Ex no.: 1

Date:07/4/22

# IMPLEMENT SIMPLE ADTs AS PYTHON CLASSES

**AIM**

To do arithmetic operation by using simple ADTs.

# ALGORITHM

Step1: Declare the class called calculator

Step2: Define arithmetic function inside the class Step3: Allocate the object name to the class

Step4: Use object name for calling the function and display the output

# PROGRAM

class Calculator:

def \_\_init\_\_(self,v1,v2):

self.a = v1

self.b = v2

def add(self):

return self.a + self.b

def sub(self):

return abs(self.a - self.b)

def mult(self):

return self.a \* self.b

v1,v2=5,16

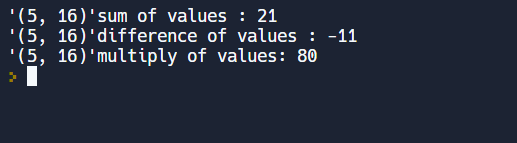
C=Calculator(v1,v2)

print(f"'{v1,v2}' sum of the values:'{C.add()}' ")

print(f"'{v1,v2}' difference of the values '{C.sub()}' ")

print(f"'{v1,v2}' product of the values '{C.mult()}' ")

# OUTPUT

****

**RESULT:**

The above program has been executed successfully and the required output is displayed.

Ex no.: 2

Date:21/4/22

**IMPLEMENT RECURSIVE ALGORITHMS IN PYTHON**

**AIM**

To do arithmetic operation by creating class.

# ALGORITHM

Step1: Declare the class called calculator

Step2: Define arithmetic function inside the class Step3: Allocate the object name to the class

Step4: Use object name for calling the function and display the output

# PROGRAM

class Factorial:

count=0

def \_\_init\_\_(self,num):

self.num= num

Factorial.count+=1

def factorial(self):

if self.num== 0:

return 1

else:

temp\_num=self.num

self.num-=1

return temp\_num\*self.factorial()

ob1=Factorial(3)

ob2=Factorial(4)

ob3=Factorial(5)

ob4=Factorial(0)

print("factorial of 3 :",ob1.factorial())

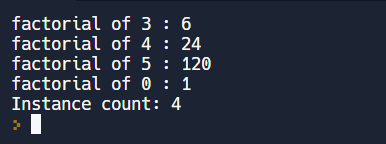
print("factorial of 4 :",ob2.factorial())

print("factorial of 5 :",ob3.factorial())

print("factorial of 0 :",ob4.factorial())

print("Instance count:",Factorial.count)

# OUTPUT



**RESULT**

The above program has been executed successfully and the required output is displayed

Ex no.: 3

Date:28/04/22

**IMPLEMENT LIST ADT USING PYTHON ARRAYS**

# AIM

To implement list ADT using array operations.

# ALGORITHM

Step1: Import array module Step2: Create a class called array

Step3: Using functions perform operation in an array Step4: Call the function using an object

# PROGRAM

class ListADT:

'''

1. Display()

2. Add(n) / Append(n)

3. Insert (index, n)

4. Delete (index)

5. Search (n)

6. Get(index)

7. Set(index, x)

8. Max() / Min()

9. Reverse()

10. Shift() / Rotate()

'''

def \_\_init\_\_(self):

self.alist=[]

#1. Display()

def display(self):

print("List Elements -> ", self.alist)

def isEmpty(self):

return self.alist==[]

#2. Add(n) / Append(n)

def addElement(self,Ele):

self.alist.append(Ele)

#3. Insert (index, n)

def addElementAt(self,Pos,Ele):

self.alist.insert(Pos,Ele)

#4. Delete (element)

def removeElement(self,Ele):

self.alist.remove(Ele)

#4. Delete (index)

def removeElementAt(self,Ele):

self.alist.pop(Ele)

#5. Search (n)

def search(self,Ele):

print(f"Element Available - {Ele in self.alist}")

#6. Get(index)

def get(self,Ele):

print(f"Element index -> {self.alist.index(Ele)}")

#7. Set(index, x)

def set(self,idx,Ele):

self.alist[idx]=Ele

#8. Max() / Min()

def MaxMin(self):

print(f" Maximum Element -> {max(self.alist)}")

print(f" Minimum Element -> {min(self.alist)}")

#9. Reverse()

def reverse(self):

print(f" Reverse the List -> {self.alist.reverse()}")

#10. Shift() / Rotate()

def Rotate(self):

for i in range(1,len(self.alist)):

print(f"shift {i} -> {alist[:i]+alist[i:]}")

l=ListADT()

l.addElement(15)

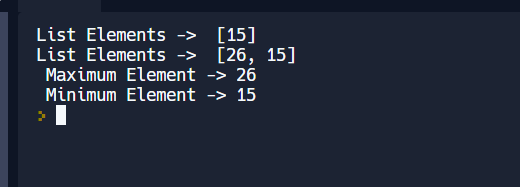
l.display()

l.addElementAt(0,26)

l.display()

l.MaxMin()

