode.keys = parentNode.keys[:mid + 1]
                          for j in parentNode.keys:
                                j.parent = parentNode
                          for j in parentdash.keys:
                                j.parent = parentdash
                          self.insert_in_parent(parentNode, value_,
    parentdash)

# Print the tree
def printTree(tree):
    lst = [tree.root]
    level = [0]
    leaf = None
    flag = 0
    lev_leaf = 0

    node1 = Node(str(level[0]) + str(tree.root.values))

    while (len(lst) != 0):
        x = lst.pop(0)
        lev = level.pop(0)
        if (x.check_leaf == False):
              for i, item in enumerate(x.keys):
                    print(item.values)
        else:
              for i, item in enumerate(x.keys):
                    print(item.values)

        if (flag == 0):

              lev_leaf = lev
                    leaf = x
```

```
                flag = 1


record_len = 3
bplustree = BplusTree(record_len)
bplustree.insert('5', '33')
bplustree.insert('15', '21')
bplustree.insert('25', '31')
bplustree.insert('35', '41')
bplustree.insert('45', '10')

printTree(bplustree)

if(bplustree.find('5', '34')):
    print("Found")
else:
    print("Not found")
```

Output:

Signature of the Faculty