SOWMYA.D.V.

Page No.
Date: / /

AVL - Trees.

```
class AVL;
class AVLNODE
{
  friend class AVL;
 Private :
    int data;
    AVLNODE * left, * right;
    int bf;
};

class AVL
{
private :
    AVLNODE *root;
public :
  AVLNODE * loc, * par;
  AVL()
  {
    root = NULL;
  }
  int insert(int);
  void displayitem();
  Void display(AVLNODE *);
  Void removeitem(int);
  void remove1(AVLNODE *, AVLNODE *, int);
  void remove2(AVLNODE *, AVLNODE *, int);
  void search(int x);
  void search1(AVLNODE *, int);
};
```

```
int AVL::insert(int x)
{
  AVLNODE *a,*b,*c,*f,*p,*q,*y,
          *clchild,*crchild;
  int found, unbalanced;
  int d;
  if(!root)
  {
    y = new AVLNODE;
    y->data = x;
    root = y;
    root->bf = 0;
    root->left = root->right = NULL;
    return TRUE;
  }
  f = NULL;
  a = p = root;
  q = NULL;
  found = FALSE;
  while(p && !found)
  {
Search for insertion -->  if(p->bf)
    {
      a = p;
      f = q;
    }
    if(x < p->data)
    {  q = p;
       p = p->left;
    }
```

```
else if (x>p->data)
{   q=p;
    p=p->right;
}
else
{   y=p;
    found = TRUE;
}
}
```

Insert → rebalance

```
if ( ! found)
{
  y = new AVLNODE;
  y->data = x;
  y->left = y->right = NULL;
  y->bf = 0;
  if (x<q->data)
      q->left = y;
  else
     q->right = y;
```

insert as right →

```
  if (x>a->data)
  {  p = a->right;
     b = p;
     d = -1;
  }
  else
  {  p = a->left;
     b = p;
     d = 1;
  }
  while (p != y)
     if (x>p->data)
     {   p->bf = -1;
         p = p->right;
     }
```

```
else
{ p->bf = 1;
  p = p->left;
}
unbalanced = TRUE;
if (!(a->bf) || !(a->bf + d))
 {
    a->bf += d;
    unbalanced = FALSE;
 }
if (unbalanced)
{
  if (d == 1)
{     if (b->bf == 1)
    {  a->left = b->right;
       b->right = a;
       a->bf = 0;
       b->bf = 0;
    }
  else
{ c = b->right;
    b->right = c->left;
    a->left = c->right;
    c->left = b;
    c->right = a;
   switch (c->bf)
   {
    case 1:
          a->bf = -1;
          b->bf = 0;
          break;
   case -1:
          b->bf = 1;
          a->bf = 0;
   break;
```

```
case 0:
    b->bf = 0;
    a->bf = 0;
    break;
}
c->bf = 0;
b = c;
}
}
else
{
  if (b->bf == -1)
  { a->right = b->left;
    b->left = a;
    a->bf = 0;
    b->bf = 0;
}
else
{  c = b->right;
   b->right = c->left;
   a->right = c->left;
   c->right = b;
   c->left = a;
  switch (c->bf)
  {
   case 1:
     a->bf = -1;
     b->bf = 0;
     break;
  case -1:
     b->bf = 1;
     a->bf = 0;
     break;
  case 0:
     b->bf = 0;
     a->bf = 0;
  }    break;
```

```
        c->bf = 0;
        b = c;
    }
    if (!f)
        root = b;
    else if (a == f->left)
        f->left = b;
    else if (a == f->right)
        f->right = b;
    }
    return FALSE;
}
void AVL :: displayitem()
{ display(root);
}
void AVL :: display (AVLNODE * temp)
{ if (temp == NULL)
    return;
  cout << temp->data << " ";
  display (temp->left);
  display (temp->right);
}
void AVL :: removeitem(int x)
  { search(x);
    if (loc == NULL)
    {
      cout << "\n item is not in tree :(";
      return;
    }
  if (loc->right != NULL && loc->left != NULL)
      remove1 (loc, par, x);
  else
      remove2 (loc, par, x);
}
```

```
void AVL :: remove1 (AVLNODE *l, AVLNODE *p,
                     int x)
{
  AVLNODE *ptr, *save, *suc, *psuc;
  ptr = l->right;
  save = l;
  while (ptr->left != NULL)
  { save = ptr;
    ptr = ptr->left
  }
  suc = ptr;
  psuc = save;
  remove2 (suc, psuc, x);
  if (p != NULL)
    if (l == p->left)
      p->left = suc;
    else
      p->right = suc;
  else
    root = l;
  suc->left = l->left;
  suc->right = l->right;
  return;
void AVL :: remove2 (AVLNODE *s, AVLNODE *p,
                     int x)
{ AVLNODE *child;
  if (s->left == NULL && s->right == NULL)
    child = NULL;
  else if (s->left != NULL)
    child = s->left;
  else
    child = s->right;
  if (p != NULL)
    if (s == p->left)
      p->left = child;
    else p->right = child;
  else root = child;
}
```