BST의 삽입 함수 끝에 AVL BALANCE하는 함수 추가.

AVL BALANCE 함수는 다음과 같음

알고리즘)

/\*

1. find the node which needs to be balanced(and its parent)

2. judge balnce type : LL/RR/LR/RL

3. a)if LL -> do LL

b)if RR - > do RR

c)if LR -> do double rotation(RR then LL)

d)if RL - >do double rotation(LL then RR)

\*/

FUNCTION ) //1. FIND

//1. find ...

struct Node\* target = \*root;

struct Node\* targetParent = NULL;

struct Node\* targetGrandParent = NULL;

struct Node\* targetChild = NULL;

char judge[3] = {'\n'};

int needBalance = false;

for( needBalance = false; needBalance ==false && strcmp(target->key, key)!=0 ; ){

targetGrandParent = targetParent;

targetParent = target;

if (abs( getMaxDepth( target->leftChild) - getMaxDepth( target->rightChild) ) >1 )

needBalance = true;

if (strcmp( target->key, key) >0 )

target = target->leftChild;

else

target = target->rightChild;

}

printf("재균형 필요 : ");

if(needBalance == true)

printf("Yes\n");

else{

printf("No\n");

return ;

}

while(abs( getMaxDepth( target->leftChild) - getMaxDepth( target->rightChild) ) > 1){

targetGrandParent = targetParent;

targetParent = target;

if (strcmp( target->key, key) >0 )

target = target->leftChild;

else

target = target->rightChild;

}

target = targetParent;

targetParent = targetGrandParent;

FUNCTION ) //2. JUDGE

//2. judge ...

if( getMaxDepth(target->leftChild) > getMaxDepth(target->rightChild) ) {//LL or LR

targetChild = target->leftChild;

judge[0] = 'L';

}

else{ //RR or RL

targetChild = target->rightChild;

judge[0] = 'R';

}

if( getMaxDepth(targetChild->leftChild) > getMaxDepth(targetChild->rightChild) )

judge[1] = 'L';

else

judge[1] = 'R';

printf("재균형 유형 : %s\n", judge);

FUNCTION ) //3. IF NEED, DO ROTATION

//3. do rotation

if (strcmp(judge, "LL") ==0){

if(\*root == target)

DoLLRotation(root, &targetParent);

else

DoLLRotation(&target, &targetParent);

}

else if(strcmp(judge, "RR") ==0){

if(\*root == target)

DoRRRotation(root, &targetParent);

else

DoRRRotation(&target, &targetParent);

}

else if(strcmp(judge, "LR") ==0){

if(\*root == target){

DoRRRotation(&targetChild, root);

DoLLRotation(root, &targetParent);

}

else {

DoRRRotation(&targetChild, &target);

DoLLRotation(&target, &targetParent);

}

}

else if(strcmp(judge, "RL") ==0){

if(\*root == target){

DoLLRotation(&targetChild, root);

DoRRRotation(root, &targetParent);

