Lab Code:18ECL31

Data Structures using Python Lab Manual

Department of Electronics & Communication Engineering

Bapatla Engineering College :: Bapatla

(Autonomous)

G.B.C. Road, Mahatmajipuram, Bapatla-522102, Guntur (Dist.)
Andhra Pradesh, India.
E-Mail:bec.principal@becbapatla.ac.in
Web:www.becbapatla.ac.in

Contents

S.No.	Title of the Experiment
1.	Write a Python program to implement bubble sort, selection sort and insertion sort.
2.	Write a Python program to implement merge sort, quick sort
3.	Write a Python program on linear search and binary search.
4.	Write a Python program to implement Singly Linked List
5.	Write a Python program to implement Doubly Linked List
6.	Write a Python program to implement Circular Linked List
7.	Write a Python programs to implement stacks using arrays and linked lists.
8.	Write a Python programs to implement queues using arrays and linked lists.
9.	Write a Python program to perform Binary Tree traversal operations.
10.	Write a Python programs to perform Binary search tree operations.
11.	Write a Python program to Travers in a graph using Depth first search.
12.	Write a Python program to Travers in a graph using breadth first search.

Bapatla Engineering College :: Bapatla
(Autonomous)

Vision

- To build centers of excellence, impart high quality education and instill high standards of ethics and professionalism through strategic efforts of our dedicated staff, which allows the college to effectively adapt to the ever changing aspects of education.

- To empower the faculty and students with the knowledge, skills and innovative thinking to facilitate discovery in numerous existing and yet to be discovered fields of engineering, technology and interdisciplinary endeavors.

Mission

- Our Mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas.

- We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Vision

To produce globally competitive and socially responsible Electronics and Communication Engineering graduates to cater the ever changing needs of the society.

Mission

- To provide quality education in the domain of Electronics and Communication Engineering with advanced pedagogical methods.

- To provide self learning capabilities to enhance employability and entrepreneurial skills and to inculcate human values and ethics to make learners sensitive towards societal issues.

- To excel in the research and development activities related to Electronics and Communication Engineering.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Educational Objectives (PEO's)

PEO-I: Equip Graduates with a robust foundation in mathematics, science and Engineering Principles, enabling them to excel in research and higher education in Electronics and Communication Engineering and related fields.

PEO-II: Impart analytic and thinking skills in students to develop initiatives and innovative ideas for Start-ups, Industry and societal requirements.

PEO-III: Instill interpersonal skills, teamwork ability, communication skills, leadership, and a sense of social, ethical, and legal duties in order to promote lifelong learning and Professional growth of the students.

Program Outcomes (PO's)

Engineering Graduates will be able to:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and Teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Specific Outcomes (PSO's)

PSO1: Develop and implement modern Electronic Technologies using analytical methods to meet current as well as future industrial and societal needs.

PSO2: Analyze and develop VLSI, IoT and Embedded Systems for desired specifications to solve real world complex problems.

PSO3: Apply machine learning and deep learning techniques in communication and signal processing.

Data Structures Using Python Lab
II B.Tech – III Semester (Code: 18ECL31)

Lectures	0	Tutorial	0	Practical	3	Credits	1
Continuous Internal Assessment			50	Semester End Examination (3 Hours)			50

Prerequisites: Data Structures

Course Objectives: Students will

- Implement various Searching and Sorting Techniques.
- Create different linear data structures like linked lists, stacks, and queues.
- Create non-linear data structures like trees and graphs.
- Understand the searching mechanism like depth first search and breadth first search.

Course Outcomes: After studying this course, the students will be able to

CO1	Compose different sorting and searching algorithms
CO2	Implement linear data structures like linked list, stacks, and queues.
CO3	Develop non-linear data structures like trees and graphs.
CO4	Demonstrate traversal techniques on non-linear data structures.

Mapping of Course Outcomes with Program Outcomes & Program Specific Outcomes

	PO's												**PSO's**		
CO	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
CO1	2	3			2				3					2	2
CO2	2	3			2				3					2	2
CO3	2	3			2				3					2	2
CO4	2	3			2				3					2	2
AVG	2	3			2				3					2	2

LIST OF LAB PROGRAMS

1. Write a Python program to implement bubble sort, selection sort and insertion sort.
2. Write a Python program to implement merge sort, quick sort
3. Write a Python program on linear search and binary search.
4. Write a Python program to implement Singly Linked List
5. Write a Python program to implement Doubly Linked List
6. Write a Python program to implement Circular Linked List
7. Write a Python programs to implement stacks using arrays and linked lists.
8. Write a Python programs to implement queues using arrays and linked lists.
9. Write a Python program to perform Binary Tree traversal operations.
10. Write a Python programs to perform Binary search tree operations.
11. Write a Python program to Travers in a graph using Depth first search.
12. Write a Python program to Travers in a graph using breadth first search.

