# ACM第六章上机题解与参考

## （二叉树的简单应用）

## **问题 A: DS\_6.1 创建和递归遍历**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。输出这棵二叉树的先序、中序和后序遍历序列。  
二叉树结点的data是字符类型数据, 其中#表示空格字符。

## **样例输入**

ABC##DE#G##F###

## **样例输出**

ABCDEGF

CBEGDFA

CGEFDBA

解：

1：根据题意 需要建立二叉树

typedef struct note

{

struct note \*left;//指向左子树

struct note \*right;//指向右子树

char data;//储存当前节点值

}Note;

2：根据题意要求 先序创建二叉树 当读到“#”时候 表示当前节点为空

我们不妨先读取一个字符data 对其内容进行判断，若这个字符data不是#，就把他存入当前节点，然后再对它的左右字数进行递归创建，将返回值保留并返回出来，若data值是#，只需要返回NULL即可。其实创建也可以通过传参数的方式进行 例如void \*creat(Note \*\*root);但这样会涉及到二级指针，不利于后续的操作，因此不妨用返回值的方式进行递归创建吧！创建函数代码如下：

Note \* creat(void)

{

Note \* leaf;

char data;

scanf("%c",&data);

if(data == '#')

{

return NULL;

}

else

{

leaf = (Note\*)malloc(sizeof(Note));

leaf ->data = data;

leaf ->left = creat();

leaf ->right = creat();

return leaf;

}

}

3：第三步只需要根据三种序列输出当前已经创建好的树。三种输出的区别只在于输出与访问次序的不同，其余部分内容一致：

void PreOrder(Note \* root)//先序遍历并输出

{

if(root != NULL)

{

printf("%c",root ->data);//优先访问

PreOrder(root ->left);

PreOrder(root ->right);

}

}

void InOrder(Note \* root)//中序遍历并输出

{

if(root != NULL)

{

InOrder(root ->left);

printf("%c",root -> data);//访问左子树后访问

InOrder(root ->right);

}

}

void PostOrder(Note \* root)//后序遍历并输出

{

if(root != NULL)

{

PostOrder(root ->left);

PostOrder(root -> right);

printf("%c",root ->data);//最后访问当前节点

}

}

4：最后只需要调用主函数测试运行它们，按要求输出。

int main()

{

Note\* root = creat();

PreOrder(root);

printf("\n");

InOrder(root);

printf("\n");

PostOrder(root);

printf("\n");

return 0;

}

## **问题 B: DS\_6.2 结点个数**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。分别统计二叉树中叶子结点、度为1的结点、度为2的结点的个数，并输出。  
第一行依次输出叶子结点个数、度为1的结点个数、度为2的结点个数，以空格隔开。  
第二行连续输出叶子结点，中间不间隔。

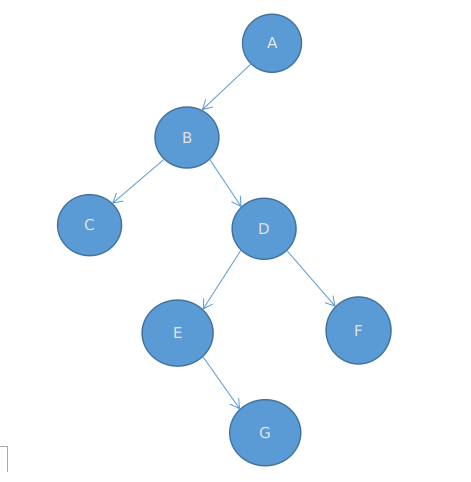
## **样例输入**

ABC##DE#G##F###

## **样例输出**

3 2 2

CGF



解：

根据样例不难得出这棵树：

该树有3个叶子节点：CGF

2个度为1的节点：ED

2个度为2的节点：BD

我们在创建这棵树时沿用上题的思路，

只需要修改先序遍历函数在访问每个节点时，

以此统计当前节点的数据类型，并保存到全局

变量，最后输出即可。

1：先声明全局变量：

char leaf\_list[1000];//保存叶子节点的数组

int nochild = 0,onechild = 0,twochild = 0;//保存不同类型节点数:

2：修改后的获得节点数及类型函数：

void get\_child(Note\*root)

