数据结构与算法

人工智能与大数据学院

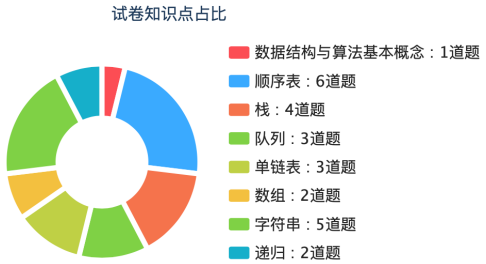
本节课内容安排

一、第1次过程考分析

二、二叉树的遍历算法及应用

第1次过程测验分析

试卷知识点占比：本试卷共29道题 包含8个知识点

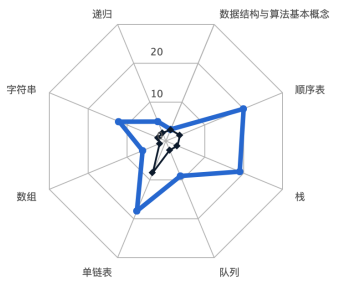


第1次过程测验20241016 （题数：29，总分：100.0）

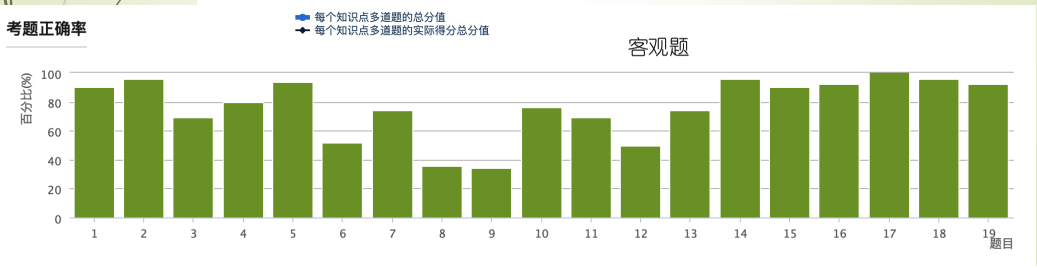
分数分布

100分	0人（0.00%）
80-99分	14人（25.45%）

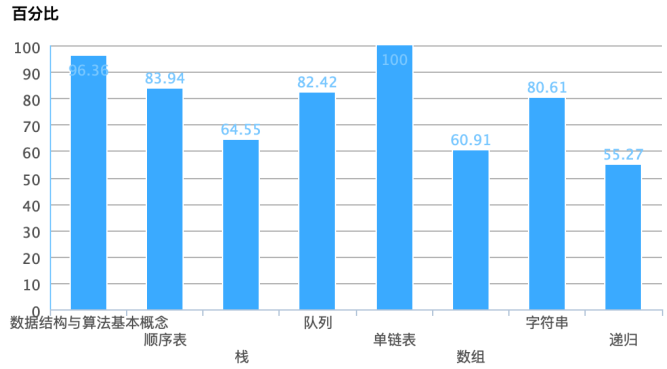
知识点雷达分布图



考题正确率



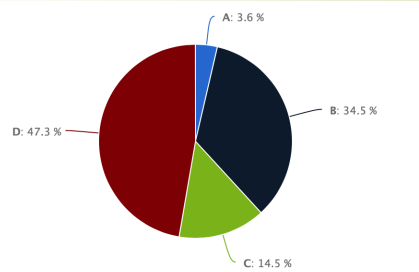
客观题平均正确率



第1次过程测验分析——递归

9 下列哪个不是递归算法的优点?

