T.J. INSTITUTE OF TECHNOLOGY

RAJIV GANDHI SALAI, KARAPAKKAM, CHENNAI-600 097

DEPARTMENT OF INFORMATION TECHNOLOGY

RECORD NOTE BOOK

CD3281- DATA STRUCTURES AND ALGORITHMS LABORATORY

Student Name : _____

Register Number : _____

T.J. INSTITUTE OF TECHNOLOGY

RAJIV GANDHI SALAI, KARAPAKKAM, CHENNAI-600097.

BONAFIDE CERTIFICATE

Certified that this is the Bonafide record of work done by the Mr/Ms _____

in the _____ Degree course _____ in the

_____Laboratory during the academic

year _____

Station : Chennai – 600097

Date :

Staff Incharge **Head of the Department**

Submitted for the University practical Examination held on …… at

T . J INSTITUTE OF TECHNOLOGY , karapakkam, chennai-600097.

Internal Examiner **External Examiner**

INDEX

Ex:	Date	Name of the experiment	Page no	Sign
		Implementation of simple class program.		
		Implementation of recursive algorithm.		
		Implement the python array using list.		
		Implementation of linked list.		
		Implementation of stack ADT.		
		Implementation of queue ADT.		
		Compute the polynomial numbers.		
		Implement the bubble sorting.		
		Implement the insertion sorting.		
		Implement the quick sorting.		
		Implement the binary search.		
		Implement the hash table.		
		Implement the tree traversal.		
		Implement the binary search tree.		
		Implement the heap.		
		Implement the graph representation.		
		Implement the shortest path algorithm.		
		Implement the minimum spanning algorithm.		

Ex.No: **IMPLEMENTATION OF SIMPLE CLASS PROGRAM**

Date:

Aim:

To Write a Python program to implement the simple class program using inheritance.

Algorithm:

Step1:Start.

Step2: Create a class.

Step 3: Create a built-in-function with reference parameter and get the no.of sides for the polygon.

Step4: Create a user-define function to print the sides.

Step5: Create a derived class.

Step 6: Print.

Step7:Stop.

Program:

```
class polygon:
    def __init__(self,no_of_sides):
        self.n=no_of_sides
        self.sides=[0 for i in range(no_of_sides)]
    def inputside(self):
        self.sides=[float(input("Enter side"+str(i+1)+":"))for i in range(self.n)]
    def dispsides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
class triangle(polygon):
    def __init__(self):
        polygon.__init__(self,3)
    def findarea(self):
```

```
        a,b,c=self.sides
        s=(a+b+c)/2
        area=(s*(s-a)*(s-b)*(s-c))**0.5
        print("The area of the triangle is %0.2f" %area)
```

Output:

```
>>> t=triangle()
        >>> t.inputside()
        Enter side1:3
        Enter side2:5
        Enter side3:4
        >>> t.dispsides()
        Side 1 is 3.0
        Side 2 is 5.0
        Side 3 is 4.0
        >>> t.findarea()
        The area of the triangle is 6.00
```

Result:

Thus the Python class program was implemented using inheritance concept were executed successfully.

Ex.No: **IMPLEMENTATION OF RECURSION ALGORITHM**

Date:

Aim:

To Write a Python program

a) To find the fibonacci series using recursion.

b) To find the factorial of number using recursion.

A) Algorithm: To find the fibonacci series using recursion.

Step1: Start.
Step 2: Define a function.
Step 3: Check the number is less than or equal to 1.
Step 4: If it is equal to 1 return it.
Step5: Else use the formula (recur_fibo(n1)+recur_fibo(n-2)).
Step6: Given the number of terms to be print.
Step 7: Print.
Step 8: Stop.

Program:

```
def recur_fibo(n):
                    if n<=1:
                        return n
                    else:
                        return(recur_fibo(n-1)+recur_fibo(n-2))
        nterms=10
        if nterms<=0:
                print("Plese enter a positive integer")
        else:
                print("Fibonacci sequence :")
                for i in range(nterms):
                        print(recur_fibo(i))
```

Output:

```
Fibonacci sequence :
0
1
1
2
3
5
8
13
21
34
```

B) Algorithm: To find the factorial of number using recursion.

Step1:Start.
Step2:Define a function.
Step3: Check the number is equal to 1.
Step4:If it is equal to 1
Step5:Else use the formula(X*factorial(X-1))
Step6: Give the number to be factorial.
Step7:Print.
Step8:Stop.