NOTE: A minimum of 10 (Ten) experiments have to be Performed and recorded by thecandidate to attain eligibility for Semester End Examination.

1. SORTING TECHNIQUES

Aim:

To Write a Python program to implement bubble sort, selection sort and insertion sort.

Software required: IDLE (python 3.11)

Source Code: Bubble sort

```
print("enter list elements")
arr=list(input())
print("the length of list is:",len(arr))
print("the list elements before
sorting",arr) i=0
while(i<len(arr)-
   1): j=0
   while(j<len(arr)-
   1):
      if arr[j]>arr[j+1]:
         arr[j],arr[j+1]=arr[j+1],arr[j]
      j=j+
   1 i=i+1
print("the list elements after sorting are",arr)
```

Output:

```
enter list
elements
8465923
the length of list is: 7
the list elements before sorting ['8', '4', '6', '5', '9', '2', '3']
the list elements after sorting are ['2', '3', '4', '5', '6', '8', '9']
```

Source Code: Selection sort

```
print("enter the list
elements")
arr=list(input())
print(("the lenght of list is:"),len(arr))
print(("the list elements before
sorting"),arr) i=0
```

```
while(i<len(arr)-
    1): j=i+1
    while(j<len(arr
    )):
        if(arr[i]>arr[j]):
          arr[i],arr[j]=arr[j],arr[i]
        j=j+1
    i=i+1
print(("the list elements after sorting"),arr)
```

Output:

```
enter the list
elements54321
the lenght of list is: 5
the list elements before sorting ['5', '4', '3', '2', '1']
the list elements after sorting ['1', '2', '3', '4', '5']
```

Source Code: Insertion sort

```
print("enter the list
elements")a=list(input())
print("array before
sorting",a)i=1
while(i<len(a)
   ):j=0
   while(j<i):
      if(a[j]>a[i]):
         temp=a[j]
         a[j]=a[i]
         k=i
         while(k>j
         ):
            a[k]=a[k-
            1]k=k-1
         a[k+1]=te
      mpj+=1
   i+=1
print("array after sorting:",a)
```

Output:

```
enter the list elements87602
array before sorting ['8', '7', '6', '0', '2']
array after sorting: ['0', '2', '6', '7', '8']
```

2. SORTING TECHNIQUES

Aim:

To Write a Python program to implement Merge sort and Quick sort.

Software required: IDLE (python 3.11)

Source Code: Merge sort

```
L1=[5,6,3,7]
L2=[8,1,2,4]
M=[0,0,0,0,0,0,
0,0]
i=0
while(i<4
):
  j=i+1
  while(j<4
  ):
    if(L1[i]>L1[j]):
      L1[i],L1[j]=L1[j],L1[
      i]
    if(L2[i]>L2[j]):
      L2[i],L2[j]=L2[j],L2[
      i]
    j+=
  1i+=1
i=0
j=0
k=
0
while(k<8):
  if(L1[i]<=L2[j])
  :
    M[k]=L1[i
    ]i+=1
    k+=1
  else:
    M[k]=L2[j
    ]j+=1
    k+=1
  if(i==4 or j==4):
    break
while(j<4):
  M[k]=L2[j
  ]j+=1
  k+=1
while(i<4):
  M[k]=L1[i
  ]i+=1
  k+=1
print("array after sorting:",M)
```

Output:

```
array after sorting: [1, 2, 3, 4, 5, 6, 7, 8]
```

Source Code: Quick sort

```
 def
   split(arr,lower,upper):
   l=lower+1
   u=upper p=arr[lower]
   while(u>=l):
      while(arr[l]<p):
         l=l+1
         if(l==len(arr)):
            break
      while(arr[u]>p):
         u=u-1 if(u>l):
         arr[l],arr[u]=arr[u
      ],arr[l]
      arr[lower],arr[u]=ar
      r[u],arr[lower]
   return u
def
   quicksort(arr,lower,uppe
   r): if(upper>lower):
      pivote=split(arr,lower,uppe
      r)
      quicksort(arr,lower,pivote-
      1)
      quicksort(arr,pivote+1,upp
      er)
print("enter list
elements:")
arr=list(input())
```

```
print("before quicksort
the array is:",arr)
quicksort(arr,0,len(arr)-
1)
print("after quicksort
the array is:",arr)
```