**OUTPUT**



**RESULT**

The above program has been executed successfully and the required output is displayed

Ex no.: 4

Date:05/05/22

**LINKED LIST IMPLEMENTATIONS OF LIST**

# AIM

To write a python program to perform a linked list operation

# ALGORITHM

Step1: Create a class called node. Step2: Create a class called linkedlist

Step3:Inside the linkedlist class perform operations using function Step4: Declare a object to the class

Step5: Call the functions using object

# PROGRAM

# Single Linked list :

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.next=None

class SingleLinkedList:

def \_\_init\_\_(self):

self.head=None

def display(self):

if self.head is None:

print("List is empty")

else:

temp=self.head

while temp:

print(temp.data,"-->",end=" ")

temp=temp.next

def insert(self,ele,pos=0):

n=Node(ele)

temp=self.head

if self.head is None: #Initialize

self.head=n

elif pos=='beginning': #insert at beginning

self.head=n

n.next=temp

elif pos=='end': #insert at ending

while temp:

if temp.next==None:

temp.next=n

break

temp=temp.next

else: #insert at middle(pos)

counter=1

while temp:

if counter==pos-1:

n.next= temp.next

temp.next=n

break

counter+=1

temp=temp.next

def delete(self,pos=0):

temp=self.head

if self.head is None: #check empty

pass

elif pos=='beginning': # at beginning

self.head=temp.next

elif pos=='end': # at ending

while temp:

if temp.next.next==None:

temp.next=temp.next.next

break

temp=temp.next

else: #DELETE at middle(pos)

counter=1

while temp:

if counter+1==pos:

temp.next=temp.next.next

#temp.next=n

break

counter+=1

temp=temp.next

L=SingleLinkedList()

choice=1

while(choice):

choice=eval(input("""

Operations :

1.insert at beginning

2.insert at middle(position)

3.insert at end

4.delete at beginning

5.delete at middle

6.delete at end

7.display

0.exit

Enter the choice : """))

if choice==1:

element=eval(input("Enter the element : "))

L.insert(element,'beginning')

if choice==2:

element=eval(input("Enter the element : "))

position=int(input("enter the position : "))

L.insert(element,position)

if choice==3:

element=eval(input("Enter the element : "))

L.insert(element,'end')

if choice==4:

L.delete('beginning')

if choice==5:

position=int(input("enter the position : "))

L.delete(position)

if choice==6:

L.delete('end')

if choice==7:

L.display()

