清華大學 Tsinghua University

随堂测试习题解答

刘烨斌

清华大学自动化系

第一讲

- 在数据结构中，与所使用的计算机无关的数据结构是：A

A.逻辑结构　B. 存储结构　C. 逻辑和存储结构　D. 物理结构

✓ 分析：数据结构中，逻辑结构是对数据元素之间关系的描述，与所使用的计算机无关。存储结构又称为数据的物理结构，是指数据在计算机内存中的表示，与所使用的计算机紧密相关。

- 某算法完成对n个元素进行处理所需要的时间是：$T(n)=100\log_2 n+200n+500$，则该算法的时间复杂度是：B

A.O(1)　B. O(n)　C. $O(n\log_2 n)$　D. $O(n\log_2 n+n)$

✓ 分析：估计算法时间复杂度时，忽略所有低次幂和最高次幂系数

第一讲

- 在定义ADT时，除数据对象和数据关系外，还需要说明：C
  A. 数据元素　B. 算法　C. 基本操作　D. 数据项
  - ✓ 分析：抽象数据类型可用三元组表示(D,R,P)

- 下面这段程序的时间复杂度为：$\log_3 n$

```
a = 1;
while (a<=n)
        a = a*3;
```

- 数据的逻辑结构包括：集合、线性、树形、图形四种类型

- 数据结构概念：
  - ✓ 相互之间存在一种或多种特定关系的数据元素的集合

第一讲

数据结构的概念

- 数据结构是计算机存储、组织数据的方式。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。通常情况下，精心选择的数据结构可以带来更高的运行或者存储效率。
- Sartaj Sahni在他的《数据结构、算法与应用》一书中称：“数据结构是数据对象，以及存在于该对象的实例和组成实 例的数据元素之间的各种联系。这些联系可以通过定义相关的函数来给出。”他将数据对象（data object）定义为“一个数据对象是实例或值的集合”。
- Clifford A.Shaffer在《数据结构与算法分析》一书中的定义是：“数据结构是ADT（抽象数据类型Abstract Data Type）的物理实现。”
- Robert L.Kruse在《数据结构与程序设计》一书中，将一个数据结构的设计过程分成抽象层、数据结构层和实现层。其中，抽象层是指抽象数据类型层，它讨论数据的逻辑结构及其运算，数据结构层和实现层讨论一个数据结构的表示和在计算机内的存储细节以及运算的实现。
- 数据结构具体指同一类数据元素中，各元素之间的相互关系，包括三个组成成分，数据的逻辑结构，数据的存储结构和数据运算结构。

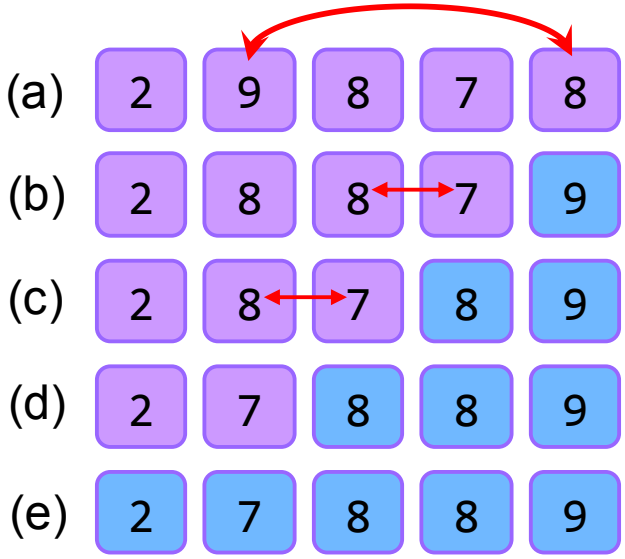
第二讲

1. 对于规模为n的无序向量的排序问题，使用下列排序算法各自的复杂度分别为：

   A. 冒泡排序： $O(n^2)$　　B. 选择排序： $O(n^2)$

   C. 归并排序： $O(n\log n)$　　D. 插入排序： $O(n^2)$

2. 以上排序算法不稳定的是：B



第一步之后两个8的顺序互换

第二讲

3. 对于规模为n的有序序列的二分查找复杂度为： O(logn)

4. 对于规模为n的无序序列的查找，最好情况下的复杂度为：O(1)　平均复杂度为： O(n)

