AVLTREE report

설명하는 곳마다 주석처리(//)를 사용하거나, 화살표(->)를 사용하였습니다.

#include<stdio.h>

#include<stdlib.h>

typedef int element;

//node안에 info는 void pointer를 사용했습니다.

typedef struct node{

void\* info; // info의 void pointer

struct node\* left, \*right; // 왼쪽 child와 오른쪽 child node pointer

int bal; // balance 값

}Node;

typedef struct head{

int count; // node의 개수

Node\* root; // root의 pointer값

int(\*compare) (void\* a, void\* b); // compare 함수

}Head; //AVL tree의 head

//struct는 head와 node 두가지인데, head에는 이 AVL tree가 가지는 기본적인 인자를 넣었습니다.

int compare(void\* a, void\* b);

Head\* AVL\_create(){

Head\* a = (Head\*)malloc(sizeof(Head));

a->count = 0;

a->root = NULL;

a->compare = compare;

return a;

}

//create함수입니다. Head struct를 만든 후 각각을 초기화하였습니다.

int intcompare(int a, int b);

int compare(void\* a, void\* b){

if (intcompare(\*(int\*)a, \*(int\*)b) == 1)

return 1;//true

else

return -1;//false

}

int intcompare(int a, int b){

if (a >= b)

return 1;//true

else

return -1;//false

}

//우선 void pointer이지만, 이 코드 상에서는 int pointer라고 생각했습니다.

그래서 compare함수는 void pointer를 인자로 받아서 int pointer만큼 concatenate한 후 intcompare 함수를 이용해서 compare를 했습니다. 이 function pointer 값을 위에 Head struct에 넣었습니다.

Node\* createnode(void\* element){

Node\* newnode = (Node\*)malloc(sizeof(Node));

if (newnode = NULL)

return NULL;

newnode->bal = 0;

newnode->info = element;

newnode->left = newnode->right = NULL;

return newnode;

} // 새로운 node를 만드는 함수

//Node를 생성한 후 각각 초기화 하는 함수입니다.

Node\* rotateRight(Node\* root){

Node\* tmp = root->left;

root->left = tmp->right;

tmp->right = root;

return tmp;

}

Node\* rotateLeft(Node\* root){

Node\* tmp = root->right;

root->right = tmp->left;

tmp->left = root;

return tmp;

}

//Rotate함수들입니다. Right는 오른쪽으로, Left는 왼쪽으로 rotate하는 함수입니다.

그래서 새로운 root를 반환합니다.

Node\* rightChange(Node\* root, int\* fix){ // fix여부를 묻는 pointer도 같이 받습니다.

switch (root->right->bal){

case 1: //RL rotation

switch (root->right->left->bal){

//RL rotation이므로, root, root->right, root->right->left 세가지의 balance값이 변합니다. 그래서 후자의 balance에 따라 각각의 balance값들을 조정해줍니다.

case 1:

root->bal = 1;

root->right->bal = 0;

break;

case 0: //

root->bal = 0;

root->right->bal = 0;

break;

case -1:

root->bal = 0;

root->right->bal = -1;

break;

}

root->right->left->bal = 0;

root->right = rotateRight(root->right);

root = rotateLeft(root);

\*fix = 0;

//실제로 RL rotation을 행합니다. 고쳐졌으므로 fix를 0으로 바꿔줍니다.

break;

case 0: //error

printf("Error\n");

\*fix = 0;

break;

//이런 경우는 존재할 수 없으므로 error입니다.

case -1://RR rotation

root->bal = 0;

root->right->bal = 0;

root = rotateRight(root);

\*fix = 0;

break;

}

return root;

}

Node\* changeLeft(Node\* root, int\* fix){

switch (root->left->bal){

case 1://LL rotation을 행합니다.

root->bal = 0;

root->left->bal = 0;

root = rotateRight(root);

\*fix = 0;

break;

case 0://존재할 수 없는 경우이므로 error입니다.

printf("error\n");

\*fix = 0;

break;

case -1://LR rotation을 행합니다. 그리고 각각 해당되는 bal값을 switch를 써서 전부 수정해줍니다.

switch (root->left->right->bal){

case 1:

root->bal = -1;

root->left->bal = 0;

break;

case 0:

root->bal = 0;

root->left->bal = 0;

break;

case -1:

root->bal = 0;

root->left->bal = 1;

break;

}

root->left->right->bal = 0;

root->left = rotateLeft(root->left);

root = rotateRight(root);

\*fix = 0;

break;

}

return root;

}

//Rightchange와 changeLeft함수는 각각 AVL\_insert를 할 때, balance값을 조정해주고, 상황에 맞춰 RR,RL,LR,LL rotation을 해주는 함수입니다. 자세한 설명은 주석처리를 붙여 하였습니다.