Program:

```
def factorial(x):

"""This is a recursive function to find the factorial of an integer"""

    if x==1:

        return 1

    else:

        return (x*factorial(x-1))

num=3

print("The factorial of",num,"is",factorial(num))
```

Output:

The factorial of 3 is 6

Result:

Thus the recursion algorithm was implemented usingPythonprogram wereexecuted successfully.

Ex.No: IMPLEMENT THE PYTHON ARRAY USING LIST

Date:

Aim:

To write a Python program for implement the array to insert,delete and traverse the data element .

Algorithm:

Step1: Start.

Step 2: Import array library.

Step3: Print the array values and then accessing the array element from index 2.

Step4: Insert an element in the array of index 3 & 4 (400,150) respectively and then print the inserted element .

Step5: Delete an element from the array (150) and then print .

Step 6: Traversing the elements from the array and then print .

Step 7: Stop.

Program:

```
import array

balance = array.array('i', [100,200,300])

print("The array given values")

print(balance)

print("Accessing of an array element from index[2] :  ")

print(balance[2])

print(" ------------Inserting an element------------")

balance.insert(3, 400)

balance.insert(4, 150)

print("After Insertion :")

print(balance)
```

```
print("Accessing the index value of an element :")
print(balance.index(400))
print(" ------------------Deleting an element from a array----------")
balance.remove(150)
print(balance)
print( " --------------------Traverse an array ---------- ")
for x in balance:
    print("Array Elememt :")
    print(x)
```

Output:

```
The array given values
array('i', [100, 200, 300])
Accessing of an array element from index[2] :
300
 ------------Inserting an element------------
After Insertion :
array('i', [100, 200, 300, 400, 150])
Accessing the index value of an element :
3
 ------------------Deleting an element from a array----------
```

```
array('i', [100, 200, 300, 400])
 ----------Traverse an array ----------
Array Elememt :
100
Array Elememt :
200
Array Elememt :
300
Array Elememt :
400
```

Result:

Thus the Python program executed successfully.

Ex.No: IMPLEMENTATION OF LINKED LIST

Date:

Aim:

To write a python program to implement the linked list.

Algorithm:

Step1: Start .

Step2:Creating a class as Node.

Step 3: Again creating a class as Linked List.

Step4:Defining a insert function.

Step 5: Defining printLL .

Step 6: Print the data element in the linked list.

Step7: Creating an object for the class Linked List .

Step 8: With the help of the object calling the insert function to insert the element .

Step 9: Print the data element.

Step10:Stop.

Program:

```
class Node:
  def __init__(self, data = None, next=None):
    self.data = data
    self.next = next
class LinkedList:
  def __init__(self):
    self.head = None
  def insert(self, data):
    newNode = Node(data)
    if(self.head):
      current = self.head
      while(current.next):
```

```
            current = current.next
          current.next = newNode
        else:
          self.head = newNode
      def printLL(self):
        current = self.head
        while(current):
          print(current.data)
          current = current.next
print("The data elements in linked list are: \n ")
LL = LinkedList()
LL.insert(1)
LL.insert(2)
LL.insert(3)
LL.insert(4)
LL.printLL()
```

Output:

The data elements in linked list are:

1

2

3

4

Result:

Thus the Python program executed successfully.

Ex.No: IMPLEMENTATION OF STACK ADT

Date:

Aim:

To write a Python program to implement of stack ADT using list.

Algorithm:

Step1: Start.

Step2: Creating a class as Node.

Step 3: Again creating a class as
Stack.

Step4:Defining push function.

Step5: Defining pop function.

Step6: Defining traverse function.

Step 7: Define Menu function.

Step 8: Print.

Step 9: Creating an object for the class Stack.

Step 10: Print the stack.

Step 11: Get the input from the user for choice and then print.

Step 12: Stop.