5. 归并排序采用分治递归策略实现，其复杂度为：O(nlogn) 。以下请使用递归复杂度方程，分析证明该复杂度

- ✓ 两路归并merge时间复杂度：O(n)
- ✓ 对长度为n的向量归并排序，需完成2长度为n/2向量的归并排序和一两路归并：
  T(n) = 2*T(n/2) + O(n)
- ✓ 边界条件：T(1) = 1

$$T(n) = 2*T(n/2) + O(n)$$
$$T(n)/n=T(n/2)/(n/2)+O(1)$$
$$=T(n/4)/(n/4)+O(2)$$
$$=T(n/2^k)/(n/2^k)+O(k)$$
当$n=2^k$时，$k=\log n$,
$$T(n)/n=O(\log n)$$
$$T(n)=O(n\log n)$$

第三讲

1. 在长度为n>1的带头节点的单链表h上，另设有尾指针r指向尾节点，则执行 **B** 操作与链表长度有关
   A. 删除单链表的第一个元素
   B. 删除单链表的最后一个元素
   C. 在单链表第一个元素前插入一个新元素
   D. 在单链表的最后一个元素后插入一个新元素

2. 带表头节点的单链表中，当删除某一指定节点时，必须找到该节点的 **前驱** 。

3. 一个非空线性链表，在由p所指结点的后面插入一个由q所指的结点的过程是：**B**

```
A. q->next=p; p->next = q;
B. q->next=p->next; p->next = q;
C. q->next=p->next; p = q;
D. p->next=q; q->next = p;
```

第三讲

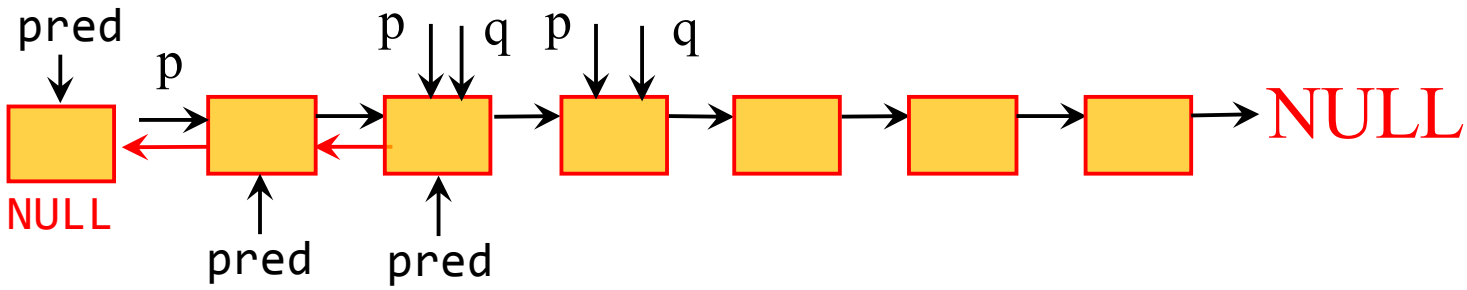
4. 已知无头单链表A和B表示两个集合，本算法实现A=A-B（集合补运算，即求出现在A中但不出现在B中的元素）

```
typedef struct node
{
        int data;
        struct node* next;
} lnode ;

lnode* setminus(lnode* A, lnode* B)
{
    lnode *p, *q;
    p=(lnode*) malloc(sizeof(lnode));
    p->next = A; A=p;
    while(B!=NULL)
    {
       p=A;
       while( P->next!= NULL )
           if(B->data==p->next->data)
           {
               q=p->next;
               p->next = q->next ;
               free(q);
           }
           else p = p->next ;
       B = B->next ;
    }
    A=A->next;
    return A;
    }
```

第三讲

5. 将一单链表逆置，要求在原表上进行，不允许重新建链表。



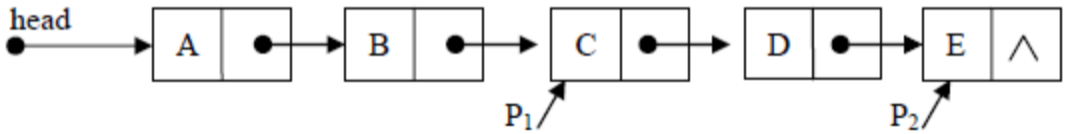

```
lnode* inverse(lnode *p)
{
        lnode *q, *pred = null;
        while(p) {
                q = p->next;
                p->next = pred;
                pred = p;
                p = q;
        }
        return pred;
}
```