Output:

```
enter list elements:
654321
before quicksort the array is: ['6', '5', '4', '3', '2', '1']
after quicksort the array is: ['1', '2', '3', '4', '5', '6']
```

3. SEARCHING TECHNIQUES

Aim:

To Write a Python program to implement Linear and Binary Search.

Software required: IDLE (python 3.11)

Source Code: Linear Search

```
print("enter the list values")
a=input()
print("enter a value to
find") n=input()
for i in
   range(len(a)):
   if(n==a[i]):
      print("the given value",n,"is found in",i,"th
      location")
      break
if(i==len(a)):
   print("the given element is not found in the list")
```

Output:

```
enter the list
values 87654
enter a value to
find 5
the given value 5 is found in 3 th location
```

Source Code: Binary Search

```
print("enter list elements")
arr=list(input())
print("the length of list is:",len(arr))
print("the list elements before sorting",arr)
i=0
while(i<len(arr)-1):
      j=0
      while(j<len(arr)-1):
      if arr[j]>arr[j+1]:
         arr[j],arr[j+1]=arr[j+1],arr[j]
      j=j+1
```

```
  i=i+1
print("the list elements after sorting are",arr)
print("enter a value to search")
n=input()
flag=0
l=0
u=(len(arr)-1)
mid=int((l+u)
/2)
while(l<=u):
   if(n==arr[mid]):
     flag=1
     print("the given number is found in the location",mid)
     if(n<arr[mid]):
     u=mid
   -1 else:
     l=mid+1
   mid=int((l+u)
   /2)
if(flag==0):
  print("the element is not found in the list")
```

Output:

```
enter list
elements
321654
the length of list is: 6
the list elements before sorting ['3', '2', '1', '6', '5', '4']
the list elements after sorting are ['1', '2', '3', '4', '5',
'6']enter a value to search
3
the given number is found in the location 2
```

4. SINGLY LINKED LIST

Aim:

To Write a Python program to implement Singly linked list.

Software required: IDLE (python 3.11)

Source Code:

```
class node:
    data=None
    link=None
    def append(self,num):
        if(self.data==None):
            self.data=num
            self.link=None
        else:
            tr=node()
            temp=node()
            tr=self
            while(tr.link!=None):
                tr=tr.link
            temp.data=num
            temp.link=None
            tr.link=temp
    def display(self):
        count=0
        while(self!=None):
            print(self.data)
            count=count+1
            self=self.link
        print("the number of nodes in the linked list is",count)
    def addatbeg(self,obj,num):
        temp=node()
        temp.data=num
        temp.link=obj
        obj=temp
        return obj
    def addafter(self,loc,num):
        temp=node()
        tr=node()
        tr=self
        for i in range(loc):
            tr=tr.link
        temp.data=num
        temp.link=tr.link
        tr.link=temp
    def delete(self,obj,num):
        old=node()
        temp=node()
        temp=self
        while(temp!=None):
            while(temp.data==num):
              if(temp.data==self.data):
              obj=self.link
              else: old.link=temp.link
               temp.data=None
               temp.link=None

            old=temp
            temp=temp.link
        return obj

obj=node()
obj.append(5)
obj.append(6)
obj.append(7)
obj.append(8)
print("the linkedlist after append")
obj.display()
obj=obj.addatbeg(obj,4)
obj=obj.addatbeg(obj,3)
print("after add at beg")
obj=obj.delete(obj,7)
obj.display()
obj.addafter(2,777)
obj.addafter(5,888)
print("after add after")
obj.display()
obj=obj.delete(obj,5)
print("the linked list after deletion")
obj.display()
```

Output:

```
the linked list after
append5 6 7  8
the number of nodes in the linked
list is 4after add at beg
3   4   5   6   8
the number of nodes in the linked
list is 5after add after
3  4  5   777  6  8  888
the number of nodes in the linked
list is 7the linked list after
deletion
3 4   777  6  8  888
the number of nodes in the linked list is 6
```

5. DOUBLY LINKED LIST

Aim:

To Write a Python program to implement Doubly linked list.

