BEST PRACTICE: 1

1. **Title of the Practice:** ProjectBased Learning (PBL) for B.E./B.Tech Students

2. **Objectives of the Practice:**

ProjectBased Learning at RSREC aims to:

- Establish a studentcentric, experiential learning model.
- Encourage students to develop socially relevant engineering projects.
- Foster the creation of socially responsible engineers who contribute to societal development.
- Provide students with realtime exposure to engineering projects that adhere to industry standards.
- Equip students with the necessary technical skills, knowledge, and industryrelevant competencies for successful placements.
- Enhance communication, collaboration, and soft skills essential for career growth.
- Promote interdisciplinary learning by integrating concepts from science, mathematics, and engineering with realworld applications.

3. **The Context:**

The RSREC ProjectBased Learning Framework, introduced in 202223, provides experiential learning aligned with realworld challenges for selected B.E./B.Tech students. This innovative approach combines ProjectBased Learning (PBL) with ProblemBased Learning, ensuring a strong synergy between basic sciences, mathematics, and their engineering applications.

This model offers a unique "bottomup" deep learning process, paired with a "topdown" projectmaking exercise, to create a balanced learning ecosystem. Students design scientific and engineering solutions to realworld problems, gaining a thorough understanding of core science and engineering concepts. PBL ensures studentcentered learning, where faculty members serve as facilitators, while students take the lead in exploring complex, realworld problems.

4. The Practice:

In the RSREC PBL model, students collaborate in small teams to identify a problem statement, dive deep into the problem, and develop potential solutions with the guidance of faculty, alumni, and industry mentors. This interdisciplinary approach allows students to explore the connections between science, mathematics, engineering, technology, and management.

Mentorship: Faculty mentors assist in connecting fundamental science and math with engineering principles. Alumni mentors provide industryspecific insights and guide students in applying technological solutions, while industry mentors ensure the relevance of projects to realworld applications.

Review Process: Each team is reviewed at least three times per semester to monitor progress. A final project presentation is conducted, where industry experts evaluate the originality, feasibility, design uniqueness, and overall effort of the project.

Assessment: Projects are assessed based on the innovation of the solution, execution potential, design uniqueness, and the student's dedication. The PBL model also incorporates multiple modes of evaluation, such as peer reviews, selfassessment, and mentor feedback.

5. Evidence of Success:

PBL has proven to be an effective learning method, enabling students to:

- Develop critical thinking and research skills, seek relevant information, and apply theoretical knowledge to practical scenarios.
- Engage with industry standards and provide technologydriven solutions to realworld challenges.
- Actively participate in selfdirected learning and interdisciplinary collaboration.

Key outcomes of the PBL initiative at RSREC include:

- 49 completed projects, including 5 industrystandard prototypes.
- Multiple research publications by students based on their project work.
- Enhanced employability of students, with many securing placements in leading organizations.
- Strong engagement with industry, resulting in realtime problemsolving and improved student confidence.

6. Problems Encountered and Resources Required:

Like any new pedagogy, PBL faced some challenges:

Time Management: Some students found it difficult to balance regular coursework with the additional workload of project development. To address this, the selection process was made more stringent, limiting participation to around 510% of learners who demonstrate high potential.

Resource Requirements: While dedicated lab space and research facilities are being established, students currently utilize lab facilities across various departments for project development. This highlights the need for enhanced infrastructure, including dedicated workspaces and access to advanced technological resources.

RSREC is continuously working to expand its resources and further refine the PBL model to offer students an even more immersive, realworld learning experience that equips them with the skills necessary for both academic success and industry readiness.

RAMIREDDY SUBBARAMI REDDY ENGINEERING COLLEGE KADANUTHALA 524142

PRINCIPAL
RAMIREDDY SUBBARAMI REDDY
ENGINEERING COLLEGE
NH-5, KADANUTHALA - 524 142.

BEST PRACTICE: 1

Project Based Learning (PBL) for B.E./B.Tech Students

SAMPLE PROOF / EVIDENCES

Dept.: EEE	Academic Year: 2022 -23
Class: B.Tech: IV Year- I Semester (EEE)	Regulations: R19

PROJECT BASED LEARNING MEASUREMENTS & SENSORS

1) **Identification of a Problem**: Electrical measurements are the methods, devices and calculations used to measure electrical quantities. Measurement of electrical quantities may be done to measure electrical parameters of a system.

To measure the value of unknown inductance with the help of Anderson's bridge and Maxwell's Bridge, comparing the results of both experiments

2) **Project Planning:** The following experiments are carried out by the IV EEE students.

   1. **Measurement of the value of unknown inductance by Anderson's bridge**
   2. **Measurement of the value of unknown inductance by Maxwell's bridge**

3) **Schedule: Schedule of different activities participated by IV EEE students**

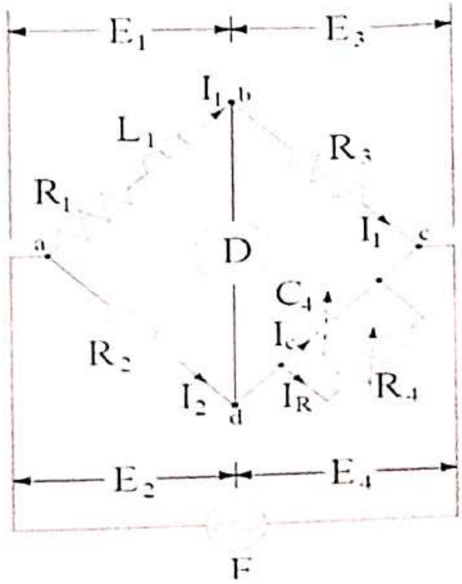
S.NO	PERIOD	ACTIVITY	NO. OF STUDENTS PARTICIPATED
1	01, NOV 2022 to 05 NOV 2022	GROUP DISCUSSIONS	36
2	07, NOV 2022 to 14 NOV 2022	SEMINARS	36
3	15, NOV 2022 to 20 NOV 2022	PROJECT	36

4) **Monitor the Progress/RESULTS:**

APPARATUS REQUIRED:

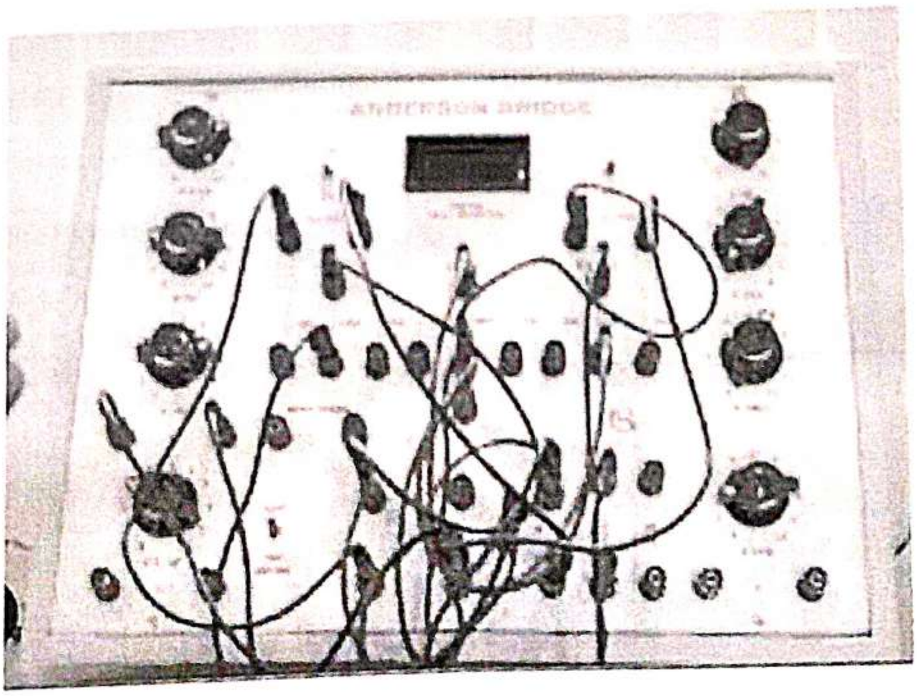
1. Anderson's bridge trainer kit.
2. Maxwell's bridge trainer kit.
3. 2mm patch chords.
4. Digital multimeter
5. Head Phone

ANDERSONS BRIDGE:



PROCEDURE:

1. Make the connections as per the circuit diagram.
2. Connect the one pair of end terminals to the input supply and other end terminals are connected to the Differential amplifier.
3. Set the amplitude knob in maximum position.
4. Rotate the Potentiometer will equal to the r1= 123.8Ω. per unknown inductor Lx1.
5. Switch on the power supply. Now connect the head phone to audio socket.
6. Rotate the potentiometer r to find a condition where null or minimum sound is generated.
7. For different combinations of L and R, vary the r where null or minimum sound is generated.
8. Switch off the power supply take the readings of variable resistance r and calculate the unknown inductance.



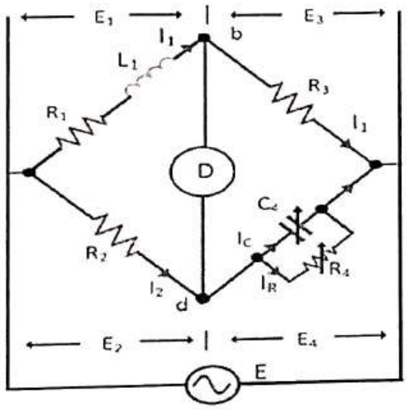
TABULAR COLUMN:

S.NO.	r(Ω)measured	r1(Ω)measured	Rx(Ω)conducted	Lx(μH)calculated
1	0.643	123.5	326.79	173.5
2	1.286	123.5	326.79	314.3
3	1.702	123.5	326.79	416.3

CALCULATIONS:

Calculated value of Lx =CR3/R4(r(R4+R2)+R4.R2)
Calculates the value of Rx =R2.R3/R4-r1

MAXWELL'S BRIDGE:



Maxewell's Inductance Capacitance Bridge

PROCEDURE:

1) Make the connections as per the circuit diagram.

2) Connect the one pair of bridge end terminals to the null detector section.

3) And other input terminals are connected to output terminals.

4) Set the R2 knob in minimum position.

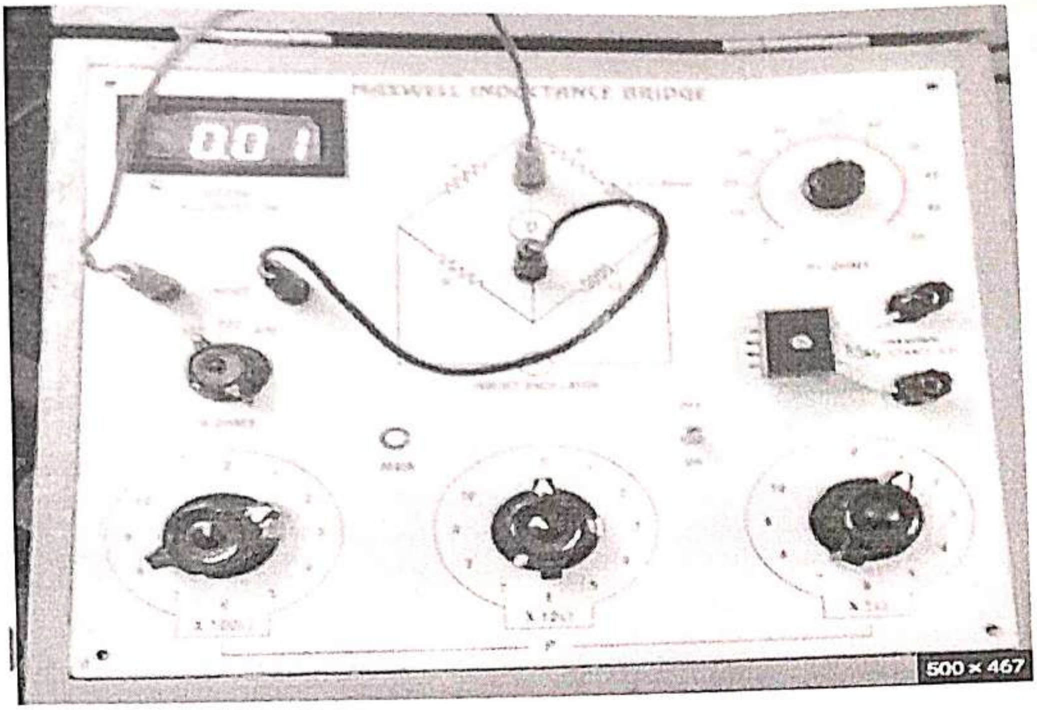
5) Set the amplitude in maximum position. i.e., 1 kHz.

6) For different combinations of R and L vary the R2 knob until the null detection section shows zero reading.

7) Switch off the power supply, note the values of R2 by using multimeter.

8) Now calculate the values of unknown resistance (Rx)and inductance (Lx), in each case.

9) Switch off the power supply to the trainer kit.



