

CS 2103

Assignment in Data Structures # 9

(**BST, AVL and 2-3 Tree**)

Submitted by:

**Robert Michael Lim, BSCS – 2**

Submitted to:

**Ms. Christine F. Peña**

typedef enum{TRUE, FALSE}Boolean;

typedef struct node{

int elem;

struct node \*left, \*right;

}\*BST, BSTType;

void insert(BST \*B, int x)

{

if((\*B)==NULL){

(\*B) = (BST)malloc(sizeof(BSTType));

(\*B)->elem = x;

(\*B)->left = (\*B)->right = NULL;

}else{

if(x < (\*B)->elem){

insert(&(\*B)->left, x);

}else if(x > (\*B)->elem){

insert(&(\*B)->right, x);

}

}

}

void deleteBST(BST \*B, int x)

{

BST temp;

if((\*B)==NULL){

\*B = NULL;

}else if(x < (\*B)->elem){

deleteBST(&(\*B)->left, x);

}else if(x > (\*B)->elem){

deleteBST(&(\*B)->right, x);

}else{

if((\*B)->left == NULL && (\*B)->right == NULL){

\*B = NULL;

}else if((\*B)->left == NULL){

temp = (\*B);

(\*B) = (\*B)->right;

free(temp);

}else if((\*B)->right == NULL){

temp = (\*B);

(\*B) = (\*B)->left;

free(temp);

}else{

temp = findMin((\*B)->right);

(\*B)->elem = temp->elem;

deleteBST(&(\*B)->right, temp->elem);

}

}

}

BST findMin(BST B)

{

if(B->left==NULL){

return B;

}else{

return findMin(B->left);

}

}

Boolean isMember(BST B, int x)

{

if(B==NULL){

return FALSE;

}

if(B->elem == x){

return TRUE;

}

if(x < B->elem){

return isMember(B->left, x);

}else{

return isMember(B->right, x);

}

}

/\*2-3 Tree\*/

#define SENTINEL 9999999

typedef enum {TRUE,FALSE}Boolean;

typedef enum{LEAF,INODE}nodeType;

typedef struct L{

char name[30];

unsigned long ID;

}Leaf;

typedef struct node{

nodeType indicator;

union{

Leaf L;

struct{

unsigned long key1, key2;

struct node \*ptr[3];

}Inode;

}U;

}\*TTT, TwoThreeTree;

Boolean isMember(TTT S, unsigned long x)

{

if(S->indicator!=LEAF){

if(x < S->U.Inode.key1){

return isMember(S->U.Inode.ptr[0],x);

}else if(x < S->U.Inode.key2){

return isMember(S->U.Inode.ptr[1],x);

}else if(x > S->U.Inode.key2){

return isMember(S->U.Inode.ptr[2],x);

}

}

return S->U.L.ID == x ? TRUE: FALSE;

}

/\*AVL TREE\*/

typedef struct node{

int elem;

struct node \*left, \*right;

int height;

}\*BST, nodeType;

void rightRotate(BST \*y)

{

BST x = (\*y)->left;

BST T2 = x->right;

x->right = (\*y);

(\*y)->left = T2;

(\*y)->height = max(height((\*y)->left), height((\*y)->right))+1;

x->height = max(height(x->left), height(x->right))+1;

}

void leftRotate(BST \*x)

{

BST y = (\*x)->right;

BST T2 = y->left;

y->left = (\*x);

(\*x)->right = T2;

(\*x)->height = max(height((\*x)->left), height((\*x)->right))+1;

y->height = max(height(y->left), height(y->right))+1;

}

void insert(BST \*B, int x)

{

if((\*B)==NULL){

(\*B) = (BST)malloc(sizeof(nodeType));

(\*B)->elem = x;

(\*B)->left = NULL;

(\*B)->right = NULL;

(\*B)->height = 1;

}

if(x < (\*B)->elem){

insert(&(\*B)->left, x);

}else if(x > (\*B)->elem){

insert(&(\*B)->right, x);

}

(\*B)->height = 1+max(height((\*B)->left), height((\*B)->right));

int balance =getBalance((\*B)) ;

printf("BALANCE: %d \n",getBalance((\*B)));

if(balance > 1 && x < (\*B)->left->elem){

rightRotate(B);

}

if(balance < -1 && x > (\*B)->right->elem){

leftRotate(B);

}

if(balance > 1 && x > (\*B)->left->elem){

leftRotate(&(\*B)->left);

rightRotate(B);

}

if(balance < -1 && x < (\*B)->right->elem){

rightRotate(&(\*B)->right);

leftRotate(B);

}

}

int max(int x, int y)

{

return (x > y)?x:y;

}

int height(BST B)

{

if(B==NULL){

return 0;

}

return B->height;

}

int getBalance(BST B)

{

if(B==NULL){

return 0;

}

return height(B->left) - height(B->right);

}

During the time I made this code, it was fairly easy because the BST and the 2-3 Trees were discussed last semester. On the other hand, the AVL Tree was the challenging part for the syntax the source I used was without curly braces. I ran into trouble with the AVL but with the help of my friends, I was able to find the balance factors for this and made improvements since last year. ☺