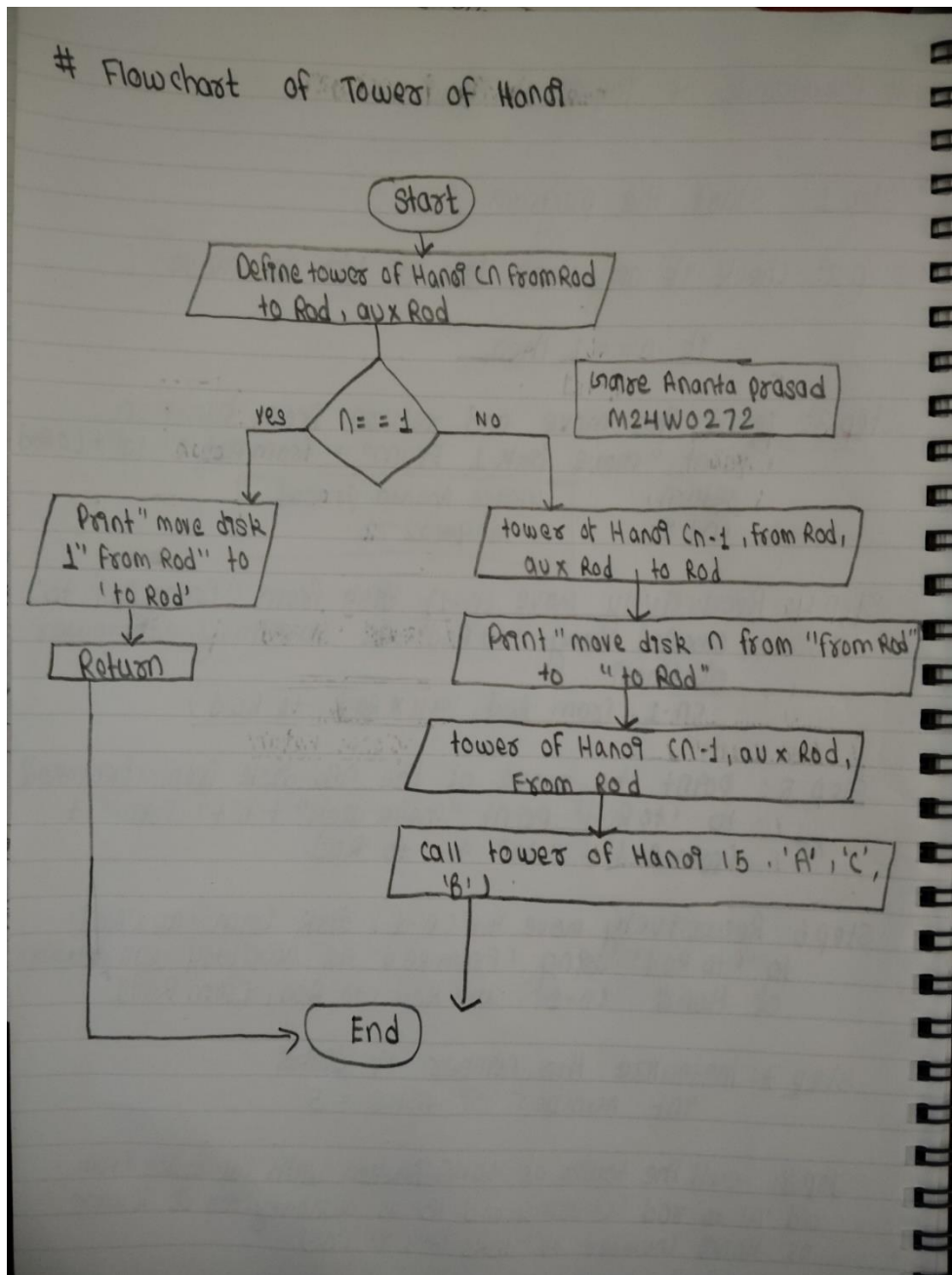


Gaire Ananta Prasad (M24W0272)