A、 代码简洁
B、 容易调试
C、 适用于解决分治问题
D、 可以减少重复计算



正确答案: B 正确: 19 人 错误: 36 人 正确率: 34.5%

第1次过程测验分析——递归

11 递归函数的执行过程与队列的规则相似，具备“先进先出”的特点

正确答案: 错误 正确: 38 人 错误: 17 人 正确率: 69.1%

第1次过程测验分析——数组

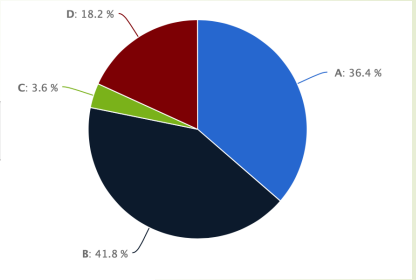
3 假设二维数组有4行4列，按照行优先顺序进行存储，每个元素占2个存储单元，第1个元素的存储单元地址为100，则a[2][3]的地址是（ ）

A、 122
B、 128
C、 126
D、 124

6 假设二维数组有4行4列，按照列优先顺序进行存储，每个元素占2个存储单元，第1个元素的存储单元地址为100，则a[2][3]的地址是（ ）

A、 122
B、 126
C、 130
D、 128

第1次过程测验分析——栈



8 栈在程序设计中有着广泛的应用，下列哪一个不是栈的典型应用?

A、 函数调用的参数传递

B、 表达式求值

C、 括号匹配检查

D、 页面访问历史的后退功能

正确答案： A　　正确： 20 人　　错误： 35 人　　正确率： 36.4%

第1次过程测验分析——栈

19 栈的插入和删除操作只能在栈顶进行

正确答案： 正确　　正确： 51 人　　错误： 4 人　　正确率： 92.7%

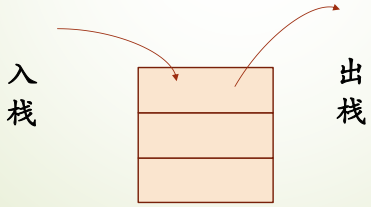
25 已知栈s，假设依次执行下列操作，请给出输出序列结果（数字之间通过逗号间隔）：_____

```
push(s, 1);
push(s, 2);
push(s, 3);
pop(s);
pop(s);
push(s, 3);
pop(s);
push(s, 4);
pop(s);
pop(s);
```

栈的应用

假设在一个算术表达式中可以包含三种括号:圆括号“(”和“)”，中括号“[”和“]” 和花括号“{”和“}”，并且这三种括号可以按任意的次序嵌套使用。

例如：算术表达式为［10*(5-2)］ -［(4+7)］]



24 根据题目填空完善程序：

设一个算术表达式中包括圆括号和方括号两种类型的括号，编写一个算法判断其中的括号是否匹配。

```
void push(SeqStack *s, char x);
char pop(SeqStack *s);

bool ismatch(SeqStack *s){
   bool ismatched = _____;
   char item;
   while(_____ && (item=getchar())!='\n'){
        if(item=='___' || item == '[')
           push(s, _____);
        if(item=='___' || item == '__'){
           if(empty(s)){
              _____=false;
           }else{
                char match;
                match = pop(___);
                ismatched = ((item=='___' && match == '(' )) || (item=='___' && match == '____' );
           }
      }
}
if(!empty(s))
   _____;
return ismatched;
```

第1次过程测验分析——字符串

5 下列哪个操作可以实现字符串的逆序?