第三讲附加

1.对于顺序存储的长度为N的线性表，访问结点和增加结点的时间复杂度分别对应为O(1)和O(N)。T
2.若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算，则利用顺序表存储最节省时间。T
3.在具有N个结点的单链表中，访问结点和增加结点的时间复杂度分别对应为O(1)和O(N)。F
4.对于顺序存储的长度为N的线性表，删除第一个元素和插入最后一个元素的时间复杂度分别对应为O(1)和O(N)。F
5.若用链表来表示一个线性表，则表中元素的地址一定是连续的。F
6.在顺序表中逻辑上相邻的元素，其对应的物理位置也是相邻的。T
7.顺序存储的线性表不支持随机存取。F
8.在顺序表上进行插入、删除操作时需要移动元素的个数与待插入或待删除元素的位置无关。F
9.在线性表的顺序存储结构中可实现快速的随机存取，而在链式存储结构中则只能进行顺序存取。T
10.)单链表不是一种随机存取的存储结构。

第三讲附加

对于下图2.29所示的单链表，下列表达式值为真的是( )。



1. head.getNext().getData()=='C'
2. P2.getNext()==null
3. head.getData()=='B'
4. P1.getData()=='D'

第三讲附加

已知指针ha和hb分别是两个单链表的头指针，下列算法将这两个链表首尾相连在一起，并形成一个循环链表（即ha的最后一个结点链接hb的第一个结点，hb的最后一个结点指向ha），返回该循环链表的头指针。请将该算法补充完整。

```
typedef struct node{
ElemType data;
    struct node *next;
}LNode;
LNode *merge(LNode *ha, LNode *hb) {
LNode *p=ha;
     if (ha==NULL || hb==NULL) {
cout<<”one or two link lists are empty!”<<endl;
          return NULL;
     }
     while ( p->next!=NULL )
           p=p->next;
     p->next=hb;
     while ( p->next!=NULL )
           p=p->next;
           _________
}
```

1. ha=p->next; return ha;
2. p->next=ha; return ha;
3. ha=p->next; return p;
4. p->next=ha; return p;

第三讲附加

```
1  //逆序打印单链表
2  void LinkListReversePrint(LinkNode* head)
3  {
4      if(head == NULL)
5      {
6          return;
7      }
8      LinkListReversePrint(head->next);
9      printf("[%c]   ",head->data);
10 }
```

第四讲

1. 循环队列的队满条件为_D___

A. (sq.rear+1) %maxsize = (sq.front+1)%maxsize

B. sq.rear = sq.front

C. (sq.front+1) %maxsize = sq.rear

D. (sq.rear+1)%maxsize = sq.front

2. 设有算数表达式$x + a \times (y - b) - c/d$，该表达式的前缀表示为 $- + x \times a - yb/cd$, 后缀表示为$xayb - \times + cd/-$。

3. 若一个栈的进栈序列是1,2,3,...,n，其输出序列为p1,p2,p3,...,pn, 若p1=3，则p2为( )
A. 可能是2　B. 一定是2　C. 可能是1　D. 一定是1

4. 向一个栈顶指针为h的带头节点的链栈中插入指针s所指的节点时，应执行的两句代码为：$s \to next = h \to next; h \to next = s;$

5. 在图书馆一共6个人在排队，3个还《面试宝典》一书，3个在借《面试宝典》一书，图书馆此时没有了面试宝典了，求他们排队的总数？_h(3)=5_

第四讲

6. 12个高矮不同的人,排成两排,每排必须是从矮到高排列,而且第二排比对应的第一排的人高,问排列方式有多少种?

__h(6)=132__ 把12个人先由低到高排，依次处理，push相当于把人放第一排，pop相当于把人放第二排

7. 写出(0!+1)×2^(3!+4)−(5!−67−(8+9))的前缀表达式，

$$-\times +!01^2+!34--!567+89$$

8. 列出中缀转RPN表达式计算中每一步栈中的内容。输入：(0!+1)×2^(3!+4)−(5!−67−(8+9))

输入	输出	输入	输出	输入	输出	输入	输出	输入	输出
(	(	x	x	+	x^(+	!	-(!	+	-(-(+
0	(	2	x	4	x^(+	-	-(-	9	-(-(+
!	(!	^	x^	)	x^	67	-(-	)	(-
+	(+	(	x^(	-	-	-	-(-	)	NULL
1	(+	3	x^(	(	-(	(	-(-(		
)	NULL	!	x^(!	5	-(	8	-(-(		