Software required: IDLE (python 3.11)

Source Code:

```
class node:
    data=None
    prev=None
    next=None
    def append(self,n):
        if(self.data==None):
            self.data=n
            self.prev=None
            self.next=None
        else:
            temp=node()
            temp.data=n
            temp.next=None
            tr=self
            while(tr.next!=None):
                tr=tr.next
            tr.next=temp
            temp.prev=tr
    def addatbeg(self,n):
        temp=node()
        temp.data=n
        temp.prev=None
        temp.next=self
        self.prev=temp
        self=temp
        return  self
    def display(self):
        count=0
        while(self!=None):
            count=count+1
            print(self.data)
            self=self.next
        print("the no of elements in linked list:",count)
```

```
      def addafter(self,loc,n):
            temp=node()
            temp.data=n
            tr=self
            for    i    in    range(loc):
              tr=tr.next
            temp.next=tr.nex
            ttemp.prev=tr
            tr.next.prev=tem
            ptr.next=temp
         def delete(self,n):
            temp=node()
            old=node()
            temp=self
            while(temp!=None):
              if(temp.data==n):
                if(temp.data==self.data):
                  self=self.next
                  temp.next=None
                  self.prev=None
                  temp.data=None
                else:
                  old.next=temp.next
                  temp.next.prev=old
                  temp.prev=None
                  temp.next=None
                  temp.data=None
              else:
                 old=temp
                 temp=temp.next
            return self
       obj=node()
       obj.append(5)
       obj.append(6)
       obj.append(7)
       obj.append(8)
       print("the linkedlist
       after append")
       obj.display()
       obj=obj.addatbeg(3)
       obj=obj.addatbeg(4)
       print("after addatbeg")
       obj.display()
       obj.addafter(2,777)
```

```
obj.addafter(5,888)
print("after add after")
```

```
obj.display()
obj=obj.delete(5)
print("after deletion:")
obj.display()
```

Output:

```
the linkedlist after
append5 6 7  8
the number of nodes in the
linkedlist is 4after add at beg
3   4   5   6   8
the number of nodes in the
linkedlist is 5after add after
3  4  5   777  6  8  888
the number of nodes in the
linkedlist is 7the lnkedlist after
deletion
3 4   777  6  8  888
the number of nodes in the linkedlist is 6
```

6. CIRCULAR LINKED LIST

Aim:

To Write a Python program to implement Circular linked list.

Software required: IDLE (python 3.11)

Source Code:

```
class node:
    data=None
    link=None
    def append(self,n):
        if(self.data==None):
            self.data=n
            self.link=self
        else:
            temp=node()
            tr=node()
            tr=self
            while(tr.link!=self):
                tr=tr.link
            temp.data=n
            temp.link=self
            tr.link=temp
    def display(self):
        tr=self
        if(self!=None):
            print(self.data)
            self=self.link
            count=1
        while(self!=tr):
            count=count+1
            print(self.data)
            self=self.link
        print("the no of nodes in linkedlist:",count)
    def addatbeg(self,obj,n):
        temp=node()
        temp.data=n
        temp.link=obj
        tr=self
```

```
while(tr.link!=
self):
   tr=tr.link
```

```
        tr.link=temp
        obj=temp
        return obj
    def addafter(self,loc,num):
        temp=node()
        tr=node()
        tr=self
        for i in range(loc):
            tr=tr.link
        temp.data=num
        temp.link=tr.link
        tr.link=temp
    def delete(self,obj,n):
        temp=node()
        old=node()
        tr=node()
        tr=self
        temp=self
        while(temp!=None):
            if(temp.data==n):
                if(temp.data==self.data):
                    while(tr.link!=self):
                        tr=tr.link
                    obj=self.link
                    tr.link=obj
                    temp.data=None
                    temp.link=None
                else:
                    old.link=temp.link
                    temp.data=None
                    temp.link=None
            else:
                old=temp
                temp=temp.link
        return obj
obj=node()
obj.append(5)
obj.append(6)
obj.append(7)
obj.append(8)
print("the linkedlist after
append")
obj.display()
obj=obj.addatbeg(obj,4)
```

```
obj=obj.addatbeg(obj,3)
print("after add at beg")
obj=obj.delete(obj,7)
obj.display()
obj.addafter(2,777)
obj.addafter(5,888)
print("after add after")
obj.display()
obj=obj.delete(obj,5)
print("the
linkedlist after
deletion")
obj.display()
```

Output:

```
the linkedlist after
append5 6 7  8
the number of nodes in the
linkedlist is 4after add at beg
3   4   5   6   8
the number of nodes in the
linkedlist is 5after add after
3  4  5   777  6  8  888
the number of nodes in the
linkedlist is 7the linked list after
deletion
3 4   777  6  8  888
the number of nodes in the linked list is 6
```

7. STACKS

Aim:

To Write a Python program to implement Stacks using Arrays and Linked list.