A、 将字符串中的所有字符按照ASCII码值进行排序

B、 将字符串中的所有字符依次插入到一个栈中，然后再依次弹出

C、 将字符串中的所有字符依次插入到一个队列中，然后再依次出队

D、 将字符串中的所有字符依次进行交换，直到整个字符串逆序

正确答案： B 正确： 52 人 错误： 3 人 正确率： 94.5%

第1次过程测验分析——字符串

12 字符串中任意个字符组成的序列称为该字符串的子串，空串是任意串的子串，任意串是自身的子串

21 串S1='ABCDEFG'，串s2='PQRST'，则

1、Concat(Substr(s1,2,Index(s2,'Q')), Substr(s1,len(s2),2))的结果为'（）'（备注：不用再写单引号）

2、Equal(Substr(s2, 2, 2), 'QR')的结果为（ ）（备注：0或者1）

23 字符串的替换操作是将字符串中的某个子串替换为另一个_____，并返回替换后的新字符串

28 子串是字符串中任意个_____字符组成的序列

第1次过程测验分析——队列

1 判断循环队列队空的条件为（）

A、 (rear+1)%maxsize==front

B、 front==rear

C、 (front+1)%maxsize==rear

D、 (rear-1)%maxsize==front

4 判断循环队列队满的条件为（ ）

A、 front==rear

B、 (front+1)%maxsize==rear

C、 (rear+1)%maxsize==front

D、 (rear-1)%maxsize==front

10 顺序队列的假溢出现象可以通过增加存储空间的方式来解决

第1次过程测验分析——顺序表

14 线性表是一种线性结构，其中的元素之间是一对一的关系

正确答案： 正确 正确： 53 人 错误： 2 人 正确率： 96.4%

15 线性表只能顺序存储

正确答案： 错误 正确： 50 人 错误： 5 人 正确率： 90.9%

16 在顺序存储的线性表中，插入和删除操作的时间复杂度都是O(1)

正确答案： 错误 正确： 51 人 错误： 4 人 正确率： 92.7%

第1次过程测验分析——顺序表

13 判断下面程序的正误（顺序表插入元素）

```
for (j=len-1 ;j>= i-1  ;j--)
    L->data[ j+1] = L->data[ j];
L->data[i –1] = x;
L->last = L->last+1 ;
```

第1次过程测验分析——顺序表

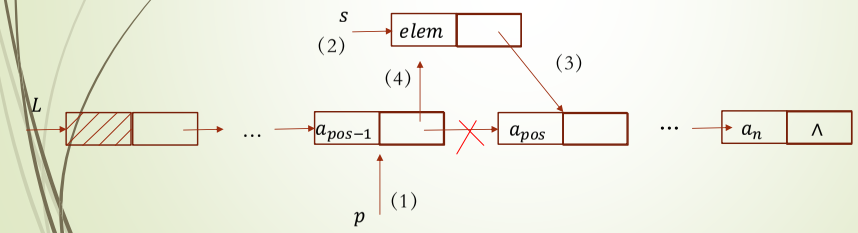
26 如下为顺序表删除数据元素程序片段，将其补充完整：

```
#define maxlen 100
typedef struct{
   int data[maxlen];
   int last;
}
int SqLlength(Sequenlist *L){
   return(L->last+1);
}
void Sqdelete(Sequenlist *L, int i){  // 算法中 i 是顺序表中数据元素的序号（从1开始）
   int j;
   for(j=i; j <=_______;j++)
      L->data[_______]=L->data[_______];
   ____________;
}
```

知识点回顾

（2）单链表的插入操作

插入操作是指将值为elem的新结点插入到单链表中第pos个结点的位置上，即a_{pos-1}和a_{pos}之间。为了实现这个操作，必须找到a_{pos-1}的位置，然后构造一个数据域为elem的新结点，将其挂在单链表上，操作过程如下图所示。

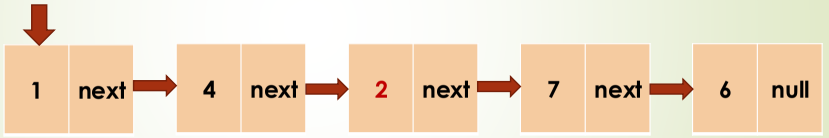


插入数据元素/结点

new

顺序表：

单链表：



第1次过程测验分析——单链表

20 在单链表中 z 结点之后插入值为 c 的新结点 x。算法思路为：

（1）生成一个新结点x；

（2）将c赋给新结点 x的数据域；

（3）将新结点插入到单链表中；

请给出步骤（2）和（3）对应的实现代码。

步骤（2）：_____;

步骤（3）：_____;

步骤（3）：_____;

22 如下程序为单链表的插入算法实现，将程序补充完整。

```
typedef struct node{
    int data;
    struct node *next;
}LinkList;
void LLinsert(LinkList *L, int i, int x){
    LinkList *P, *S;
    int j = 0;
    P=L;
    while(P!=_____ && j<i-1){
        _____;
        j++;
    }
    if(p=NULL) printf("error/n");
    else{
        S = (LinkList*)malloc(sizeof(LinkList));
        S->data = _____;
        S->next=_____;
        _____;
    }
}
```