TABULAR COLUMN:

Cases	$R_2(\Omega)$	$R_4(\Omega)$	$R_3(\Omega)$	$L_1(\mu H)$	$L_x=R_2.L_1/R_4(\mu H)$	$R_x=R_2 R_3/R_4(\Omega)$
1	270	100	120.8	12.1	170.2	326.16
2	198	100	237.9	12.1	312.8	471
3	110.9	100	470	12.1	408	521.2

RAMIREDDY SUBBARAMIREDDY ENGINEERING COLLEGE:KADANUTHALA

ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT Date:24 NOV,2022

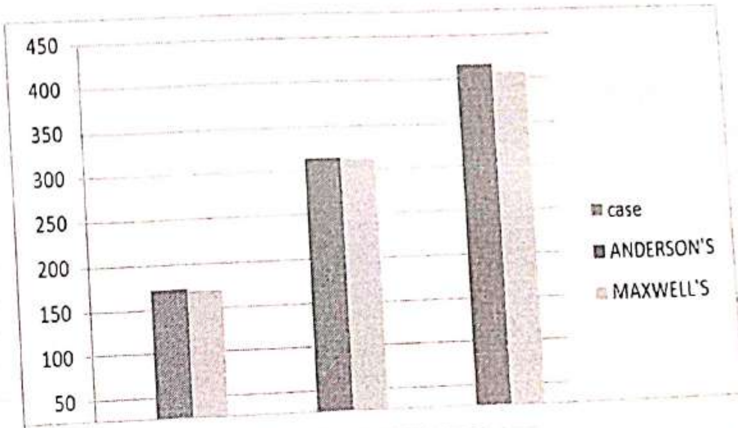
PROJECT BASED LEARNING-ASSESSMENT IV B.TECH,EEE

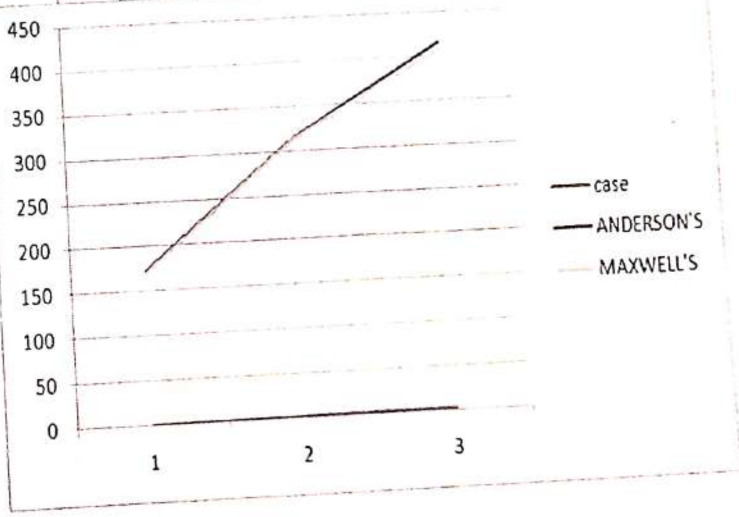
S.NO	ROLL NO	NAME	ATTENDANCE (5)	G.D (10)	SEMINAR (15)	PROJECT (25)	TOTAL
1	193R1A0201	G AJAY KUMAR REDDY	5	9	12	24	45
2	193R1A0202	A AKHILA	5	9	12	24	45
3	193R1A0206	S BHAVANA	5	9	13	25	47
4	193R1A0207	M BRAHMANI	5	10	14	25	49
5	193R1A0209	CH CHANDRASEKHAR REDDY	5	10	14	25	49
6	193R1A0210	E DAYA SANKALP	5	10	15	25	50
7	193R1A0211	I HARSHITHA	5	10	13	23	46
8	193R1A0212	D HEMANTH	5	8	12	22	42
9	193R1A0214	K KARTHEEK	5	7	12	22	41
10	193R1A0215	K KRUPA	5	9	12	23	44
11	193R1A0217	Y LAKSHMI PRIYA	5	9	14	23	46
12	193R1A0218	P LAVANYA	5	9	13	23	45
13	193R1A0219	SK LOHEENA	5	10	15	25	50
14	193R1A0220	M LOKESH	5	8	12	23	43
15	193R1A0221	K MADHURI	5	10	14	22	46
16	193R1A0223	M NANDINI	5	9	13	23	45
17	193R1A0224	G NIRAJA	5	10	15	24	49
18	193R1A0225	K NITHYA NIKHITA	5	10	15	24	49
19	193R1A0226	A PAVAN KUMAR	5	8	13	24	45
20	193R1A0227	K PAVANI	5	9	12	23	44
21	193R1A0229	S PRATAPREDDY	5	9	14	24	47
22	193R1A0230	M PRIYANKA	5	9	13	23	45
23	193R1A0231	T RAKESH	5	9	13	22	44
24	193R1A0232	SK REEMA	5	8	12	23	43
25	193R1A0233	SK SOHAIL	5	10	12	23	45
26	193R1A0234	CH SRAVANI	5	10	15	25	50
27	193R1A0235	P SRAVANI	5	10	15	24	49
28	193R1A0237	Y SRIKAR THEJA	5	8	14	24	46
29	193R1A0238	I SUPRIYA	5	9	13	24	46
30	193R1A0239	Y THANUSHA	5	9	12	23	44
31	193R1A0240	B TIRUPATHISWAMY	5	9	13	22	44
32	193R1A0241	U VAMSIKRISHNA	5	8	12	23	43
33	193R1A0242	K VENUGOPAL REDDY	5	9	13	24	46
34	203R5A0201	G VENKATA NEERAJA	5	10	14	24	48
35	203R5A0202	SD SAZID	5	7	12	23	42
36	203R5A0203	T SESHASAI KONDAIAH	5	8	12	22	42

Signature of HOD

6) Evaluation of results:

case	1	2	3
ANDERSON'S	173.5	314.8	416.3
MAXWELL'S	170.2	312.8	408





case	1	2	3
ANDERSON'S	173.5	314.8	416.3
MAXWELL'S	170.2	312.8	408

BEST PRACTICE: 1

Project Based Learning (PBL) for B.E./B.Tech Students

SAMPLE PROOF / EVIDENCES

RAMIREDDY SUBBARAMI REDDY ENGINEERING COLLEGE

(Approved by AICTE- Affiliated to JNTUA- An ISO 9001:2015 Certified Institution)

NH.16, Kadanuthala (V), Bogole (M), SPS Nellore Dt., Andhra Pradesh – 524 142

3R

Dept.: COMPUTER SCIENCE & ENGINEERING | **Academic Year: 2022-23**

Class: B.Tech. II Year- I Semester (CSE) | **Regulations: R20**

PROJECT BASED LEARNING

REPORT

ON

ADVANCED DATASTRUCTURES & ALGORITHMS

Submitted
BY

SK AFZAL	213R1A0504
R ANNAMAIAH	213R1A0510
B ANU	213R1A0511
A CHARAN KUMAR	213R1A0518
M GAYATHRI	213R1A0522
M HARSHA VARDHAN	213R1A0527
T JAYA MADHURI	213R1A0532
B KAVYA	213R1A0536
M MAHITHA SREE	213R1A0550
SK MOHAMMED YUNIS	223R5A0560