Software required: IDLE (python 3.11)

Source Code: Stack using Arrays

```
 class stack:
    arr=[ ]
    top=-1
def push(self,n):
    if(self.top==0):
        self.top=self.top+1
        self.arr.append(n)
    if(self.top==len(self.arr)-1):
        self.arr.append(n)
        self.top=self.top+1
        return
    self.top=self.top+1
    self.arr[self.top]=n
def pop(self):
     if(self.top==-1):
        print("stack is empty,we cant remove")
        return
      del_data=self.arr[self.top]
      self.arr[self.top]=None
      self.top=self.top-1
      print("removed data is:",del_data)
 print("stack elements are:",self.arr)
 obj=stack()
 obj.push(5)
 obj.push(6)
 obj.push(7)
 obj.push(8)
 print("the stack elements
 are:",obj.arr)
 obj.pop()
 obj.pop()
 obj.push(10)
obj.push(11)
print("the stack elements after
repushing are:",obj.arr)
```

Output:

```
the stack elements are: [5, 6,
6, 7, 8]
removed data is: 8
stack elements are: [5, 6, 6,
7, None]removed data is: 7
stack elements are: [5, 6, 6, None, None]
the stack elements after repushing are: [5, 6, 6, 10, 11]
```

SOURCE CODE: Stack using Linked List

```
class node:
   data=None
   link=None
def push(self,n):
   temp=node()
   temp.data=n
   temp.link=self
   self=temp
   return temp
def pop(self):
   temp=self
   self=self.link
   temp.data=None
   temp.link=None
   return self
def display(self):
   count=0
      while(self.link!=None):
         count=count+1
         print(self.data)
         self=self.link
      print("the number of elements in
stack",count) obj=node()
obj=obj.push(4)
obj=obj.push(5)
obj=obj.push(6) print("the
actual stack is")
obj.display()
obj=obj.pop()
obj=obj.pop()
```

```
print("the stack after deletion")
obj.display()
```

Output:

```
the actual stack is6 5 4
the number of elements in
stack 3the stack after
deletion
4
the number of elements in stack 1
```

8. QUEUES

Aim:

To Write a Python program to implement Queues using Arrays and Linked list.

Software required: IDLE (python 3.11)

Source Code: Queue using Arrays

```
 class queue:
   arr=[]
   front=-1
   rear=-1
 def enqueue(self,n):
   if(self.rear==len(self.arr)-1):
        self.front=0
        self.rear=self.rear+1
        self.arr.append(n)
        return
     self.rear=self.rear+1
     self.front=0
     self.arr[self.rear]=n
def dequeue(self):
     if(self.arr==self.front==-1):
        print("queue is empty")
        return
     else:
        if(self.front==self.rear):
           self.arr[self.front]=self.arr
           [self.rear]=None self.front=-1
           self.rear=-1
       else:
           self.arr[self.front]=None
           self.front=self.front+1
 obj=queue()
 obj.enqueue(5)
 obj.enqueue(6)
 obj.enqueue(7)
 obj.enqueue(8)
```

```
print(obj.arr)
obj.dequeue()
obj.dequeue()
obj.dequeue()
```

```
print(obj.arr)
obj.enqueue(9)
obj.enqueue(10)
obj.enqueue(11)
print(obj.arr)
```

Output

```
[5, 6, 7, 8]
[None, None, None, 8]
[None, None, None, 8, 9, 10, 11]
```

Source Code: Queue using Linked List

```
class node:
  data=None
  link=None
def enqueue(self,n):
  if(self.data==None):
      self.data=n
      self.link=None
  else:
      temp=node()
      tr=self
      while(tr.link!=None):
        tr=tr.link
      temp.data=n
      temp.link=None
      tr.link=temp
def dequeue(self):
  temp=self
  self=self.link
  temp.data=None
  temp.link=None
  return self
def display(self):
  count=0
  while(self!=None):
      count=count+1
      print(self.data)
      self=self.link
   print("the total number of elements in queue is:",count)
obj=node()
obj.enqueue(5)
obj.enqueue(6)
obj.enqueue(7)
print("the elements in queue are:")
obj.display()
obj=obj.dequeue()
obj=obj.dequeue()
print("the elements of queue after deleting:")
obj.display()
```

Output:

```
the elements in queue are:
5 6   7
the total number of elements in
queue is: 3the elements of queue
after deleting:
7
the total number of elements in queue is: 1
```

9. BINARY TREE TRAVERSAL

Aim:

To Write a Python program to implement Binary tree traversal operations.

