화일처리 보고서
AVL 트리

20181653 이강희

November 12, 2019

강의 슬라이드를 참조하였습니다.

1 출력 결과

Listing 1: output.txt

```
File Processing
   AVL Tree
=================

insert 40       NO      40
insert 11       NO      11 40
insert 77       NO      11 40 77
insert 33       NO      11 33 40 77
insert 20       RL      11 20 33 40 77
insert 90       NO      11 20 33 40 77 90
insert 99       RR      11 20 33 40 77 90 99
insert 70       NO      11 20 33 40 70 77 90 99
insert 88       NO      11 20 33 40 70 77 88 90 99
insert 80       LR      11 20 33 40 70 77 80 88 90 99
insert 66       RL      11 20 33 40 66 70 77 80 88 90 99
insert 10       NO      10 11 20 33 40 66 70 77 80 88 90 99
insert 22       NO      10 11 20 22 33 40 66 70 77 80 88 90 99
insert 30       LR      10 11 20 22 30 33 40 66 70 77 80 88 90 99
insert 44       LL      10 11 20 22 30 33 40 44 66 70 77 80 88 90 99
insert 55       NO      10 11 20 22 30 33 40 44 55 66 70 77 80 88 90 99
insert 50       RL      10 11 20 22 30 33 40 44 50 55 66 70 77 80 88 90 99
insert 60       LR      10 11 20 22 30 33 40 44 50 55 60 66 70 77 80 88 90 99
insert 100      RR      10 11 20 22 30 33 40 44 50 55 60 66 70 77 80 88 90 99 100
insert 28       LL      10 11 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90 99 100
insert 18       NO      10 11 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90 99 100
insert 9        NO      9 10 11 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90 99 100
insert 5        LL      5 9 10 11 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90 99 100
insert 17       NO      5 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90 99
                        100
insert 6        NO      5 6 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90 99
                        100
insert 3        NO      3 5 6 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90
                        99 100
insert 1        LL      1 3 5 6 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88 90
                        99 100
insert 4        NO      1 3 4 5 6 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88
                        90 99 100
insert 2        LL      1 2 3 4 5 6 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80 88
                        90 99 100
insert 7        LR      1 2 3 4 5 6 7 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77 80
                        88 90 99 100
insert 8        RR      1 2 3 4 5 6 7 8 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77
                        80 88 90 99 100
insert 10       ERROR   1 2 3 4 5 6 7 8 9 10 11 17 18 20 22 28 30 33 40 44 50 55 60 66 70 77
                        80 88 90 99 100
insert 12       LL      1 2 3 4 5 6 7 8 9 10 11 12 17 18 20 22 28 30 33 40 44 50 55 60 66 70
                        77 80 88 90 99 100
insert 13       NO      1 2 3 4 5 6 7 8 9 10 11 12 13 17 18 20 22 28 30 33 40 44 50 55 60 66 70
insert 14       RR      1 2 3 4 5 6 7 8 9 10 11 12 13 14 17 18 20 22 28 30 33 40 44 50 55 60 66
                        70 77 80 88 90 99 100
insert 16       LR      1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 17 18 20 22 28 30 33 40 44 50 55
                        60 66 70 77 80 88 90 99 100
insert 15       LR      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 22 28 30 33 40 44 50
                        55 60 66 70 77 80 88 90 99 100
```

