탐색

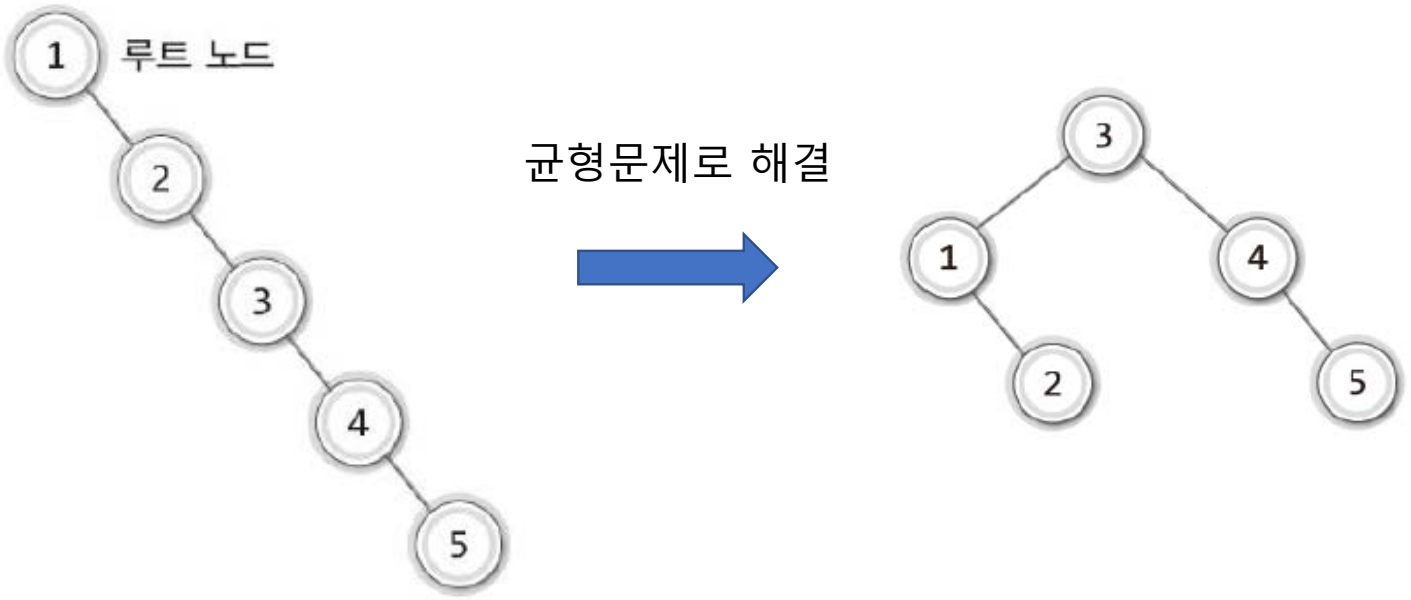
15 주차-강의

남춘성

SUNG KYUN KWAN UNIVERSITY

이진 탐색 트리 문제점과 AVL 트리

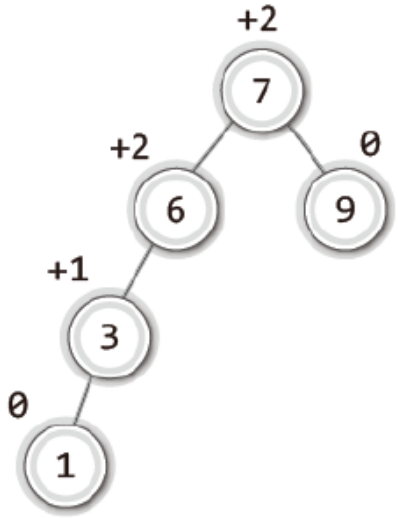
- 이진 탐색 트리의 탐색 연산은 $O(\log_2 n)$의 시간 복잡도를 보임
- 균형이 맞지 않을 수록 O(n)에 가까운 시간 복잡도를 보임