二叉树的遍历算法

1. 基础知识

“遍历”是**任何数据结构类型**均有的操作。

对线性结构而言，因为每个结点（最后一个结点除外）均只有一个后继，因而只有一条搜索路径，无须另加讨论。

二叉树是非线性结构，结点有零个、一个或者两个后继，存在如何遍历的问题，即按什么样的搜索路径遍历。

二叉树的遍历是指按照某种方法顺着某一条搜索路径巡防二叉树中的结点，使得**每个结点均被访问一次，而且仅被访问一次**。

二叉树的遍历算法

2. 递归遍历算法

一棵二叉树一般由根结点、根结点的左子树和根结点的右子树3部分组成，因而只要依次遍历这3部分，就能遍历整棵二叉树。

如果用D、L、R分别表示访问二叉树的根、遍历根结点的左子树和遍历根结点的右子树，那么二叉树的遍历方式有DLR, DRL, LDR, LRD, RDL, RLD 6种。

如果进一步限定访问子树的顺序为先左子树，再右子树，那么遍历方式有DLR(先序遍历)、LDR(中序遍历)、LRD(后序遍历)3种。

二叉树的遍历算法

2. 递归遍历算法

(1) 先序遍历算法

如果二叉树为空，则遍历结束；否则，
先访问根结点，然后先序遍历根结点的左子树，最后先序遍历根结点的右子树。

【算法1】二叉树先序遍历递归算法

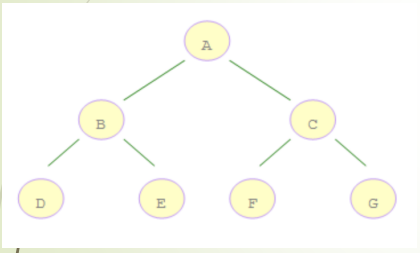
```
Void Preorder(BiTree bt){
    if(bt!=NULL){
        visit(bt->data);  //访问根结点
        Preorder(bt->lchild); //先序遍历左子树
        Preorder(bt->rchild); //先序遍历右子树
    }
}
```

二叉树的遍历算法

2. 递归遍历算法

(1) 先序遍历算法

【例题】请给出如下二叉树的先序遍历序列



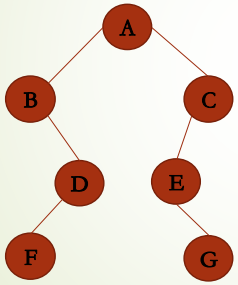
先序遍历的结果是【A, B, D, E, C, F, G】

二叉树的遍历算法

2. 递归遍历算法

(1) 先序遍历算法

【例题】请给出如下二叉树的先序遍历序列



先序遍历的结果是【A, B, D, F, C, E, G】

二叉树的遍历算法

2. 递归遍历算法

(2) 中序遍历算法

如果二叉树为空，则遍历结束；否则，
先中序遍历根结点的左子树，然后访问根结点，最后先序遍历根结点的右子树。

【算法2】二叉树中序遍历递归算法

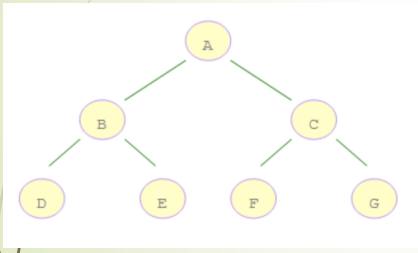
```
Void Inorder(BiTree bt){
    if(bt!=NULL){
        Inorder(bt->lchild); //中序遍历左子树
        visit(bt->data);  //访问根结点
        Inorder(bt->rchild); //中序遍历右子树
    }
}
```