Node\* AVL\_insert(Head\* AVL, Node\* root, void\* element, int\* fix){

if (root == NULL){

root = createnode(element);

\*fix = 1; //fix가 1인 경우는 수정해주고, 0인 경우는 수정하지 않습니다.

AVL->count++;

return root;

}

//root가 NULL일 경우 안에 들어있는 게 없다는 뜻이므로, root를 create해준 후 반환합니다.

if (compare(element, root->info) == 1){

//element가 더 큰 경우이므로, element는 right tree로 들어갑니다.

root->right = AVL\_insert(AVL,root->right, element, fix);

if (\*fix = 1){

switch (root->bal){

case 1: //현재 LH이므로, 지금 EH 가 되었다!

root->bal = 0;

\*fix = 0;

break;

case 0: //현재 EH이므로, 지금 RH가 되었다!

root->bal = -1;

break;

case -1://현재 RH이므로, 지금 -2가 되었다!

root = rightChange(root,fix);

//이 내부에서 bal 조정하고 rotation을 진행해줍니다.

break;

}

}

return root;

}

//모든게 끝난 후 root를 반환합니다.

else{//element가 root info보다 작음 ->left 로 insert되게 됩니다.

root->left = AVL\_insert(AVL,root->left, element, fix);

if (\*fix == 1){

switch (root->bal){

case 1://LH 2단계

root = changeLeft(root, fix);

//changeLeft함수 내에서 bal값 조정하고 rotation 해줍니다.

break;

case 0://LH가 됩니다.

root->bal = 1;

break;

case -1://EH가 됩니다. 그러므로 fix 수정

root->bal = 0;

\*fix = 0;

break;

}

}

return root;

}

}

Node\* balRight(Node\* root, int\* shorter){//left가 제거된 상황입니다.

Node\* righttree = NULL;

Node\* lefttree = NULL;

switch (root->bal){

case 1:

root->bal = 0;

break;

case 0:

//지금은 righthigh입니다.

root->bal = -1;

break;

case -1:

//지금은 bal값이 -2되어서 낮춰줘야 합니다. Leftrotate를 해줍니다.

righttree = root->right;

if (righttree->bal == 1){

//이 경우는 RLrotation이 필요합니다.

lefttree = righttree->left;

switch (lefttree->bal){

case 1:

righttree->bal = -1;

root->bal = 0;

break;

case 0:

root->bal = 0;

righttree->bal = 0;

break;

case -1:

root->bal = 1;

righttree->bal = 0;

break;

}

lefttree->bal = 0;

root->right = rotateRight(righttree);

root = rotateLeft(root);

}

else{ // 나머지 경우는 왼쪽으로 한번만 rotation시켜 주면 됩니다.

switch (righttree->bal){

case 1:

printf("error");

// 위에서 걸러져야 할 경우라 error처리 하였습니다.

break;

case 0:

righttree->bal = -1;

root->bal = 1;

\*shorter = 0;

break;

case -1:

root->bal = 0;

righttree->bal = 0;

break;

}

root = rotateLeft(root);

}

return root;

}

}

Node\* balLeft(Node\* root, int\* shorter){//right가 제거된 상황

Node\* righttree = NULL;

Node\* lefttree = NULL;

switch (root->bal){

case 1:

//이 경우 2가 되어 left high가 된다

lefttree = root->left;

if (lefttree->bal == -1){

//LRroation이 필요합니다. Switch를 써서 bal 값을 재조정해준 후에 rotation을 시켜줍니다.

righttree = lefttree->right;

switch (righttree->bal){

case 1:

root->bal = 0;

lefttree->bal = 1;

break;

case 0:

root->bal = 0;

lefttree->bal = 0;

break;

case -1:

root->bal = -1;

lefttree->bal = 0;

break;