# Double Linked list :

# class Node:

# def \_\_init\_\_(self,data):

# self.data=data

# self.next=None

# class DoubleLinkedList:

# def \_\_init\_\_(self):

# self.head=None

# 

# 

# def addFirst(self, val):

# newNode = Node(val)

# 

# if self.head == None:

# self.head = newNode

# else:

# newNode.next = self.head

# self.head.prev = newNode

# self.head = newNode

# def addLast(self, val):

# newNode = Node(val)

# if self.head == Node:

# self.head = newNode

# else:

# last = self.head

# while last.next != None:

# last = last.next

# last.next = newNode

# newNode.prev = last

# def search(self, key):

# temp = self.head

# while temp != None:

# if temp.data == key:

# return True

# temp = temp.next

# return False

# 

# def delete(self, key):

# if self.head == None:

# return

# temp =self.head

# while temp != None and temp.data != key:

# temp = temp.next

# if temp == None:

# print("Key Not Found")

# elif temp == self.head:

# self.head = self.head.next

# self.head.prev = None

# elif temp.next == None:

# temp.prev.next = None

# else:

# temp.prev.next = temp.next

# temp.next.prev = temp

# def display(self):

# if self.head is None:

# print("List is empty")

# else:

# temp=self.head

# while temp:

# print(temp.data,"-->",end=" ")

# temp=temp.next

# L=DoubleLinkedList()

# choice=1

# while(choice):

# choice=eval(input("""

# Operations :

# 1.insert at beginning

# 2.insert at end

# 3.search element

# 4.delete element

# 5.display

# 0.exit

# Enter the choice : """))

# if choice==1:

# element=eval(input("Enter the element : "))

# L.addFirst(element)

# if choice==2:

# element=eval(input("Enter the element : "))

# L.addLast(element)

# if choice==3:

# element=eval(input("Enter the key:"))

# L.search(element)

# if choice==4:

# element=eval(input("enter the element:"))

# L.delete(element)

# if choice==5:

# L.display()

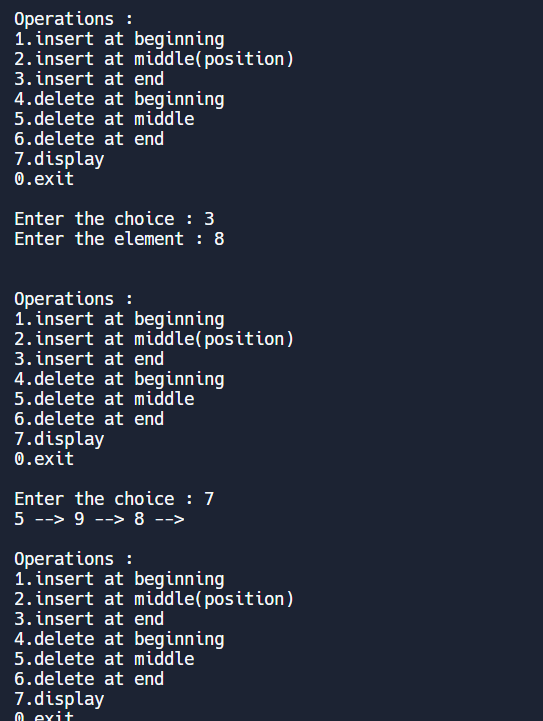
# 

# if choice==0:

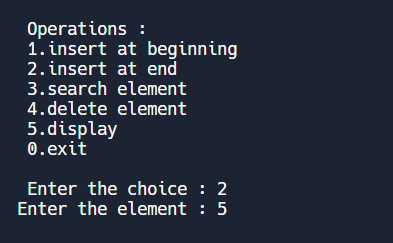
# break

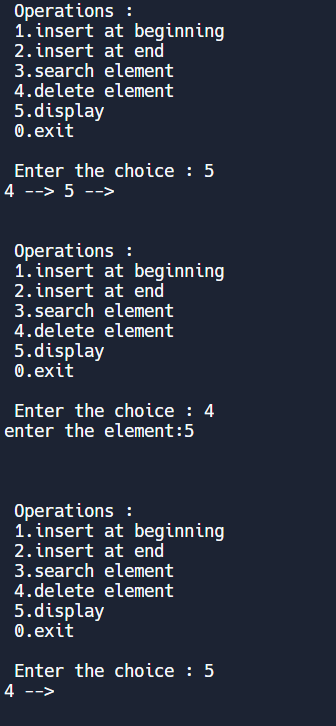
# OUTPUT

# Single linked list:

****

**Double linked list :**

****

****

**RESULT**

The above program has been executed successfully and the required output is displayed.

Ex no.: 5

Date:19/05/22

**IMPLEMENTATION OF STACK ADTS**

# AIM

To write a python program to implement the stack ADT

# ALGORITHM

Step1: Define a class Stack and implement the following methods inside the class. Step2: Define a method init () to initialize a list and the stack size.

Step3: Define a method isempty() to check if the stack is empty or not. Step4: Define a method isfull() to check if the stack is full.

Step5: Define a method push() to insert an element into the stack. Step6: Define a method pop() to delete an element from the list. Step7: Define a method disply() to display the elements in the stack**.**

Step8: Create an object reference for the class stack to access the methods inside the class. Step9: Repeat the user choices through while loop.

# PROGRAM

class Stack:

def \_\_init\_\_(self,ssize):

self.alist=[]

self.stacksize=ssize

def isempty(self):

return self.alist==[]

def isfull(self):

return len(self.alist)==self.stacksize

def push(self,ele):

if self.isfull():

print("stack is full")

else:

self.alist.append(ele)

def pop(self):

if self.isempty():

print("stack is empty")

else:

self.alist.pop()

def display(self):

print(self.alist)

choice=int(input(" 1.Stack \n 2.exit \n Enter your choice: "))

if choice==1:

StackSize=int(input("Stack size :"))

s=Stack(StackSize)

while True:

operation=int(input("""

1.push

2.pop

3.display

4.exit

Enter your choice: """))

if operation==1:

ele=input("Enter the element : ")

s.push(ele)

elif operation==2:

s.pop()

elif operation==3:

s.display()

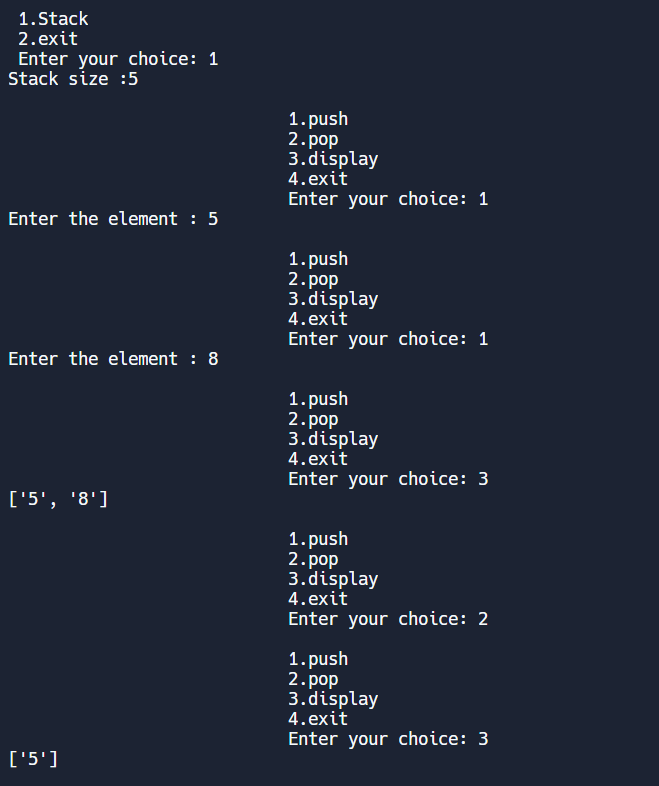
elif operation==4:

break

else:

print("option is wrong")

