P.T.Lee Chengalvaraya Naicker College of Engineering & Technology

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)

Vallal P.T.Lee Chengalvaraya Naicker Nagar, Oovery, Kanchipuram

Department of Information Technology

Regulation-2021

CD3281-Data Structure And Algorithms Laboratory

RECORD

Name :

Reg.No :

Year/Semester :

P.T.Lee Chengalvaraya Naicker College of Engineering & Technology

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)

Vallal P.T.Lee Chengalvaraya Naicker Nagar, Oovery, Kanchipuram

Department of Information Technology

BONAFIDE CERTIFICATE

RegNo: _____

Certified that this is a bonafide record of work done by Candidate Mr\Mrs _____ in CD3281-Data Structure And Algorithms Laboratory during for academic year 2024-2025

_____ _____

Lab incharge Head of the department

Submitted for the practical Examination held on _____

_____ _____

Internal Examiner External Examiner

LIST OF EXPRIMENT

S.NO	DATE	LIST OF EXPRIMENT	PAGE. NO	SIGN
1		IMPLEMENTS SIMPLE ADTs AS PYTHON CLASSES.		
2		IMPLEMENT RECURSIVE ALGORITHM IN PYTHON		
3		IMPLEMENT LIST ADT USING PYTHON ARRAYS		
4		LINKED LIST IMPLEMENTATIONS OF LIST		
5		IMPLEMENTATION OF STACK AND QUEUE ADTSs		
6		IMPLEMENTATION OF SORTING AND SEARCHING ALGORITHMS		
7		IMPLEMENTATION OF SORTING AND SEARCHING ALGORITHMS		
		7(a) LINEAR SEARCH		
		7(b) BINARY SEARCH		
		7(c) BUBBLE SORT		
		7(d) INSERTION SORT		
		7(e) MERGE SORT		
		7(f) QUICK SORT		
8		IMPLEMENTATION OF HASH TABLES		

9		**TREE REPRESENTATION AND TRAVERSAL ALGORITHMS**		
10		**IMPLEMENTATION OF BINARY SEARCH TREES**		
11		**IMPLEMENTATION OF HEAPS**		
12		**12(a) GRAPH REPRESENTATION**		
		12(b) GRAPH TRAVERSAL ALGORITHMS		
13		**IMPLEMENTATION OF SINGLE SOURCE SHORTEST PATH ALGORITHM**		
14		**IMPLEMENTATION OF MINIMUM SPANNING TREEALGORITHMS**		

EX.NO 1 IMPLEMENTS SIMPLE ADTs AS PYTHON CLASSES.

DATE:

AIM:

To implement simple ADTs as python classes

ALGORITHM:

Step1: Select the structure chosen by the user as 1 for stack and 2 for queue.

Step 2: If press 1, it will call class Stack.

Step 3: It will keep on checking till n1 becomes 5.

Step 4: If press 1, accept from the user values and it will store in the stack.

Step 5: If press 2, it will remove value from the stack.

Step 6: If press 3, it will show the size of the stack.

Step 7: If press 4, it will show items of the stack.

Step 8: If press 2, it will call class Queue.

Step 9: Accept from the user 1. 1.enqueue 2.dequeue 3.size 4.display 5.exit

Step 10: It will keep on checking till n1 becomes 5.

Step 11: If press 1, accept from the user values and it will store in the enqueue.

Step 12: If press 2, it will perform dequeue and
display the message dequeue done.

Step 13: If press 3, it will show the size of the queue.

Step 14: If press 4, it will show items of the queue.

PROGRAM/SOURCE CODE:

```
# Stack ADT
class Stack:
    def__init_(self):
        self.items = []
    def isEmpty(self):
        return self.items ==
        []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def size(self):
        return len(self.items)
# Queue ADT
class Queue:
    def__init_(self):
        self.items = []
    def isEmpty(self):
        return self.items ==
        []
    def enqueue(self,item):
        self.items.append(item)
    def dequeue(self):
        return self.items.pop(0)
    def front(self):
        return self.items[len(self.items)-1]
    def size(self):
        return len(self.items)
# Stack ADT operation example
n1 = 0
n2 = 0
print('Enter number. 1.Stack 2.Queue')
i = int(input())
if i == 1:
    s=Stack()
    while n1 < 5:
        print("Enter number. 1.push 2.pop 3.size 4.display 5.exit")
        n1 = int(input())
        if n1 == 1:
            print('Enter number')
            num = int(input())
            s.push(num)
```

```
elif n1 == 2:
    s.pop()
```

```
                print('Value popped')
            elif n1 == 3:
                print('Size:' , s.size())
            elif n1 == 4:
                print(s.items)
        elif i == 2:
            q = Queue()
            while n2 < 5:
                print("Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit")
                n2 = int(input())
                if n2 == 1:
                    print('Enter number')
                    num = int(input())
                    q.enqueue(num)
                elif n2 == 2:
                    q.dequeue()
                    print('dequeue done')
                elif n2 == 3:
                    print('Size:' , q.size())
                elif n2 == 4:
                    print(q.items)
```

OUTPUT:

OUTPUT 1:

```
Enter number. 1.Stack 2.Queue
1
Enter number. 1.push 2.pop 3.size 4.display 5.exit
1
Enter number
10
Enter number. 1.push 2.pop 3.size 4.display 5.exit
1
Enter number
20
Enter number. 1.push 2.pop 3.size 4.display 5.exit
3
Size: 2
Enter number. 1.push 2.pop 3.size 4.display 5.exit
4
[10, 20]
Enter number. 1.push 2.pop 3.size 4.display 5.exit
2
Value popped
```

Enter number. 1.push 2.pop 3.size 4.display 5.exit
4
[10]
Enter number. 1.push 2.pop 3.size 4.display 5.exit
5

```
In [4]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter number. 1.Stack 2.Queue

1
Enter number. 1.push 2.pop 3.size 4.display 5.exit

1
Enter number

10
Enter number. 1.push 2.pop 3.size 4.display 5.exit

1
Enter number

20
Enter number. 1.push 2.pop 3.size 4.display 5.exit

3
Size: 2
Enter number. 1.push 2.pop 3.size 4.display 5.exit

4
[10, 20]
Enter number. 1.push 2.pop 3.size 4.display 5.exit

2
Value popped
Enter number. 1.push 2.pop 3.size 4.display 5.exit

4
[10]
Enter number. 1.push 2.pop 3.size 4.display 5.exit

5
```

OUTPUT 2:

Enter number. 1.Stack 2.Queue
2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
1
Enter number
10
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
1
Enter number
20
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
3
Size: 2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
4
[10, 20]

Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
2
dequeue done
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
4
[20]
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
5

```
In [6]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter number. 1.Stack 2.Queue

2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

1
Enter number

10
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

1
Enter number

20
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

3
Size: 2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

4
[10, 20]
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

2
dequeue done
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

4
[20]
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

5
```

RESULT:

Thus, the Program to Implement simple ADTs using python has been executed successfully.

EX NO 2 IMPLEMENT RECURSIVE ALGORITHM IN PYTHON
DATE :

AIM:

To implement recursive algorithms in python.

ALGORITHM:

MAIN FUNCTION:

STEP 1: Start the program.

STEP 2: Call the function factorial(X)

STEP 3: Get X

STEP 4: print (Factorial)

STEP 5: Stop

SUB FUNCTION FACTORIAL(X):

STEP 1: if (X==1) then return factorial=1

STEP 2: else return fact= X* factorial(X-1)

PROGRAM/SOURCE CODE:

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
print("Enter number")
num = int(input())
print("The factorial of", num, "is", factorial(num))
```

OUTPUT:

Enter number

4

The factorial of 4 is 24

```
In [1]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter number

4
The factorial of 4 is 24
```

RESULT:

Thus, the Program to implement recursive algorithm in python has been executed successfully.

EX NO 3 IMPLEMENT LIST ADT USING PYTHON ARRAYS
DATE :

AIM:

To Implement List ADT using Python arrays.

ALGORITHM:

Step 1: Start the Program.

Step 2: Assign n=0.

Step 3: Create an empty list named as list1=[].

Step 4: while n less than 5, Then go to step 4.1.

Step 4.1: print "1.append 2.remove 3.size 4.display 5.exit"

Step 4.2: Read the input value n.

Step 4.3: if n equal to equal to 1, Then go to step 4.3.1.

Step 4.3.1: print "enter number"

Step 4.3.2: Read the input value num.

Step 4.3.3: Append num to list1.

Step 4.4: elif n equal to equal to 2, Then go to step 4.4.1.

Step 4.4.1: print "enter number to be removed"

Step 4.4.2: Read the input value num.

Step 4.4.3: if num in list1, Then go to step 4.4.3.1.

Step 4.4.3.1: Remove num from list1.

Step 4.4.4: else, go to the step 4.4.4.1.

Step 4.4.4.1: print "The number not in list"

Step 4.5: elif n equal to equal to 3, Then go to step 4.5.1.

Step 4.5.1: print "size: ", length of the list1.

Step 4.6: elif n equal to equal to 4, go to step 4.6.1.

Step 4.6.1: print the list 1

Step 5: End the program.

PROGRAM/SOURCE CODE:

```
n = 0
list1 = []
while n < 5:
        print('1.append 2.remove 3. size 4.display 5.exit')
        n = int(input())
        if n == 1:
                print('enter a number')
                num = int(input())
                list1.append(num)
        elif n == 2:
                print('enter a number to be removed')
                num = int(input())
                if num in list1:
                        list1.remove(num)
                else:   print('number not in list')

        elif n == 3:


                print('Size: ' , len(list1))
        elif n == 4:
                print(list1)
```

OUTPUT:

```
1.append 2.remove 3. size 4.display
5.exit 1
enter number
10
1.append 2.remove 3. size 4.display
5.exit 1
enter number
20
1.append 2.remove 3. size 4.display
5.exit 3
Size:  2
1.append 2.remove 3. size 4.display
5.exit 4
[10, 20]
1.append 2.remove 3. size 4.display
5.exit 2
enter number to be removed
20
1.append 2.remove 3. size 4.display
5.exit 2
```

enter number to be removed
40
number not in list
1.append 2.remove 3. size 4.display
5.exit 5

```
In [2]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
1.append 2.remove 3. size 4.display 5.exit

1
enter a number

10
1.append 2.remove 3. size 4.display 5.exit

1
enter a number

20
1.append 2.remove 3. size 4.display 5.exit

3
Size:  2
1.append 2.remove 3. size 4.display 5.exit

4
[10, 20]
1.append 2.remove 3. size 4.display 5.exit

2
enter a number to be removed

20
1.append 2.remove 3. size 4.display 5.exit

2
enter a number to be removed

40
number not in list
1.append 2.remove 3. size 4.display 5.exit

5
```

RESULT:

Thus, the Program to Implement List ADT using Python arrays has been executed successfully.

EX NO 4 LINKED LIST IMPLEMENTATIONS OF LIST
DATE :

AIM:

To Implementation of Linked list using python list

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create a node with the data value and next value as “None”.

STEP 3: Create a single linked list with the parameter self and assign head value= ”None” .

STEP 4: Calling the “printlist” function inorder to define it.

STEP 5: Assign printval=self.headval.

STEP 5.1: If printval is none it will print the headvalue.

Print(printval=self.headval).

STEP 5.2: while printval is not none it will print the data available in the list.

Print(printval.dataval).

Printval=printval.dataval.

STEP 6: Calling the “insert” function to add a new node to the existing list.

STEP 6.1: Create a new node with a new data and assign Newnode=node(newdata).

STEP 6.2: If Self.headval==None then self.headval=newnode.

STEP 6.3: To insert node at the beginning, self.headval.dataval is greater than or equal to newnode then newnode next value is self.headval then self.headval=newnode.

STEP 6.4: To insert a newnode in the specific position nodepos=self.headval.

STEP 6.4.1: While nodeposition next value is lesser than newnode Than nodeposition=nodeposition.nextval.

STEP 6.4.2: If nodeposition.nextval==none and when data value of Nodeposition is lesser than newnode then its next value is assigned to newnode.

STEP 6.4.3: Else newnode next value is declared to nodeposition next value Then nodepos.nextval=newnode .

STEP 7: Calling “search” function to search a particular element in a list with parameters

(self,newdata).

STEP 7.1: Assigning nodeposition of search as head value.

STEP 7.2: If data of nodeposition==newnode.

STEP 7.3: Print("Element present").

STEP 7.3.1- return.

STEP 7.4: If node position is next value of node

STEP 7.5: Print("Element not present")

STEP 8: Calling the "delete" function to delete a node in a list

STEP 8.1: Assign node position=head value

` **STEP 8.2:** If newdata=nodeposition data value and next value of node position is none

STEP 8.3: print("This is the only element present")

STEP 8.4: else Assign nodeposition next value=head value.

STEP 8.4.1 return

STEP 8.5: To check the next value of nodeposition and data value is not newdata.

STEP 8.5.1- Assign nodeposition to next value of nodeposition.

STEP 8.6: If next value of nodeposition is not none.

STEP 8.6.1: Assign temporary variable "temp" to the next value of node position

STEP 8.6.2: then declare next value of nod position=temp.nextval.

STEP 9: Create an object "list1" and assign value singlelinkedlist().

STEP 10: Declare ch=0.

STEP 11: while (ch<5)

STEP 12: print("1.insert 2.delete 3.print 4. Search 5.exit").

STEP 13: If (ch==1) to insert an element by getting the value as input and calling the predefined "insert" function by print("Insert the value to be inserted:").

STEP 14: If (ch==2) to delete a element by getting the value as input and calling the Predefined "delete" function by print("Enter the value to be inserted:").

STEP 15: If (ch==3) to print the elements of the list by calling the predefined "printlist" function.

By print("Values to be printed are:").

STEP 16: If (ch==4) to search an element of the list by taking the values of list by calling the.

Predefined “search function” by print(“Enter the search element:”).

STEP 17: If (ch==5) Exit the program.

STEP 18: End of the program.

PROGRAM/SOURCE CODE:

```
class node:
        def__init_(self,dataval=None):
                self.dataval=dataval;
                self.nextval=None
class singlylinkedlist:
        def__init_(self):
                self.headval=None
        def listprint(self):
                printval=self.headval
                while printval is not
                None:
                        print(printval.dataval)
                        printval=printval.nextval
        #Adding newnode in sorted way in the list
        def insert(self,newdata):
                newnode=node(newdata)
                if(self.headval==None):
                        self.headval=newnode
                elif(self.headval.dataval>=newdata):
                        newnode.nextval=self.headval
                        self.headval=newnode
                else:
                        nodepos=self.headval
                        while(nodepos.nextval and nodepos.nextval.dataval<newdata):
                                nodepos=nodepos.nextval
                        if(nodepos.nextval==None):
                                if(nodepos.dataval<newdata):
                                        nodepos.nextval=newnode
                                else:   newnode.nextval=self.headval
                                        self.headval=newnode


                        else:
                                newnode.nextval=nodepos.nextval
                                nodepos.nextval=newnode
        #Search
```

```
def search(self,newdata):
        nodepos=self.headval
        while(nodepos):
```

```
                    if(nodepos.dataval==newdata):
                        print("element present")
                        return
                    nodepos=nodepos.nextval
                print("element not present")
        #Deleting the node
        def deletenode(self,newdata):
                nodepos=self.headval
                if(nodepos.dataval==newdata):
                    if(nodepos.nextval is None):
                        print("this is the only element present")
                    else:
                        self.headval=nodepos.nextval
                        return
                while(nodepos.nextval and nodepos.nextval.dataval is not newdata):
                    nodepos=nodepos.nextval
                if(nodepos.nextval is not None):
                    temp=nodepos.nextval
                    nodepos.nextval=temp.nextval
list1=singlylinkedlist()
ch=0
while(ch<5):
        print("1. insert 2. delete 3.print 4.search 5.exit")
        ch=int(input())
        if(ch==1):
                print("insert value to be inserted")
                element=int(input())
                list1.insert(element)
                continue
        elif(ch==2):
                print("enter value to be deleted")
                element=int(input())
                list1.deletenode(element)
                continue
        elif(ch==3):
                print("values are: ")
                list1.listprint()
                continue
        elif(ch==4):
                print("Enter search element ")
                element=int(input())
                list1.search(element)
                continue
```

OUTPUT:

```
1. insert 2. delete 3.print 4.search 5.exit
1
insert value to be inserted
10
1. insert 2. delete 3.print 4.search 5.exit
1
insert value to be inserted
20
1. insert 2. delete 3.print 4.search 5.exit
3
values are:
10
20
1. insert 2. delete 3.print 4.search 5.exit
4
Enter search element
20
element present
1. insert 2. delete 3.print 4.search 5.exit
2
enter value to be deleted
20
1. insert 2. delete 3.print 4.search 5.exit
3
values are:
10
1. insert 2. delete 3.print 4.search
5.exit 5
```

```
In [3]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
1. insert 2. delete 3.print 4.search 5.exit

1
insert value to be inserted

10
1. insert 2. delete 3.print 4.search 5.exit

1
insert value to be inserted

20
1. insert 2. delete 3.print 4.search 5.exit

3
values are:
10
20
1. insert 2. delete 3.print 4.search 5.exit

4
Enter search element

20
element present
1. insert 2. delete 3.print 4.search 5.exit

2
enter value to be deleted

20
1. insert 2. delete 3.print 4.search 5.exit

3
values are:
10
1. insert 2. delete 3.print 4.search 5.exit

5
```

RESULT:

Thus, the Program to Implementation of Linked list using python list has been executed successfully.

EX NO 5 IMPLEMENTATION OF STACK AND QUEUE ADTSs
DATE :

AIM:

To Implementation of Stack and Queue ADTs using python classes

ALGORITHM:

Step1: Select the structure chosen by the user as 1 for stack and 2 for queue.

Step 2: If press 1, it will call class Stack.

Step 3: It will keep on checking till n1 becomes 5.

Step 4: If press 1, accept from the user values and it will store in the stack.

Step 5: If press 2, it will remove value from the stack.

Step 6: If press 3, it will show the size of the stack.

Step 7: If press 4, it will show items of the stack.

Step 8: If press 2, it will call class Queue.

Step 9: Accept from the user 1. 1.enqueue 2.dequeue 3.size 4.display 5.exit

Step 10: It will keep on checking till n1 becomes 5.

Step 11: If press 1, accept from the user values and it will store in the enqueue.

Step 12: If press 2, it will perform dequeue and

display the message dequeue done.

Step 13: If press 3, it will show the size of the queue.

Step 14: If press 4, it will show items of the queue.

PROGRAM/SOURCE CODE:

```
# Stack ADT
class Stack:
    def__init_(self):
        self.items = []
    def isEmpty(self):
        return self.items ==
        []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def size(self):
        return len(self.items)
# Queue ADT
class Queue:
    def__init_(self):
        self.items = []
    def isEmpty(self):
        return self.items ==
        []
    def enqueue(self,item):
        self.items.append(item)
    def dequeue(self):
        return self.items.pop(0)
    def front(self):
        return self.items[len(self.items)-1]
    def size(self):
        return len(self.items)
# Stack ADT operation example
n1 = 0
n2 = 0
print('Enter number. 1.Stack 2.Queue')
i = int(input())
if i == 1:
    s=Stack()
    while n1 < 5:
        print("Enter number. 1.push 2.pop 3.size 4.display 5.exit")
        n1 = int(input())
        if n1 == 1:
            print('Enter number')
            num = int(input())
            s.push(num)
```

```
elif n1 == 2:
    s.pop()
```

```
                print('Value popped')
            elif n1 == 3:
                print('Size:' , s.size())
            elif n1 == 4:
                print(s.items)
        elif i == 2:
            q = Queue()
            while n2 < 5:
                print("Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit")
                n2 = int(input())
                if n2 == 1:
                    print('Enter number')
                    num = int(input())
                    q.enqueue(num)
                elif n2 == 2:
                    q.dequeue()
                    print('dequeue done')
                elif n2 == 3:
                    print('Size:' , q.size())
                elif n2 == 4:
                    print(q.items)
```

OUTPUT:

OUTPUT 1:

```
Enter number. 1.Stack 2.Queue
1
Enter number. 1.push 2.pop 3.size 4.display 5.exit
1
Enter number
10
Enter number. 1.push 2.pop 3.size 4.display 5.exit
1
Enter number
20
Enter number. 1.push 2.pop 3.size 4.display 5.exit
3
Size: 2
Enter number. 1.push 2.pop 3.size 4.display 5.exit
4
[10, 20]
Enter number. 1.push 2.pop 3.size 4.display 5.exit
2
Value popped
```

Enter number. 1.push 2.pop 3.size 4.display 5.exit
4
[10]
Enter number. 1.push 2.pop 3.size 4.display 5.exit
5

```
In [4]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter number. 1.Stack 2.Queue

1
Enter number. 1.push 2.pop 3.size 4.display 5.exit

1
Enter number

10
Enter number. 1.push 2.pop 3.size 4.display 5.exit

1
Enter number

20
Enter number. 1.push 2.pop 3.size 4.display 5.exit

3
Size: 2
Enter number. 1.push 2.pop 3.size 4.display 5.exit

4
[10, 20]
Enter number. 1.push 2.pop 3.size 4.display 5.exit

2
Value popped
Enter number. 1.push 2.pop 3.size 4.display 5.exit

4
[10]
Enter number. 1.push 2.pop 3.size 4.display 5.exit

5
```

OUTPUT 2:

Enter number. 1.Stack 2.Queue
2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
1
Enter number
10
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
1
Enter number
20
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
3
Size: 2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
4
[10, 20]

Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
2
dequeue done
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
4
[20]
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit
5

```
In [6]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter number. 1.Stack 2.Queue

2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

1
Enter number

10
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

1
Enter number

20
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

3
Size: 2
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

4
[10, 20]
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

2
dequeue done
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

4
[20]
Enter number. 1.enqueue 2.dequeue 3.size 4.display 5.exit

5
```

RESULT:

Thus, the Program to Implement simple ADTs using python has been executed successfully.

EX NO 6 APPLICATIONS OF LIST, STACK AND QUEUE ADTs
DATE :

AIM:

To Implement Application of list, stack, Queue ADTs using python classes

1. Application of Stack **(Infix Expression to Postfix Expression).**
2. Application of Queue **(First Come First Serve (FCFS Scheduling)).**

ALGORITHM:

1. Application of Stack (Infix Expression to Postfix Expression).

Step 1: Start the program.

Step 2: Assign Expression exp.

Step 3: Create an object obj using the class conversion with the parameters length of the Expression.

Step 4: Call the member function infix_to_postfix with one parameter exp which defined inside the class conversion.

Step 5: Take a for loop i in exp

Step 5.1: if self.isOperand with parameter i, Then go the step 5.1.1

Step 5.2: elif i==’(‘, Then go to step 5.2.1

Step 5.2.1: call member function push which is defined inside the class Conversion with parameter i.

Step 5.3: elif i==’)’, Then go to step 5.3.1

Step 5.3.1: while ((not self.isEmpty()) and self.peek() != '('),Then do steps 5.3.1.1, 5.3.1.2, 5.3.1.3

Step 5.3.1.1: Call the member function pop which is defined inside the class conversion and assign it in a.

Step 5.3.1.2: Append a in self.output

Step 5.3.1.3: if (not self.isEmpty() and self.peek() != ('), Then return-1 else self.pop()

Step 5.4: else while(not self.isEmpty() and self.notGreater(i)), Then do steps 5.4.1, 5.4.2

Step 5.4.1: Append self.pop() in self.output

Step 5.4.2: call the member function push which is defined inside the class conversion with parameter i.

Step 5.5: while not self.isEmpty(), Then do step 5.5.1

Step 5.5.1: Append self.pop() in self.output.

Step 5.6: print ("".join(self.output))

Step 6 : End the program.

2. Application of Queue (First Come First Serve (FCFS Scheduling)).

Step 1: Start the program.

Step 2: if __name__==”__main__” ,Then go to Step 2.1

Step 2.1: Assign processes= [1,2,3]

Step 2.2: Assign n=length of the processes.

Step 2.3: Assign burst time(bt)= [10,5,8]

Step 2.4: Call the find_avg_time functionwith parameters processes,n,bt.

Step 2.4.1: Assign wt=[0]*n

Step 2.4.2: Assign tat=[0]*n

Step 2.4.3: Assign total_wt=0

Step 2.4.4: Assign total_tat=0

Step 2.4.5: Call the find_waiting_time function with parameters processes,n,bt,wt.

Step 2.4.5.1: Assign wt[0]=0

Step 2.4.5.2: Take a for loop i in range of 1,n

Step 2.4.5.2.1: Assign wt[i]=bt[i-1]+wt[i-1]

Step 2.4.6: Call the find_turn_around _time function with parameters processes,n,bt,wt,tat.

Step 2.4.6.1: Take a for loop I in range of n

Step 2.4.6.2: assign tat[i]=bt[i]+wt[i]

Step 2.4.7: “For display processes along with details” print “processes burst time” + ”waiting time”+ ”turn around time”

Step 2.4.8: “To calculate total wt and total turn around time” Take a for loop I in range of n

Step 2.4.8.1: Assign total_wt=total_wt+wt[i]

Step 2.4.8.2: Assign total_tat=total_tat+tat[i]

Step 2.4.8.3: print(“”+str(i+1)+”\t\t”+str(bt[i])+”
\t”+str(wt[i])+”\t\t”+str(tat[i]))

Step 2.4.8.4: print("Average waiting time="+str(total_wt/n))

Step 2.4.8.5: print("Average turn around time="+str(total_tat/n))

Step 3: End the program.

PROGRAM/SOURCE CODE:

1.(Application of stack (Infix Expression to postfix Expression)):

```
# Class to convert the expression
class Conversion:
    # Constructor to initialize the class variables
    def__init_(self, capacity):
        self.top = -1
        self.capacity =capacity
        #This array is used a stack
        self.array  = []
        #Precedence setting
        self.output = []
        self.precedence = {'+':1, '-':1, '*':2, '/':2, '^':3}
    # check if the stack is empty
    def isEmpty(self):
        return True if self.top == -1 else False
    # Return the value of the top of the stack
    def peek(self):
        return self.array[-1]
    # Pop the element from the stack
    def pop(self):
        if not self.isEmpty():
            self.top -= 1
            return self.array.pop()
```

```
    else:
        return "$"
# Push the element to the stack
def push(self, op):
    self.top += 1
    self.array.append(op)
# A utility function to check is the given character is operand
def isOperand(self, ch):
    return ch.isalpha()
# Check if the precedence of operator is strictly less than top of stack or not
def notGreater(self, i):
    try:
        a = self.precedence[i]
        b = self.precedence[self.peek()]
        return True if a <= b else False
    except KeyError:
        return False
     # The main function that converts given infix expression to postfix expression
def infix_To_Postfix(self, exp):
    # Iterate over the expression for conversion
    for i in exp:
        #if the character is an operand, add it to output
        if self.isOperand(i):
            self.output.append(i)
        elif i == '(':
            self.push(i)
        #if the scanned character is an ')',pop and output from the stack until and '('is found
        elif i == ')':
            while( (not self.isEmpty()) and self.peek() != '('):
                a = self.pop()
                self.output.append(a)
```