Software required: IDLE (python 3.11)

Source Code:

```
class node:
        left=None
        data=None
        right=None
def buildtree(arr,n):
        temp=node()
        if(arr[n]!='\0'):
           temp.left=buildtree(arr,2*n+1)
           temp.data=arr[n]
           temp.right=buildtree(arr,2*n+2)
        return temp
def inorder(root):
        if(root.data!=None):
           inorder(root.left)
           print(root.data)
           inorder(root.right)
def preorder(root):
        if(root.data!=None):
           print(root.data)
           preorder(root.left)
           preorder(root.right)
def postorder(root):
        if(root.data!=None):
           postorder(root.left)
           postorder(root.right)
           print(root.data);


    arr=['a','b','c','d','e','f','g','\0','\0','h','\0','\0','\0','\0','\0','\0','\0','\0','\
    0','\0','\0']
    root=node();
```

```
root=buildtree(ar
r,0);
print("\n inorder
traversel:\n");
inorder(root);
print("\n pre order
trvarsel:\n");
preorder(root);
print("\n post order
trvaresel:\n");
postorder(root);
```

Output:

```
inorder tvarulsel:
d  b  h  e  a  fc g
 pre order trvarsel:
a  b  d  e  h  c  f  g
post order
trvaresel: d h e b
f  g  c a
```

10. BINARY SEARCH TREE

Aim:

To Write a Python program to implement Binary search tree operations.

Software required: IDLE (python 3.11)

Source Code:

```
class node:
        leftchild=None
        data=None
        rightchild=None
        flag="Not found"
def insert(sr,num):
        if(sr.data==None):
           sr.leftchild=None
           sr.data=num
           sr.rightchild=None
        else:
           if(num<sr.data):
              if(sr.leftchild==None):
                 sr.leftchild=node()
                 insert(sr.leftchild,num)
              else:
                 insert(sr.leftchild,num)
           else:
              if(sr.rightchild==None):
                 sr.rightchild=node()
                 insert(sr.rightchild,num)
              else:
                 insert(sr.rightchild,num)
def delete(root,num):
        x=node()
        xsucc=node()
        par=node()
        if(root==None):
           print("tree is
           empty.\n") return
        par,x=search(root,num,par,x)
```

```
    if(x.flag=="Not Found"):
            print("data to be deleted , not found")
            return
    if(x.leftchild!=None and x.rightchild!=None):
            par=x
            xsucc=x.rightchild
            while(xsucc.leftchild!=None):
               par=xsucc
               xsucc=xsucc.leftchild
            x.data=xsucc.
            datax=xsucc
    if(x.leftchild==None and
            x.rightchild==None):
            if(par.rightchild==x):
               par.rightchild=None
            else:
               par.leftchild=None
            return
      if(x.leftchild==None and
            x.rightchild!=None):
            if(par.leftchild==x):
               par.leftchild=x.rightchild
            else:
               par.rightchild=x.rightchild
            return
        if(x.leftchild!=None and x.rightchild==None):
            if(par.leftchild==x):
               par.leftchild=x.leftchild
            else:
               par.rightchild=x.leftchild
            return

def search(root,num,x,par):
    q=root
         while(q!=None):
            if(q.data==num):
               x.flag="Found"
               x=q
               print("\n the element :",x.data,"is
               found")
               return par,x
           par=q
            if(q.data>num):
               q=q.leftchild
```

```
          else:
            q=q.rightchild
   def inorder(sr):
       if(sr!=None):
          inorder(sr.leftchild)
          print(sr.data)
          inorder(sr.rightchild)

    obj=node()
    a=[11,9,13,8,10,12,14,15,
    7]
    for i in
       range(9):
       insert(obj,a[
       i])

    print("binary tree before the
    deletion:\n")inorder(obj)
    delete(obj,10)
    print("\nbinary tree after the
    deletion:\n")inorder(obj)
    delete(obj,14)
    print("\nbinary tree after the
    deletion:\n")inorder(obj)

    delete(obj,8)
    print("\nbinary tree after the
    deletion:\n")inorder(obj)

    delete(obj,11)
    print("\nbinary tree after the
    deletion:\n")inorder(obj)
```