2 소스 코드

```
#include <iostream>
#include <string>
using namespace std;

enum rotationType {NO=1, LL, RR, LR, RL};
string typeStringArray[] = {"ERROR", "NO", "LL", "RR", "LR", "RL"};

class TreeNode{
public:
    TreeNode(int key=0, int bf=0, TreeNode *left=nullptr, TreeNode *right=nullptr)
        : key(key), bf(bf), left(left), right(right) {}
    ~TreeNode() {
        if (this->left != nullptr) delete this->left;
        if (this->right != nullptr) delete this->right;
    }
    int key;
    int bf;
    TreeNode *left;
    TreeNode *right;
};

TreeNode *getNode(int key=0, int bf=0, TreeNode *left=nullptr, TreeNode *right=nullptr) {
    return new TreeNode(key, bf, left, right);
}

bool insertBST(TreeNode *&T, int newKey) {
    if (T == nullptr) {
        T = getNode(newKey);
        return true;
    }

    TreeNode *q = nullptr;
    TreeNode *p = T;
    while (p != nullptr) {
        if (newKey == p->key) return false;
        q = p;
        if (newKey < p->key) p = p->left;
        else p = p->right;
    }

    TreeNode *newNode = getNode(newKey);

    if (T == nullptr) T = newNode;
    else if(newKey < q->key) q->left = newNode;
    else q->right = newNode;
    return true;
}

void checkBalance(TreeNode *T, int newKey, int &rotationType, TreeNode *&p, TreeNode *&q) {
    TreeNode *a, *f;
    a = p = T;
    f = q = nullptr;
    while (p != nullptr) {
        if (p->bf != 0) {
            a = p; f = q;
        }
```

```
        q = p;
        if (newKey < p->key) p = p->left;
        else p = p->right;
    }
    p = a;
    while (p->key != newKey) {
        if (newKey > p->key) {
            p->bf -= 1;
            p = p->right;
        }
        else {
            p->bf += 1;
            p = p->left;
        }
    }
    p = a;
    q = f;
    if (abs(p->bf) <= 1) {
        rotationType = NO;
        p = q = nullptr;
        return;
    }
    if (p->bf == 2) {
        if (p->left->bf == 1) rotationType = LL;
        else rotationType = LR;
    }
    else {
        if (p->right->bf == -1) rotationType = RR;
        else rotationType = RL;
    }
}

void rotateTree(TreeNode *&T, int rotationType, TreeNode *a, TreeNode *f) {
    TreeNode *b;
    if (rotationType == LL || rotationType == LR) b = a->left;
    else b = a->right;
    if (rotationType == LL) {
        a->left = b->right;
        b->right = a;
        a->bf = 0;
        b->bf = 0;
    }
    else if (rotationType == RR) {
        a->right = b->left;
        b->left = a;
        a->bf = 0;
        b->bf = 0;
    }
    else if (rotationType == LR) {
        TreeNode *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:
                a->bf = 0;
                b->bf = 1;
                break;
            case 0:
                a->bf = 0;
                b->bf = 0;
        }
        c->bf = 0;
        b = c;
    }
    else { // RL
        TreeNode *c = b->left;
        b->left = c->right;
        a->right = c->left;
        c->left = a;
        c->right = b;
        switch (c->bf) {
            case 1:
                a->bf = 0;
                b->bf = -1;
                break;
            case -1:
                a->bf = 1;
                b->bf = 0;
                break;
            case 0:
                a->bf = 0;
                b->bf = 0;

        }
        c->bf = 0;
        b = c;
    }
    if (f == nullptr) T = b;
    else if (a == f->left) f->left = b;
    else f->right = b;
}

int insertAVL(TreeNode *&T, int newKey) {
    if (!insertBST(T, newKey)) return 0;
    int rotationType;
    TreeNode *p, *q;
    checkBalance(T, newKey, rotationType, p, q);
    if (rotationType != NO) rotateTree(T, rotationType, p, q);
    return rotationType;
}

void inorder(TreeNode *T) {
    if (T == nullptr) return;
    inorder(T->left);
    cout << T->key << ' ';
    inorder(T->right);
}

int main() {
    cout << "File Processing" << endl;
```

```
    cout << "   AVL Tree" << endl;
    cout << "==================" << endl;

    int keyArray[] = {
        40, 11, 77, 33, 20, 90, 99, 70, 88, 80,
        66, 10, 22, 30, 44, 55, 50, 60, 100, 28,
        18, 9, 5, 17, 6, 3, 1, 4, 2, 7,
        8, 10, 12, 13, 14, 16, 15
        };

    TreeNode *root = nullptr;

    for (int key : keyArray) {
        int rotationType = insertAVL(root, key);
        cout << "insert " << key << '\t' << typeStringArray[rotationType] << '\t';
        inorder(root);
        cout << endl;
    }

    delete root;

    return 0;
}
```