Xilinx Zynq FPGA, TI DSP, MCU
기반의 프로그래밍 및 회로 설계
전문가 과정

강사 – **Innova Lee(**이상훈**)**
gcccompil3r@gmail.com

학생 – 안상재
sangjae2015@naver.com

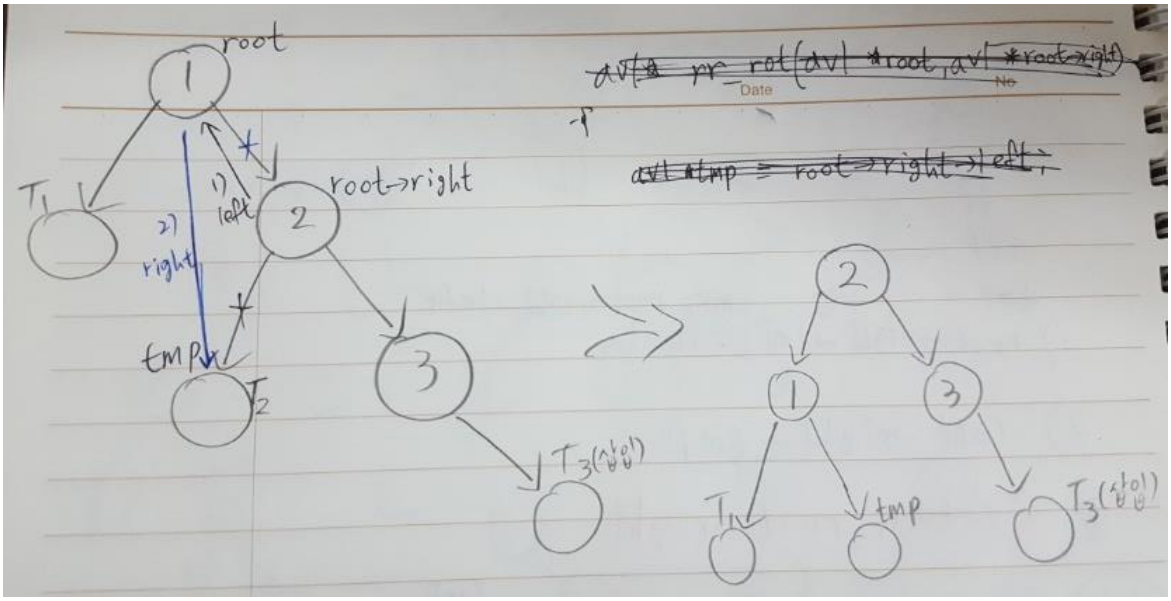
1. 이진 트리와 AVL 트리의 차이점

- ✓ **이진 트리 :** 삽입되는 데이터에 따라서 트리의 모양이 자유자재로 변할 수 있으며, 트리 구조가 일렬로 될 경우, 맨 끝의 데이터를 검색하는 너무 많은 시간이 소요됨. (트리 구조가 일렬로 될 경우 큐의 구조와 같아짐.)
- ✓ **AVL 트리 :** 각 노드의 왼쪽 서브 트리와 오른쪽 서브 트리의 높이 차가 최대 1 차이나며, 서로 균형을 이룸. 노드의 삽입과 삭제 시마다 트리의 재균형 작업을 해주어야 해서 이진 트리보다 시간이 더 걸리지만, 검색을 할 경우에는 모든 트리 종류 중에 가장 빠른 검색 속도의 장점을 가짐.

2. AVL 트리의 노드 삽입 시 회전 알고리즘

1) RR 회전

- 노드가 오른쪽으로 일직선상에 있고, 오른쪽 높이가 왼쪽 높이보다 2 이상 클 때 수행함.



- 내가 구현해본 RR 회전 코드

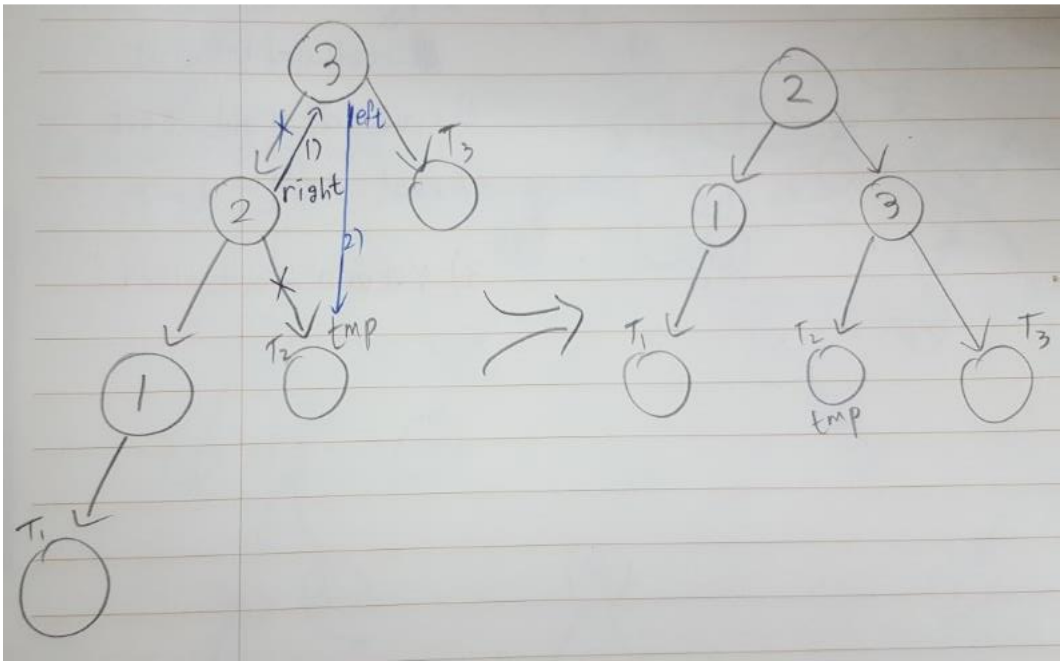
* 핵심

-> 2의 왼쪽 노드를 어떻게 처리할 것인가?

```
avl* rr_rot(avl *root, avl *root->right)        // 현재 root는 1이고 (*root)->right는 2
{
        avl *tmp = root->right->left;       // 2 노드의 left가 1 노드를 가리키면 2와 T2 노드가 끊어지므로
                                            T2에 접근할 수 없어짐. 그래서 T2를 tmp에 잠시 저장함.
        root->right->left = root;  // 2 노드의 left가 1노드를 가리키게함.
        root->right = tmp;         // 1 노드의 right가 2 노드의 left 였던 T2를 가리키게 함.
        return root->right;        // 가장 위의 노드인 2의 노드를 리턴함.
}
```

2) LL 회전

- 노드가 왼쪽으로 일직선상에 있고, 왼쪽 높이가 오른쪽 높이보다 2 이상 클 때 수행함.



- 내가 구현해본 LL 회전 코드 (RR과 동일한 방법)

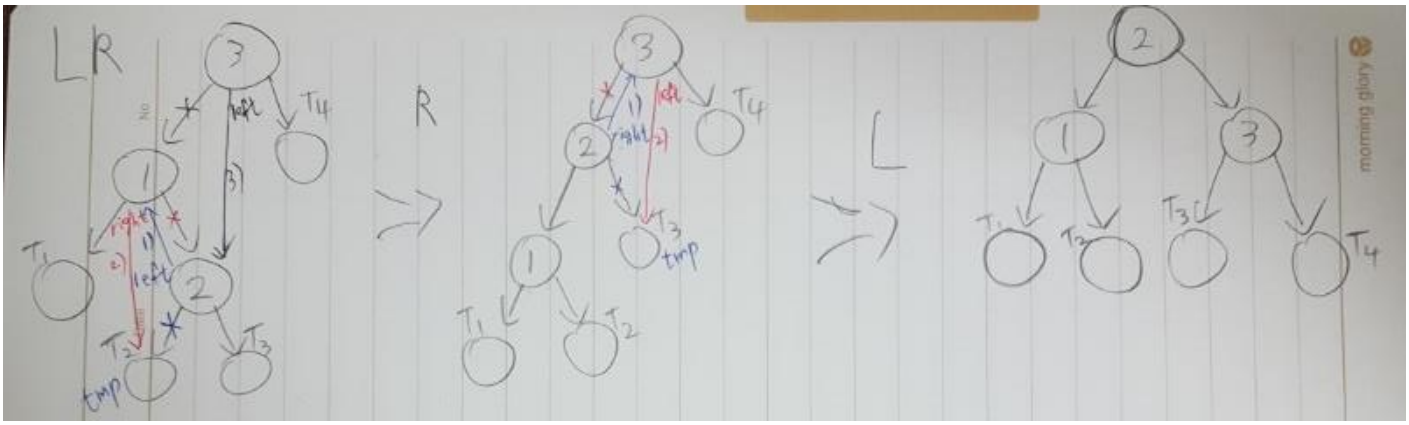
* 핵심

-> 2의 오른쪽 노드를 어떻게 처리할 것인가?

```
avl* ll_rot(avl *root, avl *root->left)
{
        avl *tmp = root->left->right;
        root->left->right = root;
        root->left = tmp;
        return root->left;
}
```

3) LR 회전 코드

- LR 회전은 1 노드의 오른쪽 자식의 왼쪽에 노드가 추가될 때 수행된다.