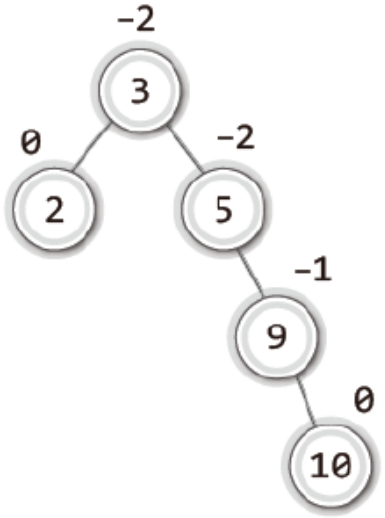


자동으로 균형 잡는 AVL 트리와 균형인수(Balance Factor)

SUNG KYUN KWAN UNIVERSITY

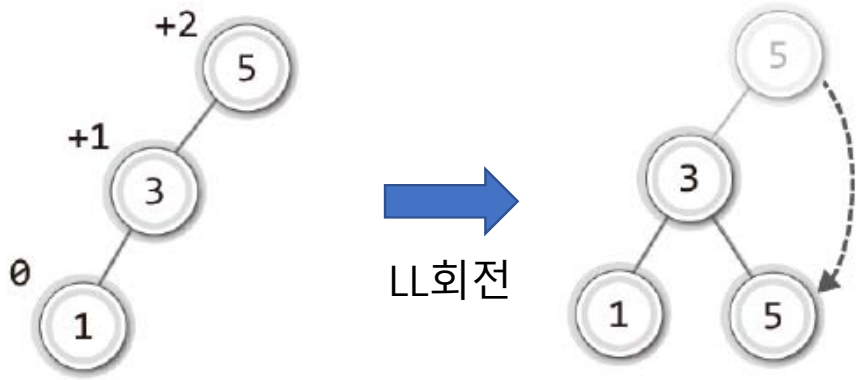
- 균형 인수 = 왼쪽 서브 트리 높이 - 오른쪽 서브 트리 높이





균형 인수의 절대값이 2이상인 경우
→ 리밸런싱

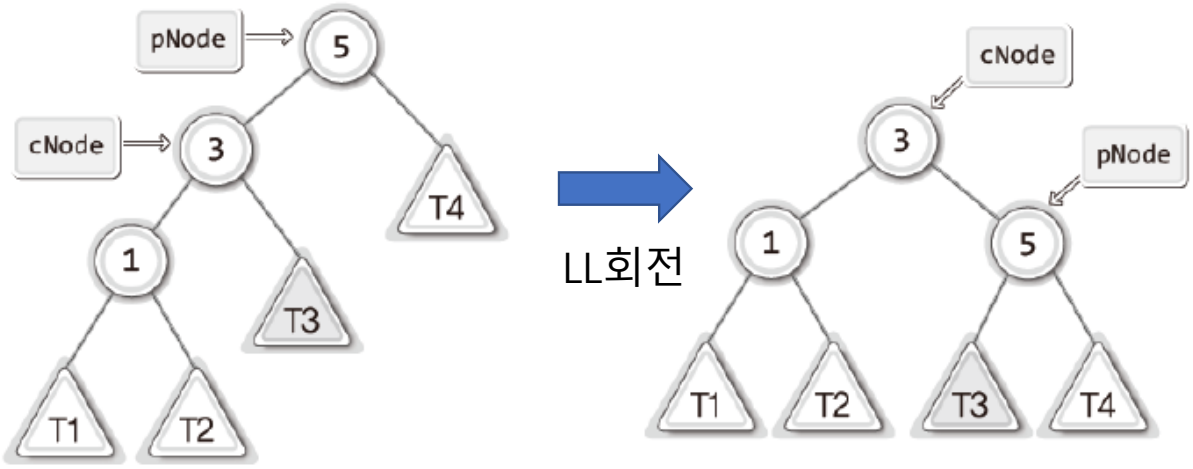
리밸런싱(Rebalancing) : 첫 번째 상태 (LL회전)



- 루트 노드 균형 인수가 +2
- 자식 노드 두개가 왼쪽으로 연이어 연결되어서 균형 인수 +2가 연출
- LL회전이란 왼쪽으로 두번 돌리는 회전을 의미하는 것이 아닌, LL상태에서 균형을 위한 회전

LL상태를 균형 잡기 위한 LL회전

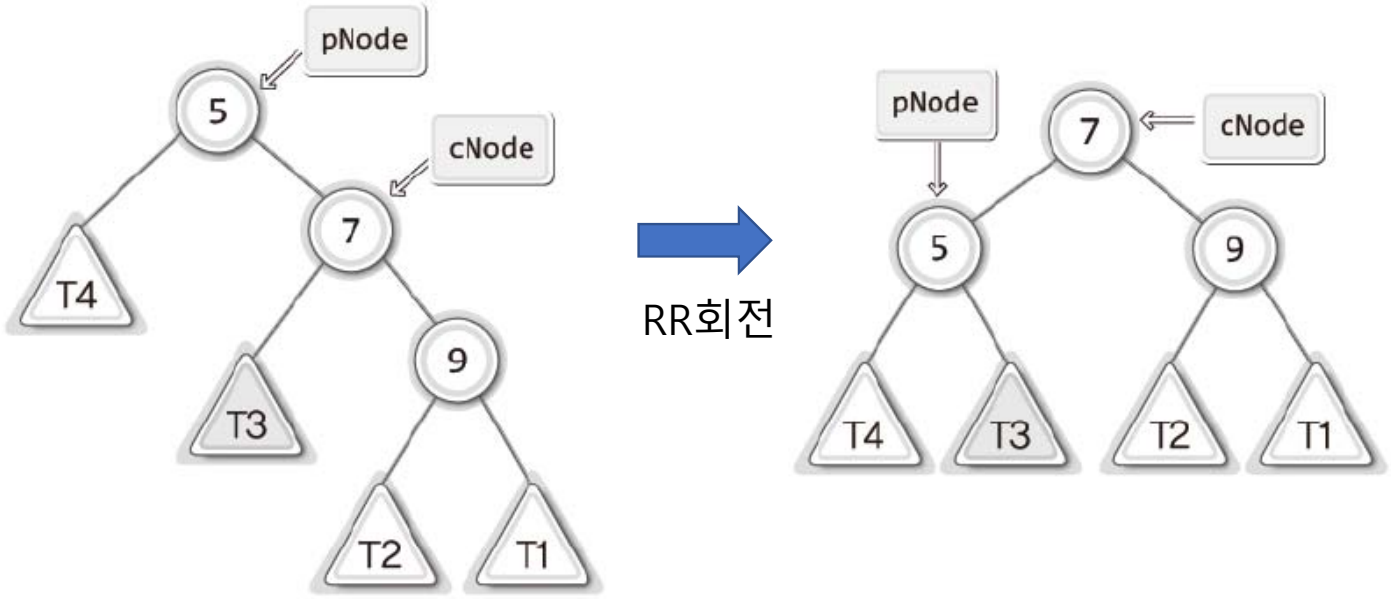
ChangeLeftSubTree(cNode, pNode)



pNode를 cNode의 오른쪽자식노드가 되게 변경
cNode의 오른쪽 서브트리를 pNode의 왼쪽 서브트리로

ChangeLeftSubTree(pNode, GetRightSubTree(cNode))
ChangeRightSubTree(cNode, pNode)

리밸런싱(Rebalancing) : 두 번째 상태 (RR회전)



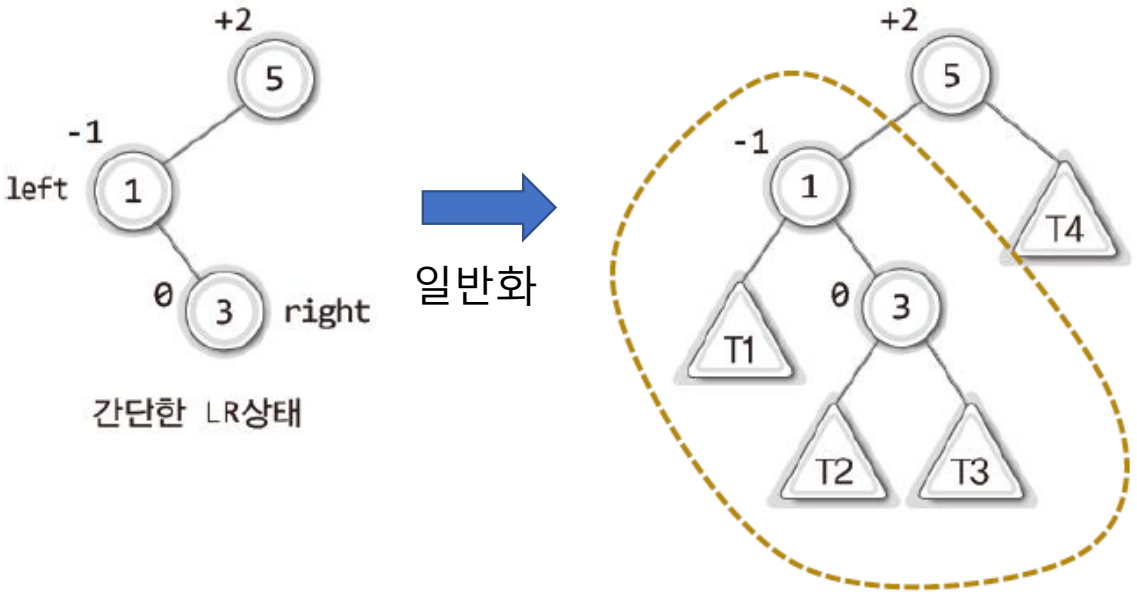
pNode를 cNode의 왼쪽자식 노드로 변경
cNode의 왼쪽 서브 트리를 pNode의 오른쪽 서브트리로

ChangeRighttSubTree(pNode, GetLeftSubTree(cNode))

ChangeLeftSubTree(cNode, pNode)

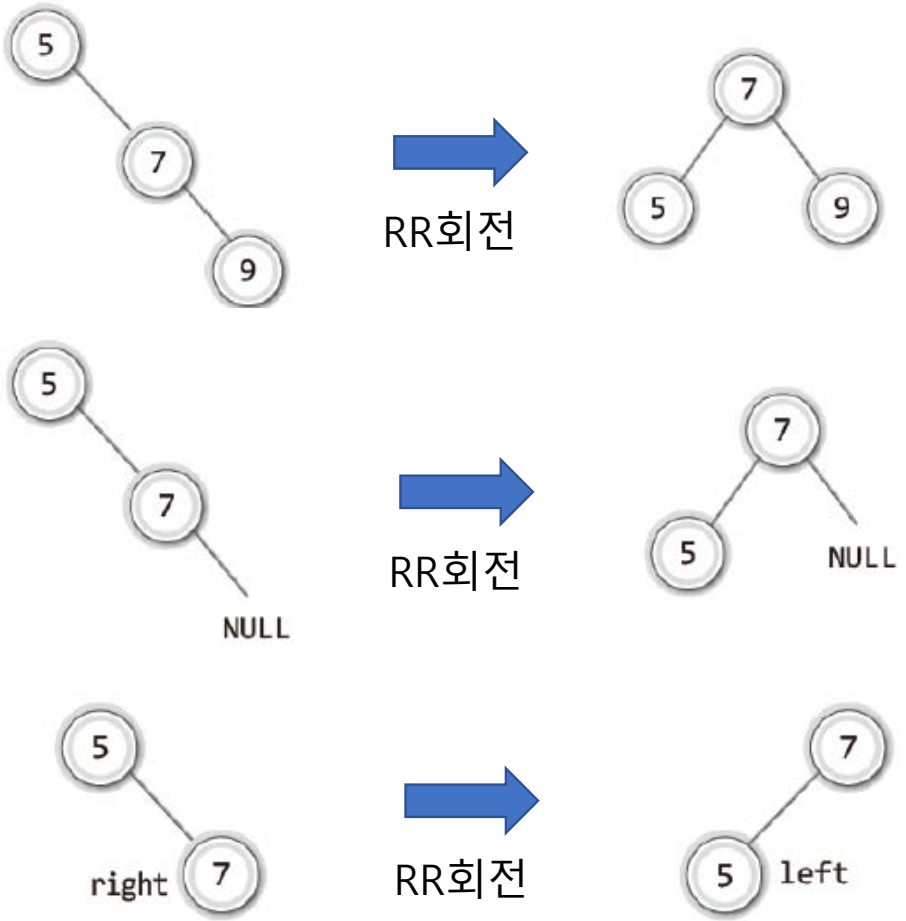
리밸런싱(Rebalancing) : 세 번째 상태 (LR회전)

- LL상태 그리고 RR상태와 같이 한 번의 회전으로 균형을 잡을 수 없음
- LR상태는 한 번의 회전으로 균형이 잡히는 LL상태 또는 RR상태가 되도록 하는 것이 우선

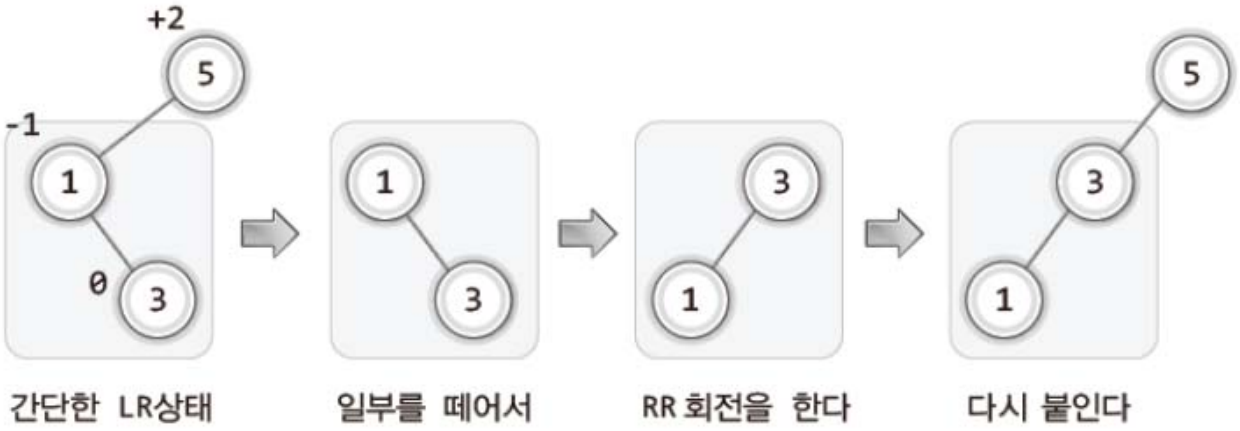


LR 상태는 RR회전을 통해서(RR회전의 부수적인 효과를 이용해서) LL상태가 되게 할 수 있음