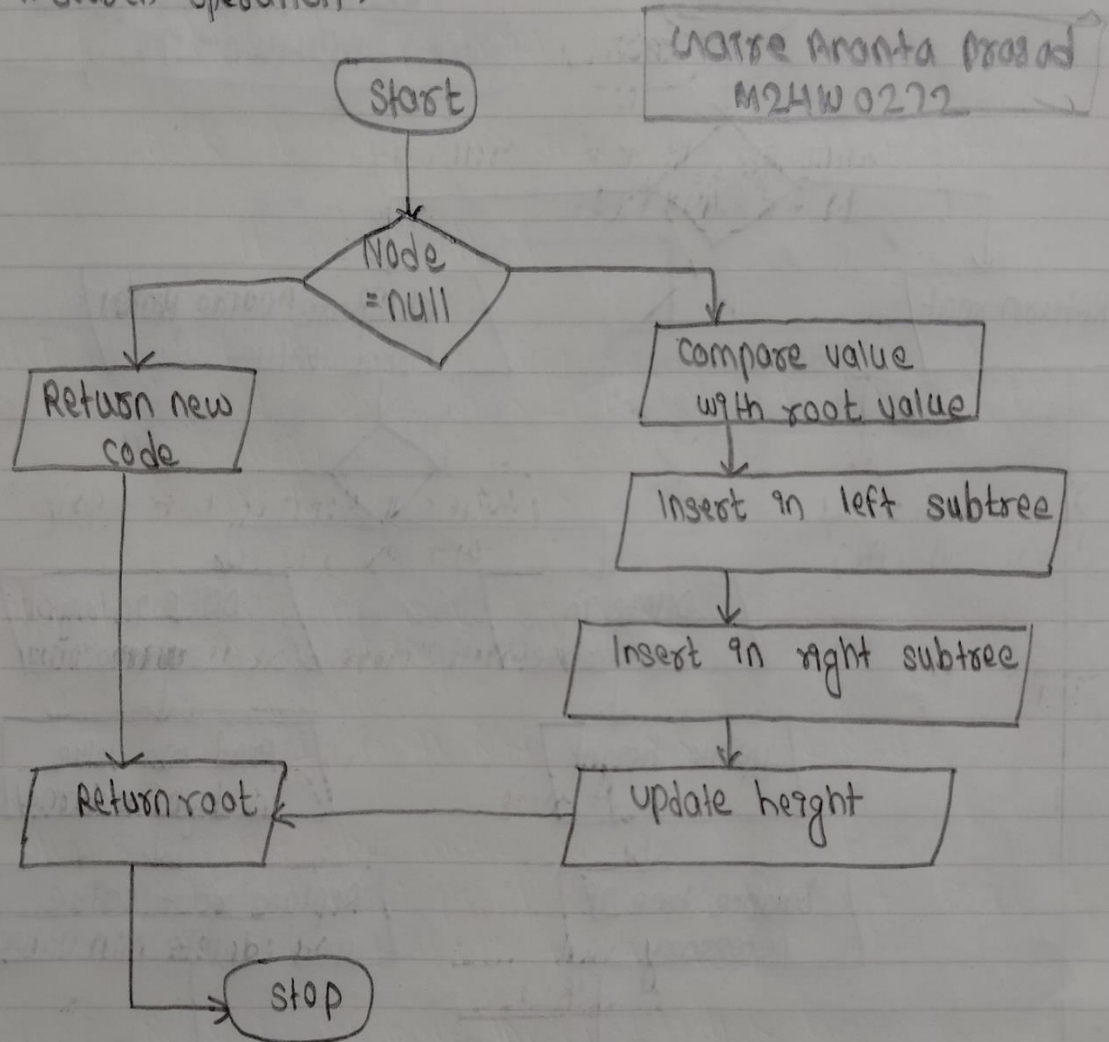
1. Binary Search Tree (BST) incorporating
2. Equilibrium Trees (AVL) incorporating rotations operations after



Gaire Ananta Prasad (M24W0272)

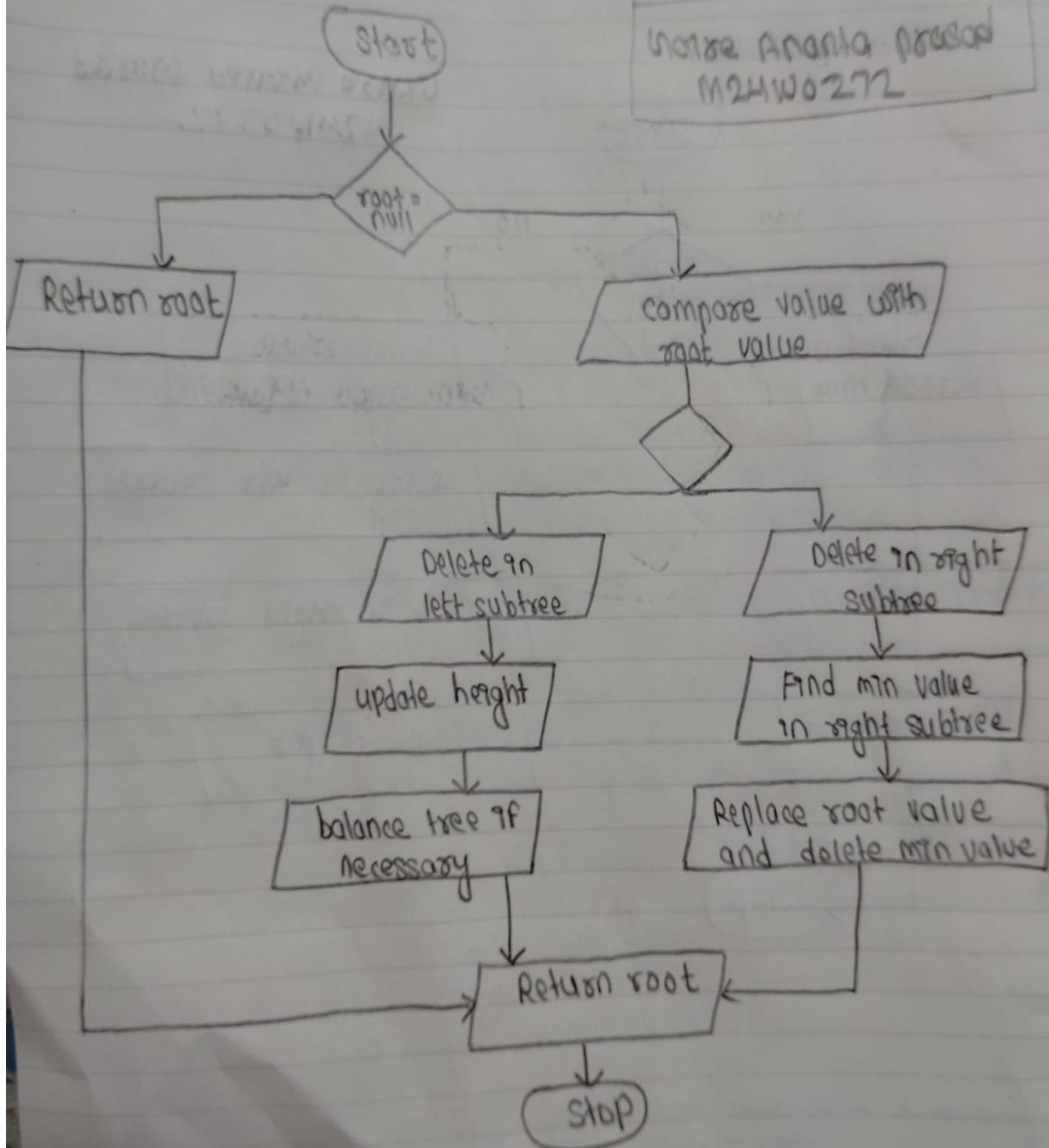
Binary search Tree (BST) Flowchart

* search operation :