第四讲

8. 列出中缀转RPN表达式计算中每一步栈中的内容。输入：(0!+1)×2^(3!+4)−(5!−67−(8+9))

输入	输出	输入	输出	输入	输出	输入	输出	输入	输出
(	(	x	x	+	x^(+	!	-(!	+	-(-(+
0	(	2	x	4	x^(+	-	-(-	9	-(-(+
!	(!	^	x^	)	x^	67	-(-	)	(-
+	(+	(	x^(	-	-	-	-(-	)	NULL
1	(+	3	x^(	(	-(	(	-(-(		
)	NULL	!	x^(!	5	-(	8	-(-(		

第四讲

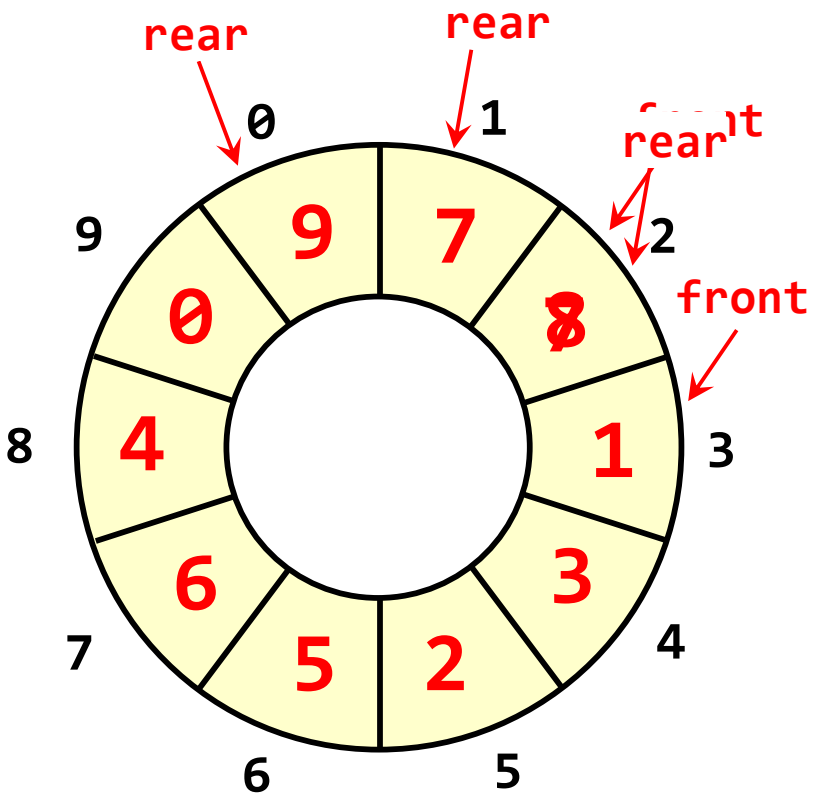
1. 若用一个大小为6的数组来实现循环队列，且当前队尾指针rear和队头指针front的值分别为0和3,。当从队列中删除一个元素，再加上两个元素后，rear和front的值分别为_B_。

A.1和5　　dequeue()

B.2和4　　enqueue(7)

C.4和2　　enqueue(8)

D.5和1



✓ 队头出队，队尾入队

✓ 入队出队秩增加周期求余

第四讲

2. 循环队列相对于基于数组实现的普通队列的优点是__降低随着元素出队产生的空间浪费__。
3. 基于单链表的队列实现中，为了实现O（1）复杂度的入队和出队操作，需要__在队尾设置rear指针__。此时，链表头进行_出队_操作，表尾进行_入队_操作。

1. 以下运算实现在链队上的入队列操作，请在____处填充适当的语句。

```
Void EnQueue(QueptrTp* lq, Data x)
{
LqueueTp *p;
P = (LqueueTp*) malloc(sizeof(LqueueTp));
__p->data__________ = x;
p->next = NULL;
(lq->rear)->next = ___p_____________________________;
_____lq->rear=p_____________________________;
}
```