Under the valuable Guidance of

MR KV PASADA REDDY,

Associate Prof. AI & DS

Date: 03.11.2022

RSREC/CSE/NOV/cir-01

CIRCULAR

PROJECT BASED LEARNING

As per the first Academic Council meeting, the Project Based Learning(PBL) system is to be adopted by the CSE Department for one course, it as follows,

Course Name: ADVANCED DATA STRUCTURES AND ALGOITHMS

Course Code: 20A5301T

Regulation: R-20

Semester: II - I

PBL Faculty Instructor: Mr. KV PRASADA REDDY, Associate Prof., AI & DS

The faculty instructor is responsible for submitting the batch-wise report for PBL. Also, the Assignment marks of the students will be evaluated through the PBL Reports for the above-mentioned course.

HOD-CSE and AI&DS

HEAD
DEPARTMENT OF CSE
RSR Engineering College
NH-5, KADANUTHALA

Copy to

1. Dept. NB and Class NB.
2. HOD - Office file.
3. IQAC
4. Course File.

Batch-1

S.No.	Roll. No	Name of the Student
1	213R1A0504	SK AFZAL
2	213R1A0510	R ANNAMAIAH
3	213R1A0511	B ANU
4	213R1A0518	A CHARAN KUMAR
5	213R1A0522	M GAYATHRI
6	213R1A0527	M HARSHA VARDHAN
7	213R1A0532	T JAYA MADHURI
8	213R1A0536	B KAVYA
9	213R1A0550	M MAHITHA SREE
10	223R5A0560	SK MOHAMMED YUNIS

Name of the Activity : AVL tress Operations

ABSTRACT

AVL tree is a height-balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced if, the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if the height of left and right children of every node differ by either -1, 0 or +1. In an AVL tree, every node maintains an extra information known as **balance factor**

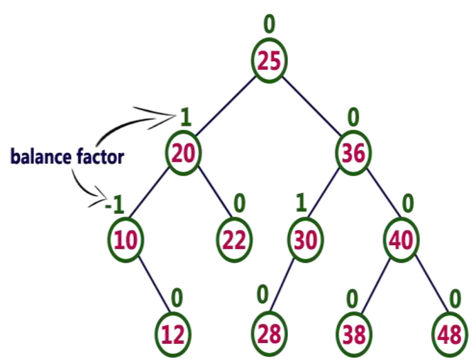
INTRODUCTION:

An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node is either -1, 0 or +1.

Balance factor of a node is the difference between the heights of the left and right subtrees of that node. The balance factor of a node is calculated either **height of left subtree - height of right subtree** (OR) **height of right subtree - height of left subtree**.

Balance factor = heightOfLeftSubtree - heightOfRightSubtree

Example of AVL Tree



The above tree is a binary search tree and every node is satisfying balance factor condition. So this tree is said to be an AVL tree.

Every AVL Tree is a binary search tree but every Binary Search Tree need not be AVL tree.

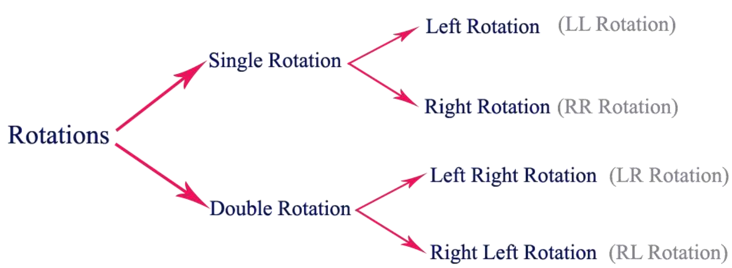
METHODOLOGY:

AVL Rotations

An AVL tree, after performing operations like insertion and deletion we need to check the **balance factor** of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. Whenever the tree becomes imbalanced due to any operation we use **rotation** operations to make the tree balanced.

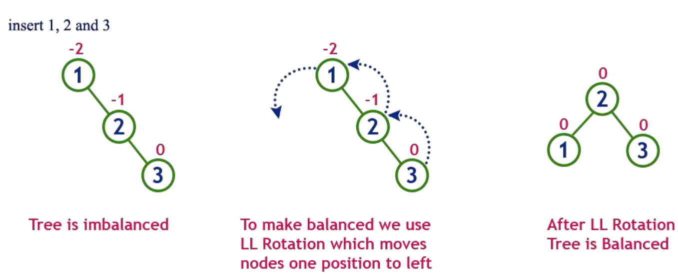
Rotation operations are used to make the tree balanced.

There are **four** rotations and they are classified into **two** types.



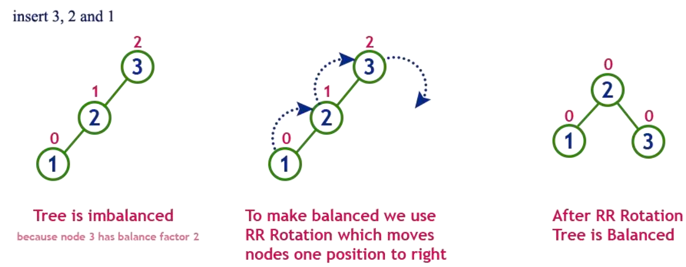
SINGLE LEFT ROTATION (LL ROTATION)

In LL Rotation, every node moves one position to left from the current position. To understand LL Rotation, let us consider the following insertion operation in AVL Tree...



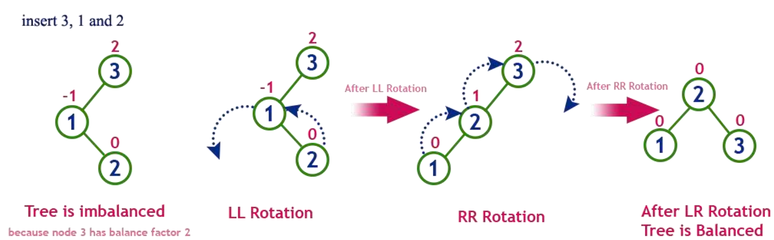
SINGLE RIGHT ROTATION (RR ROTATION)

In RR Rotation, every node moves one position to right from the current position. To understand RR Rotation, let us consider the following insertion operation in AVL Tree...



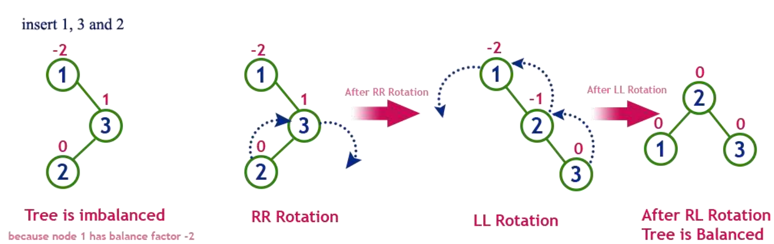
LEFT RIGHT ROTATION (LR ROTATION)

The LR Rotation is a sequence of single left rotation followed by a single right rotation. In LR Rotation, at first, every node moves one position to the left and one position to right from the current position. To understand LR Rotation, let us consider the following insertion operation in AVL Tree...



RIGHT LEFT ROTATION (RL ROTATION)

The RL Rotation is sequence of single right rotation followed by single left rotation. In RL Rotation, at first every node moves one position to right and one position to left from the current position. To understand RL Rotation, let us consider the following insertion operation in AVL Tree...



SYSTEM REQUIREMET SPECIFICATIONS

Hardware Requirements

RAM :4GB
HDD :500GB
Processor: Intel i3

Software Requirements:

OS : windows 7-10

Programming Language: Python Programming Language

ALGORITHM IMPLEMENTATION

Software implementation refers to a set of computer science activities dedicated to the process of creating, designing, deploying and supporting software. Software itself is the set of instructions or programs that tell a computer what to do. It is independent of hardware and makes computers programmable.

OPERATIONS ON AN AVL TREE

The following operations are performed on AVL tree...

1. **Search**
2. **Insertion**
3. **Deletion**

Search Operation in AVL Tree

In an AVL tree, the search operation is performed with **O(log n)** time complexity. The search operation in the AVL tree is similar to the search operation in a Binary search tree. We use the following steps to search an element in AVL tree...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Compare the search element with the value of root node in the tree.
- **Step 3 -** If both are matched, then display "Given node is found!!!" and terminate the function
- **Step 4 -** If both are not matched, then check whether search element is smaller or larger than that node value.
- **Step 5 -** If search element is smaller, then continue the search process in left subtree.
- **Step 6 -** If search element is larger, then continue the search process in right subtree.
- **Step 7 -** Repeat the same until we find the exact element or until the search element is compared with the leaf node.
- **Step 8 -** If we reach to the node having the value equal to the search value, then display "Element is found" and terminate the function.

- **Step 9 -** If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Insertion Operation in AVL Tree

In an AVL tree, the insertion operation is performed with **O(log n)** time complexity. In AVL Tree, a new node is always inserted as a leaf node. The insertion operation is performed as follows...

- **Step 1 -** Insert the new element into the tree using Binary Search Tree insertion logic.
- **Step 2 -** After insertion, check the **Balance Factor** of every node.
- **Step 3 -** If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.
- **Step 4 -** If the **Balance Factor** of any node is other than **0 or 1 or -1** then that tree is said to be imbalanced. In this case, perform suitable **Rotation** to make it balanced and go for next operation.

Deletion Operation in AVL Tree

The deletion operation in AVL Tree is similar to deletion operation in BST. But after every deletion operation, we need to check with the Balance Factor condition. If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.

CODE IMPLEMENTATION

```
class treeNode(object):
    def __init__(self, value):
        self.value = value
        self.l = None
        self.r = None
        self.h = 1

class AVLTree(object):

    def insert(self, root, key):

        if not root:
            return treeNode(key)
        elif key < root.value:
            root.l = self.insert(root.l, key)
        else:
            root.r = self.insert(root.r, key)

        root.h = 1 + max(self.getHeight(root.l),
                        self.getHeight(root.r))

        b = self.getBal(root)

        if b > 1 and key < root.l.value:
            return self.rRotate(root)

        if b < -1 and key > root.r.value:
            return self.lRotate(root)

        if b > 1 and key > root.l.value:
            root.l = self.lRotate(root.l)
            return self.rRotate(root)

        if b < -1 and key < root.r.value:
            root.r = self.rRotate(root.r)
            return self.lRotate(root)

        return root

    def lRotate(self, z):

        y = z.r
        T2 = y.l
```

```
        y.l = z
        z.r = T2

        z.h = 1 + max(self.getHeight(z.l),
                        self.getHeight(z.r))
        y.h = 1 + max(self.getHeight(y.l),
                        self.getHeight(y.r))

        return y

    def rRotate(self, z):

        y = z.l
        T3 = y.r

        y.r = z
        z.l = T3

        z.h = 1 + max(self.getHeight(z.l),
                        self.getHeight(z.r))
        y.h = 1 + max(self.getHeight(y.l),
                        self.getHeight(y.r))

        return y

    def getHeight(self, root):
        if not root:
            return 0

        return root.h

    def getBal(self, root):
        if not root:
            return 0

        return self.getHeight(root.l) - self.getHeight(root.r)

    def preOrder(self, root):

        if not root:
            return

        print("{0} ".format(root.value), end="")
        self.preOrder(root.l)
```

```
                    self.preOrder(root.r)

Tree = AVLTree()
root = None

root = Tree.insert(root, 1)
root = Tree.insert(root, 2)
root = Tree.insert(root, 3)
root = Tree.insert(root, 4)
root = Tree.insert(root, 5)
root = Tree.insert(root, 6)

# Preorder Traversal
print("Preorder traversal of the",
      "constructed AVL tree is")
Tree.preOrder(root)
print()
```

Output

4 2 1 3 5 6

```
# Python code to insert a node in AVL tree

# Generic tree node class
class TreeNode(object):
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
        self.height = 1

# AVL tree class which supports the
# Insert operation
class AVL_Tree(object):

    # Recursive function to insert key in
    # subtree rooted with node and returns
    # new root of subtree.
    def insert(self, root, key):

        # Step 1 - Perform normal BST
        if not root:
            return TreeNode(key)
        elif key < root.val:
```

```
                root.left = self.insert(root.left, key)
        else:
                root.right = self.insert(root.right, key)

        # Step 2 - Update the height of the
        # ancestor node
        root.height = 1 + max(self.getHeight(root.left),
                                                self.getHeight(root.right))

        # Step 3 - Get the balance factor
        balance = self.getBalance(root)

        # Step 4 - If the node is unbalanced,
        # then try out the 4 cases
        # Case 1 - Left Left
        if balance > 1 and key < root.left.val:
                return self.rightRotate(root)

        # Case 2 - Right Right
        if balance < -1 and key > root.right.val:
                return self.leftRotate(root)

        # Case 3 - Left Right
        if balance > 1 and key > root.left.val:
                root.left = self.leftRotate(root.left)
                return self.rightRotate(root)

        # Case 4 - Right Left
        if balance < -1 and key < root.right.val:
                root.right = self.rightRotate(root.right)
                return self.leftRotate(root)

        return root

def leftRotate(self, z):

        y = z.right
        T2 = y.left

        # Perform rotation
        y.left = z
        z.right = T2

        # Update heights
        z.height = 1 + max(self.getHeight(z.left),
```

```
                                        self.getHeight(z.right))
        y.height = 1 + max(self.getHeight(y.left),
                                        self.getHeight(y.right))

        # Return the new root
        return y

    def rightRotate(self, z):

        y = z.left
        T3 = y.right

        # Perform rotation
        y.right = z
        z.left = T3

        # Update heights
        z.height = 1 + max(self.getHeight(z.left),
                                        self.getHeight(z.right))
        y.height = 1 + max(self.getHeight(y.left),
                                        self.getHeight(y.right))

        # Return the new root
        return y

    def getHeight(self, root):
        if not root:
            return 0

        return root.height

    def getBalance(self, root):
        if not root:
            return 0

        return self.getHeight(root.left) - self.getHeight(root.right)

    def preOrder(self, root):

        if not root:
            return

        print("{0} ".format(root.val), end="")
        self.preOrder(root.left)
        self.preOrder(root.right)
```

```
# Driver program to test above function
myTree = AVL_Tree()
root = None

root = myTree.insert(root, 10)
root = myTree.insert(root, 20)
root = myTree.insert(root, 30)
root = myTree.insert(root, 40)
root = myTree.insert(root, 50)
root = myTree.insert(root, 25)

"""The constructed AVL Tree would be
                              30
                    / \
                    20 40
                    / \        \
          10 25 50"""

# Preorder Traversal
print("Preorder traversal of the",
          "constructed AVL tree is")
myTree.preOrder(root)
print()
```

Output

Preorder traversal of the constructed AVL tree is

30 20 10 25 40 50

CONCLUSION

AVL Tree is a heigh-balanced BST that maintains its height by performing tree rotations according to a Balancing Criteria. AVL Tree height is always O(log n) i.e., it has logarithmic time complexity for all the operations

Dept.: COMPUTER SCIENCE & ENGINEERING | **Academic Year: 2022-23**

Class: B.Tech. II Year- I Semester (CSE) | **Regulations: R20**

Assignment Marks

S.No.	Roll. No	Name of the Student	Operation-1 Assignment (25)	Operation-2 Assignment (25)	Operation-3 Assignment (25)	Grade/Average Marks
S.No.	Roll. No	Name of the Student	20	19	21	20
1	213R1A0504	SK AFZAL	19	19	19	19
2	213R1A0510	R ANNAMAIAH	21	22	22	22
3	213R1A0511	B ANU	19	18	20	19
4	213R1A0518	A CHARAN KUMAR	20	20	21	20
5	213R1A0522	M GAYATHRI	21	21	21	21
6	213R1A0527	M HARSHA VARDHAN	22	18	18	19
7	213R1A0532	T JAYA MADHURI	19	19	19	19
8	213R1A0536	B KAVYA	24	24	24	24
9	213R1A0550	M MAHITHA SREE	16	17	20	18

HOD-CSE and AI&DS

HEAD
DEPARTMENT OF CSE
RSR Engineering College
NH-5, KADANUTHALA

BEST PRACTICE: 1

Project Based Learning (PBL) for B.E./B.Tech Students

SAMPLE PROOF / EVIDENCES

Dept.: COMPUTER SCIENCE & ENGINEERING	Academic Year: 2022-23
Class: B.Tech. II Year- I Semester (CSE)	Regulations: R20

PROJECT BASED LEARNING

REPORT

ON

ADVANCED DATASTRUCTURES & ALGORITHMS

Submitted
BY

SK NAWZ SHAREEF	213R1A0568
I NISHANTH	213R1A0570
I NISHITHA	213R1A0571
T PRASANNA	213R1A0578
K SAGAR	213R1A0592
CH SUPRAJA	213R1A05B0
T VASANTHA LAKSHMI	213R1A05B9
M VEENA MADHURI	213R1A05C1
A VYSHNAVI	213R1A05C9
G RAVINDRA	223R5A0503

Under the valuable Guidance of

MR KV PASADA REDDY,

Associate Prof. of AI & DS

Date: 03.11.2022

RSREC/CSE/NOV/cir-01

CIRCULAR

PROJECT BASED LEARNING

As per the first Academic Council meeting, the Project Based Learning(PBL) system is to be adopted by the CSE Department for one course, it as follows,

Course Name: ADVANCED DATA STRUCTURES AND ALGOITHMS

Course Code: 20A5301T

Regulation: R-20

Semester: II - I

PBL Faculty Instructor: Mr. KV PRASADA REDDY, Associate Prof., AI & DS

The faculty instructor is responsible for submitting the batch-wise report for PBL. Also, the Assignment marks of the students will be evaluated through the PBL Reports for the above-mentioned course.

HOD-CSE and AI&DS

HEAD
DEPARTMENT OF CSE
RSR Engineering College
NH-5, KADANUTHALA

Copy to

1. Dept. NB and Class NB.
2. HOD - Office file.
3. IQAC
4. Course File.

Batch-1

S.No.	Roll. No	Name of the Student
1	213R1A0568	SK NAWZ SHAREEF
2	213R1A0570	I NISHANTH
3	213R1A0571	I NISHITHA
4	213R1A0578	T PRASANNA
5	213R1A0592	K SAGAR
6	213R1A05B0	CH SUPRAJA
7	213R1A05B9	T VASANTHA LAKSHMI
8	213R1A05C1	M VEENA MADHURI
9	213R1A05C9	A VYSHNAVI
10	223R5A0503	G RAVINDRA

Name of the Activity : Binary Search Trees operations

ABSTRACT

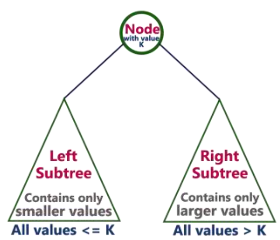
In a binary tree, every node can have a maximum of two children but there is no need to maintain the order of nodes basing on their values. In a binary tree, the elements are arranged in the order they arrive at the tree from top to bottom and left to right.

INTRODUCTION:

A Binary Search Tree (BST) is a special type of binary tree in which the left child of a node has a value less than the node's value and the right child has a value greater than the node's value. This property is called the BST property and it makes it possible to efficiently search, insert, and delete elements in the tree.