Program:

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class Stack:
    def __init__(self):
        self.head=None
        self.ctr=0
        self.top=None
    def Push(self,data):
        node=Node(data)
        if self.head==None:
            self.head=node
            self.top=node
        else:
            self.top.next=node
            self.top=node
            print("Node pushed to stack",data)
            self.ctr+=1
            return
    def Pop(self):
        if self.head==None:
            print("Stack Underflow")
        elif self.head==self.top:
            print("Deleted from Stack",self.head.data)
            self.head=self.top=None
            self.ctr-=1
        else:
            print("Deleted from Stack",self.top.data)
            temp=self.head
            while temp.next is not self.top:
                temp=temp.next
                temp.next=None
                self.top=temp
                self.ctr-=1
                return
    def Traverse(self):
        if self.head==None:
            print("No Nodes exist")
            return
        temp=self.head
        while temp is not None:
            print(temp.data)
            temp=temp.next
def Menu():
    print("1.Push\n2.Pop\n3.Traverse\n4.Number of nodes\n5.Exit")
    ch=int(input("Enter choice:"))
    return ch
```

```
s=Stack()
print("**********Stack**********")
while True:
    ch=Menu()
    if ch==1:
        data=input("Enter data:")
        s.Push(data)
    elif ch==2:
        s.Pop()
    elif ch==3:
        s.Traverse()
    elif ch==4:
        print("Number of nodes",s.ctr)
    else:
        print('Quit')
     break
```

Output:

```
**********Stack**********
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:12
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:123
Node pushed to stack 123
1.Push
2.Pop
3.Traverse
```

4.Number of nodes

5.Exit

Enter choice:2

Deleted from Stack 123

1.Push

2.Pop

3.Traverse

4.Number of nodes

5.Exit

Enter choice:3

12

123

1.Push

2.Pop

3.Traverse

4.Number of nodes

5.Exit

Enter choice:4

Number of nodes 1

1.Push

2.Pop

3.Traverse

4.Number of nodes

5.Exit

Enter choice:5

Quit

Result:

Thus the Python program executed successfully.

Ex.No: IMPLEMENTATION OF QUEUE ADT

Date:

Aim:

Write a Python program of implementation of queue.

Algorithm:

Step1: Start.

Step2:Create a empty queue.

Step3:Append a,b,c to the queue.

Step4:Print initial queue.

Step5:Print the elements dequeued from the queue.

Step6:Pop the element from the index of 0 for

twice.

Step7: Print queue after removing the element.

Step8:Stop.

Program:

```
queue = []
queue.append('a')
queue.append('b')
queue.append('c')
print("Initial queue")
print(queue)
print("\nElements dequeued from queue")
print(queue.pop(0))
print(queue.pop(0))
print("\nQueue after removing elements")
print(queue)
```

Output:

```
Initial queue
['a', 'b', 'c']

Elements dequeued from queue
a
b

Queue after removing elements
['c']
```

Result:

Thus the Python program executed successfully.

Ex.No: **APPLICATION OF LIST(POLYNOMIAL)**

Date:

Aim:

To write a Python program to find the polynomial.

Algorithm:

Step1: Start.

Step2:Creating a class as Node.

Step3:Define addnode.

Step4:Define printList.

Step5:Define remove Duplicate.

Step6:Define multiply.

Step7: Print 1st polynomial.

Step8:Print 2nd polynomial.

Step 9: Print resultant polynomial.

Step 10: Stop.

Program:

```
class Node:
    def __init__(self):
        self.coeff = None
        self.power = None
        self.next = None
def addnode(start, coeff, power):
    newnode = Node();
    newnode.coeff = coeff;
    newnode.power = power;
    newnode.next = None;
    if (start == None):
        return newnode;
    ptr = start;
    while (ptr.next != None):
        ptr = ptr.next;
    ptr.next = newnode;
    return start;
def printList(ptr):
    while (ptr.next != None):
        print(str(ptr.coeff) + 'x^' + str(ptr.power), end = '')
```