第五讲

4. 八皇后的递归实现

```
void search(int r) {
    if (r == n) {
        print();
        ++cnt;
        return;
    }
    for (int i = 0; i<n; ++i) { //循环对第r行的n个位置进行试探
        c[r] = i;
        int ok = 1;
        for (int j = 0; j<r; ++j) //循环对前r行检测是否冲突
            if (c[r]==c[j]||r-j==c[r]-c[j]||r-j==c[j]-c[r]){
                ok = 0;
                break;
            }
        if (ok) search(r + 1);  //成果则递归搜索第r+1行
    }
}
```

第五讲

1. 对于一颗具有N个结点的树，树中所有度数之和为_ N-1 _。设树T的度为4，其中度为1，2，3，4的节点个数分别为4，2，1，1，则T中的叶子数为？__4+2*2+3+4-（4+2+1+1）+1=_8____
2. 已知一棵有2015个节点的树，其叶子节点的个数为115，则该树对应的二叉树中无右孩子的节点个数为？_1901__
3. 判断题：完全二叉树所有叶子结点具有相同的高度？_Y_深度?_N_
4. 一棵二叉树的第i层（i>=1）最多有_2^i_个结点；一棵n（n>0）个结点的满二叉树共有_(n+1)/2_叶子和_(n-1)/2_内部节点。

第五讲

5. 一棵二叉树的前序遍历序列为ABCDEFG，它的中序遍历序列可能是（B）
   A. CABDEFG　B. ABCDEFG　C. DACEFBG　D. ADCFEG

6. 在二叉树结点的先序序列，中序序列和后序序列中，所有叶子结点的先后顺序（B）
   A．都不相同　　C．先序和中序相同，而与后序不同
   B．完全相同　　D．中序和后序相同，而与先序不同

7. 某二叉树的前序序列和后序序列正好相反，则该二叉树一定是（C）的二叉树。
   A．空或只有一个结点　B．任一结点无左子树
   C．高度等于其结点数　D．任一结点无右子树