Output:

```
binary tree before the deletion:
7  8   9 10  11  12  13   14  15
('\n the element :', 10, 'is
found')binary tree after
the deletion:
7 8  9  11   12  13  14  15
('\n the element :', 14, 'is
found')binary tree after
the deletion:
```

```
7   8   9   11   12   13    15
('\n the element :', 8, 'is
found')binary tree after
the deletion: 7  9 11  12
13  15
('\n the element :', 11, 'is
found')binary tree after
the deletion:
7   9  12   13   15
```

11. GRAPH USING DEPTH FIRST SERCH

Aim:

To Write a Python program to Travers in a graph using Depth first search.

Software required: IDLE (python 3.11)

Source Code:

```
class Graph:
  def__init_(self):
     # dictionary containing keys that map to the corresponding
     vertex object
     self.vertices = {}

  def add_vertex(self, key):
     """Add a vertex with the given key to the
     graph."""
     vertex = Vertex(key)
     self.vertices[key] = vertex

  def get_vertex(self, key):
     """Return vertex object with the corresponding
     key."""
     return self.vertices[key]

  def__contains_(self, key):
     return key in self.vertices

  def add_edge(self, src_key, dest_key, weight=1):
     """Add edge from src_key to dest_key with given weight."""
     self.vertices[src_key].add_neighbour(self.vertices[dest_key],
     weight)

  def does_edge_exist(self, src_key, dest_key):
     """Return True if there is an edge from src_key to
     dest_key.""" return
     self.vertices[src_key].does_it_point_to(self.vertices[dest_k
     ey])
```

```
    def___iter_(self):
        return iter(self.vertices.values())


class Vertex:
    def___init_(self,
        key): self.key =
        key self.points_to
        = {}
    def get_key(self):
        """Return key corresponding to this vertex
        object."""return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this
        vertex."""return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to
        dest."""return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to
        dest."""return dest in self.points_to


class Stack:
    def___init_(self):
        self.items = []

    def is_empty(self):
        return self.items
        == []

    def push(self, data):
        self.items.append(data)

    def pop(self):
        return self.items.pop()
```

```
def display_dfs(v):
    visited = set()
    s = Stack()
    s.push(vertex)
    while not s.is_empty():
        current = s.pop()
        if current in
            visited:
            continue
        print(current.get_key(), end=' ')
        visited.add(current)
        for dest in current.get_neighbours():
            if dest not in visited:
                s.push(dest)


g = Graph()
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest>')
print('dfs <vertex key>')
print('display')
print('quit')

while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation ==
        'add':
        suboperation =
        do[1]
        if suboperation ==
            'vertex':key =
            int(do[2])
            if key not in g:
                g.add_vertex(k
                ey)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src =
            int(do[2])
            dest =
            int(do[3]) if
            src not in g:
               print('Vertex {} does not
            exist.'.format(src)) elif dest not in g:
               print('Vertex {} does not
            exist.'.format(dest)) else:
               if not g.does_edge_exist(src,
                  dest): g.add_edge(src, dest)
               else:
                  print('Edge already exists.')

      elif operation ==
         'dfs': key =
         int(do[1])
         print('Depth-first Traversal: ',
         end='') vertex =
         g.get_vertex(key)
         display_dfs(vertex)
         print()

      elif operation == 'display':
         print('Vertices: ', end='')
         for v in g:
            print(v.get_key(), end=' ')
         print()

         print('Edges: ')
         for v in g:
            for dest in
               v.get_neighbours(): w =
               v.get_weight(dest)
               print('(src={}, dest={}, weight={}) '.format(v.get_key(),
                                               dest.get_key(), w))
         print()

      elif operation ==
         'quit': break
```

Output:

```
What would you like to do? add vertex 1
What would you like to do? add vertex 2
What would you like to do? add vertex 3
What would you like to do? add vertex 4
What would you like to do? add vertex 5
What would you like to do? add vertex 6
What would you like to do? add vertex 7
What would you like to do? add edge 1 2
What would you like to do? add edge 2 3
What would you like to do? add edge 3 4
What would you like to do? add edge 1 5
What would you like to do? add edge 1 6
What would you like to do? add edge 5 6
What would you like to do? add edge 3 7
What would you like to do? dfs 1
Depth-first Traversal: 1 5 6 2 3 7 4
What would you like to do? Quit
```

12. GRAPH USING BREADTH FIRST SERCH

Aim:

To Write a Python program to Travers in a graph using breadth first search.