二叉树的遍历算法

2. 递归遍历算法

(2) 中序遍历算法

【例题】请给出如下二叉树的中序遍历序列



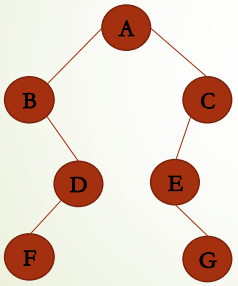
中序遍历的结果是【D, B, E, A, F, C, G】

二叉树的遍历算法

2. 递归遍历算法

(2) 中序遍历算法

【例题】请给出如下二叉树的先序遍历序列



中序遍历的结果是【B, F, D, A, E, G, C】

二叉树的遍历算法

2. 递归遍历算法

(3) 后序遍历算法

如果二叉树为空，则遍历结束；否则，
先后序遍历根结点的左子树，然后后序遍历根结点的右子树，最后访问根结点。

【算法3】**二叉树后序遍历递归算法**

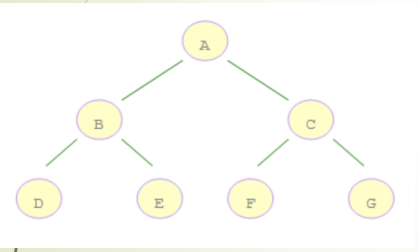
```
Void Postorder(BiTree bt){
    if(bt!=NULL){
        Postorder(bt->lchild); //后序遍历左子树
        Postorder(bt->rchild); //后序遍历右子树
        visit(bt->data);  //访问根结点
    }
}
```

二叉树的遍历算法

2. 递归遍历算法

(3) 后序遍历算法

【例题】请给出如下二叉树的后序遍历序列



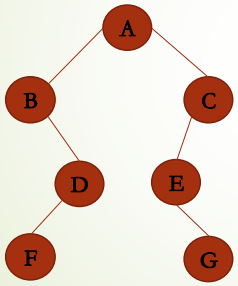
后序遍历的结果是【D, E, B, F, G, C, A】

二叉树的遍历算法

2. 递归遍历算法

(3) 后序遍历算法

【例题】请给出如下二叉树的后序遍历序列



后序遍历的结果是【F, D, B, G, E, C, A】

二叉树的遍历算法

3. 层次遍历算法

二叉树的层次遍历是指从二叉树的根结点开始，从上到下逐层遍历，同一层中按从左到右的顺序依次访问二叉树的结点。

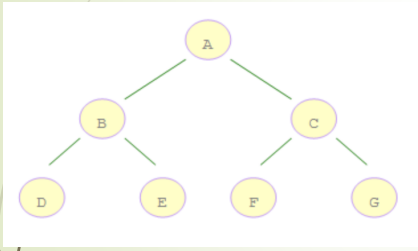
在层次遍历中，对一层的结点访问完以后，再按照它们的访问次序依次访问各个结点的左孩子和右孩子，这样一层一层地进行，先遇到的结点先访问。

这和队列的操作规则完全一致，因此，层次遍历二叉树时，可采用一个队列来进行操作。

二叉树的遍历算法

3. 层次遍历算法

【例题】请给出如下二叉树的层次遍历序列



层次遍历的结果是【A, B, C, D, E, F, G】

二叉树的遍历算法

层次遍历算法的的队列实现方法

首先将根结点入队，然后从队头取一个元素，每取一个元素，执行如下3个动作：

① 访问该元素所指结点；

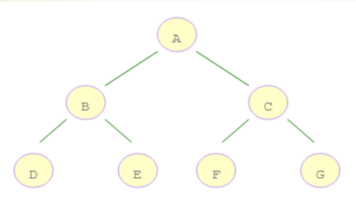
② 如果该元素所指结点有左孩子，则左孩子指针入队；

③ 如果该元素所指结点有右孩子，则右孩子指针入队。

此过程一直执行到队列空为止，此时二叉树遍历结束。

A结点是BC结点的双亲，按照根->左->右的遍历顺序，而且每个结点都是如此。因此可以采用队列的数据结构。

【例题】层次遍历的结果是【A, B, C, D, E, F, G】



- 结点A入队
- 结点A出队，结点A的左右孩子入队，队列：C、B
- 结点B出队，结点B的左右孩子入队，队列：E、D、C
- 结点C出队，结点C的左右孩子入队，队列：G、F、E、D
- 结点D出队，结点E出队，结点F出队，结点G出队