```
            if (not self.isEmpty() and self.peek() != '('):
                return -1
            else:
                self.pop()
        #An operator is encountered
        else:
            while(not self.isEmpty() and self.notGreater(i)):
                self.output.append(self.pop())
                self.push(i)
            #pop all the operator from the stack
            while not self.isEmpty():
                self.output.append(self.pop())
            print ("".join(self.output))
# Driver program to test above function
exp = "a+b*(c^d-e)^(f+g*h)-i"
obj = Conversion(len(exp))
obj.infix_To_Postfix(exp)
```

OUTPUT:

a

ab

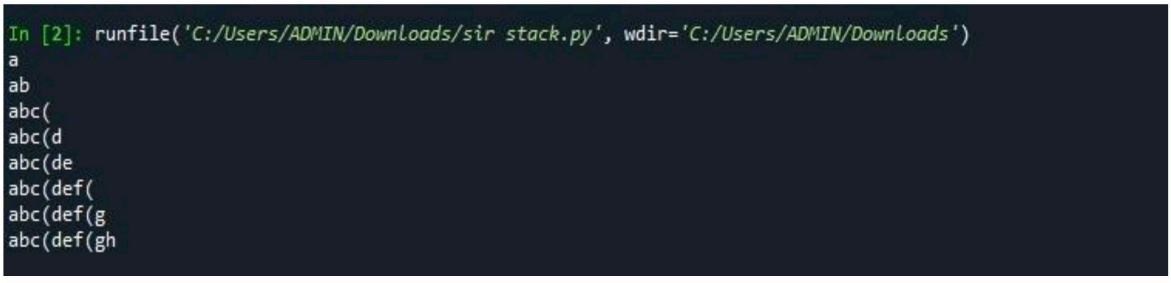
abc(abc(

d abc(de

abc(def(

abc(def(g

abc(def(gh



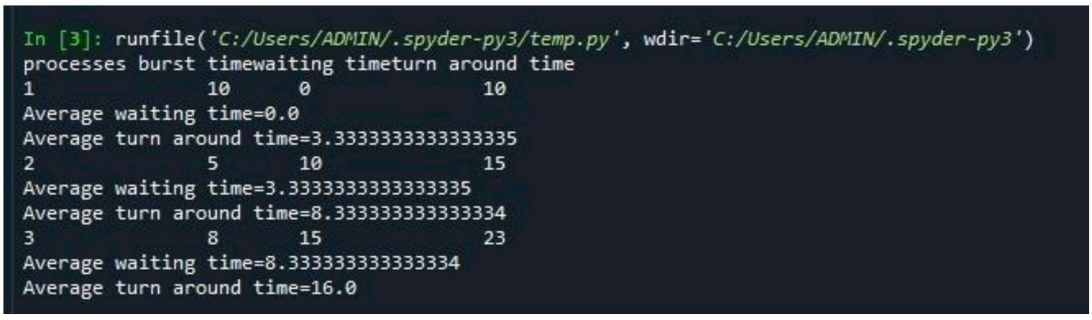
2.(Application of Queue (First come First serve (FCFS Scheduling))):

```
#Function to find the waiting time for all processes
def find_waiting_time(processes,n,bt,wt):
    #waiting time for first process is 0
    wt[0]=0
    #calculating waiting time
    for i in range(1,n):
        wt[i]=bt[i-1]+wt[i-1]
    #Function to calculate turn around time
def  find_turn_around_time(processes,n,bt,wt,tat):
    #calculating turn around time time by adding bt[i]+wt[i]
    for i in range(n):
        tat[i]=bt[i]+wt[i]
    #Function to calculate average time
def find_avg_time(processes,n,bt):
    wt=[0]*n
    tat=[0]*n
    total_wt=0
    total_tat=0
    #Function to find waiting time of all processes
    find_waiting_time(processes,n,bt,wt)
    #Function to find turn around time for all processes
    find_turn_around_time(processes,n,bt,wt,tat)
    #Display processes along with all details
    print("processes burst time"+"waiting time"+"turn around time")
    #calculate total waiting time and total turn around time
    for i in range(n):
        total_wt=total_wt+wt[i]
        total_tat=total_tat+tat[i]
        print(""+str(i+1)+"\t\t"+str(bt[i])+"\t"+str(wt[i])+"\t\t"+str(tat[i]))
        print("Average waiting time="+str(total_wt/n))
        print("Average turn around time="+str(total_tat/n))
```

```
#Driver code
if__name_=="_main_":
    #process id's
    processes=[1,2,3]
    n=len(processes)
    #Burst time of all processes
    bt=[10,5,8]
    find_avg_time(processes,n,bt)
```

OUTPUT:

```
processes burst timewaiting timeturn around
time 1          10      0               10
Average waiting time=0.0
Average turn around time=3.3333333333333335
2               5       10              15
Average waiting time=3.3333333333333335
Average turn around time=8.333333333333334
3               8       15              23
Average waiting time=8.333333333333334
Average turn around time=16.0
```



RESULT:

Thus, the program to Implement Application of List, Stack and Queue ADTs using python Classes **(for Stack (Infix Expression to postfix Expression) and for Queue (FCFS Scheduling))** has been executed successfully.

IMPLEMENTATION OF SORTING AND SEARCHING ALGORITHMS

EX NO 7(a) LINEAR SEARCH

DATE :

AIM:

To Implement Linear search using Python Code.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Define a function called linear_search (alist, key).

STEP 3: Set for i in range(len(alist)) and check if alist[i] == key, then return i.

STEP 4: Then return -1

STEP 5: Get the values of alist and split it.

STEP 6: Set alist = int(x) for x in alist.

STEP 7: And also get the value of key to find the element in the list.

STEP 8: Set index = linear_search (alist, key).

STEP 9: Check if index < 0, then print key, “was not found”.

STEP 10: Else print key, “Was found at”, index.

STEP 11: Stop the program.

PROGRAM/SOURCE CODE:

```
def linear_search(alist, key):
    """Return index of key in alist. Return -1 if key not present."""
    for i in range(len(alist)):
        if alist[i] == key:
            return i
    return -1
alist = input('Enter the list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))
index = linear_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key, index))
```

OUTPUT:

Enter the list of numbers: 1 2 3 4
The number to search for: 2
2 was found at index 1

RESULT:

Thus, the program to Implement Linear Search using Python code has been executed successfully.

EX NO 7(b) BINARY SEARCH

DATE :

AIM:

To Implement Binary Search using python code.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Define a function called binary_search (alist, key).

STEP 3: Set a start and end values to point the boundary of the list to be searched.

STEP 4: Set while start < end, then find the mid position of the given alist called mid.

STEP 5: Compare the key value with mid position element.

a. If alist[mid] > key

Set end = mid.

b. Elif alist[mid] < key

Set start = mid +1.

c. Else

Return the value of mid

STEP 7: Continue the step 4 & 5 until the element found or reach the end of the list.

STEP 8: Then return -1.

STEP 9: Get the values of the alist and split it.

STEP 10: Set alist = int(x) for x in alist.

STEP 11: And also get the value of key to find the element in the list.

STEP 12: Set index = binary_search (alist, key).

STEP 13: check if index < 0, then print key, “was not found”.

STEP 14: Else print key, “was found at”, index.

STEP 15: Stop the program.

PROGRAM/SOURCE CODE:

```
def binary_search(alist, key):
    """Search key in alist[start... end - 1]."""
    start = 0
    end = len(alist)
    while start < end:
        mid = (start + end)//2
        if alist[mid] > key:
            end = mid
        elif alist[mid] < key:
            start = mid + 1
        else:
            return mid
    return -1
alist = input('Enter the sorted list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))
index = binary_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key, index))
```

OUTPUT:

Enter the sorted list of numbers: 1 2 3 4 5
The number to search for: 3
3 was found at index 2.

RESULT:

Thus, the Program to implement Binary Search using python code has been executed successfully.

EX NO 7(c) BUBBLE SORT
DATE :

AIM:

To Implement Bubble Sort using python code.

ALGORITHM:

Step 1: Start the Program.

Step 2: Initialize unsorted list elements.

Step 3: Assign num equal to length of the elements

Step 4: Call the function bubblesort two parameters elements, number.

Step 4.1: Take a for loop i in range of number.

Step 4.1.1: Take another for loop j in range of 1,number-i

Step 4.1.1.1: if elements[j-1] greater than elements[j], Then go to the step 4.1.1.1.1

Step 4.1.1.1.1: Assign temp=elements[j-1]

Step 4.1.1.1.2: and Assign elements[j-1]=elements[j]

Step 4.1.1.1.3: Assign elements[j]=temp

Step 4.1.2: Print “Display”

Step 4.1.3: print(elements)

Step 5: End the program.

PROGRAM/SOURCE CODE:

```
def bubblesort(elements,number):
    for i in range(number):
        for j in range(1,(number-i)):
            if(elements[j-1]>elements[j]):
                #swap numbers
                temp=elements[j-1]
                elements[j-1]=elements[j]
                elements[j]=temp
    print("Display")
    print(elements)
elements=[1,4,2,7,5,10,5]
number=len(elements)
bubblesort(elements,number)
```

OUTPUT:

Display
[1, 2, 4, 5, 5, 7, 10]

