자료구조 과제 9

* –

20151523

김동현

- 실행결과



- 코드 및 알고리즘

#define \_CRT\_SECURE\_NO\_WARNINGS

#define MAX\_TERMS 100

#define IS\_FULL(ptr) (!(ptr))

#include <stdlib.h>

#include <stdio.h>

typedef struct treeNode\* treePtr;

typedef struct treeNode {

treePtr lchild;

int data;

treePtr rchild;

}treeNode;

void inorder(treePtr);

void preorder(treePtr);

void insert(treePtr\*, int);

treePtr modifiedSearch(treePtr, int);

void main()

{

int i, n, A[MAX\_TERMS];

treePtr tree = NULL;

FILE\* fp = fopen("input.txt", "r");

fscanf(fp, "%d", &n);

: n에 노드 개수를 저장한다.

for (i = 0; i < n; i++) {

fscanf(fp, "%d", &A[i]);

}

: 배열 A에 노드 데이터를 저장한다.

for (i = 0; i < n; i++)

insert(&tree, A[i]);

inorder(tree);

printf("\n");

preorder(tree);

printf("\n");

}

treePtr modifiedSearch(treePtr tree, int key) {

/\* return a pointer to the node that contains key.

If there is no such node, return NULL. \*/

while (tree) {

if (key == tree->data) return NULL;

if (key < tree->data) {

if (!tree->lchild) {

return tree;

}

else tree = tree->lchild;

}

else {

if (!tree->rchild) {

return tree;

}

else tree = tree->rchild;

}

}

return NULL;

}

: 키가 존재하거나 트리가 존재하지않으면 NULL을 반환한다. 그렇지 않다면 최근 검색했던 트리의 노드를 반환한다.

void insert(treePtr\*node, int num)

{

/\* If num is in the tree pointed at by node do nothing;

otherwise add a new node with data = num\*/

treePtr ptr, temp = modifiedSearch(\*node, num);

if (temp || !(\*node)) {

: 넣고자 하는 num이 tree에 없을 때 (Binary search tree의 노드 key는 유일하므로)

ptr = (treePtr)malloc(sizeof(treeNode));

if (IS\_FULL(ptr)) {

fprintf(stdout, "The memory is full");

exit(1);

}

: 에러체크

ptr->data = num;

ptr->lchild = ptr->rchild = NULL;

if (\*node) /\* insert as child of temp \*/

if (num < temp->data)

temp->lchild = ptr;

else temp->rchild = ptr;

: num이 더 작으면 temp의 왼쪽 자식으로, 더 크면 오른쪽 자식으로 지정한다. (Binary search tree의 정의에 의해)

else \*node = ptr;

}

}

void inorder(treePtr ptr) {

if (ptr) {

inorder(ptr->lchild);

printf("%d ", ptr->data);

inorder(ptr->rchild);

}

}

: inorder로 서브트리에 접근하는 방식이다. Inorder는 LVR순이므로 먼저 lchild로 계속 inorder를 재귀호출하여 Null이 나올 때까지 가서 첫 데이터를 출력 후, rchild가 존재하면 다시 재귀호출 후 반복, 없는 경우 call한 위치로 돌아가서 data 출력 후 반복하는 방식이다.

void preorder(treePtr ptr) {

if (ptr) {

printf("%d ", ptr->data);

preorder(ptr->lchild);

preorder(ptr->rchild);

}

}

: preorder로 서브트리에 접근하는 방식이다. Preorder는 VLR순이므로 먼저 root에서부터 데이터를 출력하고 lchild를 root로 하는 서브트리를 preorder로 재귀호출하여 lchild를 따라 Null을 만날 때까지 data를 출력해주고, rchild를 접근하는 방식이다.