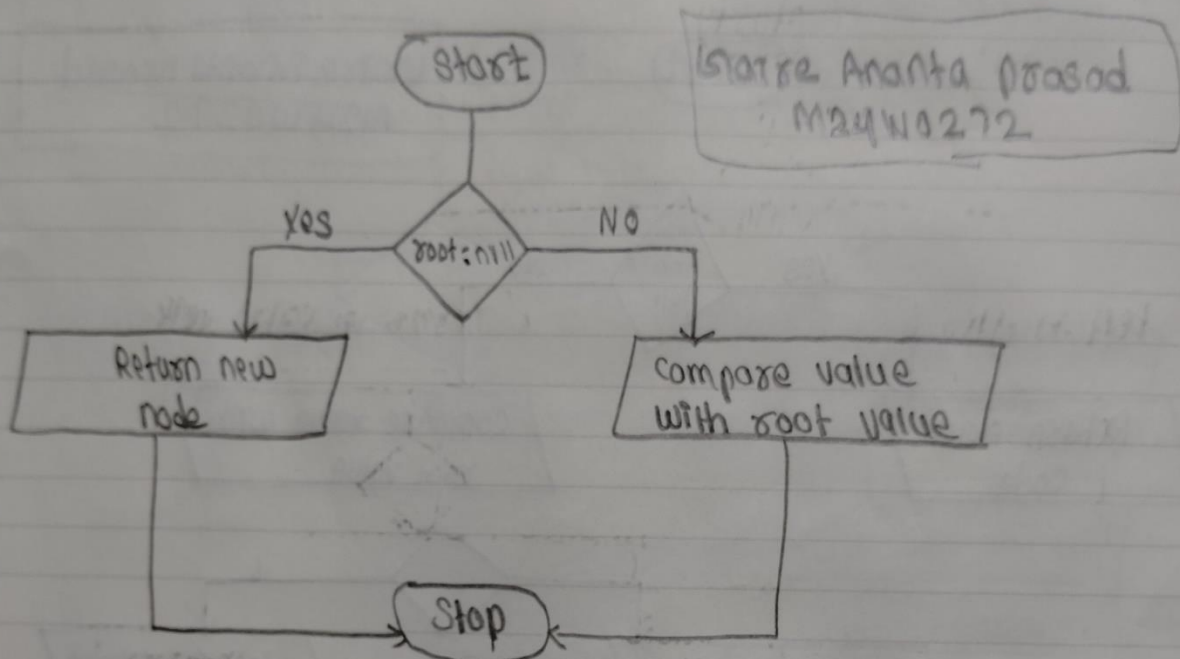


Binary delete operation flowchart

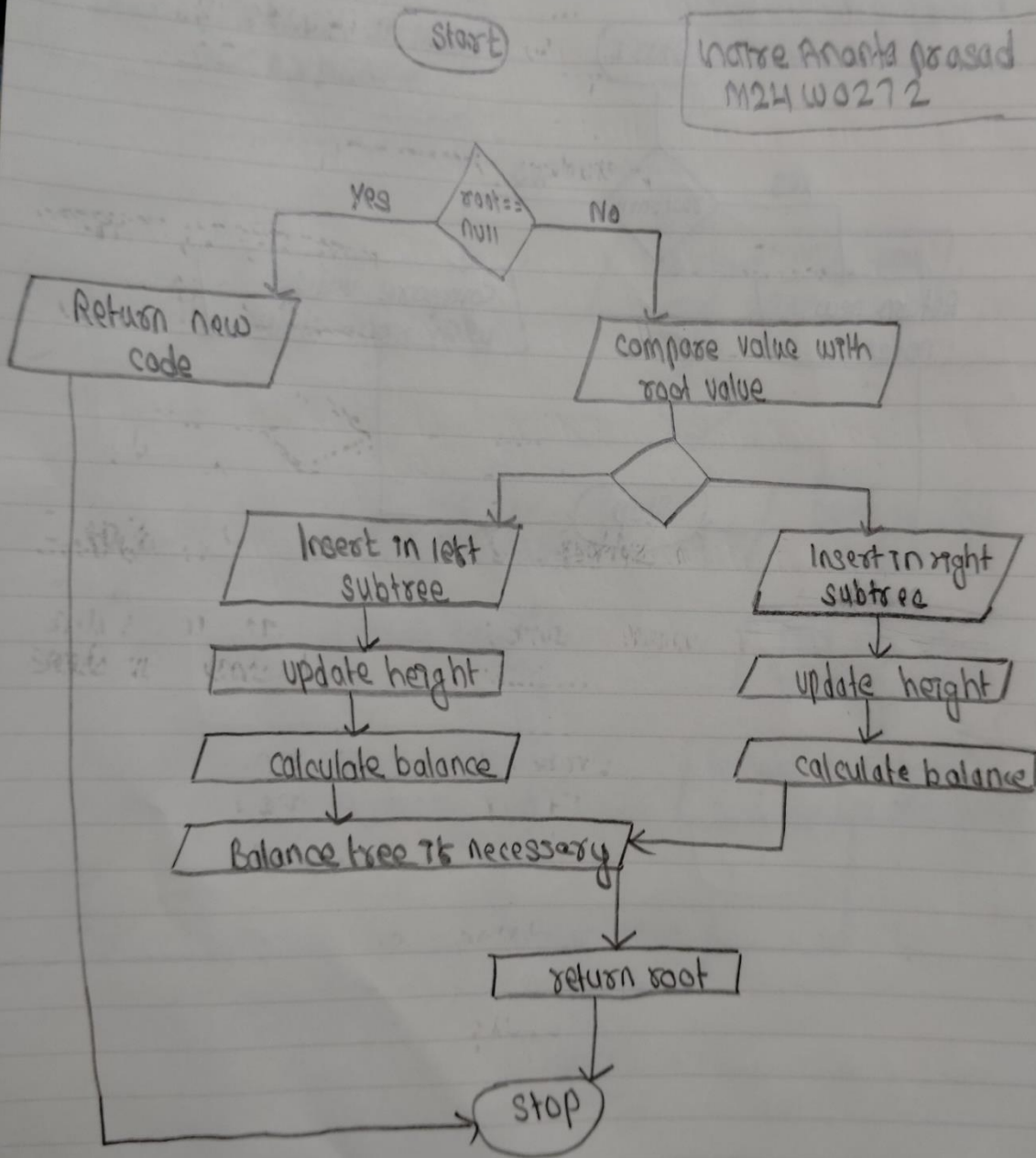
Gaire Ananta Prasad
M24W0272



Flowchart of Binary search tree Insert operation

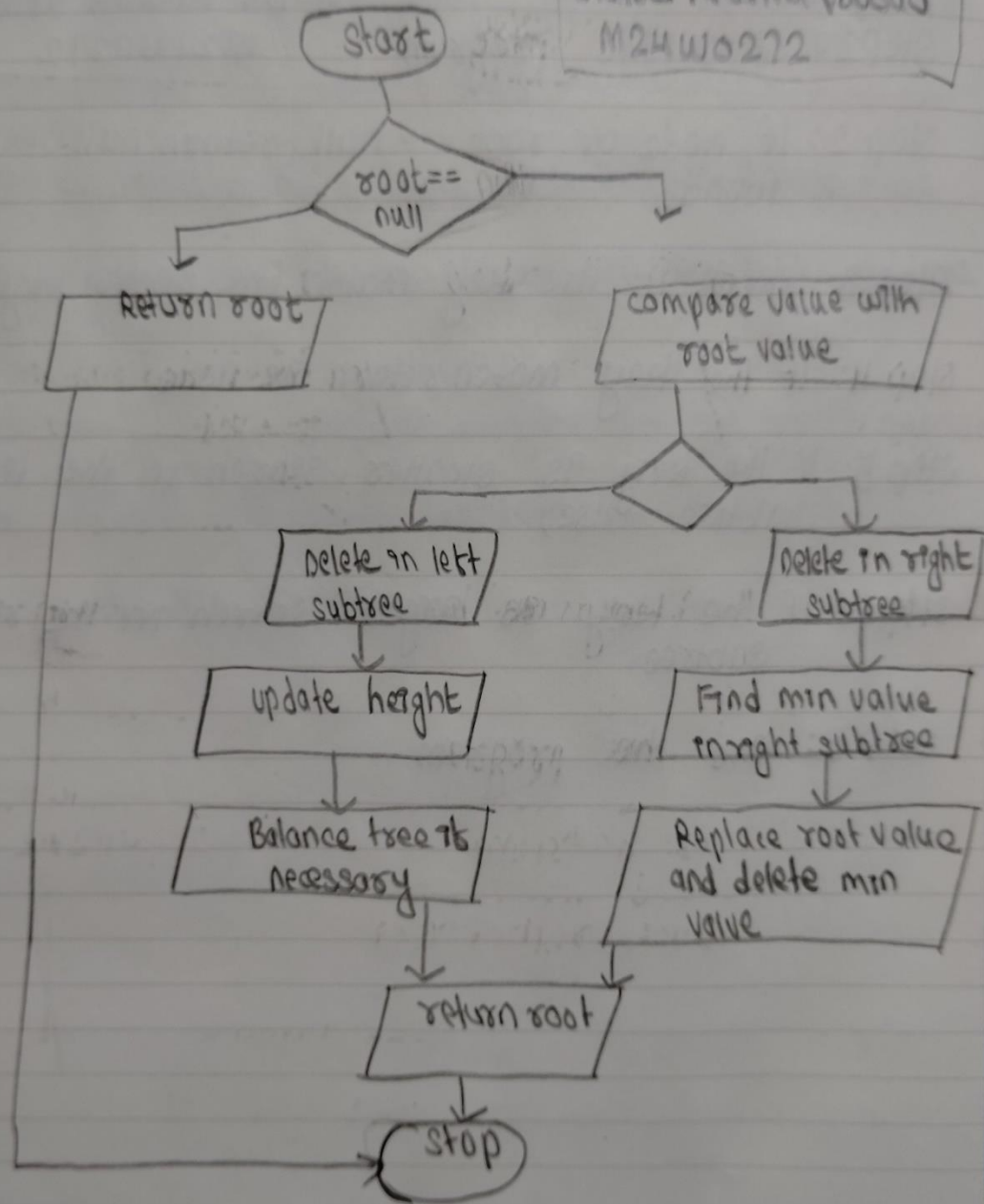


Flowchart of AVL Insert operation:



flow chart of delete operation of AVL

Gaire Ananta Prasad
M24W0272



Pseudocode of Binary search tree (search operation)

Step 1: Start the program

Gaire Ananta Prasad
M24W0272

Step 2: If Node or root == null, return null (key not found)

Step 3: Check if the key match the root's key

Step 4: If the key match return the node

Step 5: If the key is smaller, search in the left subtree else,

Step 6: If the key is larger, search in the right subtree

Step 7: end the program

pseudocode of BST insert operation)

step 1: start the program.