}

righttree->bal = 0;

root->left = rotateLeft(root->left);

root = rotateRight(root);

}

else{//나머지 경우일때는 오른쪽으로 한번만 rotation시켜 줍니다.

switch (lefttree->bal){

case 1:

root->bal = 0;

lefttree->bal = 0;

break;

case 0:

lefttree->bal = -1;

root->bal = 1;

break;

case -1:

printf("error");

break;

}

root = rotateRight(root);

}

case 0:

root->bal = 1;

\*shorter = -1;

break;

case -1:

root->bal = 0;

break;

}

return root;

}

Node\* AVL\_delete(Head\* AVL, Node\* root, void\* info, int\* shorter, int\* success){

if (root == NULL){

\*shorter = 0;

\*success = 0;

return NULL;

}

//root값이 없으면, 한마디로 AVL tree안에 아무것도 없으면 NULL값을 반환합니다.

if (AVL->compare(info, root->info) == 1){ //info가 root값보다 더 크면 오른쪽으로 가서 delete 진행해줍니다.

root->right = AVL\_delete(AVL, root->right, info, shorter, success);

if (\*shorter)//만약 shorter가 0이 아니라면, (아마 바로 위에 node에서 걸릴 것입니다.)

root = balLeft(root, shorter);

//오른쪽에서 제거되었을 것이므로, balLeft함수를 통해 balance값을 조정해주고, rotation시켜 줍니다.

}

else if (AVL->compare(info, root->info) == -1){

root->left = AVL\_delete(AVL, root->left, info, shorter, success);

if (\*shorter)

root = balRight(root, shorter);

}//위와 반대의 경우입니다.

else{

//이 경우에는 root의 info가 주어진 info와 같을 경우, 한마디로 deletion해줘야 할 경우입니다.

Node\* delPtr = root; // 지울 포인터

if (root->right == NULL){ //root의 오른쪽에 아무것도 없다면

Node\* newroot = root->left;//바로 새로운 root는 왼쪽것이 됩니다.

\*success = 1;//성공

\*shorter = 1;//bal검사가 필요합니다.

free(delPtr);

return newroot;

}

else if (root->left == NULL){//root의 왼쪽에 아무것도 없다면

Node\* newroot = root->right;//위와 반대의 상황이 벌어집니다.

\*success = 1;

\*shorter = 1;

free(delPtr);

return newroot;

}

else{

//지워야 할 root가 왼쪽과 오른쪽을 모두 가지고 있을 경우

Node\* exchPtr = root->left;

while (exchPtr != NULL)

exchPtr = exchPtr->right;

//root의 왼쪽 node에서, 가장 오른쪽으로 갑니다. 한마디로 root보다 작지만 제일 큰 값이 들어있는 node를 찾습니다.

root->info = exchPtr->info;

root->left = AVL\_delete(AVL, root->left, exchPtr->info, shorter, success);

//root에 그 info를 복사한 후, 그 node로 가서 지웁니다.(한마디로 exchange)

if (\*shorter)

root = balRight(root,shorter);

//balance검사를 수행하고 rotation합니다.

}

}

return root;

}

int AVL\_search(Node\* root, void\* element){

if (\*(int\*)(root->info) == \*(int\*)element){

return 1;

}

else if (compare(root, element) == 1){

AVL\_search(root->left, element);

}

else

AVL\_search(root->right, element);

if (root == NULL)

return 0;

}

//주어진 element를 info로 갖는 node를 찾는 재귀 함수입니다. 그래서 있다면 1, 없으면 0을 출력합니다.

int AVL\_count(Head\* AVL){

return AVL->count;

}

//AVL속에 node가 몇 개 있는지 return하는 함수입니다.

int AVL\_empty(Head\* AVL){

if (AVL->count == 0)

return 1;

else return 0;

}

//AVL이 empty라면 1 아니면 0

void AVL\_printTree(Node\* root){

if (root == NULL)

return;

AVL\_printTree(root->left);

printf("%d", \*(int\*)root->info);

AVL\_printTree(root->right);

}//left, root, right순으로 재귀적으로 출력하는 함수입니다.