# OUTPUT

****

**RESULT**

The above program has been executed successfully and the required output is displayed.

Ex no.: 6

Date: 26/05/22

**IMPLEMENTATION OF QUEUE ADT**

# AIM

To write a python program to implement queue ADT

# ALGORITHM

Step1: Define a class Queue and implement the following methods inside the class. Step2: Define a method init () to initialize a list and the queue size.

Step3: Define a method isempty() to check if the queue is empty or not. Step4: Define a method isfull() to check if the queue is full.

Step5: Define a method enquue() to insert an element into the queue. Step6: Define a method dequeue() to delete an element from the queue. Step7: Define a method disply() to display the elements in the queue.

Step8: Create an object reference for the class queue to access the methods inside the class. Step9: Repeat the user choices through while loop

# PROGRAM

class Queue:

def \_\_init\_\_(self):

self.queue = []

def isempty(self):

return self.queue==[]

def isfull(self):

return len(self.queue)==self.stacksize

def enqueue(self, ele):

self.queue.append(ele)

def dequeue(self):

if len(self.queue) < 1:

return None

return self.queue.pop(0)

def display(self):

print(self.queue)

def size(self):

return len(self.queue)

d=Queue()

choice=int(input(" 1.QUEUE \n 2.exit \n Enter your choice: "))

if choice==1:

while True:

operation=int(input("""

1.enquue

2.dequeue

3.display

4.exit

Enter your choice: """))

if operation==1:

ele=input("Enter the element : ")

d.enqueue(ele)

elif operation==2:

d.dequeue()

elif operation==3:

d.display()

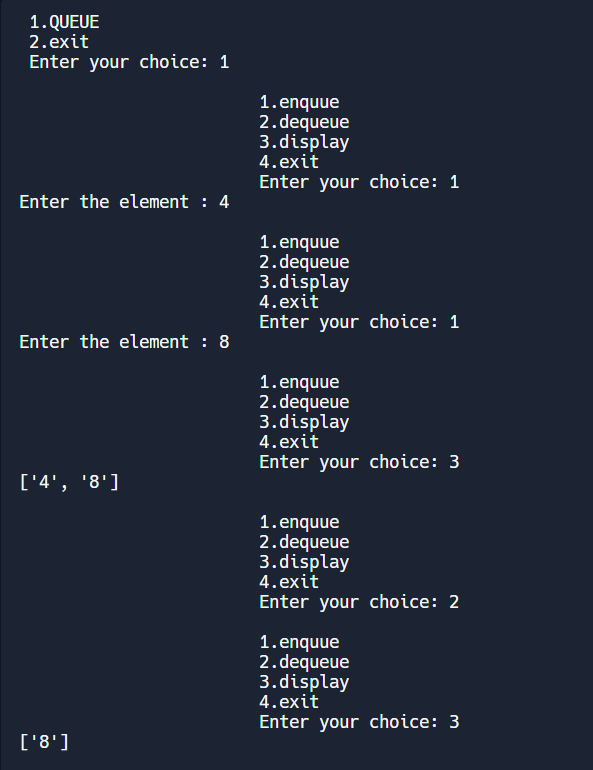
elif operation==4:

break

else:

print("option is wrong")

# OUTPUT

****

**RESULT**

The above program has been executed successfully and the required output is displayed

Ex no.: 7

Date: 29/05/22

**IMPLEMENTATION OF SORTING ALGORITHMS**

# AIM

To write a program to implement different sorting techniques used to sort a list.

# ALGORITHM

Step1: Define a class Sort.

Step2: Define a method init () to initialize the list for sorting.

Step3: Define various functions to implement various sorting techniques. Step4: Call the functions using an object.

