KARTHIK VENUGOPAL

1BM18CS043

14/10/2020

AVL tree
write up

~~Pseudocode~~

~~class Node~~
~~{~~
~~key~~

Pseudocode

```
class Node
{
    key, *left, *right
    height
}
```

```
height (Node N)
{
    if(N==NULL)
        return 0;
    return N->height;
}
```

```
rightrotate ( Node *y)
{
    *x = y -> left.
    *T2 = x -> right,
    x -> right = y;
    y -> left = T2.

    y -> height = height (y->left) > height (y -> right)
                  ? height(y->left)+1
                  : height (y -> right)+1

    x -> height = height (x -> left) > height(x-> right)
                  ? height (x -> left)+1 : height(x->righ
                  +1

    return x.
}
```

```
leftrotate (Node *x)
{
    *y  = x -> right;
    *T2 = y -> left;

      y -> left = x;
      x -> right = T2;

      x -> height = max (height (x -> left),
                         height (x -> right)) + 1
      y -> height  = max (height (y -> left),
                         height (y -> right)) + 1;

      return y;
   }


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


Node * insert ( node , 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;
```

```
int balance = getbalance(node);

if( balance >1 && key < node -> ~~right~~ left -> key )
    return rightrotate(node)

if (balance <-1 && key > node -> right-> key)
    return ~~lt~~ leftrotate(node);

if (balance >1 && key > node -> left -> key)
{
    node -> left = leftrotate (node -> left),
    return rightrotate(node),
}

if(balance <-1 && key < node -> right-> key)
{
    node -> right = rightrotate (node -> right),
    return leftrotate(node);
}
    return node;
}

deletenode ( root , key)
{
    if ( root == NULL)
        return root;

    if ( key < root -> key)
        root -> ~~lef~~ left = deletenode(root -> left,
                                          key ),

    else if ( key > root -> key)
        root -> right = deletenode( root -> right,
                                    key),
else
{
    if ((root -> left == NULL) ||
        (root -> right == NULL))
    {
      Node * temp = root -> left ? root -> left :
                    root -> right;

      if (temp == NULL)
      {
         temp = root;
         root = NULL;
       }
      else
         root = *temp;
         free(temp);
     }
     else
     {
        Node * temp = minValueNode(root -> right)
        root -> key = temp -> key;
        root -> right = deleteNode(root -> right,
                        temp -> key);
      }
  }
  if (root == NULL)
     return root;

  root -> height = 1 + max(height (root -> left)
                   height (root -> right));

  int balance = getBalance (root);
```

```
if (balance > 1 && getbalance(root -> left)
{
    root -> left = ~~te~~ leftrotate(root -> left)
    return rightrotate(root)
}

if (balance > 1 && getbalance

if (balance > 1 && getbalance(root -> left)
                    >= 0)
    return rightrotate(root);

if (balance > 1 && getbalance(root -> left) <
{
    root -> left = leftrotate(root -> left)
    return rightrotate(root)
}
if (balance < -1 && getbalance(root
                    -> right) > 0)
{
    root -> right = rightrotate(root
                    -> right)
    return leftrotate(root)
}
return root;
}
```