<자료구조 코드 모음 Chapter 5_Tree>

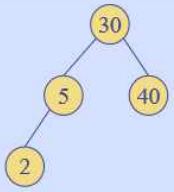
작성자_2018320161_송대선

-Searching a Binary Search Tree (Recursive)-

```
element* search(treePointer root, int k)
{
        /* return a pointer to the element whose key is k,
           if there is no such element, return NULL */

        if ( !root ) return NULL;
        if ( k == root->data.key ) return &(root->data);
        if ( k < root->data.key )
                return search(root->leftChild, k);
        return search( root->rightChild, k );
}
```

Recursive search



-Searching a Binary Search Tree (Iterative)-

```
element* iterSearch(treePointer root, int k)
{
        /* return a pointer to the element whose key is k,
           if there is no such element, return NULL */
        while( tree ) {
                if ( k == root->data.key ) return &(root->data);
                if ( k < root->data.key )
                        tree = tree->leftChild;
                else
                        tree = tree->rightChild;
        }
        return NULL;
}
```

Iterative search

-Inserting into a Binary Search Tree-

```
void insert (treePointer *node, int k, iType theItem)
{
        /* if k is in the tree pointed at by node do nothing;
           otherwise add a new node with data =(k, theItem) */
        treePointer ptr;
        /* searches the binary search tree *node for the key k;
           if the tree is empty or if k is present, returns NULL
           otherwise returns a pointer to the last node of the tree that was encountered during the search */
        treePointer temp = modifiedSearch( *node, k );
        if ( temp || !(*node) ) {
                /* k is not in the tree */
                MALLOC( ptr, sizeof(*ptr) );
                ptr->data.key = k;
                ptr->data.item = theItem;
                ptr->leftChild = prt->rightChild = NULL;

                if ( *node ) /* insert as child of temp */
                        if (k<temp->data.key)
                                temp->leftChild = ptr;
                        else
                                temp->rightChild = ptr;
                else
                        *node = ptr;
        }
}
```

-Splitting Binary Trees-

```
void split(nodePointer *theTree, int k, nodePointer *small, element *mid, nodePointer *big)
{
        /* split the binary search tree with respect to key k */
        if (!theTree ) {
                *small = *big = 0;
                (*mid).key = -1; return ;
        }       /* empty tree */
        nodePointer sHead, bHead;               /* header nodes for small and big, respectively */
        nodePointer s, b, currentNode;
        /* create header nodes for small and big */
        MALLOC( sHead, sizeof(*sHead) );
        MALLOC( bHead, sizeof(*bHead) );
        s = sHead;  b = bHead;

        /* do the split */
        currentNode = *theTree;
        while( currentNode )
                if( k<currentNode->data.key ) { /* add to big */
                        b->leftChild = currentNode;
                        b = currentNode;   currentNode = currentNode->leftChild;
                }
                else if( k>currentNode->data.key) { /* add to small */
                        s->rightChild = currentNode;
                        s = currentNode;   currentNode = currentNode->rightChild;
                }
                else { /* split at currentnode */
                        s->rightChild = currentNode->leftChild;
                        b->leftChild = currentNode->rightChild;
                        *small = sHead->rightChild;   free(sHead);
                        *big = bHead->leftChild;   free(bHead);
                        (*mid).item = currentNode->data.item;
                        (*mid).key = currentNode->data.key;
                        free(currentNode);
                        return;
                }
        /* no pair with key k */
        s->rightChild = b->leftChild = 0;
        *small = sHead->rightChild;   free(sHead);
        *big = bHead->leftChild;   free(bHead);
        (*mid).key = -1;
        return ;
}
```