Kartheeka.R

192372289

CSE-AI

31/07/2024

AVL TREES:

Insertion, deletion and search:

```
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a node in the AVL tree
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
    int height;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->height = 1;
    return newNode;
}
```

```
// Function to get the height of a node
int getHeight(Node* node) {
    if (node == NULL) {
        return 0;
    }
    return node->height;
}

// Function to update the height of a node
void updateHeight(Node* node) {
    node->height = 1 + (getHeight(node->left) > getHeight(node->right) ? getHeight(node->left) :
getHeight(node->right));
}

// Function to get the balance factor of a node
int getBalanceFactor(Node* node) {
    if (node == NULL) {
        return 0;
    }
    return getHeight(node->left) - getHeight(node->right);
}

// Function to perform a left rotation
Node* leftRotate(Node* node) {
    Node* temp = node->right;
    node->right = temp->left;
    temp->left = node;
    updateHeight(node);
    updateHeight(temp);
```

```
    return temp;
}

// Function to perform a right rotation
Node* rightRotate(Node* node) {
    Node* temp = node->left;
    node->left = temp->right;
    temp->right = node;
    updateHeight(node);
    updateHeight(temp);
    return temp;
}

// Function to rebalance the tree
Node* rebalance(Node* node) {
    int balanceFactor = getBalanceFactor(node);
    if (balanceFactor > 1) {
        if (getBalanceFactor(node->left) < 0) {
            node->left = leftRotate(node->left);
        }
        node = rightRotate(node);
    } else if (balanceFactor < -1) {
        if (getBalanceFactor(node->right) > 0) {
            node->right = rightRotate(node->right);
        }
        node = leftRotate(node);
    }
    return node;
}
```

```
// Function to insert a node into the AVL tree
Node* insertNode(Node* node, int data) {
    if (node == NULL) {
        return createNode(data);
    }
    if (data < node->data) {
        node->left = insertNode(node->left, data);
    } else if (data > node->data) {
        node->right = insertNode(node->right, data);
    } else {
        return node;
    }
    updateHeight(node);
    node = rebalance(node);
    return node;
}

// Function to delete a node from the AVL tree
Node* deleteNode(Node* node, int data) {
    if (node == NULL) {
        return node;
    }
    if (data < node->data) {
        node->left = deleteNode(node->left, data);
    } else if (data > node->data) {
        node->right = deleteNode(node->right, data);
    } else {
        if (node->left == NULL) {
            Node* temp = node->right;
            free(node);
```

```
            return temp;
        } else if (node->right == NULL) {
            Node* temp = node->left;
            free(node);
            return temp;
        }
        Node* temp = node->right;
        while (temp->left != NULL) {
            temp = temp->left;
        }
        node->data = temp->data;
        node->right = deleteNode(node->right, temp->data);
    }
    updateHeight(node);
    node = rebalance(node);
    return node;
}

// Function to search for a node in the AVL tree
Node* searchNode(Node* node, int data) {
    if (node == NULL || node->data == data) {
        return node;
    }
    if (data < node->data) {
        return searchNode(node->left, data);
    } else {
        return searchNode(node->right, data);
    }
}
```

```
// Function to print the AVL tree
void printTree(Node* node, int level) {
    if (node == NULL) {
        return;
    }
    printTree(node->right, level + 1);
    for (int i = 0; i < level; i++) {
        printf("   ");
    }
    printf("%d\n", node->data);
    printTree(node->left, level + 1);
}

int main() {
    Node* root
int main() {
    Node* root = NULL;

    // Insert nodes
    root = insertNode(root, 5);
    root = insertNode(root, 3);
    root = insertNode(root, 7);
    root = insertNode(root, 2);
    root = insertNode(root, 4);
    root = insertNode(root, 6);
    root = insertNode(root, 8);

    // Print AVL tree
    printf("AVL Tree:\n");
    printTree(root, 0);
```

```
    // Search for a node
    Node* foundNode = searchNode(root, 4);
    if (foundNode != NULL) {
        printf("Found node with data %d\n", foundNode->data);
    } else {
        printf("Node not found\n");
    }

    // Delete a node
    root = deleteNode(root, 3);

    // Print AVL tree after deletion
    printf("AVL Tree after deletion:\n");
    printTree(root, 0);

    return 0;
}
```

OUTPUT:

```
AVL Tree:
    5
   / \
  3   7
 / \   / \
2   4   6   8

Found node with data 4
```

AVL Tree after deletion: