6th assignment

2013011491

컴퓨터전공

안찬영

struct avl\_node\* rotate\_right(struct avl\_node \*parent)

{

struct avl\_node \*child = parent->left\_child;

parent->left\_child = child->right\_child;

child->right\_child = parent;

return child;

}

오른쪽으로 회전시키는 함수이다. child노드를 생성한 뒤 parent 노드의 왼쪽 subtree로 설정한다.

그 다음 parent 노드의 왼쪽 subtree를 child 노드의 오른쪽 subtree로 설정하고, child 노드의 오른쪽 subtree를 parent노드로 설정한 뒤, child노드를 반환한다.

struct avl\_node\* rotate\_right\_left(struct avl\_node \*parent)

{

struct avl\_node \*child = parent->right\_child;

parent->right\_child=rotate\_right(child);

child=rotate\_left(parent);

return child;

}

Parent노드와 오른쪽 subtree 그리고 그 subtree의 왼쪽 노드를 회전시키는 함수이다. Child 노드를 생성한 뒤 parent 노드의 오른쪽 subtree로 설정한다. 그리고 child 노드를 먼저 오른쪽 회전시키고, 그 다음 parent 노드를 왼쪽 회전시킨 뒤 child 노드를 반환한다.

struct avl\_node\* rotate\_left\_right(struct avl\_node \*parent)

{

struct avl\_node \*child = parent->left\_child;

parent->left\_child=rotate\_left(child);

child=rotate\_right(parent);

return child;

}

위의 함수 설명과 반대인 경우이다. Parent 노드와 왼쪽 subtree 그리고 그 subtree의 오른쪽 노드를 회전시키는 함수이다. 설명은 위 함수와 같다고 할 수 있다.

int height\_diff = get\_height\_diff(\*node);

if( height\_diff > 1 ){

if( get\_height\_diff((\*node)->left\_child) > 0 )

\*node = rotate\_right(\*node);

else

\*node = rotate\_left\_right(\*node);

}

재균형이 필요한 노드의 왼쪽 subtree 의 높이가 오른쪽과 2이상 차이가 나고, 재균형이 필요한 노드의 왼쪽 자식 노드 또한 왼쪽 subtree의 높이가 오른쪽과 1이상 차이가 날 경우에는, 오른쪽 단일 회전만 하고, 오른쪽 subtree의 높이가 더 큰 경우에는, left\_right 회전시킨다.

else if ( height\_diff < -1 ){

if( get\_height\_diff((\*node)->right\_child) < 0 )

\*node = rotate\_left(\*node);

else

\*node = rotate\_right\_left(\*node);

}

위 함수 설명과 반대의 경우라고 보면 된다. 재균형이 필요한 노드의 오른쪽 subtree 의 높이가 왼쪽과 2이상 차이가 나는 경우이다.

else if( new\_key < (\*root)->data ){

(\*root)->left\_child=avl\_add(&((\*root)->left\_child), new\_key);

\*root = rebalance(root);

}

Add 함수에서 key값이 탐색중인 data 보다 작을 경우, 왼쪽 자식 노드로 이동해 다시 add함수를 시행해서 위치를 찾는다. 그리고 rebalance 함수를 통해 재균형 시키는 함수이다.

<결과>

C:\Users\Administrator\Desktop\결과.jpg