Gaire Ananta Prasad
M24W0272

step 2: check if root = Null

step 3: If root = null, create and return a new node.

step 4: If key is smaller, insert in the left subtree
else,

step 5: If key is larger, insert in the right subtree

step 6: Return the node pointer.

step 7: end the program

pseudocode of BST delete operation)

Step 1: start the program

Step 2: check if $root = null$

Step 3: If $root = null$, create and return a new node

Step 4: If key is smaller, insert in the left subtree

else,

Gaire Ananta Prasad

M24W0272

Step 5: If key is larger, insert in the right subtree

Step 6: Return the node pointer

Step 7: end the program.

Pseudocode of BST delete operation

step 1: start the program

step 2: check, Node or root is null

step 3: If root is null, return it (key not found)

step 4: If key is smaller, delete in the left subtree
else,

step 5: If key is larger, delete in the right subtree

step 6: If key match delete the node

step 7: Return the node pointer

step 8: end the program.

Gaire Ananta Prasad
M24W0272

Pseudocode of AVL Insert operations.

Step 1: start the program

Step 2: check, root is Null or not

Step 3: If root is Null, create and return a new node

Step 4: If the key is larger, insert in right subtree
else

Step 5: Insert in left subtree, if key is smaller

Step 6: update the height of this ancestor node.

Step 7: set the balance factor of this ancestor node

Step 8: If the node becomes unbalanced, then 4 cases

Case 1: left, left

Case 2: right, right

Case 3: left, right

Case 4: right, left

Gaire Ananta Prasad
M24W0272

Step 9: return the pointer (unchanged order)

Step 10: end the program.

pseudocode of AVL delete operation with rotation.

step 1: start the program.

step 2: check if Node is null or not

step 3: If Node is null return it (key not found)

step 4: If the key is smaller, delete the left subtree

else,

step 5: If the key is larger, delete the right subtree

step 6: If the key match, delete the node

case 1: If Node \rightarrow left == null

temp = node \rightarrow right

free (node)

return temp

Gaire Ananta Prasad
M24W0272

else,

case 2 If Node \rightarrow right == null

temp = node \rightarrow left

free (node)

return temp

step 7: Get the inorder successor (smallest in the right subtree)

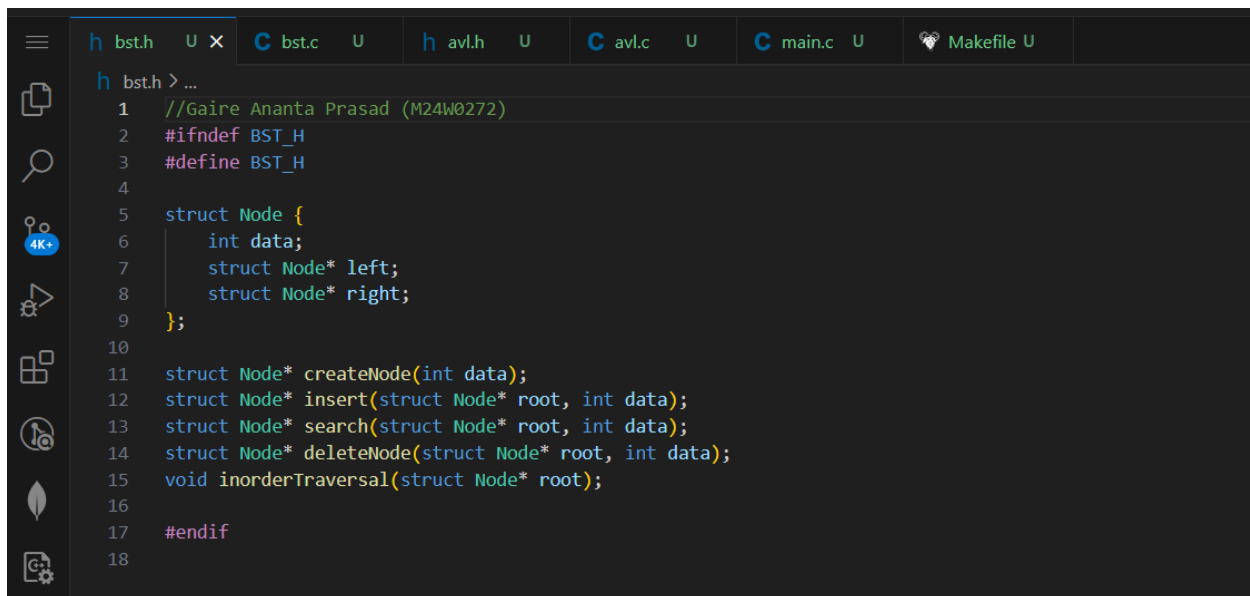
step 8: copy the inorder successor's key to this node.

step 9: Delete the inorder successor.

step 10: update the height of this ancestor node.