# PROGRAM

class sorting:

def \_\_init\_\_(self, arr, size):

self.arr = arr

self.size = size

def insertion\_sort(self):

insertion\_sort(self.arr, self.size)

def selection\_sort(self):

selectionSort(self.arr, self.size)

def bubble\_sort(self):

bubbleSort(self.arr, self.size)

def heap\_Sort(self):

heapSort(self.arr, self.size)

def quick\_sort(self):

quickSort(self.arr, 0, self.size-1)

def merge\_sort(self):

mergsort(self.arr, 0, self.size-1)

#insertion sort

def insertion\_sort(arr, size):

for i in range(size):

value = arr[i]

index = 1

while index > 0 and arr[index - 1] > value:

arr[index] = arr[index - 1]

index = index - 1

arr[index] = value

#selection sort

def selectionSort(arr, size):

for i in range(size-1):

for j in range(i+1, size):

if arr[i] > arr[j]:

arr[i],arr[j] = arr[j],arr[i]

#bobbleSort

def bubbleSort(arr, size):

for i in range(size-1):

flag = 0

for j in range(size-1-i):

if arr[j] > arr[j+1]:

arr[j],arr[j+1] = arr[j+1],arr[j]

flag = 1

if flag == 0:

break

#quickSort

def quickSort(arr, start, end):

if start < end:

pivot = partition(arr, start, end)

quickSort(arr, start, pivot-1)

quickSort(arr, pivot+1, end)

def partition(arr, start, end):

PIndex = start

pivot = arr[end]

for i in range(start, end):

if arr[i] <= pivot:

arr[i], arr[PIndex] = arr[PIndex], arr[i]

PIndex = PIndex + 1

arr[PIndex], arr[end] = arr[end], arr[PIndex]

return PIndex

#mergeSort

def mergsort(arr, start, end):

if start < end:

mid = (start + end)//2

mergsort(arr, start, mid)

mergsort(arr, mid+1, end)

merge(arr, start, mid, end)

def merge(arr, start, mid, end):

temp = [None] \* (end - start + 1)

i = start

j = mid + 1

k = 0

while i <= mid and j <= end:

if arr[i] < arr[j]:

temp[k] = arr[i]

i = i + 1

k = k + 1

else:

temp[k] = arr[j]

j = j + 1

k = k + 1

while i <= mid:

temp[k] = arr[i]

i = i + 1

k = k + 1

while j <= end:

temp[k] = arr[j]

j = j + 1

k = k + 1

k = 0

for i in range(start, end+1):

arr[i] = temp[k]

k = k + 1

#HeapSort

def heapSort(arr, n):

buildHeap(arr, n)

for i in range(n-1, 0, -1):

arr[i], arr[0] = arr[0], arr[i]

heapify(arr, i, 0)

def buildHeap(arr, n):

for i in range(n//2 - 1, -1, -1):

heapify(arr, n, i)

def heapify(arr, n, i):

largest = i

left = (2 \* i) + 1

right = (2 \* i) + 2

if left < n and arr[largest] < arr[left]:

largest = left

if right < n and arr[largest] < arr[right]:

largest = right

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

heapify(arr, n, largest)

sort = sorting([3,9,2,1,4,5], 6)

sort.insertion\_sort()

print("Insertion Sort :",sort.arr)

sort.selection\_sort()

print("Selection Sort :",sort.arr)

sort.bubble\_sort()

print("Bubble Sort :",sort.arr)

sort.heap\_Sort()

print("Heap Sort :",sort.arr)

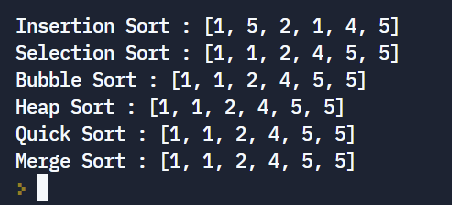
sort.quick\_sort()

print("Quick Sort :",sort.arr)

sort.merge\_sort()

print("Merge Sort :",sort.arr)

# OUTPUT



**RESULT**

The above program has been executed successfully and the required output is displayed.

Ex no.: 8

Date:29/05/22

**IMPLEMENTATION OF SEARCH ALGORITHMS**

# AIM

To write a program to implement different searching techniques used to search elements in a list.

# ALGORITHM

Step1: Define a class Search.

Step2: Define a method init() to initialize the list, low, high and the element to be found for searching. Step3: Define a function LinearSearch() to perform linear searching inside the class.

Step4: Define a function BinarySearch() to perform binary searching inside the class Step4: Call the functions using a object.

# PROGRAM

class Search:

def \_\_init\_\_ (self,alist,low,high,x):

self.alist=alist

self.low=low

self.high=high

self.x=x

def LinearSearch(self):

for i in range(len(self.alist)):

if self.alist[i]==key:

return i

def binarySearch(self):

while self.low<=self.high:

mid = (self.low + self.high) // 2

if self.alist[mid] == self.x:

return mid

elif self.alist[mid] < self.x:

self.low = mid + 1

else:

self.high = mid - 1

list1=[1,2,3,4,5,6,7,8,9]

print("given list",list1)

key=int(input("Enter the element:"))

x=int(input("Enter a element:"))

obj=Search(list1,0,len(list1)-1,x)

print("Linear Search:",obj.LinearSearch())

print("Binary Search:",obj.binarySearch())

res = obj.binarySearch()

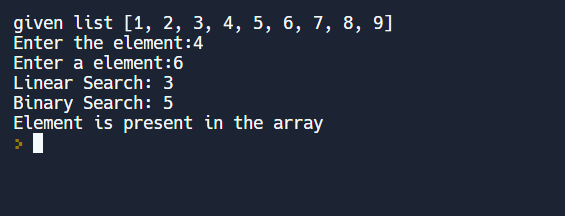
if res!= -1:

print("Element is present in the array")

else:

print("Element is not present in array")

# OUTPUT

****

**RESULT**

The above program has been executed successfully and the required output is displayed.

