Binary and AVL Trees in C

Jianye Hao
Overview

- Binary tree
  - Degree of tree is 2

```c
struct node_s {
    Datatype element;
    struct node_s *leftChild;
    struct node_s *rightChild;
};
typedef struct node_s node;
```
void preorder(node *t) {
    if (t != NULL) {
        printf("%d ", t->element);    /* V */
        preorder(t->leftChild);       /* L */
        preorder(t->rightChild);      /* R */
    }
}

void inorder(node *t) {
    if (t != NULL) {
        inorder(t->leftChild);      /* L */
        printf("%d ", t->element);  /* V */
        inorder(t->rightChild);     /* R */
    }
}
Trees – traversal (Recursion Method)

Postorder

```c
void postorder(node *t) {
    if (t != NULL) {
        postorder(t->leftChild); /* L */
        postorder(t->rightChild); /* R */
        printf("%d ", t->element); /* V */
    }
}
```
Trees - traversal

- Preorder
  A B D G H E C F I

- Inorder
  G D H B E A F I C

- Postorder
  G H D E B I F C A
Definition

- An AVL tree (or Height-Balanced tree) is a binary search tree such that:
  - The height of the left and right subtrees of the root differ by at most 1.
  - The left and right subtrees of the root are AVL trees.
AVL Tree
Non-AVL Tree
To keep track of whether a binary search tree is an AVL tree, we associate with each node a balance factor, which is

\[ \text{Height(right subtree)} - \text{Height(left subtree)} \]
AVL tree

- Height(right subtree) – Height(left subtree)
Non-AVL tree

Height(right subtree) – Height(left subtree)
AVL tree structure in C

For each node, the difference of height between left and right are no more than 1.

```c
struct AVLnode_s {
    Datatype element;
    struct AVLnode *left;
    struct AVLnode *right;
};
typedef struct AVLnode_s AVLnode;
```
There are four models about the operation of AVL Tree:

- LL
- RR
- LR
- RL
<table>
<thead>
<tr>
<th>Case 1: insertion to right subtree of right child</th>
<th>Case 2: insertion to left subtree of left child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution: <strong>Left</strong> rotation</td>
<td>Solution: <strong>Right</strong> rotation</td>
</tr>
<tr>
<td><img src="image1" alt="Case 1 Diagram" /></td>
<td><img src="image2" alt="Case 2 Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 3: insertion to right subtree of left child</th>
<th>Case 4: insertion to left subtree of right child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution: <strong>Left-right</strong> rotation</td>
<td>Solution: <strong>Right-left</strong> rotation</td>
</tr>
<tr>
<td><img src="image3" alt="Case 3 Diagram" /></td>
<td><img src="image4" alt="Case 4 Diagram" /></td>
</tr>
</tbody>
</table>
Case 1: insertion to right subtree of right child

Solution: **Left** rotation

Left-Rotation
Left-Rotation

Case 1: insertion to right subtree of right child

Solution: Left rotation

Add 23

Left-Rotation
Right-Rotation

Case 2: insertion to left subtree of left child

Solution: Right rotation

```
  \begin{aligned}
    &T_1 \quad T_2 \quad T_3 \\
    &T_1 \quad T_2 \quad T_3 \\
  \end{aligned}
```

```
  \begin{aligned}
    &T_1 \quad T_2 \quad T_3 \\
    &T_1 \quad T_2 \quad T_3 \\
  \end{aligned}
```

```
  \begin{aligned}
    &T_1 \quad T_2 \quad T_3 \\
    &T_1 \quad T_2 \quad T_3 \\
  \end{aligned}
```

Add 20
Right-Rotation

Case 2: insertion to left subtree of left child
Solution: Right rotation
Left-Right Rotation

Case 3: insertion to right subtree of left child

Solution: Left-right rotation

```
           17
          /   
         9     23
        / 
       7   8

Add 8

           17
          /   
         9     23
        /     
       7      8

Left-Rotation

           17
          /   
         9     23
        /     
       7      8

Left-Rotation

           17
          /   
         9     23
        /     
       7      8

Right-Rotation

           17
          /   
         9     23
        /     
       7      8
```

```
```

```
```
Left-Right Rotation

Case 3: insertion to right subtree of left child

Solution: Left-right rotation

Add 21

Left-rotation

Right-rotation

Right-rotation
Right-Left Rotation

Case 4: insertion to left subtree of right child
Solution: Right-left rotation

Add 27

Right-Left Rotation

Left-Rotation

Left-Rotation
Right-Left Rotation

Case 4: insertion to left subtree of right child

Solution: Right-left rotation
How to identify rotations?

- First find the node that cause the imbalance (balance factor)
- Then find the corresponding child of the imbalanced node (left node or right node)
- Finally find the corresponding subtree of that child (left or right)
How to identify rotations?

Add 23

Add 12

Add 16
Balancing an AVL tree after an insertion

- Begin at the node containing the item which was just inserted and move back along the access path toward the root. {
  - For each node determine its height and check the balance condition. {
    - If the tree is AVL balanced and no further nodes need be considered.
    - else If the node has become unbalanced, a rotation is needed to balance it.
  }
}

} we proceed to the next node on the access path.
AVLnode *insert(Datatype x, AVLnode *t) {
    if (t == NULL) {
        /* CreateNewNode */
    }
    else if (x < t->element) {
        t->left = insert(x, t->left);
        /* DoLeft */
    }
    else if (x > t->element) {
        t->right = insert(x, t->right);
        /* DoRight */
    }
}
AVL tree

CreateNewNode

```c
struct AVLnode *t = malloc(sizeof(struct AVLnode));
t->element = x;
t->left = NULL;
t->right = NULL;
```
AVL tree

DoLeft

if (height(t->left) - height(t->right) == 2)
    if (x < t->left->element)
        t = singleRotateWithLeft(t); // LL
    else
        t = doubleRotateWithLeft(t); // LR
AVL tree

- DoRight

```c
if (height(t->right) - height(t->left) == 2)
    if (x > t->right->element)
        t = singleRotateWithRight(t); // RR
    else
        t = doubleRotateWithRight(t); // RL
```
Demo

http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html
You can insert, delete and locate nodes in the tree using control buttons.

The data can be entered manually or randomly generated.

By pressing <Insert> button only, you can quickly build a large tree.
The End

Any Questions?