**5th assignment**

2013011491

컴퓨터 전공

안찬영

TreeNode \*search(TreeNode \*node, int key)

{while(node!=NULL){

if(key==node->key)

return node;

else if(key<node->key)

node=node->left;

else

node=node->right;}

return NULL; }

**탐색 함수입니다. 탐색하는 node가 NULL이 아닐 경우 계속 반복하게 되고, 탐색 성공 할 경우 해당 노드를 반환합니다. 찾고자 하는 키 가 탐색 중인 노드의 키보다 작을 경우 노드는 왼쪽 자식 노드로 설정하고, 클 경우 오른쪽 자식 노드로 설정합니다. 탐색 하지 못 했을 경우 에는 NULL을 반환합니다.**

void insert\_node(TreeNode \*\*root, int key)

{TreeNode\*p, \*q;

TreeNode\*n;

p=\*root;

q=NULL;

while(p!=NULL){

if(key==p->key)

return;

q=p;

if(key<p->key)

p=p->left;

else

p=p->right;}

n=(TreeNode\*)malloc(sizeof(TreeNode));

n->key=key;

n->left=NULL;

n->right=NULL;

if(q==NULL)

\*root=n;

else if(n->key < q->key)

q->left=n;

else

q->right=n; }

**삽입함수 입니다. 먼저 반복문을 통해 삽입될 위치의 부모 노드를 설정합니다. 그 다음 삽입할 노드를 생성 및 데이터를 설정 하고. 트리가 생성되 있지 않을 경우에는 루트 노드로 설정합니다. 트리가 이미 생성되어 있는 경우에는, 키가 먼저 탐색해 놓은 부모 노드의 키보다 작을 경우 왼쪽 subtree로, 반대일 경우 오른쪽 subtree로 설정합니다.**

void delete\_node(TreeNode \*node, int key)

{TreeNode \*d, \*pa, \*min, \*pmin;

d=node;

pa=NULL;

if(search(d,key)==NULL){

printf("key is not in the tree: %d\n",key);

exit(1);

} while(key!=d->key){

pa=d;

if(key<d->key)

d=d->left;

else

d=d->right;}

if(d->left==NULL && d->right==NULL){

if(pa->right==d)

pa->right=NULL;

else

pa->left=NULL;

free(d);

}else if(d->left!=NULL && d->right==NULL){

if(pa->right==d)

pa->right=d->left;

else

pa->left=d->left;

free(d);

}else if(d->left==NULL && d->right!=NULL){

if(pa->right==d)

pa->right=d->right;

else

pa->left=d->right;

free(d);

}else if(d->left!=NULL && d->right!=NULL){

pmin = d;

min=d->right;

while(min->left!=NULL){

pmin=min;

min=min->left;

}

d->key=min->key;

if(min->right!=NULL){

if(pmin==d)

pmin->right=min->right;

else

pmin->left=min->right;

else{

if(pmin==d)

pmin->right=NULL;

else

pmin->left=NULL;

free(min);}}//최소노드 동적할당 해제

**삭제 함수입니다. 먼저 삭제할 데이터가 트리 속에 존재하는지 체크합니다. 없을 경우 에러 메세지와 함께 함수를 탈출합니다. 그 다음 삽입 함수에서처럼 반목문을 통해 삭제할 데이터가 있는 노드와 그 부모 노드를 탐색하고 설정합니다. 삭제할 노드의 유형은 네 가지로 구분 하였습니다.**

**첫 번째는 자식 노드가 아예 없는 경우. 이때는 삭제될 노드가 부모 노드의 오른쪽 노드인지, 왼쪽 노드인지 확인 후 NULL로 설정하고, 메모리를 해제해줍니다.**

**두 번째는 왼쪽 자식 노드만 가지고 있는 경우. 마찬가지로 삭제될 노드가 부모 노드의 오른쪽 노드인지, 왼쪽 노드인지 확인후에 삭제될 노드의 왼쪽 자식 노드로 연결해 준 뒤 메모리를 해제 해줍니다.**

**세 번째의 경우 오른쪽 자식 노드만 가지고 있는 경우. 이 때는 두 번째 경우와 왼쪽, 오른쪽만 반대로 실행합니다.**

**네 번째의 경우 자식 노드들이 모두 있는 경우. 삽입 함수처럼 반복문을 통해 삭제될 노드의 오른쪽 노드에서 최솟값을 보유한 노드와 그 부모 노드를 탐색하고 설정합니다.**

**그 다음 최소 노드의 데이터를 삭제될 노드로 데이터만 수정 해준 뒤, 최소 노드를 메모리 해제 해줍니다. 이때 최소 노드가 오른쪽 자식 노드를 가지고 있을 경우에, 부모 노드의 subtree로 연결해 준 뒤 삭제합니다.**