```
        if( ptr.next != None and ptr.next.coeff >= 0):
            print('+', end = '')
        ptr = ptr.next
    print(ptr.coeff)
def removeDuplicates(start):
    ptr2 = None
    dup = None
    ptr1 = start;
    while (ptr1 != None and ptr1.next != None):
        ptr2 = ptr1;
        while (ptr2.next != None):
            if (ptr1.power == ptr2.next.power):
                ptr1.coeff = ptr1.coeff + ptr2.next.coeff;
                dup = ptr2.next;
                ptr2.next = ptr2.next.next;
            else:
                ptr2 = ptr2.next;
        ptr1 = ptr1.next;
def multiply(poly1, Npoly2, poly3):
    ptr1 = poly1;
    ptr2 = poly2;
    while (ptr1 != None):
        while (ptr2 != None):
            coeff = ptr1.coeff * ptr2.coeff;
            power = ptr1.power + ptr2.power;
            poly3 = addnode(poly3, coeff, power);
            ptr2 = ptr2.next;
        ptr2 = poly2;
        ptr1 = ptr1.next;
    removeDuplicates(poly3);
    return poly3;
if __name__=='__main__':
    poly1 = None
    poly2 = None
    poly3 = None;
    poly1 = addnode(poly1, 3, 3);
    poly1 = addnode(poly1, 6, 1);
    poly1 = addnode(poly1, -9, 0);
    poly2 = addnode(poly2, 9, 3);
    poly2 = addnode(poly2, -8, 2);
    poly2 = addnode(poly2, 7, 1);
    poly2 = addnode(poly2, 2, 0);
    print("1st Polynomial:- ", end = "");
    printList(poly1);
    print("2nd Polynomial:- ", end = "");
    printList(poly2);
    poly3 = multiply(poly1, poly2, poly3);
    print("Resultant Polynomial:- ", end = "");
    printList(poly3);
```

Output:

1st Polynomial:- 3x^3+6x^1-9
2nd Polynomial:- 9x^3-8x^2+7x^1+2
Resultant Polynomial:- 27x^6-24x^5+75x^4-123x^3+114x^2-51x^1-18

Result:

Thus the Python program executed successfully

Ex.No: **IMPLEMENT THE BUBBLESORT**

Date:

Aim:

To write a Python program to sort the elements using bubblesort.

Algorithm:

Step1: Start.

Step2: Define bubble_sort.

Step 3: For i in range(0,len(list1)-1)

Step4: For j in range(len(list1)-1)

Step5:Print the unsorted list.

Step 6: Print the sorted list.

Step 7: Stop.

Program:

```
def bubble_sort(list1):
    for i in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp
    return list1
list1 = [5, 3, 8, 6, 7, 2]
print("The unsorted list is: ", list1)
print("The sorted list is: ", bubble_sort(list1))
```

Output:

```
The unsorted list is:  [5, 3, 8, 6, 7, 2]
The sorted list is:  [2, 3, 5, 6, 7, 8]
```

Result:

Thus the Python program executed successfully

Ex.No: **IMPLEMENT THE INSERTIONSORT**

Date:

Aim:

To write a Python program to sort the elements using insertionsort.

Algorithm:

Step1: Start.

Step2: Define insertion_sort.

Step 3: For i in range(1,len(alist)).

Step4: Get the input from the user to enter the list of numbers.

Step5:Print the sorted list.

Step 6: Stop.

Program:

```
def insertion_sort(alist):
    for i in range(1,len(alist)):
        temp=alist[i]
        j=i-1
        while(j>=0 and temp<alist[j]):
            alist[j+1]=alist[j]
            j=j-1
        alist[j+1]=temp
alist=input('Enter the list of numbers :').split()
alist=[int(x)for x in alist]
insertion_sort(alist)
print('Sorted list :',end='')
print(alist)
```

Output:

Enter the list of numbers :3 7 1 9 2

Sorted list :[1, 2, 3, 7, 9]

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENT THE QUICKSORT**

Date:

Aim:

To write a Python program to sort elements using quick sort method.

Algorithm:

Step1:Start.

Step2:Define quick sort.

Step3:Define partition.

Step4: Get the input from the user to enter the list of numbers.

Step5:Print the sorted list.

Step 6:Stop.

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 :',end='')
```

```
print(alist)
```

Output:

```
Enter the list of numbers :22 7 4 9 1
Sorted list :[1, 4, 7, 9, 22]
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENT THE BINARY SEARCH**

Date:

Aim:

To write a Python program to search an element in a list of elements using Binary Search technique.

Algorithm:

Step1:Start.

Step2:Define binary_sort.

Step3:Get the input from the user to enter the size of the list.

Step4:Again get the input from the user to enter any number.

Step 5: Print the list will be sorted

Step6: Get the input from the user to enter the number to be searched.

Step7:Print the entered number which is present at the position.

Step8: Stop.

Program:

```
def binary_sort(sorted_list,length,key):
    start=0
    end=length-1
    while start<=end:
        mid=int((start+end)/2)
        if key==sorted_list[mid]:
            print("\n Entered number %d is present at position : %d" %(key,mid))
            return-1
        elif key<sorted_list[mid]:
            end=mid-1
        elif key>sorted_list[mid]:
            start=mid+1
    print("\n Elementnot found!")
    return-1
```