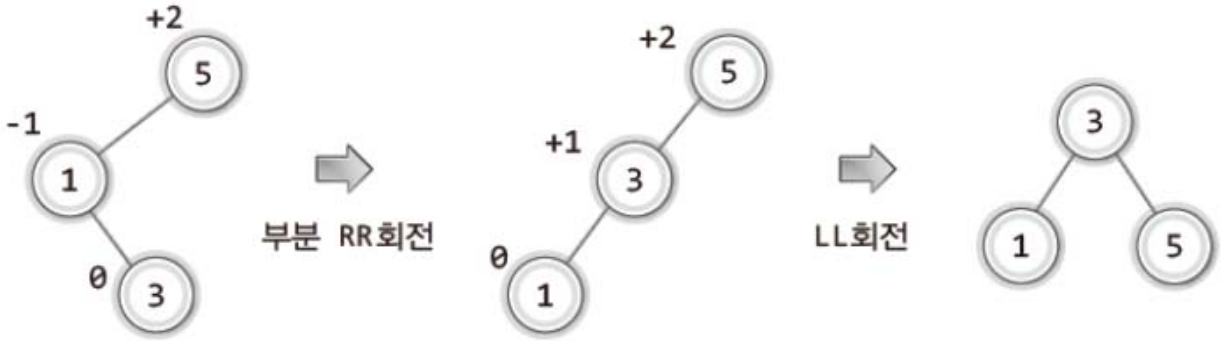
RR회전의 부수적인 효과



LR 회전 방법

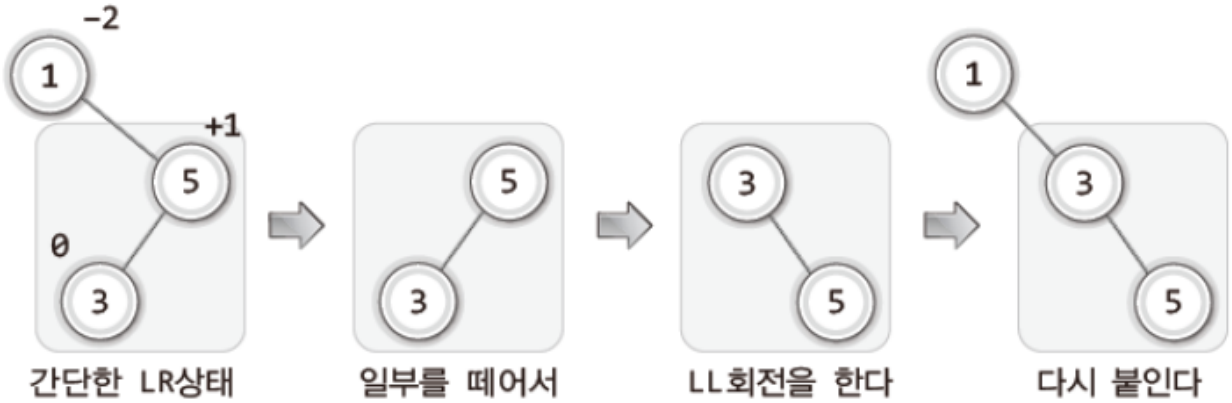


LR상태를 LL상태로 변경

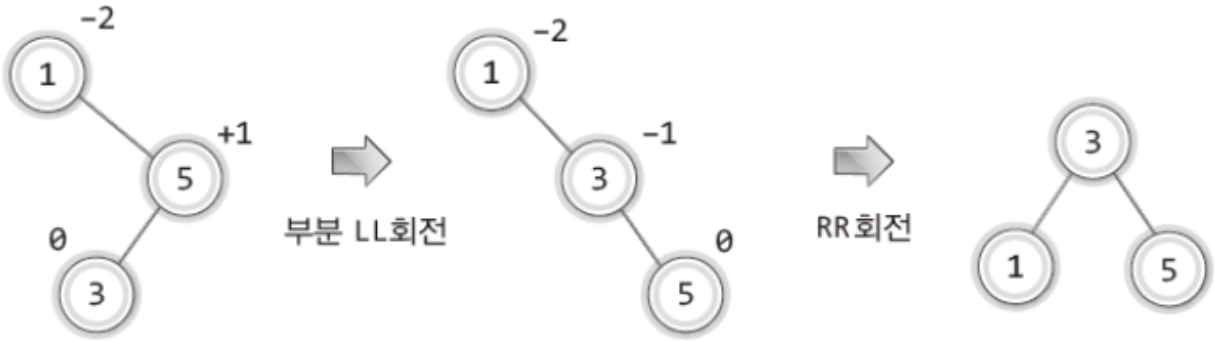


LR 회전

리밸런싱(Rebalancing) : 네 번째 상태 (RL회전)



