TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

2018.03.08. – 011일차 수업

---

* AVL 트리 Delete 함수

트리 Level 리턴 함수

```
int getTreeLvl(Tree** tree){
	int Lvl, Lvl_L, Lvl_R ;

	if(*tree == NULL){
		return 0;
	}
	else if( (*tree)->Right == NULL && (*tree)->Left == NULL ){
		return 1;
	}
	else
	{
		Lvl_L = getTreeLvl(&((*tree)->Left));
		Lvl_R = getTreeLvl(&((*tree)->Right));
		Lvl = ( (Lvl_L < Lvl_R) ? Lvl_R : Lvl_L ) + 1;
		return Lvl;
	}
}
```

트리 균형 상태 리턴 함수

```
int TreeLvlCmp(Tree** tree){
	if(*tree == NULL){
		return 0;
	}
	Tree* rightSubtree = (*tree)->Right;
	Tree* leftSubtree = (*tree)->Left;
	return getTreeLvl(&rightSubtree) - getTreeLvl(&leftSubtree);
}
```

RR, LL, RL, LR 회전 후 바뀐 루트노드 반환 함수

```
Tree* RRrot(Tree** tree){
	Tree* tmpTree = (*tree)->Right;
	(*tree)->Right = tmpTree->Left;
	tmpTree->Left = *tree;
	return tmpTree;
}
Tree* LLrot(Tree** tree){
	Tree* tmpTree = (*tree)->Left;
	(*tree)->Left = tmpTree->Right;
	tmpTree->Right =(*tree);
	return tmpTree;
}
Tree* RLrot(Tree** tree){
	Tree* tmpTree = (*tree)->Right;
	tmpTree = LLrot(&tmpTree);
	(*tree)->Right = tmpTree;
	tmpTree = RRrot(tree);
	return tmpTree;
}
Tree* LRrot(Tree** tree){
	Tree* tmpTree = (*tree)->Left;
	tmpTree = RRrot(&tmpTree);
	(*tree)->Left = tmpTree;
	tmpTree = LLrot(tree);
	return tmpTree;
}
```

루트노드 기준 양옆 균형 맞추는 함수

```
Tree* Rebalance(Tree** root){
	if(*root == NULL){
		return *root;
	}

	Tree* cmpTree = *root;
	Tree* rootTree = *root;

	while(1)
	{
		//RR or RL 필요
		if (TreeLvlCmp(&rootTree) > 1)
		{
```

```
            cmpTree = rootTree->Right;
            //RR 실행
            if (TreeLvlCmp(&cmpTree) >= 0)
            {
                cmpTree = RRrot(&rootTree);
            }
            //RL 실행
            else
            {
                cmpTree = RLrot(&rootTree);
            }
        }
        //LL or LR 필요
        else if (TreeLvlCmp(&rootTree) < -1)
        {
            cmpTree = rootTree->Left;
            //LL 실행
            if (TreeLvlCmp(&cmpTree) <= 0)
            {
                cmpTree = LLrot(&rootTree);
            }
            //LR 실행
            else
            {
                cmpTree = LRrot(&rootTree);
            }
        }
        //균형 잡힘
        else
            break;
rootTree = cmpTree;
    }
    return rootTree;
}
```

트리 균형 맞추는 함수(재귀 이용)

```
Tree* RebalanceTree(Tree** tree){
    if(*tree == NULL){
        return NULL;
    }
    if((*tree)->Left != NULL){
        (*tree)->Left = Rebalance(&((*tree)->Left));
        RebalanceTree( &((*tree)->Right) );
    }
    if((*tree)->Right != NULL){
        (*tree)->Right = Rebalance(&((*tree)->Right));
        RebalanceTree( &((*tree)->Right) );
    }
    return *tree;
}
```

트리 데이터 삭제 함수(재귀)

```
Tree* delete_tree(Tree *root, int data){
    int num;
    //루트노드 NULL인 경우
    if(root == NULL){
        printf("not found₩n");
        return NULL;
    }
    //노드의 데이터보다 찾는 값이 작을때
    else if(root->data > data)
        root->Left = delete_tree(root->Left, data);
    //노드의 데이터보다 찾는 값이 클때
    else if(root->data < data)
        root->Right = delete_tree(root->Right,
data);
    //노드의 왼쪽과 오른쪽 주소 값이 모두 존재할
경우 == 서브트리일 경우
    else if(root->Left && root->Right){
        find_max(root->Left, &num);
        root->data = num;
    }
    else
        root = chg_node(root);
    return root;
}
```

AVL 트리 데이터 삭제 함수(재귀)

```
Tree* AVL_delete_tree(Tree *root, int data){
    delete_tree(root,data);
    root = RebalanceTree(&root);
    return root;
}
```