**Exercise - 1**

**SPLAY TREES**

**Aim:** To write a python code to implement insertion and deletion of splay trees.

**Algorithm:**

**Insertion:**

1. If the tree is empty, create a new node with the given value as the root and return.
2. Perform a standard BST insertion by recursively traversing the tree:

* If the value to be inserted is less than the current node's value, go to the left subtree.
* If the value to be inserted is greater than the current node's value, go to the right subtree.
* If the value is already present, update the node's frequency or perform any other required operations.

1. After the insertion, splay the inserted node to the root using splay operations.

**Deletion:**

1. If the tree is empty, return.
2. Perform a standard BST deletion by recursively searching for the node to delete:

* If the value to delete is less than the current node's value, go to the left subtree.
* If the value to delete is greater than the current node's value, go to the right subtree.
* If the value is found, perform the deletion.

1. After the deletion, splay the parent of the deleted node to the root using splay operations.

**Splaying:**

1. Start by finding the target node using standard BST search.
2. Perform a series of rotations until the target node becomes the root:

* Zig-Zig case: If both the target node and its parent are left children or both are right children, perform a double rotation in the same direction.
* Zig-Zag case: If the target node and its parent are opposite children, perform a double rotation in opposite directions.
* Zig case: If the target node is a direct child of the root, perform a single rotation.

1. Repeat steps 1 and 2 until the target node becomes the root.

**Source code:**

class Node:

       def \_\_init\_\_(self,data,left=None,right=None,parent=None):

              self.data=data

              self.parent=parent

              self.left=left

              self.right=right

class Splaytree:

       def \_\_init\_\_(self):

              self.root=None

       def min(self,node):

              while node.left is not None:

                     node=node.left

              return node

       def search(self,node,key):

              if node==None:

                     return None

              if node.data==key:

                     return node.data

              if key<node.data:

                     return self.search(node.left,key)

              return self.search(node.right,key)

       def insert(self,val):

              node=Node(val)

              x=self.root

              y=None

              while x is not None:

                     y=x

                     if node.data<x.data:

                            x=x.left

                     else:

                            x=x.right

              node.parent=y

              if node.parent is None:

                     self.root=node

              elif node.data<y.data:

                     y.left=node

              else:

                     y.right=node

              self.splay(node)

       def delete(self,data):

              self.delete1(self.root,data)

       def delete1(self,node,key):

              if node is None:

                     return node

              elif key<node.data:

                     node.left=self.delete1(node.left,key)

              elif key>node.data:

                     node.right=self.delete1(node.right,key)

              else:

                     if node.left==None and node.right==None:

                            node.data=None

                     if node.left==None:

                            temp=node.right

                            node=temp

                            return temp

                     if node.right==None:

                            temp=node.left

                            node=temp

                            return temp

                     if node.left!=None and node.right!=None:

                            temp=self.min(node.right)

                            node.data=temp.data

                            node.right=self.delete1(node.right,temp.data)

              return node

       def inorder1(self,node):

              if node!=None:

                     self.inorder1(node.left)

                     print(node.data)

                     self.inorder1(node.right)

       def inorder(self):

              self.inorder1(self.root)

       def leftRotate(self,x):

              y=x.right

              x.right=y.left

              if y.left!=None:

                     y.left.parent=x

              y.parent=x.parent

              if x.parent==None:

                     self.root=y

              elif x==x.parent.left:

                     x.parent.left=y

              else:

                     x.parent.right=y

              y.left=x

              x.parent=y

       def rightRotate(self,x):

              y=x.left

              x.left=y.right

              if y.right!=None:

                     y.right.parent=x

              y.parent=x.parent

              if x.parent==None:

                     self.root=y

              elif x==x.parent.right:

                     x.parent.right=y

              else:

                     x.parent.left=y

              y.right=x

              x.parent=y

       def splay(self,node):

              while node.parent!=None:

                     if node.parent==self.root:

                            if node==node.parent.left:

                                   self.rightRotate(node.parent)

                            else:

                                   self.leftRotate(node.parent)

                     else:

                            p=node.parent

                            g=p.parent

                            if node.parent.left==node and p.parent.left==p:

                                   self.rightRotate(g)

                                   self.rightRotate(p)

                            elif node.parent.right==node and p.parent.right==p:

                                   self.leftRotate(g)

                                   self.leftRotate(p)

                            elif node.parent.right==node and p.parent.left==p:

                                   self.leftRotate(p)

                                   self.rightRotate(g)

                            elif node.parent.left==node and p.parent.right==p:

                                   self.rightRotate(p)

                                   self.leftRotate(g)

flag=True

splay=Splaytree()

while flag:

       print("1.INSERT\n2.SEARCH\n3.DELETE\n4.EXIT")

       ch=int(input("Enter your choice:"))

       if ch==1:

              a=int(input("Enter the number of nodes:"))

              for i in range(a):

                     b=int(input("Enter the value:"))

                     splay.insert(b)

              print()

              splay.inorder()

              print()

       elif ch==2:

              c=int(input("Enter the node to be searched:"))

              print((splay.search(splay.root,c)))

       elif ch==3:

              d=int(input("Enter the node to be deleted:"))

              splay.delete(d)

              splay.inorder()

       else:

              flag=False

**Sample Input and Output:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**Result:**

Thus, the deletion and insertion of splay trees has been successfully implemented using Python code and the output is verified.