RL상태를 RR상태로



RL 회전

AVL 트리 구현

SUNG KYUN KWAN UNIVERSITY

- 실습에서 탐색트리를 구현했다면
  - 리밸런싱 관현 함수들만 정의하고 구현하면 됨

- 탐색트리에서 노드 추가 및 제거과정에서 균형이 깨지게 되기 때문에
  - 확장의 형태는 Rebalance(pRoot)로 노드 추가 후 리밸런싱을 함

```
Void BSTInsert(BTreeNode ** pRoot, BSTData data) {
        …
        //Rebalance(pRoot);                 //노드 추가 후 리밸런싱
        *pRoot = Rebalance(pRoot);      //회전과정에서 루트 노드가 변경될 수 있기 때문에 포인터 변수
}

BTreeNode * BSTRemove(BTreeNode ** pRoot, BSTData target) {
        …
        //Rebalance(pRoot);                 //노드 제거 후 리밸런싱
        *pRoot = Rebalance(pRoot);       //회전과정에서 루트 노드가 변경될 수 있기 때문에 포인터 변수
        rturn dNode=;
}
```

높이 비교 및 높이 재기

SUNG KYUN KWAN UNIVERSITY

```
int GetHeightDiff(BTreeNode * bst) {
	int lsh;	//왼쪽 서브트리 높이
	int rsh;	//오른쪽 서브트리 높이

	if (bst == NULL)
		return 0;

	lsh = GetHeight(GetLeftSubTree(bst));
	rsh = GetHeight(GetRightSubTree(bst));

	return lsh - rsh;	//균형 인수
}
```

```
int GetHeight(BTreeNode * bst) {
	int leftH;	//트리 왼쪽 높이
	int rightH;	//트리 오른쪽 높이

	if (bst == NULL)
		return 0;

	leftH = GetHeight(GetLeftSubTree(bst));
	rightH = GetHeight(GetRightSubTree(bst));

	if (leftH > rightH)
		return leftH + 1;
	else
		return rightH + 1;
}
```

LL 회전 및 RR회전

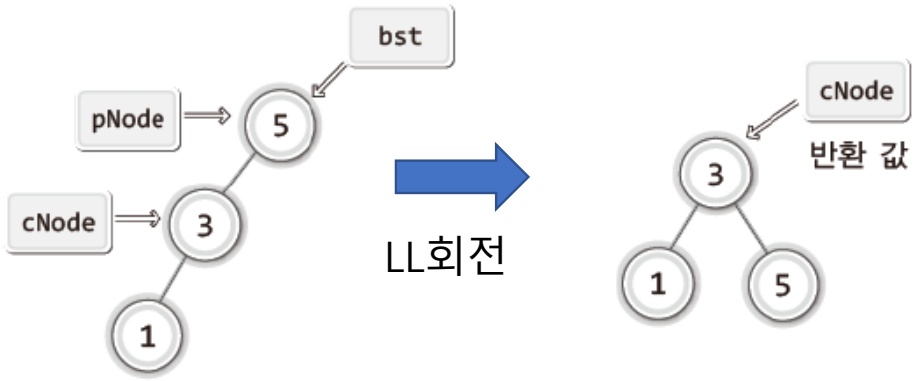
```
BTreeNode * RotateLL(BTreeNode * bst) {
	BTreeNode * pNode;	//부모(parent) node
	BTreeNode * cNode;	//자식(child) node

	pNode = bst;
	cNode = GetLeftSubTree(pNode);

	//실제 LL회전을 담당하는 부분 – 방향만 차이를 갖음
	ChangeLeftSubTree(pNode, GetRightSubTree(cNode));
	ChangeRightSubTree(cNode, pNode);

	//실제 RR회전을 담당하는 부분 – 방향만 차이를 갖음
	ChangeRightSubTree(pNode, GetLeftSubTree(cNode));
	ChangeLeftSubTree(cNode, pNode);

	//LL회전으로 인해 변경된 루트 노드의 주소 값 반환
	return cNode;
}
```