```
lst=[]
size=int(input("Enter size of list :\t"))
for n in range(size):
    numbers=int(input("Enter any number :\t"))
    lst.append(numbers)
lst.sort()
print('\n\n The list will be sorted,the sorted list is :',lst)
x=int(input("\n Enter the number to be search :"))
binary_sort(lst,size,x)
```

Output:

```
Enter size of list :    3
Enter any number :  2
Enter any number :  8
Enter any number :  22

 The list will be sorted,the sorted list is : [2, 8, 22]

 Enter the number to be search :22

 Entered number 22 is present at position : 2
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENT THE HASH TABLE**

Date:

Aim:

To write a Python program to implement the hash table.

Algorithm:

Step1:Start.

Step2:Define display_hash.

Step3:For i in range(len(hashtable)).

Step4:Print.

Step 5: Define Hashing.

Step6:Define insert .

Step7:display_hash(hashtable).

Step8: Stop.

Program:

```
def display_hash(hashtable):
    for i in range(len(hashtable)):
        print(i,end="")
        for j in hashtable[i]:
            print("-->",end="")
            print(j,end="")
        print()
hashtable=[[]for _ in range(10)]
def Hashing(keyvalue):
    return keyvalue % len(hashtable)
def insert(hashtable,keyvalue,value):
    hash_key=Hashing(keyvalue)
    hashtable[hash_key].append(value)
insert(hashtable,0,"Allahabad")
insert(hashtable,5,"Mumbai")
insert(hashtable,3,"Mathura")
insert(hashtable,9,"Delhi")
insert(hashtable,1,"Punjab")
insert(hashtable,1,"Noida")
display_hash(hashtable)
```

Output:

0--> Allahabad
1--> Punjab--> Noida
2
3--> Mathura
4
5--> Mumbai
6
7
8
9--> Delhi

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENT THE TREE TRAVERSAL**

Date:

Aim:

To write a Python program to implement the tree traversal.

Algorithm:

Step1:Start.

Step2:Creating a class as Node.

Step3:Define print Inorder.

Step4:Define print Postorder.

Step 5: Define print Preorder.

Step6:Print preorder traversal of binary tree.

Step7:Print inorder traversal of binary tree.

Step8: Print postorder traversal of binary tree.

Step 9: Stop.

Program:

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
def printInorder(root):
    if root:
        printInorder(root.left)
        print(root.val),
        printInorder(root.right)
def printPostorder(root):
    if root:
        printPostorder(root.left)
        printPostorder(root.right)
        print(root.val),
def printPreorder(root):
    if root:
        print(root.val),
        printPreorder(root.left)
        printPreorder(root.right)
root = Node(1)
```

```
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print ("\nPreorder traversal of binary tree is")
printPreorder(root)
print ("\nInorder traversal of binary tree is")
printInorder(root)
print ("\nPostorder traversal of binary tree is")
printPostorder(root)
```

Output:

```
Preorder traversal of binary tree is
1
2
4
5
3

Inorder traversal of binary tree is
4
2
5
1
3

Postorder traversal of binary tree is
4
5
2
3
1
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENT THE BINARY SEARCH TREE**

Date:

Aim:

To write a Python program to implement the binary search tree.

Algorithm:

Step1:Start.

Step2:Creating a class as BTSNode.

Step3:Define insert function.

Step4:Define inorder.

Step 5: Define replace_node_of_parent.

Step6:Define find_min.

Step7:Define remove function.

Step8: Define search function.
Step 9: Creating a class as BStree.
Step 10: Define inorder.
Step 11: Define add function.
Step 12: Define remove function.
Step 13: Define search function.
Step 14: Creating an object bstree for the class BSTree.
Step 15: Print Menu (this assumes no duplicate keys)
Step 16: Print add <key>.
Step 17: Print remove <key>.
Step 18: Print inorder.
Step 19: Print quit.
Step 20: Get the input from the user to what would you like to do.
Step 21: Print the inorder traversal.
Step 22: Stop.