{

int chid\_num = 0;

if(root != NULL)

{

if(root ->left != NULL)

{

chid\_num++;

}

if(root ->right != NULL)

{

chid\_num++;

}

switch(chid\_num)

{

case 0:

{

leaf\_list[nochild] = root ->data;

nochild++;

break;

}

case 1:

{

onechild++;

break;

}

case 2:

{

twochild++;

break;

}

default:

{

break;

}

}

get\_child(root ->left);

get\_child(root ->right);

}

}

3：测试运行：

#include<stdio.h>

#include<stdlib.h>

typedef struct note

{

struct note \*left;

struct note \*right;

char data;

}Note;

char leaf\_list[1000];

int nochild = 0,onechild = 0,twochild = 0;

Note \*creat ();

void get\_child(Note\*root);

int main()

{

int i;

Note\* root = creat();

get\_child(root);

printf("%d %d %d\n",nochild,onechild,twochild);

for(i = 0; i<nochild;i++)

{

printf("%c",leaf\_list[i]);

}

return 0;

}

## **问题 C: DS\_6.3 非递归先序和中序遍历**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。采取非递归方法输出这棵二叉树的先序、中序遍历序列。

## **样例输入**

ABC##DE#G##F###

## **样例输出**

ABCDEGF

CBEGDFA

解：

非递归时需要建立栈，并进行栈操作，我们先用链表模拟栈实现，之后的题我们再用数组模拟栈实现。

新增：

1：栈类型声明：

typedef struct stack

{

Note\* data;//储存当前节点的指针

struct stack \*next;

}Stack;

栈

2：栈类型操作：

新增：

void push(Stack\* head,Note\* root)//入栈操作函数

{

Stack\* now = (Stack\*)malloc(sizeof(Stack));

now->data = root;

now ->next = head ->next;

head ->next = now;

}

Note \* top(Stack \*head)//取栈顶元素操作函数

{

return(head ->next -> data);

}

void pop(Stack\* head)//出栈操作函数

{

Stack \*p;

if(head != NULL && head ->next != NULL)

{

p = head->next;

//root ->data = p->dat;

head->next = p->next;

free(p);

}

}

void init\_stack(Stack \*\*head)//栈初始化函数

{

(\*head) = (Stack\*)malloc(sizeof(Stack));

(\*head) ->next = NULL;

}

int IsEmpty(Stack \* head)//判断栈空函数

{

if(head == NULL || head ->next == NULL)

return 1;

else

return 0;

}

3：第三步，编写非递归先序、中序遍历函数：

void PerOrder2(Note \*root)//非递归先序遍历

{

Stack \*S;

Note \*p;

init\_stack(&S);

p = root;

while((p != NULL) || !(IsEmpty(S)))

{

while(p != NULL)

{

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

push(S,p);

p = p->left;

}

if(!IsEmpty(S))

{

p = top(S);

pop(S);

p = p->right;

}

}

}

void InOrder2(Note \*root)//非递归中序遍历

{

Stack \*S;

Note \*p;

init\_stack(&S);

p = root;

while(p != NULL || !IsEmpty(S))

{

if( p != NULL)

{

push(S,p);

p = p->left;

}

else

{

p = top(S);

pop(S);

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

p = p->right;

}

}

}

以上算法可以参考课本源码。

4：最后编写主函数测试：

#include<stdio.h>

#include<stdlib.h>

int main()

{

Note\* root = creat();

PerOrder2(root);

printf("\n");

InOrder2(root);

printf("\n");

return 0;

}

## **问题 D: DS\_6.4 非递归后序遍历**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。采取非递归方法输出这棵二叉树的后序遍历序列。

## **样例输入**

ABC##DE#G##F###

## **样例输出**

CGEFDBA

解：接上，新增后序非递归遍历函数：

void PostOrder2(Note \*root)

{

Stack \*S;

Note\* p,\*q;

init\_stack(&S);

p = root;

q = NULL;

while(p!= NULL||!IsEmpty(S))

{

while(p != NULL)

{

push(S,p);

p = p->left;

}

if(!IsEmpty(S))

{

p = top(S);

if((p->right == NULL)||(p->right == q))

{

p = top(S);

pop(S);