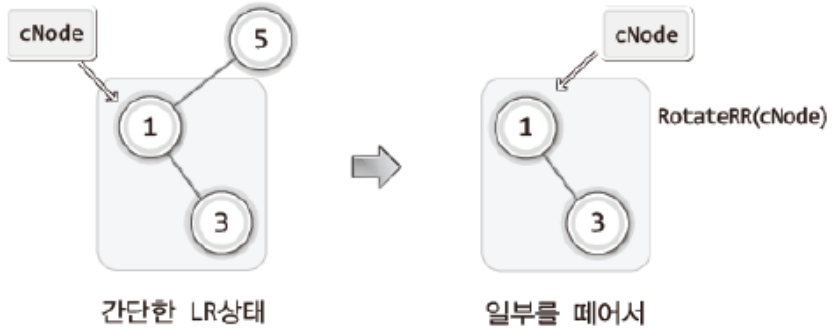
LR 회전 및 RL회전

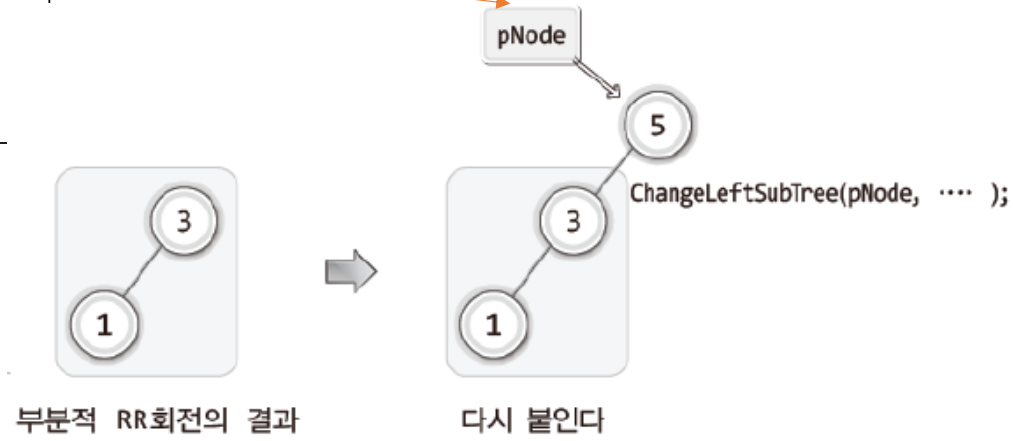
```
BTreeNode * RotateLR(BTreeNode * bst) {
	BTreeNode * pNode;
	BTreeNode * cNode;

	//pNode와 cNode가 LR회전을 위해 적절한 위치를 가리키게 함
	pNode = bst;
	cNode = GetLeftSubTree(pNode);

	//실제 LR회전을 담당하는 두 개의 문장 – 방향만 차이
	ChangeLeftSubTree(pNode, RoateRR(cNode));
	return RotateLL(pNode);

.	//실제 RL회전을 담당하는 두 개의 문장 – 방향만 차이
	ChangeRightSubTree(pNode, RoateLL(cNode));
	return RoateRR(pNode);
}
```





Rebalance 함수

```
BTreeNode * Rebalance(BTreeNode ** pRoot) {
	int hDiff = GetHeightDiff(*pRoot);	//균형 인수 계산

	//균형 인수가 +2 이상이면 LL상태 또는 LR상태
	if (hDiff > 1) {	//왼쪽 서브 트리 방향으로 높이가 2이상 크다면,
		if (GetHeightDiff(GetLeftSubTree(*pRoot)) > 0)
			*pRoot = RotateLL(*pRoot);
		else
			*pRoot = RotateLR(*pRoot);
	}

	//균형 인수가 -2이하이면 RR상태 또는 RL상태
	if (hDiff < -1) {		//오른쪽 서브 트리 방향으로 2이상 크다면,
		if (GetHeightDiff(GetRightSubTree(*pRoot)) < 0)
			*pRoot = RotateRR(*pRoot);
		else
			*pRoot = RotateRL(*pRoot);
	}

	return *pRoot;
}
```