자료구조론

Binary Search Tree

```
u@hataeseong-ui-MacBook-Pro:~/Desktop/2017_CSE2010_2016025041/HW5$./a.out input.txt
(((1)2(3))4((5)6))
key is in the tree: 3
key is not in the tree: 7
(((1)2(3))4(6))
u@hataeseong-ui-MacBook-Pro:~/Desktop/2017_CSE2010_2016025041/HW5$
```

- 실행 결과 : 일치

```
void display(TreeNode *p)
{
	if (p != NULL) {
		printf("(");
		display(p->left);
		printf("%d", p->key);
		display(p->right);
		printf(")");
	}
}
```

- 출력 함수 : 재귀함수를 사용하여 구현

```
TreeNode *search(TreeNode *node, int key)
{
	while(node != NULL){
		if(key == node->key)
			return node;
		else if(key < node->key)
			node = node->left;
		else if(key > node->key)
			node = node->right;
	}
	return NULL;
}
```

- NULL이 되기전(마지막 leaf)까지 내려가면서 binary search 실행

```
void insert_node(TreeNode **root, int key)
{
   TreeNode *p, *q; // p is current node, q is parent node
   TreeNode *n;      // n is new node
   p = *root;
   q = NULL;

   //search if key is in the node or not
   while (p != NULL){
      if( key == p->key) return;
      q = p;
      if( key < p->key ) p = p->left;
      else p = p->right;
   }

   // key isn't in the tree alloc new node and insert the key
   n = (TreeNode *) malloc(sizeof(TreeNode));
   if(n == NULL)
            return;

   // set the data
   n->key = key;
   n->left = n->right = NULL결;

   //link with parent node
  if(q != NULL)
      if( key < q->key ) q->left = n;
      else q->right = n;
  else *root = n;
}
```

- 키 값이 트리에 없으면 binary search 후 그 위치에 새로운 노드 생성 후 연결

```
int delete_node(TreeNode *node, int key)
{
            TreeNode *Current = node;
            TreeNode *Par_Parent = NULL;
            TreeNode *Parent = NULL;
            TreeNode *Child = NULL;
            TreeNode *Tmp_left = NULL;
```

```
TreeNode *Tmp_right = NULL;

int deleted = 0;

// go to the node with the key
while(Current->key != key){
        if(key > Current->key){
                Par_Parent = Parent;
                Parent = Current;
                Current = Current->right;
        }
        else{
                Par_Parent = Parent;
                Parent = Current;
                Current = Current->left;
        }
}

// if the node has no child
if(Current->left == NULL && Current->right == NULL){
        deleted = Current->key;
        if(Parent->left == Current)
                Parent->left = NULL;
        if(Parent->right == Current)
                Parent->right = NULL;
        return deleted;
}

// if the node has first child
if(Current->left == NULL || Current->right == NULL){
        Child = (Current->left != NULL) ? Current->left : Current->right;
        // check current node is left node or right node of the child
        // and link the child to the parent node
        if(Parent->left == Current)
                Parent->left = Child;
        else
                Parent->right = Child;
        deleted = Current->key;
        return deleted;
}
```

```
        // if the node has second children
        if(Current->left != NULL && Current->right != NULL){
                Tmp_right = Current->right;
                // go to right node and there is no left child
                if(Tmp_right->left == NULL){
                        Tmp_left = Current->left;
                        Child = Tmp_right;
                        if(Parent->right == Current){
                                Parent->right = Child;
                                Child->left = Tmp_left;
                        }
                        else if(Parent->left == Current){
                                Parent->left = Child;
                                Child->left = Tmp_left;
                        }
                        deleted = Current->key;
                        return deleted;
                }
                // go to right node and if there is left child
                // go to the minimum node
                while(Tmp_right->left != NULL){
                         Parent = Tmp_right;
                         Tmp_right = Tmp_right->left;
                }
                Parent->left = NULL;
                Current->key = Tmp_right->key;
        }
        return 0;
}
```

- 밑에 노드가 없을경우, 자식노드 레벨(차수)가 1인 경우, 2단계 이상의 자식이 있을경우로 나누어서
- 자식이 없는경우(= leaf node인 경우) 그냥 삭제
- 자식노드가 1단계(1차) 인경우 자식 노드와 부모노드를 연결 후 삭제
- 자식노드가 2단계(2차) 이상인경우 오른쪽 자식 트리에서 가장 작은 값을 가진 노드를 찾아 그 노드를 지금 현재 위치로 옮겨온 후 자식 연결