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

q = p;

p = NULL;

}

else

{

p = p->right;

}

}

}

}

主函数测试：

int main()

{

Note\* root = creat();

PostOrder2(root);

printf("\n");

return 0;

}

## **问题 E: DS\_6.5 层次遍历**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。输出这棵二叉树的层次遍历序列。

## **样例输入**

ABC##DE#G##F###

## **样例输出**

ABCDEFG

解：

1：层次遍历需要用到队列操作，队列与栈主要区别在于出栈顺序的不同，因此，只需要新增出队函数即可。

Note \*top\_last(Stack \*head)

{ Stack \*p=head;

while(p ->next->next != NULL)

{

p = p->next;

}

return p ->next->data;

}

void delete(Stack \*head)

{

Stack \*p=head,\*q;

while(p ->next->next != NULL)

{

p = p->next;

}

q = p->next;

free(q);

p->next = NULL;

}

2：编写层次遍历函数：

void LevelOrder(Note \*root)

{

Stack \*Q;

Note \* p;

init\_stack(&Q);

push(Q,root);

while(!IsEmpty(Q))

{

p = top\_last(Q);

delete(Q);

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

if(p->left != NULL)

{

push(Q,p->left);

}

if(p->right != NULL)

{

push(Q,p->right);

}

}

}

对于树，先将根入队，再判断，如果队列不为空就取当前队列里所保存的节点后出队，再将这个节点的左右子树入队后操作。

3：测试运行：

int main()

{

Note\* root = creat();

LevelOrder(root);

printf("\n");

return 0;

}

## **问题 F: DS\_6.6 依据先序、中序遍历序列创建二叉树**

## **题目描述**

根据先序序列和中序遍历序列，建立二叉树。输出这棵二叉树的后序遍历序列。  
结点个数<=50  
第一行：先序序列  
第二行：中序序列

## **样例输入**

ABDECFG

DBEACGF

## **样例输出**

DEBGFCA

解：

1：编写算法，核心算法参考如下函数

新增：

void BinaryTreeFromOrderings(char \*perorder,char \*inorder,int length)

{

if(length == 0)

{

return;

}

Note \*node =(Note\*) malloc(sizeof(Note));

node -> data = \*perorder;

int rootIndex = 0;

for(;rootIndex<length;rootIndex++)

{

if(inorder[rootIndex] == \*perorder)

break;

}

BinaryTreeFromOrderings(perorder+1,inorder,rootIndex);

BinaryTreeFromOrderings(perorder+rootIndex+1,inorder+rootIndex+1,length - (rootIndex+1));

printf("%c",node->data);

}

2：测试运行：

int main()

{

int length;

char inorder[60];

char perorder[60];

scanf("%s\n%s",perorder,inorder);

length = strlen(inorder);

BinaryTreeFromOrderings(perorder,inorder,length);

printf("\n");

return 0;

}

## **问题 G: DS\_6.7 依据中序、后序遍历序列创建二叉树**

## **题目描述**

根据中序序列和后序序列，建立二叉树。输出这棵二叉树的先序遍历序列。  
结点个数<=50  
第一行：中序序列  
第二行：后序序列

## **样例输入**

DBEACGF

DEBGFCA

## **样例输出**

ABDECFG

解：

2：编写核心算法：

Note\* BinaryTreeFromPostings(char \*inorder,char\*aftorder,int length)

{

if(length ==0)

{

return NULL;

}

Note\*node = (Note\*)malloc(sizeof(Note));

node->data = \*(aftorder+length-1);

printf("%c",node->data);

int rootIndex = 0;

for(;rootIndex < length; rootIndex++)

{

if(inorder[rootIndex] == \*(aftorder+length-1))

break;

}

node->left = BinaryTreeFromPostings(inorder,aftorder,rootIndex);

node->right = BinaryTreeFromPostings(inorder+rootIndex+1,aftorder+rootIndex,length-(rootIndex+1));

return node;

}

2：测试运行：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main()

{

int length;

char inorder[60];

char aftorder[60];

scanf("%s\n%s\n",inorder,aftorder);

length = strlen(inorder);

BinaryTreeFromPostings(inorder,aftorder,length);

printf("\n");

return 0;