}

else {

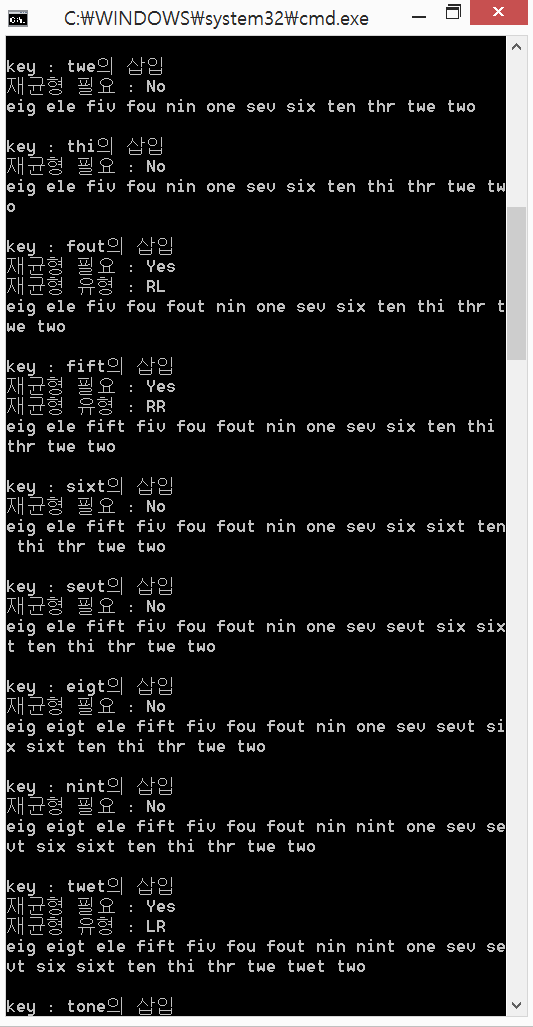
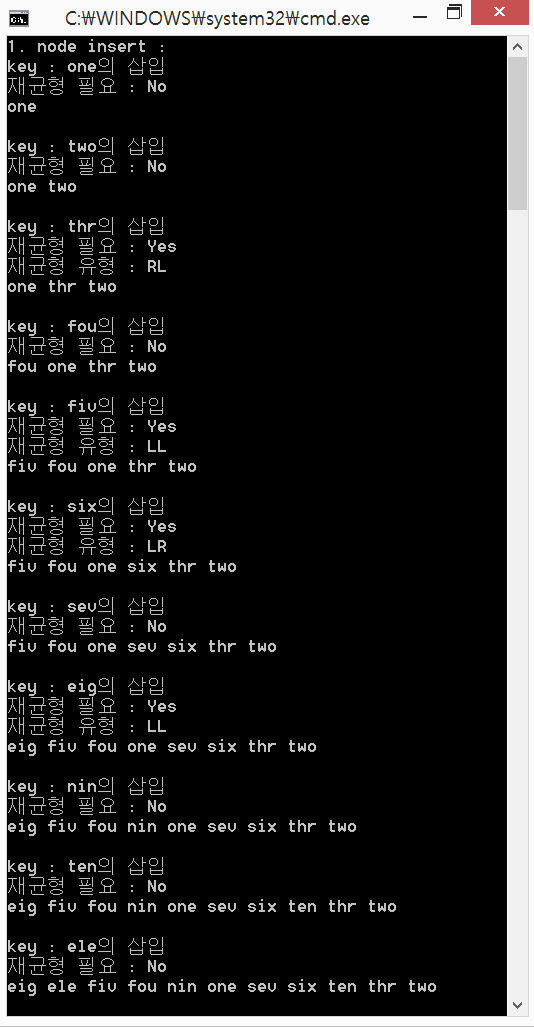
DoLLRotation(&targetChild, &target);

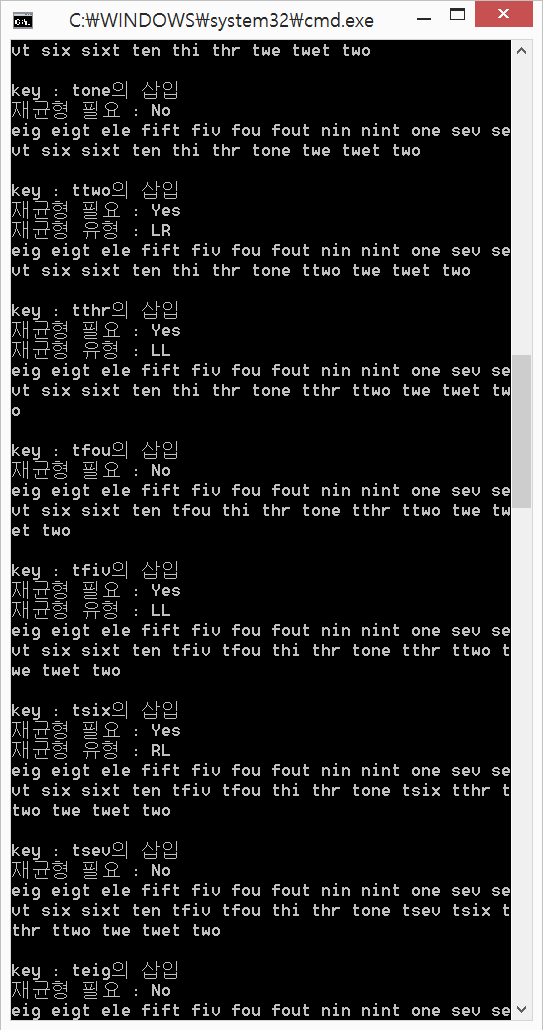
DoRRRotation(&target, &targetParent);

