ADA Lab 5

ADITH RAVI PAI
1BM18CS005

papergrid
Date: / /

Program to Insert and delete a node from AVL Tree

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{ int key;
  struct Node *left;
  struct Node *right;
  int height;
};
int max (int a, int b) { return (a>b) ? a :b ; }
int height (struct Node *N)
{ if (N == NULL)
    return 0;
  return N->height;
}
struct Node* newNode (int key)
{ struct Node *node = (struct Node*) malloc (sizeof (struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1; // new node initially added at leaf.
  return(node);
}
struct Node *rightRotate (struct Node *y)
{ struct Node *x = y->left;
  struct Node *T2 = x->right;
    // perform rotation
      x->right = y;
      y->left = T2;
    // update heights
    x->height = max(height (x->left), height (x->right))+1;
    y->height = max (height (y->left), height (y->right))+1;
    return y; // return new root
}
```

papergrid
Date: / /

```
int getBalance (struct Node *N)
{   if (N == NULL)
        return 0;
    return height (N -> left) - height (N -> right);
}

struct Node* insert (struct Node* node, int key)
{
    if (node == NULL)
        return (newNode (key));
    if (key < node -> key)
        node -> left = insert (node -> left, key);
    else if (key > node -> key)
        node -> right = insert (node -> right, key);
    else
        return node;
    node -> height = 1 + max (height (node -> left),
                              height (node -> right));

    int balance = getBalance (node);
// if this node becomes unbalanced, there are 4 cases,
    // left left case
    if (balance > 1 && key < node -> left -> key)
        return rightRotate (node);
    // right right case
    if (balance < -1 && key > node -> right -> key)
        return leftRotate (node);
    // left Right case
    if (balance > 1 && key > node -> left -> key)
    {   node -> left = leftRotate (node -> left);
        return rightRotate (node);
    }
```

papergrid
Date: / /

```
// right left case
if (balance < -1 && key < node->right->key)
{ node->right = rightRotate(node->right);
  return leftRotate(node);
}

return node;

struct Node* minValueNode(struct Node* node)
{

struct Node* deleteNode(struct Node* root, int key)
{ if (root == NULL)  // STEP 1 : Standard BST
    return root;                          delete
  if (key < root->key)
      root->left = deleteNode(root->left, key);
  else
  { if ((root->left == NULL) || (root->right == NULL))
    { struct Node *temp = root->left ? root->left :
                                       root->right;

      // No child case
      if (temp == NULL)
      { temp = root;
        root = NULL;
      }

      else
      *root = *temp;
       free(temp);
    }

    else { struct Node* temp = minValueNode
// Copy inorder successor's data to this node.   (root->right);
      root->key = temp->key;
      // delete the inorder successor
      root->right = deleteNode
                    (root->right, temp->key);
    }
```

papergrid

Date: / /

```
// if the tree had only one node then return
if (root == NULL)
    return root;
//STEP 2: update height of current node
root -> height = 1 + max(height(root -> left),
                         height(root -> right));
//STEP 3: Get the balance factor of this node (to check if
This node has become unbalanced).
int balance = getBalance(root);
// if ~~node~~ this node becomes unbalanced,
    // left left case
if (balance > 1 && getBalance(root -> left) >= 0)
    return rightRotate(root);
    // left right case
if (balance > 1 && getBalance(root -> left) < 0)
{   root -> left = leftRotate(root -> left);
    return rightRotate(root);
    // right right case
if (balance < -1 && getBalance(root -> right) <= 0)
    return leftRotate(root);
    //right left case
if (balance < -1 && getBalance(root -> right);
{   return leftRotate(root); }
    return root;
}
```