Program:

```
class BSTNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.parent = None
    def insert(self, node):
        if self.key > node.key:
```

```
            if self.left is None:
                self.left = node
                node.parent = self
            else:
                self.left.insert(node)
        elif self.key < node.key:
            if self.right is None:
                self.right = node
                node.parent = self
            else:
                self.right.insert(node)
    def inorder(self):
        if self.left is not None:
            self.left.inorder()
        print(self.key, end=' ')
        if self.right is not None:
            self.right.inorder()
    def replace_node_of_parent(self, new_node):
        if self.parent is not None:
            if new_node is not None:
                new_node.parent = self.parent
            if self.parent.left == self:
                self.parent.left = new_node
            elif self.parent.right == self:
                self.parent.right = new_node
        else:
            self.key = new_node.key
            self.left = new_node.left
            self.right = new_node.right
            if new_node.left is not None:
                new_node.left.parent = self
            if new_node.right is not None:
                new_node.right.parent = self
    def find_min(self):
        current = self
        while current.left is not None:
            current = current.left
        return current
    def remove(self):
        if (self.left is not None and self.right is not None):
            successor = self.right.find_min()
            self.key = successor.key
            successor.remove()
        elif self.left is not None:
            self.replace_node_of_parent(self.left)
        elif self.right is not None:
            self.replace_node_of_parent(self.right)
        else:
            self.replace_node_of_parent(None)
    def search(self, key):
```

```
            if self.key > key:
                if self.left is not None:
                    return self.left.search(key)
                else:
                    return None
            elif self.key < key:
                if self.right is not None:
                    return self.right.search(key)
                else:
                    return None
            return self
    class BSTree:
        def __init__(self):
            self.root = None
        def inorder(self):
            if self.root is not None:
                self.root.inorder()
        def add(self, key):
            new_node = BSTNode(key)
            if self.root is None:
                self.root = new_node
            else:
                self.root.insert(new_node)
        def remove(self, key):
            to_remove = self.search(key)
            if (self.root == to_remove
                and self.root.left is None and self.root.right is None):
                self.root = None
            else:
                to_remove.remove()
        def search(self, key):
            if self.root is not None:
                return self.root.search(key)
    bstree = BSTree()
    print('Menu (this assumes no duplicate keys)')
    print('add <key>')
    print('remove <key>')
    print('inorder')
    print('quit')
    while True:
        do = input('What would you like to do? ').split()
        operation = do[0].strip().lower()
        if operation == 'add':
            key = int(do[1])
            bstree.add(key)
        elif operation == 'remove':
            key = int(do[1])
            bstree.remove(key)
        elif operation == 'inorder':
            print('Inorder traversal: ', end='')
```

```
            bstree.inorder()
            print()
        elif operation == 'quit':
            break
```

Output:

```
Menu (this assumes no duplicate keys)
add <key>
remove <key>
inorder
quit
What would you like to do? add 1
What would you like to do? add 5
What would you like to do? add 2
What would you like to do? add 7
What would you like to do? add 3
What would you like to do? inorder
Inorder traversal: 1 2 3 5 7
What would you like to do? remove 7
What would you like to do? add 4
What would you like to do? inorder
Inorder traversal: 1 2 3 4 5
What would you like to do? quit
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENTATION OF HEAP**

Date:

Aim:

To write a Python program to implement the heap.

Algorithm:

Step1:Start.

Step2: Import heapq library.

Step3:Create a list with elements and then heapify.

Step4:Print the created heap.

Step 5: Push the element 4 with the help of heappush.

Step6:Print the modified heap after push.

Step7:Pop the smallest element from the list with the help of heappop.

Step8: Print the popped and smallest element.
Step 9: Stop.

Program:

```
import heapq
li=[5, 7, 9, 1, 3]
heapq.heapify(li)
print("The created heap is : ",end="")
print(list(li))
heapq.heappush(li,4)
print("The modified heap after push is : ",end="")
print(list(li))
print("The popped and smallest element is : ",end="")
print(heapq.heappop(li))
```

Output:

```
The created heap is : [1, 3, 9, 7, 5]
The modified heap after push is : [1, 3, 4, 7, 5, 9]
The popped and smallest element is : 1
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENTATION OF GRAPH REPRESENTATION**

Date:

Aim:

To write a Python program to implement the graph representation.

Algorithm:

Step1:Start.

Step2:Import default dict.

Step3:Creating a graph with default dict (list).

Step4:Define addEdge.

Step 5: Define generate_edges.

Step6:Create a edge with a empty list.

Step7:For node in graph.

For neighbour in graph [node].

Step8: Append the edge.
Step 9: Print the generated graph.
Step 10: Stop.

Program:

```
from collections import defaultdict
graph = defaultdict(list)
def addEdge(graph,u,v):
    graph[u].append(v)
def generate_edges(graph):
    edges = []
    for node in graph:
        for neighbour in graph[node]:
            edges.append((node, neighbour))
    return edges
addEdge(graph,'a','c')
addEdge(graph,'b','c')
addEdge(graph,'b','e')
addEdge(graph,'c','d')
addEdge(graph,'c','e')
addEdge(graph,'c','a')
addEdge(graph,'c','b')
addEdge(graph,'e','b')
addEdge(graph,'d','c')
addEdge(graph,'e','c')
```

```
print(generate_edges(graph))
```

Output:

```
[('a', 'c'), ('b', 'c'), ('b', 'e'), ('c', 'd'), ('c', 'e'), ('c', 'a'), ('c', 'b'), ('e', 'b'), ('e', 'c'), ('d', 'c')]
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENTATION OF SHORTEST PATH ALGORITHM**

Date:

Aim:

To write a Python program to implement the shortest path algorithm.

Algorithm:

Step1:Start.

Step2:Creating a class as graph.

Step3:Define print solution.

Step4:Print vertex t distance from source.

Step 5: Define minDistance.

Step6:Define dijkstra.

Step7:Creating an object for the class graph.

Step8: Print.
Step 9: Stop.

Program:

```
import sys
class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]
    def printSolution(self, dist):
        print("Vertex t Distance from Source")
        for node in range(self.V):
            print(node, "t", dist[node])
    def minDistance(self, dist, sptSet):
        min = sys.maxsize
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v
        return min_index
    def dijkstra(self, src):
        dist = [sys.maxsize] * self.V
        dist[src] = 0
```

```
            sptSet = [False] * self.V
            for cout in range(self.V):
                u = self.minDistance(dist, sptSet)
                sptSet[u] = True
                for v in range(self.V):
                    if self.graph[u][v] > 0 and sptSet[v] == False and dist[v] > dist[u] + self.graph[u]
[v]:
                        dist[v] = dist[u] + self.graph[u][v]
            self.printSolution(dist)
    g = Graph(9)
    g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
            [4, 0, 8, 0, 0, 0, 0, 11, 0],
            [0, 8, 0, 7, 0, 4, 0, 0, 2],
            [0, 0, 7, 0, 9, 14, 0, 0, 0],
            [0, 0, 0, 9, 0, 10, 0, 0, 0],
            [0, 0, 4, 14, 10, 0, 2, 0, 0],
            [0, 0, 0, 0, 0, 2, 0, 1, 6],
            [8, 11, 0, 0, 0, 0, 1, 0, 7],
            [0, 0, 2, 0, 0, 0, 6, 7, 0]
            ]
    g.dijkstra(0)
```

Output:

```
Vertex t Distance from Source
0 t 0
1 t 4
2 t 12
3 t 19
4 t 21
5 t 11
6 t 9
7 t 8
8 t 14
```

Result:

Thus the Python program executed successfully.

Ex.N0: **IMPLEMENT THE MINIMUM SPANNING TREE ALGORITHM**
Date:

Aim:

To write a Python program to implement the minimum spanning tree algorithm.

Algorithm:

Step1:Start.

Step2:Import default dict.

Step3:Creating a class graph.

Step4:Define add edge.

Step 5: Define find function.

Step6:Define union function.

Step7:Define KruskalMST.

Step8: Print the edges in the constructed MST.
Step 9: Creating an object for the class graph.
Step 10: Print the minimum spanning tree.
Step 11: Stop.

Program:

```
from collections import defaultdict
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(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 KruskalMST(self):
                result = []
                i = 0
                e = 0
                self.graph = sorted(self.graph,key=lambda item: item[2])
                parent = []
                rank = []
                for node in range(self.V):
                        parent.append(node)
                        rank.append(0)
                while e < self.V - 1:
                        u, v, w = self.graph[i]
                        i = i + 1
                        x = self.find(parent, u)
                        y = self.find(parent, v)
                        if x != y:
                                e = e + 1
                                result.append([u, v, w])
                                self.union(parent, rank, x, y)
                minimumCost = 0
                print ("Edges in the constructed MST")
                for u, v, weight in result:
                        minimumCost += weight
                        print("%d -- %d == %d" % (u, v, weight))
                print("Minimum Spanning Tree" , minimumCost)
g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)
g.KruskalMST()
```

Output:

```
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19
```

Result:

Thus the Python program executed successfully.