```
In [9]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Display
[1, 2, 4, 5, 5, 7, 10]
```

RESULT:

Thus, the Program to implement Bubble Sort using python code has been executed successfully.

EX NO 7(d) INSERTION SORT
DATE :

AIM:

To Implement Insertion Sort using python code.

ALGORITHM:

Step 1: Start the Program.

Step 2: Initialize unsorted list elements.

Step 3: Assign num equal to length of the elements

Step 4: Call the function insertionsort two parameters elements, number.

Step 4.1: Take a for loop i in range of 1, number

Step 4.1.1: Assign temp= elements[i]

Step 4.1.2: Assign j=i

Step 4.1.3: while j greater than or equal to zero and tmp less than elements[j-1], Then go to step 4.1.3.1.

Step 4.1.3.1: Assign elements[j]=elements[j-1]

Step 4.1.3.2: Assign j=j-1

Step 4.1.3.3: Assign elements[j]=tmp

Step 4.1.4: print “sorted elements”

Step 4.1.5: print(elements)

Step 5: End the program.

PROGRAM/SOURCE CODE:

```
def insertionsort(elements,number):
    for i in range(1,number):
        tmp=elements[i]
        j=i
        while(j>=0 and tmp<elements[j-1]):
            elements[j]=elements[j-1]
            j=j-1
        elements[j]=tmp
    print("sorted elements")
    print(elements)
elements=[1,4,3,6,100,8]
number=len(elements)
insertionsort(elements,number)
```

OUTPUT:

sorted elements
[1, 3, 4, 6, 8, 100]

RESULT:

Thus, the Program to Implement Insertion Sort using python code has been executed successfully.

EX NO 7(e) MERGE SORT
DATE :

AIM:

To Implement Merge Sort using python code

ALGORITHM:

Step 1: Start the program.

Step 2: Assign unsorted list and set it to elements.

Step 3: Set arraysize=length of the elements.

Step 4: Set temp=[0]*arraysize

Step 5: Call the mergesort function with the parameters elements,temp,arraysize

Step 5.1: Call the another m_sort function with parameters elements,temp,left,right

Step 5.2: if right greater than left, Then do steps 5.2.1,5.2.2,5.2.3,5.2.4.

Step 5.2.1: int((right+ left)/2) and assign it in mid.

Step 5.2.2: Call the member function m_sort with parameters element,temp,left,mid.

Step 5.2.3: Call the member function m_sort again with parameters element,temp,mid+1,right.

Step 5.2.4: Call the member function merge with parameters element,temp,left,mid+1,right.

Step 6: End the program.

PROGRAM/SOURCE CODE:

```
def mergesort(elements,temp,arraysize):
    m_sort(elements,temp,0,arraysize-1)
def m_sort(elements,temp,left,right):
    if(right>left):
        mid=int((right+left)/2)
        m_sort(elements, temp, left, mid)
        m_sort(elements, temp, mid+1, right)
        merge(elements,temp,left,mid+1,right)
def merge(elements,temp,left,mid,right):
    left_end=mid-1
    temp_pos=left
    num_elements=right-left+1
    while((left<=left_end) and (mid<=right)):
        if(elements[left]<=elements[mid]):
            temp[temp_pos]=elements[left]
            temp_pos=temp_pos+1
            left=left+1
        else:   temp[temp_pos]=elements[mid]
            temp_pos=temp_pos+1
            mid=mid+1
    while(left<=left_end):
        temp[temp_pos]=elements[left]
        temp_pos=temp_pos+1
        left=left+1
    while(mid<=right):
        temp[temp_pos]=elements[mid]
        temp_pos=temp_pos+1
        mid=mid+1
    for i in range(num_elements):
        elements[right]=temp[right]
        right=right-1
    print(elements)
elements=[5,3,7,8,1,2,9]
arraysize=len(elements)
temp=[0]*arraysize
mergesort(elements, temp, arraysize)
```

OUTPUT:

[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[1, 2, 3, 5, 7, 8, 9]

```
In [11]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[3, 5, 7, 8, 1, 2, 9]
[1, 2, 3, 5, 7, 8, 9]
```

RESULT:

Thus, the program to Implement Merge Sort using python code has been executed successfully.

EX NO 7(f) QUICK SORT
DATE :

AIM:

To Implement Quick Sort using python code.

ALGORITHM:

Step 1: Start the program.

Step 2: Assign the unsorted list and set it to the elements.

Step 3: Call the quicksort function with parameters elements,0,length of the elements.

Step 3.1: Set pivot=right-1

Step 3.2: Set i=left

Step 3.3: Set j=right-1

Step 3.4: if left less than right, Then go to step 3.4.1

Step 3.4.1: Take a while loop as True

Step 3.4.1.1: Take another while loop as elements[i] less than elements[pivot]

Step 3.4.1.1.1: Set i=i+1

Step 3.4.1.2: Take another while loop as elements[j] greater than elements[pivot]

Step 3.4.1.2.1: Set j=j+1

Step 3.4.1.3: if i less than j, Then go to step 3.4.1.3.1.

Step 3.4.1.3.1: Set temp=elements[i]

Step 3.4.1.3.2: Set elements[i]=elements[j]

Step 3.4.1.3.2: Set elements[j]=temp

Step 3.4.1.4: else break

Step 3.4.1.5: Set temp=elements[i]

Step 3.4.1.6: Set elements[i]=elements[pivot]

Step 3.4.1.7: Set elements[pivot]=temp

Step 3.4.1.8: print (elements, 'mid:' i)

Step 3.4.1.9: Call the quick sort function with left sub array

Step 3.4.1.10: Call the quick sort function with right sub array

Step 4: End the program.

PROGRAM/SOURCE CODE:

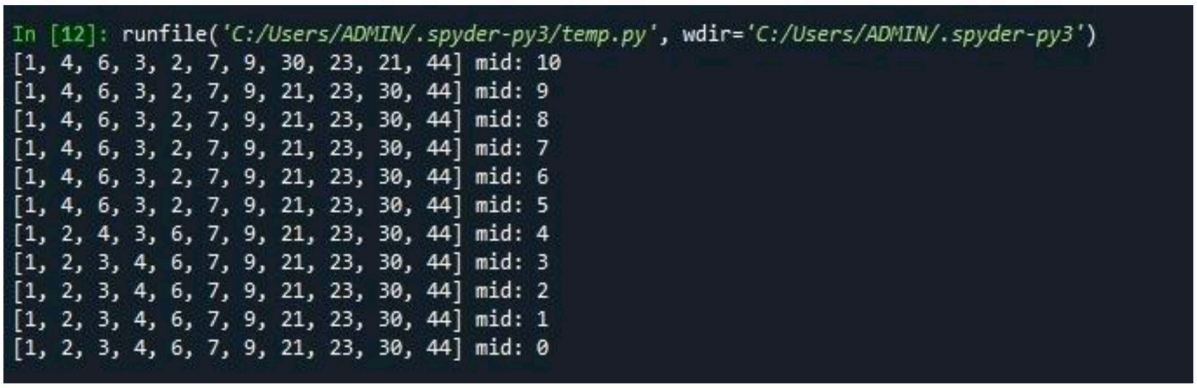
```
def quicksort(elements,left,right):
        pivot=right-1
        i=left
        j=right-1
        if(left<right):
                while(True):
                        while(elements[i]<elements[pivot]):
                                i=i+1
                        while(elements[j]>elements[pivot]):
                                j=j-1
                        if(i<j): temp=elements[i]
                                elements[i]=elements[j]
                                elements[j]=temp

                        else:   break

                temp=elements[i]
                elements[i]=elements[pivot]
                elements[pivot]=temp
                print(elements,'mid:',i)
                quicksort(elements, left, i)
                quicksort(elements, i+1, right)
elements=[1,4,6,3,2,7,9,30,23,21,44]
quicksort(elements, 0, len(elements))
```

OUTPUT:

[1, 4, 6, 3, 2, 7, 9, 30, 23, 21, 44] mid: 10
[1, 4, 6, 3, 2, 7, 9, 21, 23, 30, 44] mid: 9
[1, 4, 6, 3, 2, 7, 9, 21, 23, 30, 44] mid: 8
[1, 4, 6, 3, 2, 7, 9, 21, 23, 30, 44] mid: 7
[1, 4, 6, 3, 2, 7, 9, 21, 23, 30, 44] mid: 6
[1, 4, 6, 3, 2, 7, 9, 21, 23, 30, 44] mid: 5
[1, 2, 4, 3, 6, 7, 9, 21, 23, 30, 44] mid: 4
[1, 2, 3, 4, 6, 7, 9, 21, 23, 30, 44] mid: 3
[1, 2, 3, 4, 6, 7, 9, 21, 23, 30, 44] mid: 2
[1, 2, 3, 4, 6, 7, 9, 21, 23, 30, 44] mid: 1
[1, 2, 3, 4, 6, 7, 9, 21, 23, 30, 44] mid: 0



RESULT:

Thus, the program to Implement Quick Sort using python code has been executed successfully.

EX NO 8 IMPLEMENTATION OF HASH TABLES
DATE :

AIM:

To Implement Hash Tables using Python code.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create a Hashtable of size 10 as nested list.

STEP 3: Call the function hashing(key value) that returns, Keyvalue%len(hashtable)

STEP 4: Inserting elements in Hashtable.

Step 4.1: Call the function insert with parameters as hashtable, key value, value.

Step 4.2: Assign Hashing(key value) to hash key.

Step 4.3: Use append function to insert key-value pairs in hash Table.

STEP 5: Displaying Hash table

Step 5.1: Call the function display hash(Hash table)

Step 5.2: Print()

SUB-FUNCTION OF DISPLAY_HASH(HASH TABLE)

STEP 1: For i in range(len(hashtable))

Step 1.1: Print(i)

STEP 2: For j in hashtable[i]:

Step 2.1: Print (j)

PROGRAM/SOURCE CODE:

```
# Function to display hashtable
def display_hash(hashTable):
    for i in range(len(hashTable)):
        print(i, end = " ")
        for j in hashTable[i]:
            print("-->", end = " ")
            print(j, end = " ")
        print()
# Creating Hashtable as
# a nested list.
HashTable = [[] for _ in range(10)]
# Hashing Function to return
# key for every value.
def Hashing(keyvalue):
    return keyvalue % len(HashTable)
# Insert Function to add
# values to the hash table
def insert(Hashtable, keyvalue, value):
    hash_key = Hashing(keyvalue)
    Hashtable[hash_key].append(value)
# Driver Code
insert(HashTable, 10, 'Allahabad')
insert(HashTable, 25, 'Mumbai')
insert(HashTable, 20, 'Mathura')
insert(HashTable, 9, 'Delhi')
insert(HashTable, 21, 'Punjab')
insert(HashTable, 21, 'Noida')
display_hash (HashTable)
```

OUTPUT:

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

```
In [13]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
0 --> Allahabad --> Mathura
1 --> Punjab --> Noida
2
3
4
5 --> Mumbai
6
7
8
9 --> Delhi
```

RESULT:

Thus, the program to Implement Hash Tables using Python code has been executed successfully.

EX NO 9 TREE REPRESENTATION AND TRAVERSAL ALGORITHMS

DATE :

AIM:

To Implement Tree Representation and Traversal algorithms using python code.

ALGORITHM:

Step 1: Start the program.

Step 2: Read the input value r.

Step 3: Create an object root with Node(r)

Step 4: Read the input value i.

Step 5: Take a for loop with c in range of i.

Step 5.1: Read the input value c until the loop ends.

Step 5.2: Insert the values of c into the root.

Step 6: Print "INORDER TRAVERSAL(LEFT->RIGHT->ROOT->RIGHT) ALWAYS IN ASCENDING ORDER".

Step 7: Call the member function pt1 which is defined inside the class Node.

Step 7.1: if self.left ,then go to step 7.1.1.

Step 7.1.1: Call the member function pt1 with self.left recursively which is defined inside the class Node.

Step 7.2 Print self.data.

Step 7.3 if self.right,then go to step 7.3.1.

Step 7.3.1: Call the member function pt1 with self.right recursively which is defined inside the class Node.

Step 8: Print "POSTORDER TRAVERSAL(LEFT->RIGHT->ROOT):".

Step 9: Call the member function pt2 which is defined inside the class Node.

Step 9.1: if self.left,then go to step 9.1.1.

Step 9.1.1: Call the member function pt2 with self.left Recursively which is defined inside the class Node.

Step 9.2: if self.right,then go to step 9.2.1.

Step 9.2.1 Call the member function pt2 with self.right

Recursively which is defined inside the class Node.

Step 9.3: Print self.data.

Step 10: End the program.

PROGRAM/SOURCE CODE:

```
print("code for binary search tree traversal")
class Node:
 #ROOT
   def__init_(self, data):
      self.left = None
      self.right = None
      self.data = data
   def insert(self, data):
# Compare the new value with the parent node
      if self.data:
         if data < self.data:
            if self.left is None:
               self.left = Node(data)
            else:
               self.left.insert(data)
         elif data > self.data:
            if self.right is None:
               self.right = Node(data)
            else:
               self.right.insert(data)
      else:
         self.data = data
# Print the tree - INORDER TRAVERSAL -ALWAYS IN ASCENDING ORDER
   def pt1(self):
      if self.left:
         self.left.pt1()
      print( self.data) #INORDER
      if self.right:
         self.right.pt1()
# Print the tree - POSTORDER TRAVERSAL
   def pt2(self):
      if self.left:
         self.left.pt2()
      if self.right:
         self.right.pt2()
      print( self.data)     #POSTORDER
```

```
# Use the insert method to add nodes
r=int(input("root"))
root = Node(r)
i=int(input("enter number of child nodes"))
for c in range(i):
    c=int(input("child"))
    root.insert(c)
print("\nINORDER TRVERSAL(LEFT->ROOT->RIGHT)ALWAYS IN ASCENDING
ORDER :")
root.pt1()
print("\nPOSTORDER TRVERSAL(LEFT->RIGHT->ROOT) :")
root.pt2()
```

OUTPUT:

code for binary search tree traversal

root4

enter number of child nodes5

child1

child2

child3

child4

child5

INORDER TRVERSAL(LEFT->ROOT->RIGHT)ALWAYS IN ASCENDING ORDER :
1
2
3
4
5

POSTORDER TRVERSAL(LEFT->RIGHT->ROOT) :
3
2
1
5

4

```
In [15]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
code for binary search tree traversal

root4

enter number of child nodes5

child1

child2

child3

child4

child5

INORDER TRVERSAL(LEFT->ROOT->RIGHT)ALWAYS IN ASCENDING ORDER :
1
2
3
4
5

POSTORDER TRVERSAL(LEFT->RIGHT->ROOT) :
3
2
1
5
4
```

RESULT:

Thus, the program to Implement Tree Representation and Traversal algorithm using python code has been executed successfully.

EX NO 10 IMPLEMENTATION OF BINARY SEARCH TREES
DATE :

AIM:

To Implementation of Binary Search Trees using python code.

ALGORITHM:

Step 1: Start of the program

Step 2: create a object “r”, for class Node()

Step 3: insert the tree values by Calling the insert(ROOT, KEY) function

Step 3.1: if ROOT.KEY is NONE, return NODE(key)

ELSE

IF ROOT.VAL < KEY, return ROOT

ELSEIF ROOT.VAL<KEY, assign

ROOT.RIGHT = INSERT (ROOT.RIGHT, KEY)

ELSE ROOT.LEFT = INSERT(ROOT.LEFT, KEY)

Step 3.2: return ROOT

Step 4: Repeat Step 3, till desired

Step 5: Call Inorder(ROOT) function, to perform inorder traversal

Step 5.1: if ROOT, traverse LEFT of Tree by Reapting Step 5

Step 5.1.1: Print Value

Step 5.1.2: Traverse RIGHT of Tree by Repeating Step 5

Step 6: End of the Program.

PROGRAM/SOURCE CODE:

```
# Python program to demonstrate insert operation in binary search tree
# A utility class that represents an individual node in a BST
class Node:
    def __init_(self, key):
        self.left = None
        self.right = None
        self.val = key
# A utility function to insert a new node with the given key
def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if root.val == key:
            return root
        elif root.val < key:
            root.right = insert(root.right, key)
        else:    root.left = insert(root.left, key)

    return root


# A utility function to do inorder tree traversal
def inorder(root):
    if root:
        inorder(root.left)
        print(root.val)
        inorder(root.right)
# Driver program to test the above functions
# Let us create the following BST
# 50
# /       \
# 30      70
# / \ / \
# 20 40 60 80
r = Node(50)
r = insert(r, 30)
r = insert(r, 20)
r = insert(r, 40)
r = insert(r, 70)
r = insert(r, 60)
r = insert(r, 80)
print("Inorder traversal of the BST")
inorder(r)
```

OUTPUT:

Inorder traversal of the BST
20
30
40
50
60
70
80

```
In [16]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Inorder traversal of the BST
20
30
40
50
60
70
80
```

RESULT:

Thus, the program to Implement Binary Search Tree using python code has been executed successfully.

EX NO 11 IMPLEMENTATION OF HEAPS
DATE :

AIM:

To Implement Heaps using python code.

ALGORITHM:

Step 1: Start the Program**.**

Step 2: Initialize a=[]

Step 3: Read the input value r.

Step 4: Append r to a.

Step 5: Read the input Value i.

Step 6: Take a for loop with c in range of i.

Step 6.1: Read the input value of c until the loop ends**.**

Step 6.2: Append c to a.

Step 7: ***Call the in-build function Sort*** the values in the list a.

Step 8: Print "Tree Representation in Min Heap".

Step 9: Call the function traversed with three Parameters a, i=0, d=0.

Step 9.1: if i greater than or equal to len(a), Then go the step: 9.1.1.

Step 9.1.1: return

Step 9.2: Call the function ChildNodes with parameter i and assign it in l,r.

Step 9.3: Call the function traversed recursively with parameters a, r, d=d+1.

Step 9.4: Print " " *d+ str(a[i])

Step 9.5: Call the function traversed recursively with parameters a, l, d=d+1.

Step 10: ***Call the in-build function reverse*** with respect to a.

Step 11: Print "Tree Representation in Max Heap".

Step 12: Call the function traversed with three parameters a, i=0, d=0.

Step 12.1: if i greater than or equal to len(a), Then go the step: 12.1.1.

Step 12.1.1: return

Step 12.2: Call the function ChildNodes with parameter i and assign it in l,r.

Step 12.3: Call the function traversed recursively with parameters a, r, d=d+1.

Step 12.4: Print " " *d+ str(a[i])

Step 12.5: Call the function traversed recursively with parameters a, l, d=d+1.

Step 13: End the Program.

PROGRAM/SOURCE CODE:

```
def childNodes(i):
    return (2*i)+1, (2*i)+2  #(2*i)+1->LEFT NODE
                             #(2*i)+2->RIGHT NODE
def traversed(a, i=0, d = 0):
        if i >= len(a):
            return
        l, r = childNodes(i)
        traversed(a, r, d = d+1)
        print(" "*d + str(a[i]))
        traversed(a, l, d = d+1)
#instead of taking inputs directly give values a=[1,2,3]
a=[]
r=int(input("root"))
a.append(r)
i=int(input("enter number of child nodes")) #Enter values in ascending order
for c in range(i):
    c=int(input("child"))
    a.append(c)
a.sort()
print("\nTREE REPRESENTATION IN MIN HEAP\n")
traversed(a)
a.reverse()
print("\nTREE REPRESENTATION IN MAX HEAP\n")
traversed(a)
```

OUTPUT:

root5

enter number of child nodes6

child1

child3

child4

child6

child7

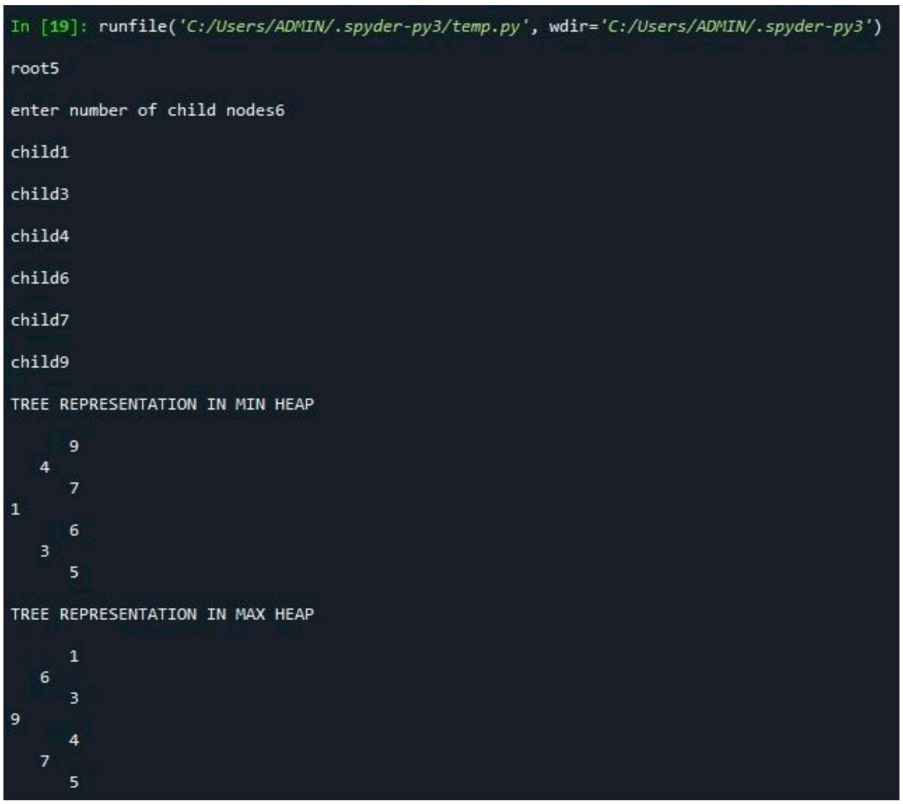
child9

TREE REPRESENTATION IN MIN HEAP

```
      9
   4
      7
1
      6
   3
      5
```

TREE REPRESENTATION IN MAX HEAP

```
      1
   6
      3
9
      4
   7
      5
```



RESULT:

Thus, the program to Implement Heaps using python code has been executed successfully.

GRAPH REPRESENTATION AND TRAVERSAL ALGORITHMS

EX NO 12(a) GRAPH REPRESENTATION

DATE :

AIM:

To Implement Graph Representation using python code.

ALGORITHM:

Step 1: Start of Program

Step 2: Size of the array list or Total Number of Vertices "V" is read

Step 3: Object 'graph' is created with "V" vertices

Step 4: Call "add edge" function with source and destination,

that adds edges to the undirected graph.

Step 4.1: An object "node" is created of class

AdjNode(destination)

Step 4.2: Adding the node to source node,

node = AdjNode(dest)

node.next=self.graph[src]

self.graph[src]=node

Step 4.3: Now, Add the source node to the Destination,

node = AdjNode(src)

node.next=self.graph[dest]

self.graph[dest[dest]=node

Step 5: Repeat Step 4, until desired (undirected edges)

Step 6: Call print_graph() function to Print the adjacency list of the vertex.

Step 6.1: print all the adjacency vertex by looping through it.

Step 7: End the Program.

PROGRAM/SOURCE CODE:

```
class AdjNode:
    def __init_(self, data):
        self.vertex = data
        self.next = None
# A class to represent a graph.
# A Graph is the list of the adjacency lists.
# Size of the array will be the no. of the vertices "V"
class Graph:
    def __init_(self, vertices):
        self.V = vertices
        self.graph = [None] * self.V
    # Function to add an edge in an undirected graph
    def add_edge(self, src, dest):
        # Adding the node to the source node
        node = AdjNode(dest)
        node.next = self.graph[src]
        self.graph[src] = node
    #Adding the source node to the destination as it is the undirected graph
        node = AdjNode(src)
        node.next = self.graph[dest]
        self.graph[dest] = node
    # Function to print the graph
    def print_graph(self):
        for i in range(self.V):
            print("Adjacency list of vertex {}\n head".format(i),
            end="") temp = self.graph[i]
            while temp:
                print(" -> {}".format(temp.vertex), end="")
                temp = temp.next
            print(" \n")
# Driver program to the above graph class
if__name__== "_main_":
    V = 5
    graph = Graph(V)
    graph.add_edge(0, 1)
    graph.add_edge(0, 4)
    graph.add_edge(1, 2)
    graph.add_edge(1, 3)
    graph.add_edge(1, 4)
    graph.add_edge(2, 3)
    graph.add_edge(3, 4)
    graph.print_graph()
```

OUTPUT:

Adjacency list of vertex 0
 head -> 4 -> 1

Adjacency list of vertex 1
 head -> 4 -> 3 -> 2 -> 0

Adjacency list of vertex 2
 head -> 3 -> 1

Adjacency list of vertex 3
 head -> 4 -> 2 -> 1

Adjacency list of vertex 4
 head -> 3 -> 1 -> 0

RESULT:

Thus, the program to Implement Graph Representation using python code has been executed successfully.

EX NO 12(b) GRAPH TRAVERSAL ALGORITHMS
DATE :

AIM:

1. To write a python program to Implement Breadth first Search **(BFS)**
2. To write a python program to Implement Depth First Search **(DFS)**

ALGORITHM:

1. Breadth first Search (BFS):

Step 1: Start of the program

Step 2: Read the root value and assign ch=0

Step 3: Select from operation "1. Insert 2.display 3.exit" using while loop

Step 4: if chosen option ch==1,

Step 4.1: insert "elements" into the insert(element) function

Step 5: if chosen option ch==2,

Step 5.1: Call Display() function to print the value

Step 6: if chosen option ch==3, EXIT the program

Step 7: Repeat Step 3

ALGORITHM OF SUB-FUNCTION insert():

Step 1: if (SELF.DATA < DATA):

If SELF.RIGHT == None, assign SELF.RIGHT = node(DATA)

Else Call RIGHT.INSERT() function

If SELF.LEFT == None, assign SELF.LEFT= node(DATA)

Else Call LEFT.INSERT() function

2. Depth first Search (DFS):

Step 1: Start of the program.

Step 2: Read the root value and assign ch=0, stack=[]

Step 3: Select from operation "1. Insert 2.display 3.exit" using while loop

Step 4: if chosen option ch==1,

Step 4.1: insert "elements" into the insert(element) function

Step 5: if chosen option ch==2,

Step 5.1: Call Display() function to print the value

Step 6: if chosen option ch==3, EXIT the program.

Step 7: Repeat Step 3.

ALGORITHM OF SUB-FUNCTION insert():

Step 1: if SELF.DATA < DATA

if SELF.RIGHT == None, assign SELF.RIGHT=node(DATA)

Else insert into SELF.RIGHT(DATA)

If SELF.LEFT == None, assign SELF.LEFT=node(DATA)

Else insert into SELF.LEFT(DATA)

PROGRAM/SOURCE CODE:

#Breadth First Search

```
class node:
    def__init_(self,data):
        self.data=data
        self.left=None
        self.right=None
    def insert(self,data):
        if(self.data<data):
            if(self.right==None):
                self.right==node(data)
            else:
                self.right.insert(data)
        elif(self.left==None):
            self.left=node(data)
        else:
            self.left.insert(data)
    def display(self):
        if(self.left is not None):
            self.left.display()
        print(self.data)
        if(self.right is not None):
            self.right.display()
print("Enter Root value:")
obj=node(int(input()))
ch=0
while(ch<4):
    print("1.insert 2.display 3.Exit")
    ch=int(input())
    if(ch==1):
```