}

## **问题 H: DS\_6.8 结点及其所在层次**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。按先序遍历次序输出各结点的内容及相应的层次数，要求以二元组的形式输出，其所对应的输出结果为：（data,level）  
data是二叉树结点数据域值，level是该结点所在的层次。  
设根节点在第一层。   
输出的元素间不用间隔，（）都是英文字符

## **样例输入**

AB#DG###CE##FH###

## **样例输出**

(A,1)(B,2)(D,3)(G,4)(C,2)(E,3)(F,3)(H,4)

解：

1：修改数据结构，新增内容保存当前层数：

修改：

typedef struct note

{

struct note \*left;

struct note \*right;

char data;

int level;//新增！

}Note;

2：修改创建函数，在创建时自动保存当前层数，把根节点作为第一次存入，以后每次递归时传入层数+1即可。

修改：

Note \* creat(int level)

{

Note \* leaf;

char data;

scanf("%c",&data);

if(data == '#')

{

return NULL;

}

else

{

leaf = (Note\*)malloc(sizeof(Note));

leaf ->data = data;

leaf ->level = level;

leaf ->left = creat(level+1);

leaf ->right = creat(level+1);

return leaf;

}

}

3：再修改先序遍历输出函数，输出数据时新增当前层数即可。

修改：

void PreOrder(Note \* root)

{

if(root != NULL)

{

printf("(%c,%d)",root ->data,root->level);

PreOrder(root ->left);

PreOrder(root ->right);

}

}

4：最后修改主函数测试运行：

int main()

{

Note\*head = creat(1);

PreOrder(head);

printf("\n");

return 0;

}

## **问题 I: DS\_6.9 某层叶子结点个数**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。并计算指定的第K层的叶子结点个数。设根结点在第一层。  
第一行：扩展先序序列  
第二行：k的值

## **样例输入**

ABC##DE#G##F###

3

## **样例输出**

1

解：在解决上一题的数据结构基础上新增函数：

新增全局变量保存所求层的叶子节点数，每访问到一个就+1：

int level\_leaf = 0;

后修改遍历函数为如下求某曾节点数的函数：

void find\_level\_leaf(Note\*root,int num)

{

if(root != NULL)

{

if(root -> level == num)

{

if(root->left == NULL && root ->right == NULL)

{

level\_leaf++;

}

}

find\_level\_leaf(root ->right,num);

find\_level\_leaf(root ->left,num);

}

}

测试运行：

int main()

{

int num;

Note\*head = creat(1);

scanf("%d",&num);

find\_level\_leaf(head,num);

printf("%d",level\_leaf);

printf("\n");

return 0;

}

## **问题 J: DS\_6.10 交换左右子树**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。将这棵二叉树的左右子树进行交换，并输出交换后二叉树的先序、中序、后序遍历序列。

## **样例输入**

ABC##DE#G##F###

## **样例输出**

ABDFEGC

AFDGEBC

FGEDCBA

解：

新增交换左右子树函数：

void exchange(Note\* root)

{

Note\*temp;

if(root != NULL)

{

temp = (root) ->left;

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

(root) ->right = temp;

exchange((root)->right);

exchange((root)-> left);

}

}

2：修改主函数，测试运行，并调用之前函数输出：

int main()

{

int num;

Note\*head = creat(1);

exchange(head);

PreOrder(head);

printf("\n");

InOrder(head);

printf("\n");

PostOrder(head);

printf("\n");

return 0;

}

## **问题 K: DS\_6.11 根结点到叶子结点的路径**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。输出从根结点到每个叶子结点的路径。  
输出的元素间不用间隔，都是英文字符,每个路径均占一行。

## **样例输入**

AB#DG###CE##FH###

## **样例输出**

G:ABD

E:AC

H:ACF

解：

在找路过程中，需要新增全局变量保存当前路径，之后用栈操作，对不同节点的路径进行修改，当访问到叶子时，输出当前路径即可。本次对栈的访问我们用数组实现，数组操作更为简便：

1：新增全局变量：

char arry[1000];//保存路径的数组

int flag = 0;//表示当前访问的节点数:

2：新增对数组的栈操作函数：

void arry\_push(char a)//入栈

{

arry[flag++] = a;

