ASSIGNMENT 11

Name: Aniket Kalbhor

Roll: 48

PRN: 12210601

AVL

#include <stdio.h>

#include <stdlib.h>

struct Node

{

    int data;

    struct Node\* left;

    struct Node\* right;

    int height;

};

int getHeight(struct Node\* root)

{

    if (root == NULL)

    {

        return 0;

    }

    return root->height;

}

int getMax(int a, int b)

{

    return(a > b)? a : b;

}

struct Node\* createNode(int key)

{

    struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

    node->data = key;

    node->left = NULL;

    node->right = NULL;

    node->height = 1;

    return(node);

};

struct Node\* rightRotate(struct Node\* y)

{

    struct Node\* x = y->left;

    struct Node\* t2 = x->right;

    x->right = y;

    y->left = t2;

    y->height = getMax(getHeight(y->left), getHeight(y->right)) + 1;

    x->height = getMax(getHeight(x->left), getHeight(x->right)) + 1;

    return x;

};

struct Node\* leftRotate(struct Node\* x)

{

    struct Node\* y = x->right;

    struct Node\* t2 = y->left;

    y->left = x;

    x->right = t2;

    x->height = getMax(getHeight(x->left), getHeight(x->right)) + 1;

    y->height = getMax(getHeight(y->left), getHeight(y->right)) + 1;

    return y;

};

int getBalanceFactor(struct Node\* root)

{

    if (root == NULL)

    {

        return 0;

    }

    return getHeight(root->left) - getHeight(root->right);

}

struct Node\* insertNode(struct Node\* root, int key)

{

    if (root == NULL)

    {

        root = createNode(key);

    }

    if (key < root->data)

    {

        root->left = insertNode(root->left, key);

    }

    else if (key > root->data)

    {

        root->right = insertNode(root->right, key);

    }

    else

    {

        return root;

    }

    root->height = 1 + getMax(getHeight(root->left), getHeight(root->right));

    int balance = getBalanceFactor(root);

    if (balance > 1 && key < root->left->data)

    {

        return rightRotate(root);

    }

    if (balance < -1 && key > root->right->data)

    {

        return leftRotate(root);

    }

    if (balance > 1 && key > root->left->data)

    {

        root->left = leftRotate(root->left);

        return rightRotate(root);

    }

    if (balance < -1 && key < root->right->data)

    {

        root->right = rightRotate(root->right);

        return leftRotate(root);

    }

    return root;

};

void preOrderTraversal(struct Node \*root)

{

    if(root != NULL)

    {

        printf("%d ", root->data);

        preOrderTraversal(root->left);

        preOrderTraversal(root->right);

    }

}

void inOrderTraversal(struct Node \*root)

{

    if(root != NULL)

    {

        inOrderTraversal(root->left);

        printf("%d ", root->data);

        inOrderTraversal(root->right);

    }

}

void postOrderTraversal(struct Node \*root)

{

    if(root != NULL)

    {

        postOrderTraversal(root->left);

        postOrderTraversal(root->right);

        printf("%d ", root->data);

    }

}

int main()

{

  struct Node\* root = NULL;

  int n;

  while (1)

  {

      printf("\n");

      printf("1. Insert \t2. Traversal\n");

      int choice;

      printf("Choice:");

      scanf("%d",&choice);

      switch(choice)

      {

      case 1:

        printf("Enter value to insert:");

        scanf("%d",&n);

        root = insertNode(root, n);

        break;

      case 2:

        while (1)

        {

            printf("\n");

            printf("1. Inorder Traversal\t2. Preorder Traversal\t3. Postorder Traversal\t4. Exit\n");

            int choice;

            printf("Choice:");

            scanf("%d",&choice);

            switch(choice)

            {

                case 1:

                    inOrderTraversal(root);

                    break;

                case 2:

                    preOrderTraversal(root);

                    break;

                case 3:

                    postOrderTraversal(root);

                    break;

                case 4:

                    exit(0);

            }

        }

        break;

        exit(0);

      }

  }

  return 0;

}