Ex no.: 9

Date:02/06/22

**IMPEMENTATION OF HASH TABLE**

# AIM

To write a python program to implement hash table

# ALGORITHM

Step1: Define an empty list with range as variable. Step2: Define a function and create hash table.

Step3: Define another function to insert the data values. Step4: Define a function to display the result.

Step5: Insert the data using the function. Step6: Display the Output.

# PROGRAM

def display\_hash(hashTable):

  for i in range(len(hashTable)):

    print(i, end = " ")

    for j in hashTable[i]:

      print("-->", end = " ")

      print(j, end = " ")

    print()

HashTable = [[] for \_ in range(26)]

def Hashing(keyvalue):

   return keyvalue % len(HashTable)

def insert(Hashtable, keyvalue, value):

   hash\_key = Hashing(keyvalue)

   Hashtable[hash\_key].append(value)

insert(HashTable, 10, 'Allahabad')

insert(HashTable, 25, 'Mumbai')

insert(HashTable, 20, 'Mathura')

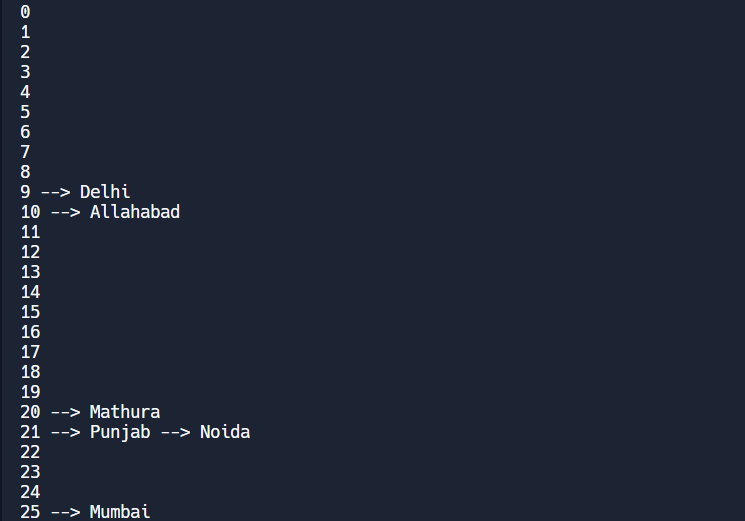
insert(HashTable, 9, 'Delhi')

insert(HashTable, 21, 'Punjab')

insert(HashTable, 21, 'Noida')

display\_hash(HashTable)

# OUTPUT

****

**RESULT**

The above program has been executed successfully and the required output is displayed.

**TREE REPRESENTATION AND TRANVERSAL ALGORITHM**

Ex no.: 10

Date:

02/06/22

# AIM

To write a python program to implement tree representation and traversal algorithm.

# ALGORITHM

Step1: Create a Class and implement the following. Step2: Recursively traverse left subtree.

Step3: Visit root node. Step4: Recursively traverse right subtree.

# PROGRAM

class Node:

  def \_\_init\_\_(self, key):

    self.left = None

    self.right = None

    self.val = key

  if root:

    # First recur on left child

    printInorder(root.left)

    # then print the data of node

    print(root.val),

    # now recur on right child

    printInorder(root.right)

  if root:

    printPostorder(root.left)

    printPostorder(root.right)

    print(root.val),

  if root:

    print(root.val),

    printPreorder(root.left)

    printPreorder(root.right)

root = Node(1)

root.left = Node(2)

root.right = Node(3)

root.left.left = Node(4)

root.left.right = Node(5)

print ("Preorder traversal of binary tree is")

printPreorder(root)

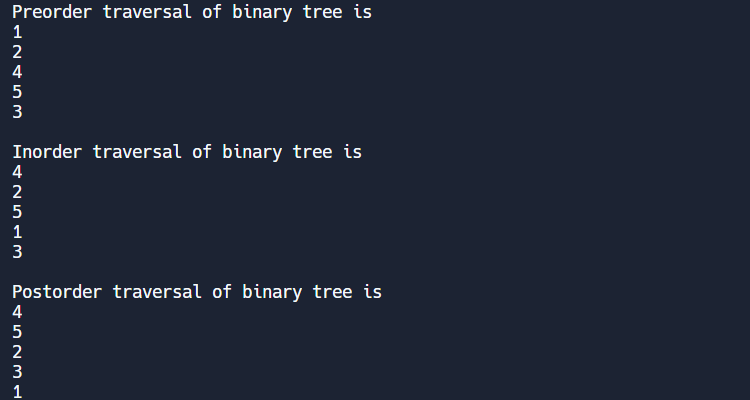
print ("\nInorder traversal of binary tree is")

printInorder(root)

print ("\nPostorder traversal of binary tree is")

printPostorder(root)

# OUTPUT

****

**RESULT**

The above program has been executed successfully and the required output is displayed.

Ex no.: 11

Date: 05/06/22

**IMPLEMENTATION OF BINARY SEARCH TREES**

# AIM:

To write a program to implement binary search tree.

# ALGORITHM:

Step1: Compare the element to be searched with the root element of the tree.

Step2: If root is matched with the target element, then return the node's location.

Step3: If it is not matched, then check whether the item is less than the root element. Step4: If it is smaller than the root element, then move to the left subtree.

Step5: If it is larger than the root element, then move to the right subtree.

Step6: Repeat the above procedure recursively until the match is found.

Step7: If the element is not found or not present in the tree, then return NULL.

# PROGRAM:

class Node:

  def \_\_init\_\_(self, key):

     self.key = key

     self.left = None

     self.right = None

def inorder(root):

    if root is not None:

     inorder(root.left)

     print(root.key, end=" ")

     inorder(root.right)

def insert(node, key):

   if node is None:

     return Node(key)

   if key < node.key:

     node.left = insert(node.left, key)

    else:

      node.right = insert(node.right, key)

   return node

def deleteNode(root, key):

   if root is None:

     return root

   if key < root.key:

     root.left = deleteNode(root.left, key)

     return root

    elif(key > root.key):

     root.right = deleteNode(root.right, key)

     return root

    if root.left is None and root.right is None:

      return None

   if root.left is None:

     temp = root.right

      root = None

     return temp

   elif root.right is None:

      temp = root.left

     root = None

     return temp

  succParent = root

  succ = root.right

  while succ.left != None:

    succParent = succ

    succ = succ.left

  if succParent != root:

    succParent.left = succ.right

  else:

    succParent.right = succ.right

  root.key = succ.key

  return root

""" Let us create following BST

      50

    /  \

    30   70

    / \ / \

  20 40 60 80 """

root = None

root = insert(root, 50)

root = insert(root, 30)

root = insert(root, 20)

root = insert(root, 40)

root = insert(root, 70)

root = insert(root, 60)

root = insert(root, 80)

print("Inorder traversal of the given tree")

inorder(root)

print("\nDelete 20")

root = deleteNode(root, 20)

print("Inorder traversal of the modified tree")

inorder(root)

print("\nDelete 30")

root = deleteNode(root, 30)

print("Inorder traversal of the modified tree")

inorder(root)

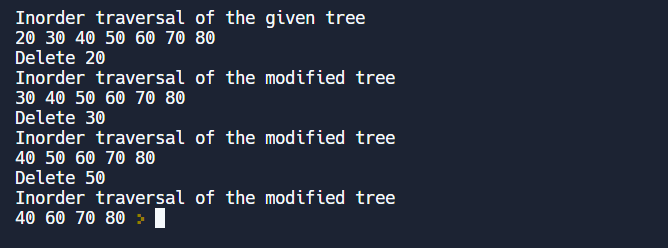
print("\nDelete 50")

root = deleteNode(root, 50)

print("Inorder traversal of the modified tree")

inorder(root)

# OUTPUT:

****

**RESULT:**

The above program is executed successfully and output is displayed.

Ex no.: 12

Date:09/06/22

**IMPLEMENTATION OF HEAPS**

# AIM:

To write a program to implement heaps.

# ALGORITHM:

Step1: Define a function and implement the following.

Step2: Isolate the end element and permute the K-1 elements. . Step3: Pick the next unique element and swap with the end element.

Step4: Repeat step 1 and 2 until all the elements became the end element once Step5: Call the function and display the output.

# PROGRAM:

def min\_heapify(A,k):

    l = left(k)

    r = right(k)

    if l < len(A) and A[l] < A[k]:

        smallest = l

    else:

        smallest = k

    if r < len(A) and A[r] < A[smallest]:

        smallest = r

    if smallest != k:

        A[k], A[smallest] = A[smallest], A[k]

        min\_heapify(A, smallest)

def left(k):

    return 2 \* k + 1

def right(k):

    return 2 \* k + 2

def build\_min\_heap(A):

    n = int((len(A)//2)-1)

    for k in range(n, -1, -1):

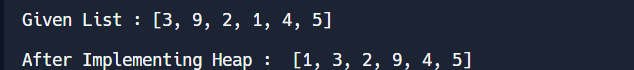
        min\_heapify(A,k)

A = [3,9,2,1,4,5]

build\_min\_heap(A)

print(A)

# OUTPUT:

****

**RESULT:**

The above program is executed successfully and output is displayed.