}

}

실행 결과





전체 SOURCE

SOURCE.C

#include "FS.h"

void main(){

int i=0;

struct Node\* root = NULL;

printf("1. node insert : \n");

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

insertNode(&root, keyArr[i]);

printTree(root);

printf("\n\n");

}

printf("\n");

}

SOURCE.H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define KEYNUM 30

#define MAXLEN 5

#define true 1

#define false 0

struct Node{

struct Node\* leftChild;

struct Node\* rightChild;

char key[MAXLEN];

};

void allocNode(struct Node\*\* node, char\* key){

(\*node)->leftChild = NULL;

(\*node)->rightChild = NULL;

strcpy((\*node)->key, key);

}

char keyArr[KEYNUM][MAXLEN] = {

"one",

"two",

"thr",

"fou",

"fiv",

"six",

"sev",

"eig",

"nin",

"ten",

"ele",

"twe",

"thi",

"fout",

"fift",

"sixt",

"sevt",

"eigt",

"nint",

"twet",

"tone",

"ttwo",

"tthr",

"tfou",

"tfiv",

"tsix",

"tsev",

"teig",

"tnin",

"thit"

};

int getMaxDepth(const struct Node\* node){

int leftDepth, rightDepth;

if(node==NULL)

return 0;

leftDepth = getMaxDepth(node->leftChild);

rightDepth = getMaxDepth(node->rightChild);

if(leftDepth>rightDepth)

return leftDepth+1;

else

return rightDepth+1;

}

void searchNode(char \*key, struct Node\*\* p, struct Node \*\*q){

if( (\*p) ==NULL) //node is null

return ;

else if( strcmp((\*p)->key, "") == 0 ) //root is empty

return ;

else if ( (\*p)!=NULL && strncmp(key, (\*p)->key, MAXLEN ) == 0) //rootkey == key

return ;

else if( (\*p)!=NULL && strncmp(key, (\*p)->key, MAXLEN ) < 0) {

\*q=\*p;

\*p = (\*p)->leftChild;

return searchNode(key, p, q);

}

else {

\*q=\*p;

\*p = (\*p)->rightChild;

return searchNode(key, p, q);

}

}

void DoLLRotation(struct Node\*\* node, struct Node\*\* parent){

struct Node\* left = (\*node)->leftChild;

(\*node)->leftChild = left->rightChild;

left->rightChild = \*node;

if(\*parent==NULL)

\*node = left;

else if( strcmp((\*parent)->key, (\*node)->key ) > 0)

(\*parent)->leftChild = left;

else

(\*parent)->rightChild = left;

}

void DoRRRotation(struct Node\*\* node, struct Node\*\* parent ){

struct Node\* right = (\*node)->rightChild;

(\*node)->rightChild = right->leftChild;

right->leftChild = (\*node);

if(\*parent==NULL)

\*node = right;

else if( strcmp((\*parent)->key, (\*node)->key ) > 0)

(\*parent)->leftChild = right;

else

(\*parent)->rightChild = right;