Software required: IDLE (python 3.11)

Source Code:

```
class Graph:
  def__init_(self):
     # dictionary containing keys that map to the corresponding vertex
     object
     self.vertices = {}

  def add_vertex(self, key):
     """Add a vertex with the given key to the
     graph."""
     vertex = Vertex(key)
     self.vertices[key] = vertex

  def get_vertex(self, key):
     """Return vertex object with the corresponding key."""
     return self.vertices[key]

  def__contains_(self, key):
     return key in self.vertices

  def add_edge(self, src_key, dest_key, weight=1):
     """Add edge from src_key to dest_key with given weight."""
     self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)

  def does_edge_exist(self, src_key, dest_key):
     """Return True if there is an edge from src_key to
     dest_key."""
```

```
        return
        self.vertices[src_key].does_it_point_to(self.vertices[dest_key])

    def__iter_(self):
        return iter(self.vertices.values())


class Vertex:
    def__init_(self, key):
        self.key = key
        self.points_to = {}
    def get_key(self):
        """Return key corresponding to this vertex
        object."""return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this vertex."""
        return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to
        dest."""return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to
        dest."""return dest in self.points_to


class Queue:
    def__init_(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def enqueue(self, data):
        self.items.append(data)

    def dequeue(self):
        return self.items.pop(0)


def display_bfs(vertex):
    """Display BFS Traversal starting at vertex."""
    visited = set()
    q = Queue()
    q.enqueue(vertex)
    visited.add(vertex)
    while not q.is_empty():
        current = q.dequeue()
        print(current.get_key(), end='
        ')
        for dest in
            current.get_neighbours():if dest
            not in visited:
                visited.add(dest)
                q.enqueue(dest)


g = Graph()
print('Menu')
print('add vertex <key>')
print('add edge <src>
<dest>')print('bfs <vertex
key>') print('display')
print('quit')

while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation == 'add':
        suboperation = do[1]
        if suboperation ==
            'vertex':key = int(do[2])
            if key not in g:
                g.add_vertex(key)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src = int(do[2])
            dest = int(do[3]) if
            src not in g:
                print('Vertex {} does not
            exist.'.format(src)) elif dest not in g:
                print('Vertex {} does not
            exist.'.format(dest)) else:
                if not g.does_edge_exist(src,
                    dest): g.add_edge(src, dest)
                else:
                    print('Edge already
    exists.') elif operation == 'bfs':
        key = int(do[1])
        print('Breadth-first Traversal: ',
        end='') vertex = g.get_vertex(key)
        display_bfs(vertex)
        print()
    elif operation == 'display':
        print('Vertices: ', end='')
        for v in g:
            print(v.get_key(),  end='  ')
        print()

        print('Edges: ')
        for v in g:
            for dest in
                v.get_neighbours(): w =
                v.get_weight(dest)
                print('(src={}, dest={}, weight={}) '.format(v.get_key(),
                                                  dest.get_key(), w))
        print()

    elif operation == 'quit':
        break
```

Output:

```
What would you like to do? add vertex 1
What would you like to do? add vertex 2
What would you like to do? add vertex 3
What would you like to do? add vertex 4
What would you like to do? add vertex 5
What would you like to do? add vertex 6
What would you like to do? add vertex 7
What would you like to do? add vertex 8
What would you like to do? add vertex 9
What would you like to do? add vertex 10
What would you like to do? add edge 1 2
What would you like to do? add edge 1 3
What would you like to do? add edge 1 5
What would you like to do? add edge 2 6
What would you like to do? add edge 3 7
What would you like to do? add edge 3 8
What would you like to do? add edge 4 8
What would you like to do? add edge 8 10
What would you like to do? add edge 5 10
What would you like to do? add edge 6 9
What would you like to do? add edge 9 10
What would you like to do? bfs 1
Breadth-first Traversal: 1 3 2 5 7 8 6 10 9
What would you like to do? quit
```