- 내가 구현해본 LR 회전 코드

* 핵심

-> RR,LL 회전과 달리 회전을 2 번 하게 된다.(1.LL -> 2.RR) 이를 어떻게 구현할 것인가?

-> 노드를 연결하는 과정에서 끊어져서 접근 불가능하게 되는 노드를 어떻게 처리할 것인가?

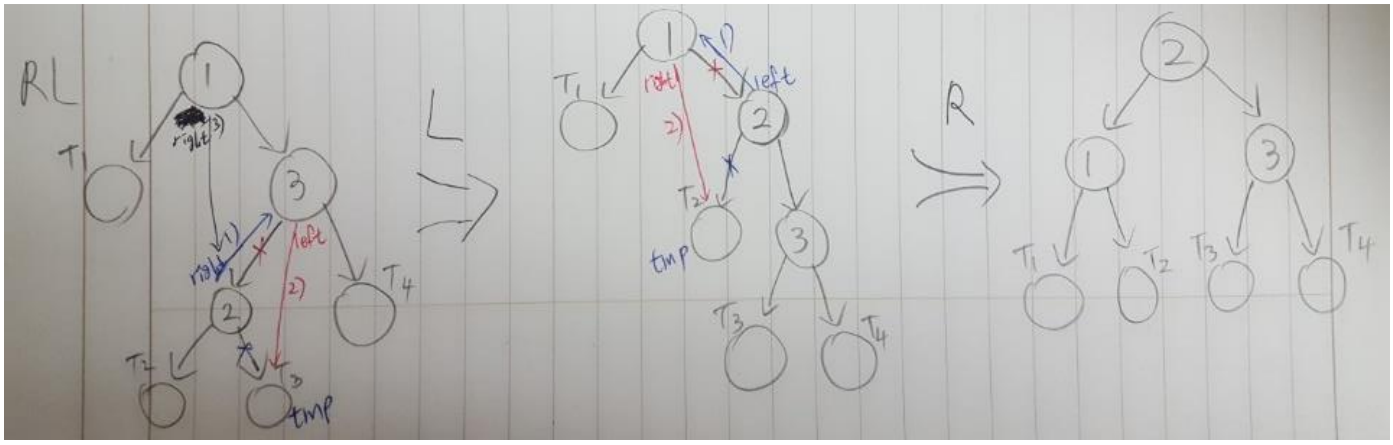
```
avl* lr_rot(avl *root, avl *root->left)
{
        avl *tmp1;        // 2 노드를 저장할 포인터 변수.
        avl *tmp = root->left->right->left; // 2 노드의 left가 1 노드를 가리키면 2와 T2 노드가 끊어지므로
                                            T2에 접근할 수 없어짐. 그래서 T2를 tmp에 저장함.
        root->left->right->left = root->left;   // 2 노드의 left가 1 노드를 가리키게 함.
        root->left->right = tmp;            // 1 노드의 right가 T2 가리키게 함.
        root->left = root->left->right;     // 3 노드의 left가 2 노드를 가리키게 함.

        tmp1 = root->left;          // 3 노드의 left가 T3을 가리키면 3 노드에서 2 노드를 접근할 수 없게됨.
                                    그래서 2 노드를 tmp1에 저장함.
        tmp = root->left->right;            // 2 노드의 right가 3 노드를 가리키면 T3에 접근할 수 없게됨.
                                                그래서 T3을 tmp에 저장함.
        root->left->right = root;  // 2 노드의 right가 3 노드를 가리키게 함.
        root->left = tmp;          // 3 노드의 left가 T3을 가리키게 함.

        return tmp1;      // root가 된 2 노드의 주소를 리턴함.
}
```

4) RL 회전 코드

- 3의 왼쪽 자식의 오른쪽 노드에 새로운 노드가 삽입되면 LR 회전이 수행된다. 1, 2에 대해 RR회전이 수행되고 그 후 3을 기준으로 LL 회전이 수행된다.



- 내가 구현해본 RL 회전 코드 (LR 회전 코드와 동일한 방법)

```
avl* rl_rot(avl *root, avl *root->right)
{
	avr *tmp1;
	avl *tmp = root->right->left->right;
	root->right->left->right = root->right;
	root->right->left = tmp;
	root->right = tmp;
	root->right = root->right->left;

	tmp1 = root->right;
	tmp = root->right->left;
	root->right->left = root;
	root->right = tmp;

	return tmp1;
}
```