AVL.h

#ifndef \_\_AVLTREE\_H\_\_

#define \_\_AVLTREE\_H\_\_

typedef struct avl\_node

{

struct avl\_node \*left;

struct avl\_node \*right;

int height;

} avl\_node\_t;

avl\_node\_t \* insert(avl\_node\_t \* avl, void \*element, int(\*compare)(void \*elementA,void \*elementB), void(\*print)(avl\_node\_t \* elementA));

avl\_node\_t \* Delete(avl\_node\_t \* root, void \* element, int (\*compare)(void \* elementA, void \* elementB)/\*, void(\*replace)(void \* elementA, void \* elementB)\*/);

avl\_node\_t \*find(avl\_node\_t \* avl, void \*element,int(\*compare)(void \*elementA, void \*elementB));

avl\_node\_t \* MaxNode(avl\_node\_t \* avl);

avl\_node\_t \* MinNode(avl\_node\_t \* avl);

static avl\_node\_t \*balanceNode(avl\_node\_t \*avl);

static avl\_node\_t \*rotateright(avl\_node\_t \* avl);

static avl\_node\_t \*rotateleft(avl\_node\_t \* avl);

static avl\_node\_t \*RR(avl\_node\_t \* avl);

static avl\_node\_t \*LL(avl\_node\_t \* avl);

static avl\_node\_t \*LR(avl\_node\_t \* avl);

static avl\_node\_t \*RL(avl\_node\_t \* avl);

static int height( avl\_node\_t \* avl);

static int balanceFactor(avl\_node\_t \* avl);

#endif

AVL.c

#include<stdio.h>

#include<stdlib.h>

#include <string.h>

#include "AVL.h"

avl\_node\_t \* insert(avl\_node\_t \* avl,void \*element, int(\*compare)(void \*elementA, void \*elementB),void(\*print)(avl\_node\_t \* elementA))

{

if(avl == NULL)

{

avl = (avl\_node\_t \*)element;

return avl;

}

if(compare(avl, element) == 1)

avl->left = insert(avl->left, element, compare, print);

else if(compare(avl, element) == -1)

avl->right = insert(avl->right, element, compare, print);

else if(compare(avl, element) == 0)

{

puts("already have this data");

return NULL;

}

avl = balanceNode(avl);

return avl;

}//跟一般的插法一樣, 插完後做balance

avl\_node\_t \* MinNode(avl\_node\_t \* avl)

{

if(avl == NULL || avl->left == NULL)

return avl;

return MinNode(avl->left);

}

avl\_node\_t \* MaxNode(avl\_node\_t \* avl)

{

if(avl == NULL || avl->right == NULL)

return avl;

return MaxNode(avl->right);

}

avl\_node\_t \* Delete(avl\_node\_t \* avl, void \* element, int (\*compare)(void \* elementA, void \* elementB)/\*, void(\*replace)(void \* elementA, void \* elementB)\*/)

{

if(avl == NULL)

{

puts("No data");

return NULL;

}

if(compare(avl, element) == 1)

{

avl->left = Delete(avl->left, element, compare/\*, replace\*/);

}

else if(compare(avl, element) == -1)

{

avl->right = Delete(avl->right, element, compare/\*, replace\*/);

}

else

{

if(avl->left != NULL && avl->right != NULL)

{

if(balanceFactor(avl) <= -1)

{

avl\_node\_t \*cur = avl;

cur->left = avl->left;

cur->right = avl->right;

avl\_node\_t \*tmp = MinNode(avl->right);

avl = tmp;

avl->left = cur->left;

cur->right = cur->right;

avl->right = Delete(avl->right, tmp, compare/\*, replace\*/);

}

else

{

avl\_node\_t \*cur = avl;

cur->left = avl->left;

cur->right = avl->right;

avl\_node\_t \*tmp = MaxNode(avl->left);

avl = tmp;

avl->left = cur->left;

cur->right = cur->right;

avl->left = Delete(avl->left, tmp, compare/\*, replace\*/);

}

}

else

{

avl\_node\_t \*tmp = avl;

avl = avl->left ? avl->left : avl->right;

free(tmp);

}

}

balanceNode(avl);

return avl;

}//跟一般的刪法一樣, 刪除完後做balance

avl\_node\_t \*find(avl\_node\_t \*avl , void \*element, int(\*compare)(void \*elementA, void \*elementB))

{

if(avl == NULL || element == NULL)

{

puts("NO data");

return NULL;

}

if(compare(avl, element) == 0)

return avl;

else if(compare(avl, element) == 1)

return find(avl->left, element, compare);

else if(compare(avl, element) == -1)

return find(avl->left, element, compare);

}//跟一般的找法一樣, 當走到空的時候則沒data回傳NULL

static avl\_node\_t \*rotateright(avl\_node\_t \*avl)

{

avl\_node\_t \*bvl;

bvl = avl->left;

avl->left = bvl->right;

bvl->right=avl;

avl->height = height(avl);

bvl->height = height(bvl);

return bvl;

}}//把左子抬高上面放下

static avl\_node\_t \*rotateleft(avl\_node\_t \*avl)

{

avl\_node\_t \*bvl;

bvl = avl->right;

avl->right = bvl->left;

bvl->left=avl;

avl->height = height(avl);

bvl->height = height(bvl);

return bvl;

}//把右子抬高上面放下

static avl\_node\_t \*balanceNode(avl\_node\_t \*avl)

{

if(avl == NULL)

return NULL;

if(avl->left != NULL)

avl->left = balanceNode(avl->left);

if(avl->right != NULL)

avl->right = balanceNode(avl->right);

if(balanceFactor(avl) >= 2)

{

if(balanceFactor(avl->left) >= 1)

return LL(avl);

else

return LR(avl);

}

else if(balanceFactor(avl) <= -2)

{

if(balanceFactor(avl->right) <= -1)

return RR(avl);

else

return RL(avl);

}

else

return avl;

}//基本上就是判斷各個情況然後找到屬於哪個類型的後座平衡

static avl\_node\_t \*RR(avl\_node\_t \*avl)

{

avl = rotateleft(avl);

return avl;

}//往左轉

static avl\_node\_t \*LL(avl\_node\_t \*avl)

{

avl = rotateright(avl);

return avl;

}//往右轉

static avl\_node\_t \*LR(avl\_node\_t \*avl)//not sure

{

avl->left = rotateleft(avl->left);

avl = rotateright(avl);

return avl;

}//先左後右

static avl\_node\_t \*RL(avl\_node\_t \*avl)

{

avl->right = rotateright(avl->right);

avl = rotateright(avl);

return avl;

}//先右後左

static int height( avl\_node\_t \*avl)

{

int left\_height = 0;

int right\_height = 0;

if(avl == NULL)

return 0;

if(avl->left != NULL)

left\_height = height(avl->left);

if(avl->right != NULL)

right\_height = height(avl->right);

if(left\_height > right\_height)

avl->height = left\_height+1;

else

avl->height = right\_height+1;

return avl->height;

}//取得高度, 當為NULL時要回傳0要不然會出錯

static int balanceFactor(avl\_node\_t \*avl)

{

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

}//左高-右高

main.c