Ex : 7

Title: Develop a C program to implement height balanced tree.

Problem Description: Given an imbalanced tree, restructure itas a balanced tree.

Method: Make use of AVL tree.

Ensure that the program performs proper rotations such as Left-Left, LR, RR and RL to create a height balanced binary search tree and display the tree created as a output .

Theory Reference: Module 4

Explanation:

1. Struct Definition (struct AVL)

   - int key: The value of the node.
   - struct AVL *left: Pointer to the left child node.
   - struct AVL *right: Pointer to the right child node.
   - int height: The height of the node, which is used to maintain the balance of the AVL tree.

2. Function: getHeight(node *n)

   - Returns the height of the node n.

3. Function: createNode(int key)

   - Allocates memory for a new node and initializes its values.

4. Function: max(int a, int b)

   - Returns the maximum of two integers.

5. Function: getBalanceFactor(node *n)

   - Computes the balance factor of a node.

6. Function: rightRotate(node *y)

   - Performs a right rotation around node y.

7. Function: leftRotate(node *x)

- Performs a left rotation around node x.

8. Function: insert(node *n, int key)

- Inserts a new key into the AVL tree while maintaining its balance.

9. Function: printtree(node *root, int space, int n)

- Prints the AVL tree in a structured format for visualization.

Algorithm:

Step 1. getHeight(node *n)

- Input: A pointer to a node n.
- If n is NULL, return 0.
- Otherwise, return the height of the node (n->height).

Step 2. createNode(int key)

- Input: An integer key.
- Allocate memory for a new node.
- Set the node's key to the given key.
- Initialize the left and right pointers to NULL.
- Set the height to 1 (new node).
- Return the pointer to the newly created node.

Step 3. max(int a, int b)

- Input: Two integers a and b.
- Return the larger of the two values.

Step 4. getBalanceFactor(node *n)

- Input: A pointer to a node n.
- If n is NULL, return 0.
- Calculate the balance factor as the height of the left subtree minus the height of the right subtree (getHeight(n->left) - getHeight(n->right)).
- Return the balance factor.

Step 5. rightRotate(node *y)

- Input: A pointer to a node y (the root of the subtree).
- Set x to the left child of y.
- Set T2 to the right child of x.
- Perform the rotation:
    - Set x->right to y.
    - Set y->left to T2.
- Update the heights of x and y.
- Return x (new root of the subtree).

Step 6. leftRotate(node *x)

- Input: A pointer to a node x (the root of the subtree).
- Set y to the right child of x.
- Set T2 to the left child of y.
- Perform the rotation:
    - Set y->left to x.
    - Set x->right to T2.
- Update the heights of y and x.
- Return y (new root of the subtree).

Step 7. insert(node *n, int key)

- Input: A pointer to a node n and an integer key.
- If n is NULL, create and return a new node with the given key.
- If key is less than n->key, recursively insert in the left subtree.
- If key is greater than n->key, recursively insert in the right subtree.
- Update the height of n.
- Calculate the balance factor of n.
- Check for and perform necessary rotations based on the balance factor:
    - **Left Left Case**: Perform right rotation.
    - **Right Right Case**: Perform left rotation.
    - **Left Right Case**: Perform left rotation on the left child, then right rotation on n.
    - **Right Left Case**: Perform right rotation on the right child, then left rotation on n.
- Return the root n.

Step 8. printtree(node *root, int space, int n)

- Input: A pointer to the root node, an integer space, and an integer n.
- If root is NULL, return.
- Increment space by n.
- Recursively print the right subtree.

- Print the current node's key with appropriate indentation based on space.
- Recursively print the left subtree.

Step 9. main()

- Declare a pointer root initialized to NULL.
- Read the number of nodes n from user input.
- Loop n times:
  - Read an integer key from user input.
  - Insert the key into the AVL tree by calling insert(root, key).
- Print the tree structure by calling printtree(root, 0, n).
- Return 0.