8. 二叉树结点的中序序列为A,B,C,D,E,F,G,后序序列为B,D,C,A,F,G,E,则该二叉树结点的前序序列为_ EACBDGF __

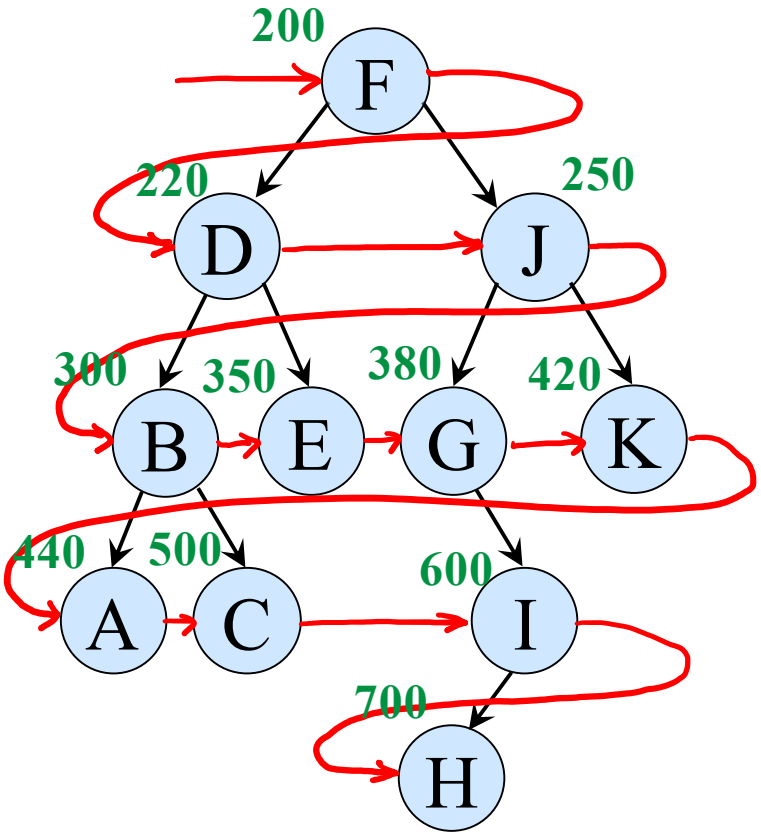
第五讲

9. 二叉树采用链表存储结构，设计算法计算二叉树的叶子节点个数。（请采用三种深度优先遍历算法之一）

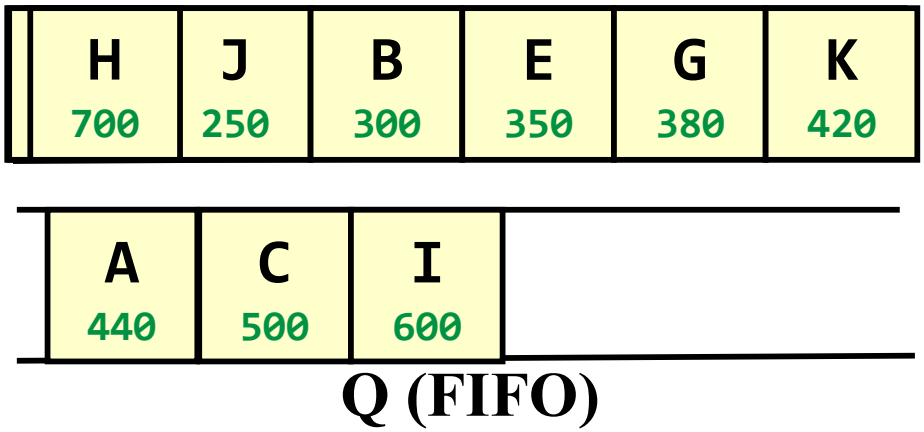
```
int nleaf(BinNode* x)
{
   if(x==NULL) return 0;
   else if(x->left==NULL&&x->right==NULL)  return 1;
   else return nleaf(x->left)+nleaf(x->right);
}
```

第五讲

10.编写计算二叉树最大宽度的算法（二叉树的最大宽度是指二叉树所有层中结点个数的最大值）。简述算法核心思想，再写伪代码 int widthTree(BinNode* x)



使用队列，层次遍历二叉树。在上一层遍历完成后，下一层的所有节点已经放到队列中，此时队列中的元素个数就是下一层的宽度。以此类推，依次遍历下一层即可求出二叉树的最大宽度。



F D J B E G K A C I H

第六讲

1. 有一份电文中共使用 6个字符:a,b,c,d,e,f,它们的出现频率依次为2,3,4,7,8,9，试构造一棵哈夫曼树，则其加权路径长度WPL为__80__，字符c的编码是__001(不唯一)_

2. 根据n个元素建立一棵二叉搜索树时，其时间复杂度大致为D

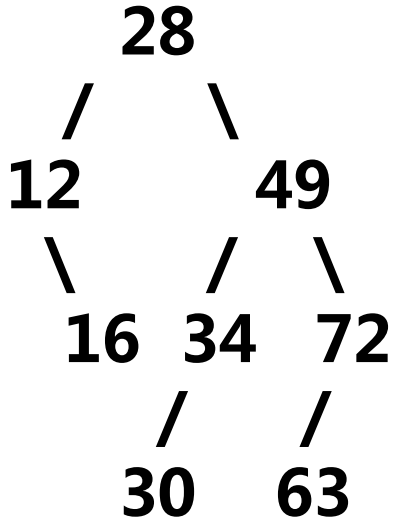
   A．O(1)　B．$O(\log_2 n)$　C．O(n)　D．$O(n\log_2 n)$

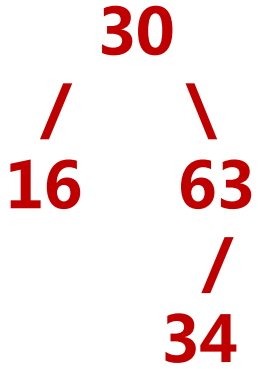
3. 具有5层节点的AVL树至少有_fib(4+3)-1=12_节点

4. 给定1,2,..10共10个整数，求这10个数的置换序列，当按置换序列顺序输入构造二叉搜索树时，所得的二叉搜索树为完全二叉树。这个置换序列可为：_7,4,9,2,6,8,10,1,3,5_