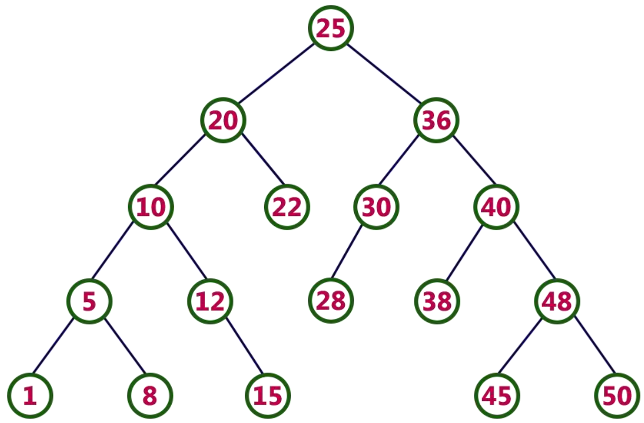
The root of a BST is the node that has the largest value in the left subtree and the smallest value in the right subtree. Each left subtree is a BST with nodes that have smaller values than the root and each right subtree is a BST with nodes that have larger values than the root.

Node contains larger values as shown in the following figure...



Example:

The following tree is a Binary Search Tree. In this tree, left subtree of every node contains nodes with smaller values and right sub tree of every node contains larger values.



METHODOLOGY:

Binary is a node-based binary tree data structure that has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- This means everything to the left of the root is less than the value of the root and everything to the right of the root is greater than the value of the root. Due to this performing, a binary search is very easy.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes(BST may have duplicate values with different handling approaches)

SYSTEM REQUIREMET SPECIFICATIONS

Hardware Requirements

RAM :4GB

HDD :500GB

Processor: Intel i3

Software Requirements:

OS : windows 7-10

Programming Language: Python Programming Language

ALGORITHM IMPLEMENTATION

Software implementation refers to a set of computer science activities dedicated to the process of creating, designing, deploying and supporting software. Software itself is the set of instructions or programs that tell a computer what to do. It is independent of hardware and makes computers programmable.

OPERATIONS ON A BINARY SEARCH TREE

The following operations are performed on a binary search tree...

1. **Search**
2. **Insertion**
3. **Deletion**

Search Operation in BST

In a binary search tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Compare the search element with the value of root node in the tree.
- **Step 3 -** If both are matched, then display "Given node is found!!!" and terminate the function
- **Step 4 -** If both are not matched, then check whether search element is smaller or larger than that node value.
- **Step 5 -** If search element is smaller, then continue the search process in left subtree.
- **Step 6-** If search element is larger, then continue the search process in right subtree.
- **Step 7 -** Repeat the same until we find the exact element or until the search element is compared with the leaf node
- **Step 8 -** If we reach to the node having the value equal to the search value then display "Element is found" and terminate the function.
- **Step 9 -** If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Insertion Operation in BST

In a binary search tree, the insertion operation is performed with **O(log n)** time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

- **Step 1 -** Create a newNode with given value and set its **left** and **right** to **NULL**.
- **Step 2 -** Check whether tree is Empty.
- **Step 3 -** If the tree is **Empty**, then set **root** to **newNode**.
- **Step 4 -** If the tree is **Not Empty**, then check whether the value of newNode is **smaller** or **larger** than the node (here it is root node).
- **Step 5 -** If newNode is **smaller** than **or equal** to the node then move to its **left** child. If newNode is **larger** than the node then move to its **right** child.
- **Step 6-** Repeat the above steps until we reach to the **leaf** node (i.e., reaches to NULL).
- **Step 7 -** After reaching the leaf node, insert the newNode as **left child** if the newNode is **smaller or equal** to that leaf node or else insert it as **right child**.

Deletion Operation in BST

In a binary search tree, the deletion operation is performed with **O(log n)** time complexity. Deleting a node from Binary search tree includes following three cases...

- **Case 1: Deleting a Leaf node (A node with no children)**
- **Case 2: Deleting a node with one child**
- **Case 3: Deleting a node with two children**

Case 1: Deleting a leaf node

We use the following steps to delete a leaf node from BST...

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** Delete the node using **free** function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

We use the following steps to delete a node with one child from BST...

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** If it has only one child then create a link between its parent node and child node.
- **Step 3 -** Delete the node using **free** function and terminate the function.

Case 3: Deleting a node with two children

We use the following steps to delete a node with two children from BST...

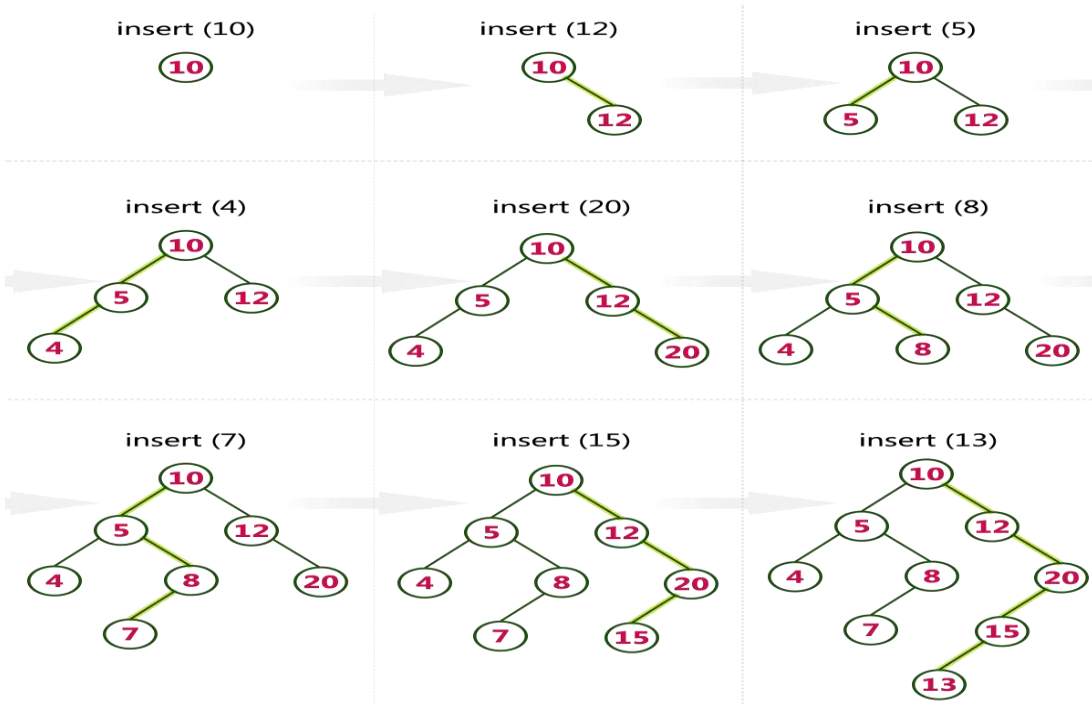
- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** If it has two children, then find the **largest** node in its **left subtree** (OR) the **smallest** node in its **right subtree**.
- **Step 3 - Swap** both **deleting node** and node which is found in the above step.
- **Step 4 -** Then check whether deleting node came to **case 1** or **case 2** or else goto step 2
- **Step 5 -** If it comes to **case 1**, then delete using case 1 logic.
- **Step 6-** If it comes to **case 2**, then delete using case 2 logic.
- **Step 7 -** Repeat the same process until the node is deleted from the tree.

