14/10/2020
Wednesday

Advanced Data Structures

Arjun. A. S
1BM18CS019

Week-4 Writeup.

AVL - Trees.

```
// AVL Tree node
class Node
{
        public:
            int data;
            Node * left;
            Node * right;
            int height;

};
class AVL
{
    Node * getnode(int data)
    {
            node *p = new Node();
            p -> data = data;
            p -> left = NULL;
            p -> right = NULL;
            p -> height = 1
            return p;
    }
    Node *rotateleft (Node * a)
    {       Node * b = a -> right;
            Node * s = a -> left;
              b -> left = a;
              a -> right = s;
            a -> height = max(height (a -> right), height (a -> left)) + 1;
            b -> height = max (height (b -> left), height (b -> right) + 1;

            return b;
    }
```

Arjun. A. S
1BM18CS019

```
Node * rotateright (Node *b)
{
    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;
}

int balance_factor (Node *p)
{
    if (p == NULL)
        return 0;
    return height(p->left) - height(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 fact = balance_factor(root)
    if (fact > 1 && data < root->left->data)
        return rotateright(root);
    if (fact < -1 && data > root->right->data)
        return rotateleft(root);
```

Arjun A S
IBM18CS019

```
if (fact > 1 && data < root -> lft -> data)
    { root -> left = rotateleft (root -> left)
      return rotateright (root)
    }
if (fact <-1 && data < root -> lft -> data)
    {   root -> right = rotateright (root -> right);
        return rotateleft (root)
    }
return root
}
// Deletion of a node
Node *deletion (Node * root, int item)
{
    if (root == NULL)
        return root.
    if [item < root -> data)
        root -> lft = deletion (root -> lft, item);
    else if (item > root -> data)
        root -> right = deletion (root -> right, item);
    else
    { if (root -> lft == NULL || root -> right == NULL)
      {   Node * s = root -> right ? root -> right : root -> left;
          if (s == NULL)
          {  s = 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)
      }
}
```

Arjun. A.S
1BM18CS019

```
int fact = balance_factor (root)
balance_factor
if (fact > 1 && (root -> left) >= 0)
        return rotateright (root);
if (fact < -1 && balance_factor (root -> left) <= 0)
        return rotateleft (root);
if (fact > 1  && balance_factor (root -> left) < 0)
    {   root -> left = rotateleft (root -> left)
        return rotateright (root);
    }
if (fact < -1 && balance_factor (root -> right) > 0)
    {       root -> right = rotateright ( root -> right);
            return rotateleft (root);
    }
return root;
}
```

Arjun. A.S