Ex no.: 13

Date:16/06/22

**IMPLEMENTATION OF SINGLE SOURCE SHORTEST PATH ALGORITHM**

# AIM:

To write a program to implement single source shortest path algorithm.

# ALGORITHM:

Step1: Create a set spt set (shortest path tree set).

Step2: Assign a distance value to all vertices in the input graph. Step3: Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

Step4: While spt set doesn’t include all vertices

….a) Pick a vertex u which is not there in spt set and has a minimum distance value.

….b) Include u to sptSet.

….c) Update distance value of all adjacent vertices of u,To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

**PROGRAM :**

import sys

class Graph():

  def \_\_init\_\_(self, vertices):

    self.V = vertices

    self.graph = [[0 for column in range(vertices)]

          for row in range(vertices)]

  def printSolution(self, dist):

    print("Vertex \tDistance from Source")

    for node in range(self.V):

      print(node, "\t", dist[node])

  def minDistance(self, dist, sptSet):

    min = sys.maxsize

    for u in range(self.V):

      if dist[u] < min and sptSet[u] == False:

        min = dist[u]

        min\_index = u

    return min\_index

  def dijkstra(self, src):

    dist = [sys.maxsize] \* self.V

    dist[src] = 0

    sptSet = [False] \* self.V

    for cout in range(self.V):

      x = self.minDistance(dist, sptSet)

      sptSet[x] = True

      for y in range(self.V):

        if self.graph[x][y] > 0 and sptSet[y] == False and \

        dist[y] > dist[x] + self.graph[x][y]:

            dist[y] = dist[x] + self.graph[x][y]

    self.printSolution(dist)

g = Graph(9)

g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

    [4, 0, 8, 0, 0, 0, 0, 11, 0],

    [0, 8, 0, 7, 0, 4, 0, 0, 2],

    [0, 0, 7, 0, 9, 14, 0, 0, 0],

    [0, 0, 0, 9, 0, 10, 0, 0, 0],

    [0, 0, 4, 14, 10, 0, 2, 0, 0],

    [0, 0, 0, 0, 0, 2, 0, 1, 6],

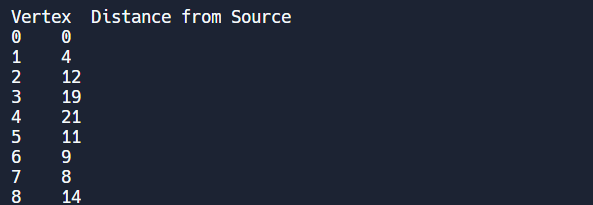
    [8, 11, 0, 0, 0, 0, 1, 0, 7],

    [0, 0, 2, 0, 0, 0, 6, 7, 0]

    ];

g.dijkstra(0)

# OUTPUT:

****

**RESULT:**

The above program is executed successfully and output is displayed.

Ex no.: 14

Date:

**IMPLEMENTATION OF MINIMUM SPANNING TREE ALGORITHMS**

19/06/22

# AIM:

To write a program to implement minimum spanning tree algorithms.

# ALGORITHM:

Step1: Sort all edges in increasing order of their edge weights.

Step2: Pick the smallest edge . Step3 Check if the new edge creates a cycle or loop in a spanning tree.

Step4: If it doesn’t form the cycle, then include that edge in MST. Otherwise, discard it. Step4: Repeat from step 2 until it includes |V| - 1 edges in MST.

# PROGRAM:

import sys

class Graph():

   def \_\_init\_\_(self, vertices):

     self.V = vertices

     self.graph = [[0 for column in range(vertices)]

           for row in range(vertices)]

   def printMST(self, parent):

     print ("Edge \tWeight")

     for i in range(1, self.V):

       print (parent[i], "-", i, "\t", self.graph[i][parent[i]])

   def minKey(self, key, mstSet):

     min = sys.maxsize

     for v in range(self.V):

       if key[v] < min and mstSet[v] == False:

         min = key[v]

         min\_index = v

     return min\_index

   def primMST(self):

     key = [sys.maxsize] \* self.V

     parent = [None] \* self.V

     key[0] = 0

     mstSet = [False] \* self.V

     parent[0] = -1

     for cout in range(self.V):

       u = self.minKey(key, mstSet)

       mstSet[u] = True

       for v in range(self.V):

         if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:

             key[v] = self.graph[u][v]

             parent[v] = u

     self.printMST(parent)

g = Graph(5)

g.graph = [ [0, 2, 0, 6, 0],

       [2, 0, 3, 8, 5],

       [0, 3, 0, 0, 7],

       [6, 8, 0, 0, 9],

       [0, 5, 7, 9, 0]]

g.primMST();

# OUTPUT:

# 

**RESULT:**

The above program is executed successfully and output is displayed.