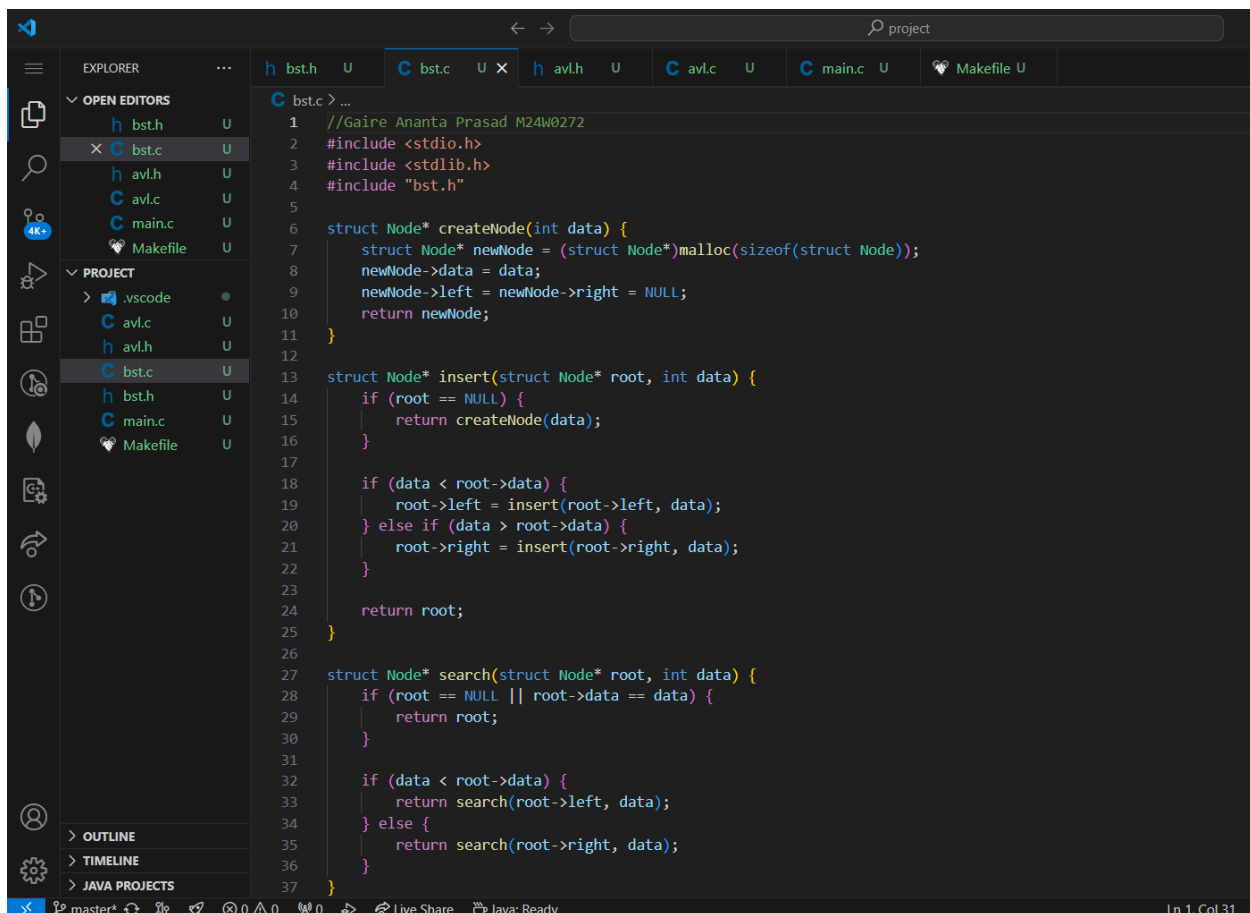
step 11: Get the balance factor.

step 12: end the program.



The screenshot shows a code editor with a single file named `bst.h` open. The file contains the following code:

```
1 //Gaire Ananta Prasad (M24W0272)
2 #ifndef BST_H
3 #define BST_H
4
5 struct Node {
6     int data;
7     struct Node* left;
8     struct Node* right;
9 };
10
11 struct Node* createNode(int data);
12 struct Node* insert(struct Node* root, int data);
13 struct Node* search(struct Node* root, int data);
14 struct Node* deleteNode(struct Node* root, int data);
15 void inorderTraversal(struct Node* root);
16
17 #endif
18
```



The screenshot shows a code editor with multiple files open. The `bst.c` file is the active editor, showing the following code:

```
1 //Gaire Ananta Prasad M24W0272
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "bst.h"
5
6 struct Node* createNode(int data) {
7     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
8     newNode->data = data;
9     newNode->left = newNode->right = NULL;
10    return newNode;
11 }
12
13 struct Node* insert(struct Node* root, int data) {
14     if (root == NULL) {
15         return createNode(data);
16     }
17
18     if (data < root->data) {
19         root->left = insert(root->left, data);
20     } else if (data > root->data) {
21         root->right = insert(root->right, data);
22     }
23
24     return root;
25 }
26
27 struct Node* search(struct Node* root, int data) {
28     if (root == NULL || root->data == data) {
29         return root;
30     }
31
32     if (data < root->data) {
33         return search(root->left, data);
34     } else {
35         return search(root->right, data);
36     }
37 }
```

The left sidebar shows the Explorer view with the following files listed:

- h bst.h
- C bst.c
- h avl.h
- C avl.c
- C main.c
- Makefile

The bottom status bar shows the current file is `bst.c` at line 1, column 31.

