자료구조론
AVL Tree

u@hataeseong-ui-MacBook-Pro:~/Desktop/2017_CSE2010_2016025041/HW6$./a.out input.txt
((((1)2)3((4)5(6)))7(8(9)))
- 실행 결과 일치

```
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;
}
```
- LL rotation

```
struct avl_node* rotate_left(struct avl_node *parent)
{
   struct avl_node *child = parent->right_child;
   parent->right_child = child->left_child;
   child->left_child = parent;
   return child;
}
```
- RR rotation

```
struct avl_node* rotate_right_left(struct avl_node *parent)
{
   struct avl_node *child = parent->right_child;
   parent->right_child = rotate_right(child);
   return rotate_left(parent);
}
```
- RL rotation(LL rotation -> RR rotation)

```
struct avl_node* rotate_left_right(struct avl_node *parent)
{
   struct avl_node *child = parent->left_child;
```

```
    parent->left_child = rotate_left(child);
    return rotate_right(parent);
}
```
- LR rotation(RR rotation -> LL rotation)

```
int get_height(struct avl_node *node)
{
    int height=0;
    if( node != NULL )
            height = 1 + max(get_height(node->left_child), get_height(node->right_child));
    return height;
}
```
- 가장 큰 height를 구함

```
int get_height_diff(struct avl_node *node)
{
    if( node == NULL ) return 0;
    return get_height(node->left_child) - get_height(node->right_child);
}
```
- height의 차이

```
struct avl_node* rebalance(struct avl_node **node)
{
    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);
    }
    else if ( height_diff < -1 ){
            if( get_height_diff((*node)->right_child) < 0 )
                    *node = rotate_left(*node);
            else
                    *node = rotate_right_left(*node);
    }
    return *node;
}
```
- 차이가 2 이상일 경우 rotate

```
struct avl_node * avl_add(struct avl_node **root, int new_key)
{
```

```
    if( *root == NULL ){
            *root = (struct avl_node *)malloc(sizeof(struct avl_node));
            if( *root == NULL ){
                    exit(1);
            }
            (*root)->data = new_key;
            (*root)->left_child = (*root)->right_child = NULL;
    }
    else if( new_key < (*root)->data ){
            (*root)->left_child = avl_add(&((*root)->left_child), new_key);
            *root = rebalance(root);
    }
    else if( new_key > (*root)->data ){
            (*root)->right_child = avl_add(&((*root)->right_child), new_key);
            *root = rebalance(root);
    }
    else{
            printf("ÁßºÎ¸µÈ Å°\n");
            exit(1);
    }
    return *root;
}
```

- 새로운 노드 생성 후 추가한뒤 rebalance

```
void avl_delete(struct avl_node **root, int key){
    ~~~~~~~~~~~~~~
    avl_delete_balance(root);
}
```

- binary search tree랑 같은 방식으로 delete 먼저 실행 후 balance 처리

```
void avl_delete_balance(struct avl_node **node){
    if(*node == NULL)
            return ;
    else{
            //using recursive rebalance at all node
            *node = rebalance(node);
            if((*node)->left_child != NULL)
                    avl_delete_balance(&(*node)->left_child);
            if((*node)->right_child != NULL)
                    avl_delete_balance(&(*node)->right_child);
    }
}
```

- 재귀형식을 통해 각각의 노드에서 전부 rebalance 실행