```
            print("Enter the value to insert:")
            element=int(input())
            obj.insert(element)
        elif(ch==2):
            print("Depth First Traversal")
            obj.display()
        else:
            break
```

OUTPUT:

```
Enter Root value:
5
1.insert 2.display 3.exit
1
Enter the value to insert:
1
1.insert 2.display 3.exit
1
Enter the value to insert:
2
1.insert 2.display 3.exit
1
Enter the value to insert:
3
1.insert 2.display 3.exit
2
Depth First Traversal
1
5
1.insert 2.display 3.exit
3
```

```
In [22]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter Root value:

5
1.insert 2.display 3.Exit

1
Enter the value to insert:

1
1.insert 2.display 3.Exit

1
Enter the value to insert:

2
1.insert 2.display 3.Exit

1
Enter the value to insert:

3
1.insert 2.display 3.Exit

2
Depth First Traversal
1
5
1.insert 2.display 3.Exit

3
```

#Depth First Search

```
class node:
    def__init_(self,data):
        self.data=data
        self.left=None
        self.right=None
    def insert(self,data):
        if(self.data<data):
            if(self.right==None):
                self.right==node(data)
            else:
                self.right.insert(data)
        elif(self.left==None):
            self.left=node(data)
        else:
            self.left.insert(data)
    def display(self):
        global stack
        print(self.data)
        if(self.left is not
        None):
            stack.append(self.left)
        if(self.right is not None):
            stack.append(self.right)
        if(stack):
```

```
            temp=stack[0]
            del(stack[0])
            temp.display()
print("Enter Root value:")
obj=node(int(input()))
stack=[]
ch=0
while(ch<4):
    print("1.insert 2.display 3.exit")
    ch=int(input())
    if(ch==1):
        print("Enter the value to insert:")
        element=int(input())
        obj.insert(element)
    elif(ch==2):
        print("Breath First Traversal")
        obj.display()
    else:
        break
```

OUTPUT:

```
Enter Root value:
5
1.insert 2.display 3.exit
1
Enter the value to insert:
1
1.insert 2.display 3.exit
1
Enter the value to insert:
2
1.insert 2.display 3.exit
1
Enter the value to insert:
3
1.insert 2.display 3.exit
1
Enter the value to insert:
4
1.insert 2.display 3.exit
2
Breath First Traversal
5
```

1
1.insert 2.display 3.exit
3

```
In [24]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Enter Root value:

5
1.insert 2.display 3.exit

1
Enter the value to insert:

1
1.insert 2.display 3.exit

1
Enter the value to insert:

2
1.insert 2.display 3.exit

1
Enter the value to insert:

3
1.insert 2.display 3.exit

1
Enter the value to insert:

4
1.insert 2.display 3.exit

2
Breath First Traversal
5
1
1.insert 2.display 3.exit

3
```

RESULT:

Thus, the program to Implement Graph Traversal algorithms **(Depth First Search and Breath First Search)** using python code has been executed successfully.

EX NO 13 IMPLEMENTATION OF SINGLE SOURCE SHORTEST PATH ALGORITHM

DATE :

AIM:

To Implement of Single Source Shortest path algorithm using python code.

ALGORITHM:

Step 1: Start the program.

Step 2: Set the number of vertex(no0fvertex).

Step 3 Set the adjgraph.

Step 4: Create an object obj using the class graph with the parameters no_0f_vertex,adjgraph, So that the constructor function in the class graph will be called with respect to the given parameters.

Step 5: Call the member function dijistra which is defined inside the class graph.

Step 6: Assign distance=[999]*self.vertex.

Step 7: Assign distance[0]=0.

Step 8: Take a variable selsource=[False]*self.vertex.

Step 9: Take a for loop with i in range of self.vertex.

Step 9.1: Call the member function minDistance which is defined inside the class graph with the parameters “distance,selsource” and assign it to a variable min_index.

Step 9.2: Then Assign selsource[min_index]=True.

Step 9.3: Take an another for loop with j in range of self.vertex.

Step 9.3.1: if self.adjgraph[min_index][j] is greater than 0 and selsource is False and distance[j] greater than distance[min_index] +self.adjgraph[min_index][j],Then go to step 9.3.1.1.

Step 9.3.1.1: Add distance[min_index] and self.adjgraph[min_index][j] and assign it to distance[j].

Step 10: Call the member function display which is defined inside the class graph with the parameter “distance”.

Step 10.1: Print the sentence "0 is source".

Step 10.2: Take a for loop with i in range self.vertex.

Step 10.2.1: Print "node 0—{}---> {}".format(i.distance[i]).

Step 11 : End the program.

PROGRAM/SOURCE CODE:

```
class graph:
    def__init_(self,vertex,adjgraph):
        self.vertex=vertex
        self.adjgraph=adjgraph
        graph=[[0 for col in range(self.vertex)]for row in range(self.vertex)]
    def minDistance(self,distance,selSource):
        min=9999
        min_index=0
        for i in range(self.vertex):
            if(min>distance[i] and selSource[i]==False):
                min=distance[i]
                min_index=min
        return min_index
    def display (self,distance):
        print("0 is the source")
        for i in
        range(self.vertex):
            print("node 0-{} ---> {}".format(i,distance[i]))
    def dijistra(self):
        distance=[999]*self.vertex
        distance[0]=0
        selSource=[False]*self.vertex
        for i in range(self.vertex):
            min_index=self.minDistance(distance,selSource)
            selSource[min_index]=True
            for j in range(self.vertex):
                if(self.adjgraph[min_index][j]>0 and selSource[j]==False
and distance[j]>distance[min_index]+self.adjgraph[min_index][j]):
                    distance[j]=distance[min_index]+self.adjgraph[min_index][j]
        self.display(distance)
no_of_vertex=6 adjgraph=[[0,1,0,0,3,0],[1,0,2,0,1,0],[0,2,0,3,0,2],
[0,0,3,0,0,1],[3,1,0,0,0,2],[0, 0,2,1,2,0]]
obj=graph(no_of_vertex,adjgraph)
obj.dijistra()
```

OUTPUT:

0 is the source
node 0-0 ---> 0
node 0-1 ---> 1
node 0-2 ---> 3
node 0-3 ---> 6
node 0-4 ---> 2
node 0-5 ---> 5

```
In [26]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
0 is the source
node 0-0 ---> 0
node 0-1 ---> 1
node 0-2 ---> 3
node 0-3 ---> 6
node 0-4 ---> 2
node 0-5 ---> 5
```

RESULT:

Thus, the program to Implement of Single Source Shortest path algorithm using python code has been executed successfully.

EX NO 14 IMPLEMENTATION OF MINIMUM SPANNING TREE ALGORITHMS

DATE :

AIM:

To Implement Minimum Spanning Tree algorithm using python code.

ALGORITHM:

Step 1: Start the program.

Step 2: Create an object g using the class Graph with giving the parameter vertex, So that the constructor function in the class Graph will be called with respect to the given parameter.

Step 3: Add the values using the member function add_edge which is defined inside the class Graph.

Step 4: Call the member function Kruskal which is defined inside the class Graph.

Step 5: Assign result=[]

Step 6: Assign i=0 and e=0.

Step 7: Assign sorted(self.graph,key=lambda item:item[2]) to self.graph.

Step 8: Assign parent=[].

Step 9: Assign rank=[].

Step 10: Take a for loop node in range self.v.

Step 10.1: Append node to parent.

Step 10.2: Append 0 to rank.

Step 11: Take a while loop for when e less than self.v – 1.

Step 11.1: Assign u,v,w=self.graph[i].

Step 11.2: Assign i=i+1.

Step 11.3: Call the member Function search which is defined inside the class Graph with parameters parent and u and assign it to the variable x.

Step 11.4: Call the member function search which is defined inside the class Graph with parameters parent,v and assign it to the variable y.

Step 11.5: if x not equal to y, Then go to step 11.5.1.

Step 11.5.1: Assign e=e+1.

Step 11.5.2: Append [u,v,w] to result.

Step 11.5.3: Call the member function apply Union which is defined inside the class Graph with parameters parent,rank,x,y.

Step 12: Take a for loop with u,v,weight in result.

Step 12.1: Print "Edge:",u,v,end=""

Step 12.2: print "-",weight.

Step 13: End the program.

PROGRAM/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 = []
        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.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,end =" ")
            print("-",weight)
g = Graph(5)
g.add_edge(0, 1, 8)
g.add_edge(0, 2, 5)
g.add_edge(1, 2, 9)
g.add_edge(1, 3, 11)
g.add_edge(2, 3, 15)
g.add_edge(2, 4, 10)
g.add_edge(3, 4, 7)
g.kruskal()
```

OUTPUT:

Edge: 0 2 - 5
Edge: 3 4 - 7
Edge: 0 1 - 8
Edge: 2 4 - 10

```
In [27]: runfile('C:/Users/ADMIN/.spyder-py3/temp.py', wdir='C:/Users/ADMIN/.spyder-py3')
Edge: 0 2 - 5
Edge: 3 4 - 7
Edge: 0 1 - 8
Edge: 2 4 - 10
```

RESULT:

Thus, the program to Implement Minimum Spanning Tree algorithm using python code has been executed successfully.