树的结构：

A

B C

D E F

G H

#include <cstdio>

#include <cstdlib>

#include <stack>

using namespace std;

typedef struct point{

bool visited;

char data;

point \*lc,\*rc;

bool RTag;

}point;

//构建树

point\* settree()

{

point\* root=(point\*)malloc(8\*sizeof(point));

for(int i=0;i<8;++i)

{

root[i].visited=false;

root[i].data='A'+i;

root[i].lc=NULL;

root[i].rc=NULL;

root[i].RTag=false;

}

root[0].lc=&root[1]; //树的结构如下图

root[0].rc=&root[2]; // A

root[1].lc=&root[3]; // B C

root[1].rc=&root[4]; // D E F

root[2].lc=&root[5]; // G H

root[4].lc=&root[6];

root[5].rc=&root[7];

return root;

}

//重置树

void resettree(point\* root)

{

for(int i=0;i<8;i++)

root[i].visited=false;

}

//6.36

void q36(point\* r1,point\* r2)

{

bool b=true;

if(r1==r2) //判断是否为同一棵树或空树

{

printf("两棵树相似！\n");

return;

}

point \*p1=r1,\*p2=r2;

stack<point\*> sp1,sp2;

sp1.push(p1);

sp2.push(p2);

while(!sp1.empty())

{

point \*t1=sp1.top();

point \*t2=sp2.top();

bool b1l=true,b1r=true;

bool b2l=true,b2r=true;

if(t1->lc==NULL || t1->lc->visited) b1l=false;

if(t1->rc==NULL || t1->rc->visited) b1r=false;

if(t2->lc==NULL || t2->lc->visited) b2l=false;

if(t2->rc==NULL || t2->rc->visited) b2r=false;

if(!(b1l||b1r||b2l||b2r)) //叶子结点或左右子树都已遍历过的结点

{

t1->visited=true;

t2->visited=true;

sp1.pop();

sp2.pop();

}

else if(b1l&&b2l) //左结点

{

sp1.push(t1->lc);

sp2.push(t2->lc);

}

else if(b1r&&b2r) //右结点

{

sp1.push(t1->rc);

sp2.push(t2->rc);

}

else{

b=false; break;

}

}

if(!sp2.empty()) b=false;

printf("两棵树%s相似！\n",b?"":"不");

}

//6.37（先序遍历加后继线索化）

void q37(point\* root)

{

stack<point\*> sp;

sp.push(root);

printf("%c\n",root->data);

bool brtag=false;

point\* t=NULL;

while(!sp.empty())

{

point\* p=sp.top();

bool bl=true,br=true;

if(!p->lc||p->lc->visited) bl=false;

if(p->RTag||(!p->rc&&!p->RTag)||p->rc->visited) br=false;

if(!p->rc)

{

p->RTag=true;

brtag=true;

t=p;

}

if(!(bl||br))

{

p->visited=true;

sp.pop();

}

else if(bl)

{

if(brtag)

{

t->rc=p->lc;

brtag=false;

}

printf("%c\n",p->lc->data);

sp.push(p->lc);

}

else if(br&&!p->RTag)

{

if(brtag)

{

t->rc=p->rc;

brtag=false;

}

printf("%c\n",p->rc->data);

sp.push(p->rc);

}

}

if(brtag) t->rc=NULL;

}

//6.56

void q56(point\* p)

{

point\* t=NULL;

if(p->lc)

t=p->lc;

else

t=p->rc;

if(t==NULL)

printf("不存在后继！\n");

else

printf("%c %c\n",p->data,t->data);

}

//6.60

int thread(point\* p,int n)

{

if(p->lc&&!p->lc->visited)

n=thread(p->lc,n);

else

{

++n;

printf("%c %d\n",p->data,n);

}

if(p->rc&&!p->rc->visited)

n=thread(p->rc,n);

p->visited=true;

return n;

}

void q60(point\* root)

{

int n=thread(root,0);

printf("%d\n",n);

}

//删除树

void deltree(point\* root)

{

free(root);

root=NULL;

}

int main()

{

point \*t1,\*t2;

t1=settree();

t2=settree();

printf("(6.36)\n");

q36(t1,t2); //t1,t2为相同的树

resettree(t1);

resettree(t2);

printf("\n(6.37)\n");

q37(t1);

resettree(t1);

printf("\n(6.56)\n");

q56(&t1[6]); //选取C作为定结点

resettree(t1);

printf("\n(6.60)\n");

q60(t1);

deltree(t1);

deltree(t2);

return 0;

}

