**EXERCISE 1**

**SPLAY TREES**

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

**ALGORITHM:**

**Insertion:**

**Deletion:**

1. If(root==NULL)
2. return NULL
3. Splay(root, data)
4. If data!= root->data
5. Element is not present
6. If root->left==NULL
7. root=root->right
8. else
9. temp=root
10. Splay(root > left, data)
11. root1->right=root>right
12. free(temp)
13. return root
14. Insert(T, n)
15. temp= T\_root
16. y=NULL
17. while(temp!=NULL)
18. y=temp
19. if(n->data <temp->data)
20. temp=temp->left
21. else
22. temp=temp->right
23. n.parent= y
24. if(y==NULL)
25. T\_root = n
26. Else if (n->data < y->data)
27. y->left = n
28. else
29. y->right = n
30. Splay(T, n)

**Splaying:**

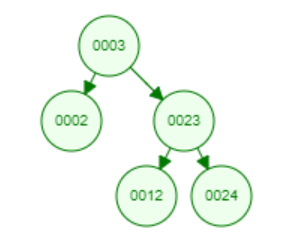
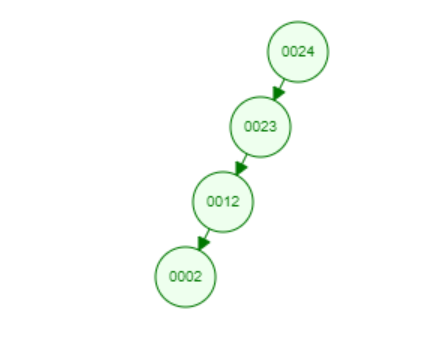
1. Splay(T, N)
2. while (n->parent !=Null)
3. if(n->parent==T->root)
4. if(n==n->parent->left)
5. right\_rotation(T, n->parent)
6. else
7. left\_rotation(T, n->parent)
8. else
9. p= n->parent
10. g = p->parent
11. if(n=n->parent->left && p=p->parent->left)
12. right.rotation(T, g), right.rotation(T, p)
13. else if(n=n->parent->right && p=p->parent->right)
14. left.rotation(T, g), left.rotation(T, p)
15. else if(n=n->parent->left && p=p->parent->right)
16. right.rotation(T, p), left.rotation(T, g)
17. else
18. left.rotation(T, p), right.rotation(T, g)
19. Implementation of right.rotation(T, x)
20. right.rotation(T, x)
21. y= x->left
22. x->left=y->right
23. y->right=x

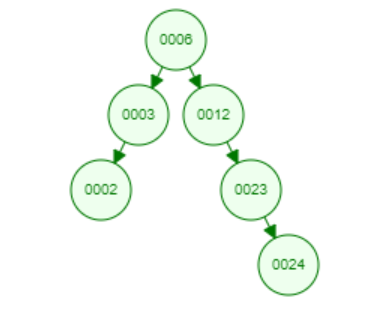
return y

**Illustrations:**

Insertion:

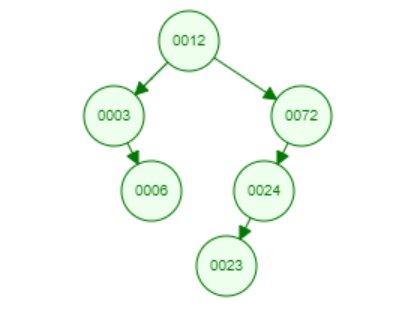
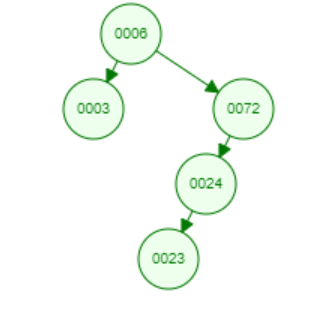
  

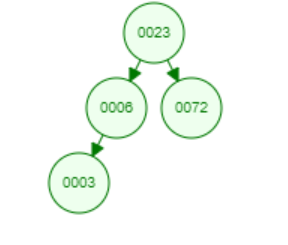


A picture containing circle, design

Description automatically generated

Deletion:



**Program 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

**OUTPUT:**

A screenshot of a computer

Description automatically generated