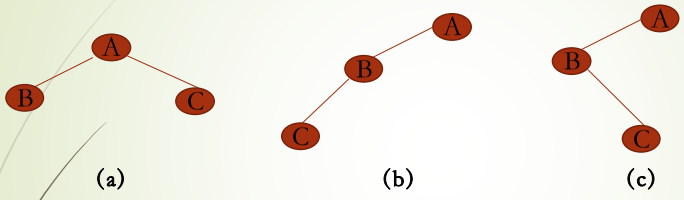
二叉树遍历算法的应用

根据二叉树的遍历规则，二叉树的先序遍历序列、中序遍历序列、后序遍历序列和层次遍历序列都是**唯一**的。

但是，对于一棵树的先中后三种顺序的遍历方式，任何一种单独拿出来都无法确定一棵树

二叉树遍历算法的应用

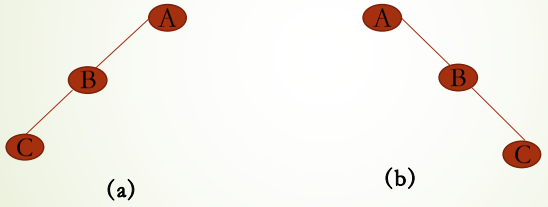
如果已知二叉树的先序遍历序列为ABC，那么下面三棵二叉树都满足这个要求。



只知道二叉树的先序遍历序列，不能唯一确定一棵二叉树

二叉树遍历算法的应用

如果已知二叉树的后序遍历序列为CBA，那么下面两棵二叉树都满足这个要求。



只知道二叉树的后序遍历序列，不能唯一确定一棵二叉树

二叉树遍历算法的应用

如果已知二叉树的中序序遍历序列为BAC，那么下面两棵二叉树都满足这个要求。

只知道二叉树的中序遍历序列，不能唯一确定一棵二叉树

二叉树遍历算法的应用

那么，两种遍历方式序列结合起来能否唯一确定一棵二叉树呢？

答案是肯定的！

三种组合：

① 先序序列 + 中序序列

② 后序序列 + 中序序列

③ 先序序列 + 后序序列 ✖

以上三种组合都可以组成二叉树，但是只有前两种组合可以唯一确定一棵二叉树

二叉树遍历算法的应用

为什么先序和后序不能唯一确定二叉树呢？

先序和后序可以告诉我们根节点，只不过先序遍历的根节点从前往后，后序遍历的根节点从后往前，也正是因为先序遍历和后序遍历都只能告诉我们根节点这个信息，所以他们两个在一起是没办法得到足够信息去构建二叉树的二叉树呢？

二叉树遍历算法的应用

由先序遍历序列和中序遍历序列建立二叉树

第一种情况：如果先序遍历序列和中序遍历序列为空，那么这棵二叉树为空二叉树。

第二种情况：对于具有n个结点的二叉树

① 先序遍历序列的第一个结点一定是二叉树的根结点；

② 在中序遍历序列中找到根结点，根结点左边的序列一定是二叉树根结点左子树上的中序遍历序列，同理，根结点右边的序列一定是二叉树根结点右子树上的中序遍历序列

③ 继续先序遍历序列的第二个结点，在中序遍历序列中该结点的左右序列即为它的左右子树的序列

二叉树遍历算法的应用

【例题】由先序遍历序列和中序遍历序列建立二叉树

一棵二叉树的先序遍历序列为{A, B, D, H, I, E, J, K,C,F,G}，中序遍历序列为{H,D,I,B,J,E,K,A,F,C,G}，请画出这个二叉树的图？