}

void balanceAVL(struct Node\*\* root, char key[MAXLEN]){

/\*

1. find the node which needs to be balanced(and its parent)

2. judge balnce type : LL/RR/LR/RL

3. a)if LL -> do LL

b)if RR - > do RR

c)if LR -> do double rotation(RR then LL)

d)if RL - >do double rotation(LL then RR)

\*/

//1. find ...

struct Node\* target = \*root;

struct Node\* targetParent = NULL;

struct Node\* targetGrandParent = NULL;

struct Node\* targetChild = NULL;

char judge[3] = {'\n'};

int needBalance = false;

for( needBalance = false; needBalance ==false && strcmp(target->key, key)!=0 ; ){

targetGrandParent = targetParent;

targetParent = target;

if (abs( getMaxDepth( target->leftChild) - getMaxDepth( target->rightChild) ) >1 )

needBalance = true;

if (strcmp( target->key, key) >0 )

target = target->leftChild;

else

target = target->rightChild;

}

printf("재균형 필요 : ");

if(needBalance == true)

printf("Yes\n");

else{

printf("No\n");

return ;

}

while(abs( getMaxDepth( target->leftChild) - getMaxDepth( target->rightChild) ) > 1){

targetGrandParent = targetParent;

targetParent = target;

if (strcmp( target->key, key) >0 )

target = target->leftChild;

else

target = target->rightChild;

}

target = targetParent;

targetParent = targetGrandParent;

//2. judge ...

if( getMaxDepth(target->leftChild) > getMaxDepth(target->rightChild) ) {//LL or LR

targetChild = target->leftChild;

judge[0] = 'L';

}

else{ //RR or RL

targetChild = target->rightChild;

judge[0] = 'R';

}

if( getMaxDepth(targetChild->leftChild) > getMaxDepth(targetChild->rightChild) )

judge[1] = 'L';

else

judge[1] = 'R';

printf("재균형 유형 : %s\n", judge);

//3. do rotation

if (strcmp(judge, "LL") ==0){

if(\*root == target)

DoLLRotation(root, &targetParent);

else

DoLLRotation(&target, &targetParent);

}

else if(strcmp(judge, "RR") ==0){

if(\*root == target)

DoRRRotation(root, &targetParent);

else

DoRRRotation(&target, &targetParent);

}

else if(strcmp(judge, "LR") ==0){

if(\*root == target){

DoRRRotation(&targetChild, root);

DoLLRotation(root, &targetParent);

}

else {

DoRRRotation(&targetChild, &target);

DoLLRotation(&target, &targetParent);

}

}

else if(strcmp(judge, "RL") ==0){

if(\*root == target){

DoLLRotation(&targetChild, root);

DoRRRotation(root, &targetParent);

}

else {

DoLLRotation(&targetChild, &target);

DoRRRotation(&target, &targetParent);

}

}

}

void insertNode(struct Node\*\* root, char\* key){

struct Node\* p = \*root;

struct Node\* q = \*root;

printf("key : %s의 삽입\n", key);

if(p!=NULL){

searchNode(key, &p, &q);

}

if( \*root == NULL){

\*root = (struct Node\*)calloc(1, sizeof(struct Node));

allocNode(root, key);

}

else if( \*root !=NULL){

if(strncmp(key, q->key, MAXLEN) <0){

q->leftChild = (struct Node\*)calloc(1, sizeof(struct Node));

allocNode(&(q->leftChild), key);

}

else if(strncmp(key, q->key, MAXLEN) >0){

q->rightChild = (struct Node\*)calloc(1, sizeof(struct Node));

allocNode(&(q->rightChild), key);

}

}

balanceAVL(root, key);

}

void printTree(const struct Node\* p){

if( (p->leftChild!=NULL))

printTree(p->leftChild);

printf("%s ", p->key);

if((p->rightChild!=NULL))

printTree(p->rightChild);

}