Chandan C Bagas
IBM18CS026

ADS Lab 4 coritecup

```
class node{
  public:
    int data;
    node * left;
    node * right;
    int height;
};

int heightfun(node* p){
  if (p == NULL)
    return 0;
  return p->height;
} // height is returned

node * getnode(int data){ // Function to create node
  node * p = new node();
  p->data = data;
  p->left = NULL;
  p->right = NULL;
  p->height = 1
  return p;
}

node rotateright(node *b){ // Rotate right function

  node * a = b->left;
  node * t = b->right;
  a->right = b;
  b->left = t;
  b->height = max(height(b->left),
                  height(b->right))+1;
  a->height = max(height(a->left),
                  height(a->right))+1;

  return a;
}
```

```
node * rotateleft (node *a) { // rotate left function
    node *b = a->right;
    node *t = a->left;
    b->left = a;
    a->right = b;
    a->height = max (heightfun(a->right),
                     height(a->left))+1;
    b->height = max (heightfun (b->left), heightfun(b->
    return b;
}

int balance ( node *p) { // Balance factor of node
    if (p==NULL)
        return 0;
    return heightfun(p->left) - heightfun (p->right);
}

node * insertion (node *root, int data) {
    if (root==NULL)
        return getnode (data);
    if (data<root->data)
        root->left = insertion (root->left, data);
    else if (data>root->data)
        root->right = insertion (root->right, data);
    else
        return root;
    root->height = max (height (root->left),
                        height (root->right))+1;
```

```
int bl = balance(root)
  if (bl>1 && data < root->left->data)
      root->left = rotateleft(root->left);
      return rotateright(root);
}
if (bl<-1 && data > root->right->data)
      return rotateleft(root);

if (bl>1 && data > root->left->data) {
      root->left = rotateleft(root->left);
      return rotateright(root);

if (bl<-1 && data < root->left->data) {
      root->right = rotateright(root->right);
      return rotateleft(root);
  }
return root;
}

Node * deletion(node * root, int item) {
    if (root == NULL)
        return root;
    if (item < root->data)
        root->left = deletion(root->left, item);

    else if (item > root->data)
        root->right = deletion(root->right, item);

    else {
        if (root->left == NULL || root->right == NULL)
        {
            node * t = root->left ? root->left : root->right
```

```
if (t == NULL) {
    t = root;
    root = NULL;
}
else
{
    *root = *temp;
    free(temp);
}
else {
    node *t = minvalue(root->right);
    root->data = t->data;
    root->right = deletion(root->right, t->data);
}
if (root == NULL)
    return root;
root->height = max(height(root->left),
                   height(root->right));

int bl = balance(root);
if (bl > 1 && balance(root->left) >= 0)
    return rotateleft(root);

if (bl < -1 && balance(root->right) <= 0)
    return rotateleft(root);

if (bl > 1 && balance(root->left) < 0) {
    root->left = rotateleft(root->left);
    return rotateright(root);
}
if (bl < -1 && balance(root->right) > 0) {
    root->right = rotateright(root->right);
    return returnleft(root);
}
return root;
}
```