Example

Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13

Above elements are inserted into a Binary Search Tree as follows...



1) Node to be deleted is the leaf: Simply remove it from the tree.

```
       50                              50
     /    \         delete(20)        /  \
   30      70       ———>          30     70
   / \    / \                          \    / \
 20   40  60   80                      40  60   80
```

2) Node to be deleted has only one child: Copy the child to the node and delete the node.

```
       50                           50
     /    \        delete(30)      /  \
   30      70      ———>       40     70
     \    / \                          / \
     40  60   80                      60   80
```

3) Node to be deleted has two children: Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor.
Note: Inorder predecessor can also be used.

```
      50                          60
    /    \        delete(50)      /  \
  40      70      ———>     40    70
         /  \                       \
       60    80                      80
```

Note: Inorder successor is needed only when the right child is not empty. In this particular case, in-order successor can be obtained by finding the minimum value in the right child of the node.
Recommended Problem
Delete a node from BST
Follow the below steps to solve the problem:

- If the root is NULL, then return root (Base case)
- If the key is less than the root's value, then set root->left = deleteNode(root->left, key)
- If the key is greater than the root's value, then set root->right = deleteNode(root->right, key)
- Else check
    - If the root is a leaf node then return null
    - else if it has only the left child, then return the left child
    - else if it has only the right child, then return the right child
    - else set the value of root as of its inorder successor and recur to delete the node with the value of the inorder successor
- Return

CODE IMPLEMENTATION


```
# Python program to insert a node
# in a BST
# Given Node
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

# Function to insert a new node with
# given key in BST
def insert(node, key):
    # If the tree is empty, return a new node
    if node is None:
```

```
        return Node(key)

    # Otherwise, recur down the tree
    if key < node.key:
        node.left = insert(node.left, key)
    elif key > node.key:
        node.right = insert(node.right, key)

    # Return the node pointer
    return node

# Function to do inorder traversal of BST
def inorder(root):
    if root is not None:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)

# Driver Code
if __name__ == '__main__':
    """
    Let us create following BST
          50
        /    \
       30     70
      / \    / \
    20  40  60   80
    """
    root = None

    # Inserting value 50
    root = insert(root, 50)

    # Inserting value 30
    insert(root, 30)

    # Inserting value 20
    insert(root, 20)

    # Inserting value 40
    insert(root, 40)

    # Inserting value 70
```

```
    insert(root, 70)

    # Inserting value 60
    insert(root, 60)

    # Inserting value 80
    insert(root, 80)

    # Print the BST
    inorder(root)

#This code is contributed by japmeet01
```

Output

20 30 40 50 60 70 80

```
A utility function to search a given key in BST
def search(root,key):

    # Base Cases: root is null or key is present at root
    if root is None or root.val == key:
        return root

    # Key is greater than root's key
    if root.val < key:
        return search(root.right,key)

    # Key is smaller than root's key
    return search(root.left,key)
```

When we delete a node, three possibilities arise.

Below is the implementation of the above approach:

```
# Python program to demonstrate delete operation
# in binary search tree

# A Binary Tree Node

class Node:
     # Constructor to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
  # A utility function to do inorder traversal of BST
def inorder(root):
    if root is not None:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)

# A utility function to insert a
# new node with given key in BST
def insert(node, key):

    # If the tree is empty, return a new node
    if node is None:
        return Node(key)

    # Otherwise recur down the tree
    if key < node.key:
        node.left = insert(node.left, key)
    else:
        node.right = insert(node.right, key)

    # return the (unchanged) node pointer
    return node

# Given a non-empty binary
# search tree, return the node
# with minimum key value
# found in that tree. Note that the
# entire tree does not need to be searched
```

```
def minValueNode(node):
    current = node

    # loop down to find the leftmost leaf
    while(current.left is not None):
        current = current.left

    return current

# Given a binary search tree and a key, this function
# delete the key and returns the new root

def deleteNode(root, key):

    # Base Case
    if root is None:
        return root

    # If the key to be deleted
    # is smaller than the root's
    # key then it lies in  left subtree
    if key < root.key:
        root.left = deleteNode(root.left, key)

    # If the kye to be delete
    # is greater than the root's key
    # then it lies in right subtree
    elif(key > root.key):
        root.right = deleteNode(root.right, key)

    # If key is same as root's key, then this is the node
    # to be deleted
    else:

        # Node with only one child or no child
        if root.left is None:
            temp = root.right
            root = None
            return temp

        elif root.right is None:
            temp = root.left
```

```
            root = None
            return temp

        # Node with two children:
        # Get the inorder successor
        # (smallest in the right subtree)
        temp = minValueNode(root.right)

        # Copy the inorder successor's
        # content to this node
        root.key = temp.key

        # Delete the inorder successor
        root.right = deleteNode(root.right, temp.key)

    return root


# Driver code
""" 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)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output

Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80

Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80

CONCLUSION

A binary search tree is the data structure in which each node should have a maximum of two child nodes, and the values of all the nodes on the left should have a value that is less than the current node, while on the right should have a value greater than the current one.

Assignment Marks

S.No.	Roll. No	Name of the Student	Operation-1 Assignment (25)	Operation-2 Assignment (25)	Operation-3 Assignment (25)	Grade/Average Marks
1	213R1A0568	SK NAWZ SHAREEF	20	19	21	20
2	213R1A0570	I NISHANTH	19	19	19	19
3	213R1A0571	I NISHITHA	21	22	22	22
4	213R1A0578	T PRASANNA	19	18	20	19
5	213R1A0592	K SAGAR	20	20	21	20
6	213R1A05B0	CH SUPRAJA	21	21	21	21
7	213R1A05B9	T VASANTHA LAKSHMI	22	18	18	19
8	213R1A05C1	M VEENA MADHURI	19	19	19	19
9	213R1A05C9	A VYSHNAVI	24	24	24	24
10	223R5A0503	G RAVINDRA	16	17	20	18

HOD-CSE and AI&DS

HEAD
DEPARTMENT OF CSE
RSR Engineering College
NH-5, KADANUTHALA