Gaire Ananta Prasad (M24W0272)


```

39 struct Node* minValueNode(struct Node* node) {
40     struct Node* current = node;
41     while (current && current->left != NULL) {
42         current = current->left;
43     }
44     return current;
45 }
46
47 struct Node* deleteNode(struct Node* root, int data) {
48     if (root == NULL) {
49         return root;
50     }
51
52     if (data < root->data) {
53         root->left = deleteNode(root->left, data);
54     } else if (data > root->data) {
55         root->right = deleteNode(root->right, data);
56     } else {
57         if (root->left == NULL) {
58             struct Node* temp = root->right;
59             free(root);
60             return temp;
61         } else if (root->right == NULL) {
62             struct Node* temp = root->left;
63             free(root);
64             return temp;
65         }
66
67         struct Node* temp = minValueNode(root->right);
68         root->data = temp->data;
69         root->right = deleteNode(root->right, temp->data);
70     }
71
72     return root;
73 }
74
75 void inorderTraversal(struct Node* root) {

```

```

73 }
74
75 void inorderTraversal(struct Node* root) {
76     if (root != NULL) {
77         inorderTraversal(root->left);
78         printf("%d ", root->data);
79         inorderTraversal(root->right);
80     }
81 }
82

```

This screenshot shows the Visual Studio Code editor with the `avl.h` header file open. The Explorer sidebar on the left shows the project structure with files `bst.h`, `bst.c`, `avl.h`, `avl.c`, `main.c`, and `Makefile`. The Editor pane displays the following code:

```
1 //Gaire Ananta Prasad M24W0272
2 #ifndef AVL_H
3 #define AVL_H
4
5 struct AVLNode {
6     int data;
7     struct AVLNode* left;
8     struct AVLNode* right;
9     int height;
10 };
11
12 struct AVLNode* createAVLNode(int data);
13 struct AVLNode* insertAVL(struct AVLNode* node, int data);
14 struct AVLNode* deleteAVLNode(struct AVLNode* root, int data);
15 void inorderTraversalAVL(struct AVLNode* root);
16
17 #endif
18
```

This screenshot shows the Visual Studio Code editor with the `avl.c` source file open. The Explorer sidebar on the left shows the project structure with files `bst.h`, `bst.c`, `avl.h`, `avl.c`, `main.c`, and `Makefile`. The Editor pane displays the following code:

```
1 //Gaire Ananta Prasad (M24W0272)
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "avl.h"
5
6 int max(int a, int b) {
7     return (a > b) ? a : b;
8 };
9
10 struct AVLNode* createAVLNode(int data) {
11     struct AVLNode* newNode = (struct AVLNode*)malloc(sizeof(struct AVLNode));
12     newNode->data = data;
13     newNode->left = newNode->right = NULL;
14     newNode->height = 1;
15     return newNode;
16 };
17
18 int height(struct AVLNode* node) {
19     if (node == NULL) {
20         return 0;
21     };
22     return node->height;
23 };
24
25 int getBalance(struct AVLNode* node) {
26     if (node == NULL) {
27         return 0;
28     };
29     return height(node->left) - height(node->right);
30 };
31
32 struct AVLNode* rightRotate(struct AVLNode* y) {
33     struct AVLNode* x = y->left;
34     struct AVLNode* T2 = x->right;
35
36     x->right = y;
37     y->left = T2;
38 }
39
```

Gaire Ananta Prasad (M24W0272)

The screenshot shows a Visual Studio Code editor window with a dark theme. The Explorer sidebar on the left lists files: bst.h, bst.c, avl.h, avl.c, main.c, and Makefile. The avl.c file is open in the editor, showing C code for AVL tree operations. The code includes functions for rightRotate, leftRotate, and insertAVL. The rightRotate function is at line 32, leftRotate at line 45, and insertAVL at line 58. The code uses a struct AVLNode and a height variable to maintain balance. The status bar at the bottom indicates 'Ln 1, Col 32'.

```
32 struct AVLNode* rightRotate(struct AVLNode* y) {
38
39     y->height = max(height(y->left), height(y->right)) + 1;
40     x->height = max(height(x->left), height(x->right)) + 1;
41
42     return x;
43 };
44
45 struct AVLNode* leftRotate(struct AVLNode* x) {
46     struct AVLNode* y = x->right;
47     struct AVLNode* T2 = y->left;
48
49     y->left = x;
50     x->right = T2;
51
52     x->height = max(height(x->left), height(x->right)) + 1;
53     y->height = max(height(y->left), height(y->right)) + 1;
54
55     return y;
56 };
57
58 struct AVLNode* insertAVL(struct AVLNode* node, int data) {
59     if (node == NULL) {
60         return createAVLNode(data);
61     }
62
63     if (data < node->data) {
64         node->left = insertAVL(node->left, data);
65     } else if (data > node->data) {
66         node->right = insertAVL(node->right, data);
67     } else {
68         return node;
69     }
70
71     node->height = 1 + max(height(node->left), height(node->right));
72     int balance = getBalance(node);
73 }
```

```

174 struct AVLNode* insertAVL(struct AVLNode* node, int data) {
175     if (balance > 1 && data < node->left->data) {
176         return rightRotate(node);
177     }
178     if (balance < -1 && data > node->right->data) {
179         return leftRotate(node);
180     }
181     if (balance > 1 && data > node->left->data) {
182         node->left = leftRotate(node->left);
183         return rightRotate(node);
184     }
185     if (balance < -1 && data < node->right->data) {
186         node->right = rightRotate(node->right);
187         return leftRotate(node);
188     }
189     return node;
190 };
191
192 struct AVLNode* minValueNode(struct AVLNode* node) {
193     struct AVLNode* current = node;
194     while (current && current->left != NULL) {
195         current = current->left;
196     }
197     return current;
198 }
199
200 struct AVLNode* deleteAVLNode(struct AVLNode* root, int data) {
201     if (root == NULL) {
202         return root;
203     }
204     if (data < root->data) {
205         root->left = deleteAVLNode(root->left, data);