}

void arry\_pop(void)//出栈

{

if(flag>0)

{

flag--;

}

}

void print(void)//打印当前数组内所以元素

{

int i;

printf("%c:",arry[flag-1]);

for(i = 0 ; i<flag-1;i++)

{

putchar(arry[i]);

}

printf("\n");

}

3：新增路径访问函数，依据先序遍历，当访问到叶子时就调用print函数输出：

void get\_road(Note\* root)

{

if(root != NULL)

{

arry\_push(root->data);

if(root ->left != NULL || root ->right != NULL)

{

get\_road(root ->left);

get\_road(root ->right);

}

else

{

print();

}

arry\_pop();

}

}

4：修改主函数测试运行：

int main()

{

int num;

Note\*head = creat(1);

get\_road(head);

return 0;

}

## **问题 L: DS\_6.12 最近共同祖先**

## **题目描述**

从键盘接收扩展先序序列，以二叉链表作为存储结构，建立二叉树。求两个不同结点ch1,ch2的最近共同祖先。

第一行：扩展先序序列  
第二行：ch1,ch2两个不同结点值，用一个空格间隔。

## **样例输入**

ABC##DE#G##F###

C F

## **样例输出**

B

解：

最后一题不妨设立如下情景 先建立两个数组用来保存两个点的路径，之后对数进行先序遍历，每次访问一个节点时，如果尚未找到该节点就入栈，之后判断这个节点是不是想要找的节点，如果是 ，就标明当前节点已经找到。之后再对左右子树递归遍历，运行完以上所以步骤后，如果当前节点没有找到，就出栈。

1：因此第一步，要新增全局变量：字符型数组，用来保存两个节点的路径：

char arry1[1000];

char arry2[1000];

接下来 再声明两个整形全局变量，保存当前对应两个栈已经入栈的节点数：

int flag1 = 0,flag2 = 0;

再声明两个标志位（全局变量）标注所求节点有没有访问到：

int Flag1 = 0,Flag2 = 0;

0表示还没有找到，1表示已经找到。

2：接下来，按照上述思路，新增对两个数组栈操作的函数：

void arry1\_pop(void)

{

if(flag1 >0)

{

flag1--;

}

}

void arry2\_pop(void)

{

if(flag2 >0)

{

flag2 --;

}

}

void arry1\_push(char a)

{

arry1[flag1++] = a;

}

void arry2\_push(char a)

{

arry2[flag2++] = a;

}

3：新增编写求两个节点路径的函数，路径保存再数组当中：（方法与求根节点到叶子节点路径方法类似）

void get\_near\_common\_node(char a,char b,Note\*root)

{

if(root != NULL)

{

if(Flag1 == 0)

{

arry1\_push(root->data);

}

if(Flag2 == 0)

{

arry2\_push(root->data);

}

if(root->data == a)

{

Flag1 = 1;

}

if(root ->data == b)

{

Flag2 =1;

}

get\_near\_common\_node(a,b,root->left);

get\_near\_common\_node(a,b,root->right);

if(Flag1 == 0)

{

arry1\_pop();

}

if(Flag2 == 0)

{

arry2\_pop();

}

}

}

4：接下来我们会得到两个存有所求点路径的函数，最后只用倒叙查找两个数组中最先出现一致的那个节点（除去它本身）就行了，因为数据不会太大，所以用一个简单的查找方式实现：

void find\_node\_char(void)

{

int i,j;

for(i = flag1-2;i>=0;i--)

{

for(j = flag2-2;j >= 0;j--)

{

if(arry1[i] == arry2[j])

{

printf("%c",arry1[i]);

return;

}

}

}

}

5：最后修改主函数测试运行：

int main()

{

int num;

char a,b;

Note\*head = creat(1);

scanf(" %c %c",&a,&b);

get\_near\_common\_node(a,b,head);

find\_node\_char();

return 0;

}

※这里要特别提醒：在最后读取两个字符时 要先增加一个空格“ ”，即： scanf(" %c %c",&a,&b);而不是scanf("%c %c",&a,&b);！

**以上为第六章上机题所有内容题解及参考思路，若有问题QQ联系919745190.**

## End