先序遍历： *先访问根结点，然后先序遍历根结点的左子树，最后先序遍历根结点的右子树*

中序遍历： *先中序遍历根结点的左子树，然后访问根结点，最后中序遍历根结点的右子树*

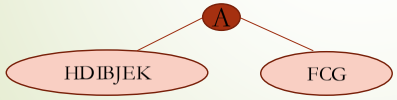
二叉树遍历算法的应用

先序遍历序列为{A, B, D, H, I, E, J, K, C, F, G}

中序遍历序列为{H, D, I, B, J, E, K, A, F, C, G}

第一步：通过先序序列的第一个元素确定根结点为A，通过中序序列可以看出A的左子树结点有哪些？右子树结点有哪些？

{H, D, I, B, J, E, K, A, F, C, G}



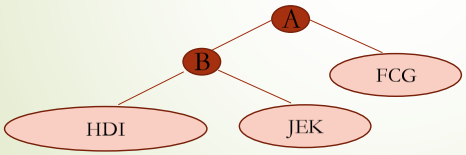
二叉树遍历算法的应用

先序遍历序列为{A, B, D, H, I, E, J, K, C, F, G}

中序遍历序列为{H, D, I, B, J, E, K, A, F, C, G}

第二步：通过看先序序列中的第二个结点B在中序序列的左右结点，可以知道哪些结点是B的左子树，哪些结点是B的右子树？

{H, D, I, B, J, E, K, A, F, C, G}



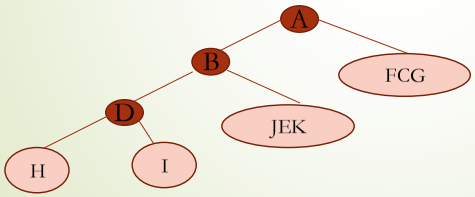
二叉树遍历算法的应用

先序遍历序列为{A, B, D, H, I, E, J, K, D, F, G}

中序遍历序列为{H, D, I, B, J, E, K, A, F, C, G}

第三步：通过看先序序列中的第三个结点D在中序序列的左右结点，可以知道哪些结点是D的左子树，哪些结点是D的右子树？

{H, D, I, B, J, E, K, A, F, C, G}

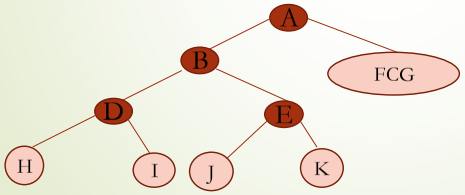


先序遍历序列为{A, B, D, H, I, E, J, K, C, F, G}

中序遍历序列为{H, D, I, B, J, E, K, A, F, C, G}

第四步：先序序列中的第四个结点H和第五个结点I已经是叶子结点了，不需要进行分析。接下来看第六个结点E，从中序序列的左右结点，可以知道哪些结点是E的左子树，哪些结点是E的右子树？

{H, D, I, B, J, E, K, A, F, C, G}

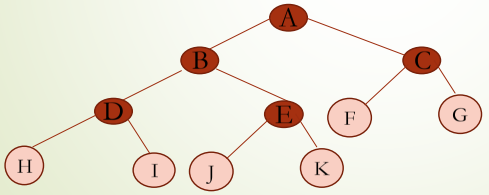


先序遍历序列为{A, B, D, H, I, E, J, K, C, F, G}

中序遍历序列为{H, D, I, B, J, E, K, A, F, C, G}

第五步：先序序列中的第7个结点J和第8个结点K已经是叶子结点了，不需要进行分析。接下来看第9个结点C，从中序序列的左右结点，可以知道哪些结点是C的左子树，哪些结点是C的右子树？

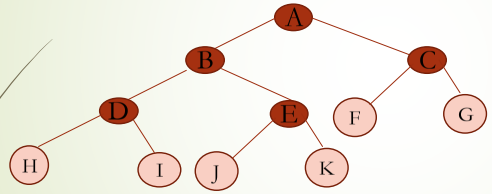
{H, D, I, B, J, E, K, A, F, C, G}



先序遍历序列为{A, B, D, H, I, E, J, K, C, F, G}

中序遍历序列为{H, D, I, B, J, E, K, A, F, C, G}

经过分析，可知二叉树如下图所示。请给出它的后序遍历序列



后序遍历：先后序遍历根结点的左子树，然后后序遍历根结点的右子树，最后访问根结点

后序遍历序列为：{H, I, D, J, K, E, B, F, G, C,A}