```

```

206 struct AVLNode* deleteAVLNode(struct AVLNode* root, int data) {
207     root->left = deleteAVLNode(root->left, data);
208 } else if (data > root->data) {
209     root->right = deleteAVLNode(root->right, data);
210 } else {
211     if ((root->left == NULL) || (root->right == NULL)) {
212         struct AVLNode* temp = root->left ? root->left : root->right;
213         if (temp == NULL) {
214             temp = root;
215             root = NULL;
216         } else {
217             *root = *temp;
218         }
219         free(temp);
220     } else {
221         struct AVLNode* temp = minValueNode(root->right);
222         root->data = temp->data;
223         root->right = deleteAVLNode(root->right, temp->data);
224     }
225 }
226
227 if (root == NULL) {
228     return root;
229 }
230
231 root->height = 1 + max(height(root->left), height(root->right));
232 int balance = getBalance(root);
233
234 if (balance > 1 && getBalance(root->left) >= 0) {
235     return rightRotate(root);
236 }
237
238 if (balance > 1 && getBalance(root->left) < 0) {
239     root->left = leftRotate(root->left);
240     return rightRotate(root);
241 }

```

```

103 struct AVLNode* deleteAVLNode(struct AVLNode* root, int data) {
144     }
145
146     if (balance < -1 && getBalance(root->right) <= 0) {
147         return leftRotate(root);
148     }
149
150     if (balance < -1 && getBalance(root->right) > 0) {
151         root->right = rightRotate(root->right);
152         return leftRotate(root);
153     }
154
155     return root;
156 }
157
158 void inorderTraversalAVL(struct AVLNode* root) {
159     if (root != NULL) {
160         inorderTraversalAVL(root->left);
161         printf("%d ", root->data);
162         inorderTraversalAVL(root->right);
163     }
164 }
165

```

```

1 //Gaire Ananta Prasad (M24W0272)
2 #include <stdio.h>
3 #include "bst.h"
4 #include "avl.h"
5
6 int main() {
7     struct Node* root = NULL;
8     root = insert(root, 50);
9     root = insert(root, 30);
10    root = insert(root, 20);
11    root = insert(root, 40);
12    root = insert(root, 70);
13    root = insert(root, 60);
14    root = insert(root, 80);
15
16    printf("BST Inorder Traversal: ");
17    inorderTraversal(root);
18    printf("\n");
19
20    root = deleteNode(root, 20);
21    printf("BST Inorder Traversal after deleting 20: ");
22    inorderTraversal(root);
23    printf("\n");
24
25    root = deleteNode(root, 30);
26    printf("BST Inorder Traversal after deleting 30: ");
27    inorderTraversal(root);
28    printf("\n");
29
30    root = deleteNode(root, 50);
31    printf("BST Inorder Traversal after deleting 50: ");
32    inorderTraversal(root);
33    printf("\n");
34
35    // AVL Tree
36    struct AVLNode* avlRoot = NULL;
37    avlRoot = insertAVL(avlRoot, 50);

```

Gaire Ananta Prasad (M24W0272)


```
main.c > ...
6  int main() {
37     avlRoot = insertAVL(avlRoot, 50);
38     avlRoot = insertAVL(avlRoot, 30);
39     avlRoot = insertAVL(avlRoot, 20);
40     avlRoot = insertAVL(avlRoot, 40);
41     avlRoot = insertAVL(avlRoot, 70);
42     avlRoot = insertAVL(avlRoot, 60);
43     avlRoot = insertAVL(avlRoot, 80);
44
45     printf("AVL Tree Inorder Traversal: ");
46     inorderTraversalAVL(avlRoot);
47     printf("\n");
48
49     avlRoot = deleteAVLNode(avlRoot, 20);
50     printf("AVL Tree Inorder Traversal after deleting 20: ");
51     inorderTraversalAVL(avlRoot);
52     printf("\n");
53
54     avlRoot = deleteAVLNode(avlRoot, 30);
55     printf("AVL Tree Inorder Traversal after deleting 30: ");
56     inorderTraversalAVL(avlRoot);
57     printf("\n");
58
59     avlRoot = deleteAVLNode(avlRoot, 50);
60     printf("AVL Tree Inorder Traversal after deleting 50: ");
61     inorderTraversalAVL(avlRoot);
62     printf("\n");
63
64     return 0;
65 }
```

```
Makefile
1  # Gaire Ananta Prasad (M24W0272)
2  CC = gcc
3  CFLAGS = -Wall -Wextra
4
5  all: bst_avl
6
7  bst_avl: main.o bst.o avl.o
8      $(CC) $(CFLAGS) -o bst_avl main.o bst.o avl.o
9
10 main.o: main.c bst.h avl.h
11     $(CC) $(CFLAGS) -c main.c
12
13 bst.o: bst.c bst.h
14     $(CC) $(CFLAGS) -c bst.c
15
16 avl.o: avl.c avl.h
17     $(CC) $(CFLAGS) -c avl.c
18
19 clean:
20     rm -f *.o bst_avl
21
```

Gaire Ananta Prasad (M24W0272)