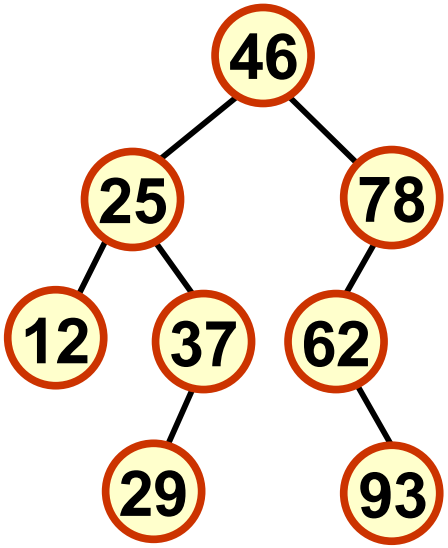
第六讲

5. 已知一个二叉搜索树如下图所示，试画出依次删除72,12,49,28节点后的得到的每个二叉搜索树（不考虑平衡调整）。



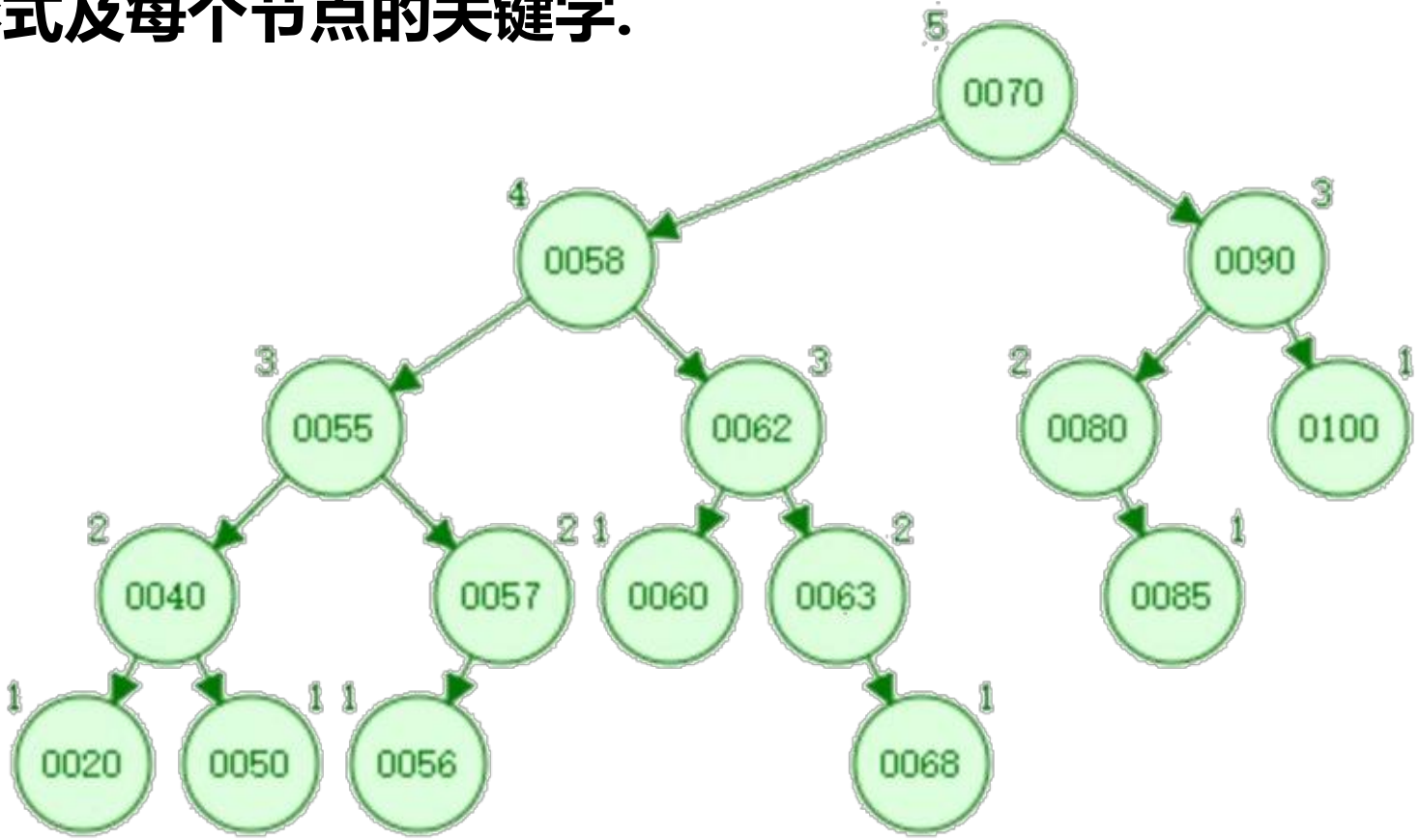


6. 已知一组元素{46,25,78,62,12,37,70,29}，试画出按顺序插入生成二叉搜索树的结果



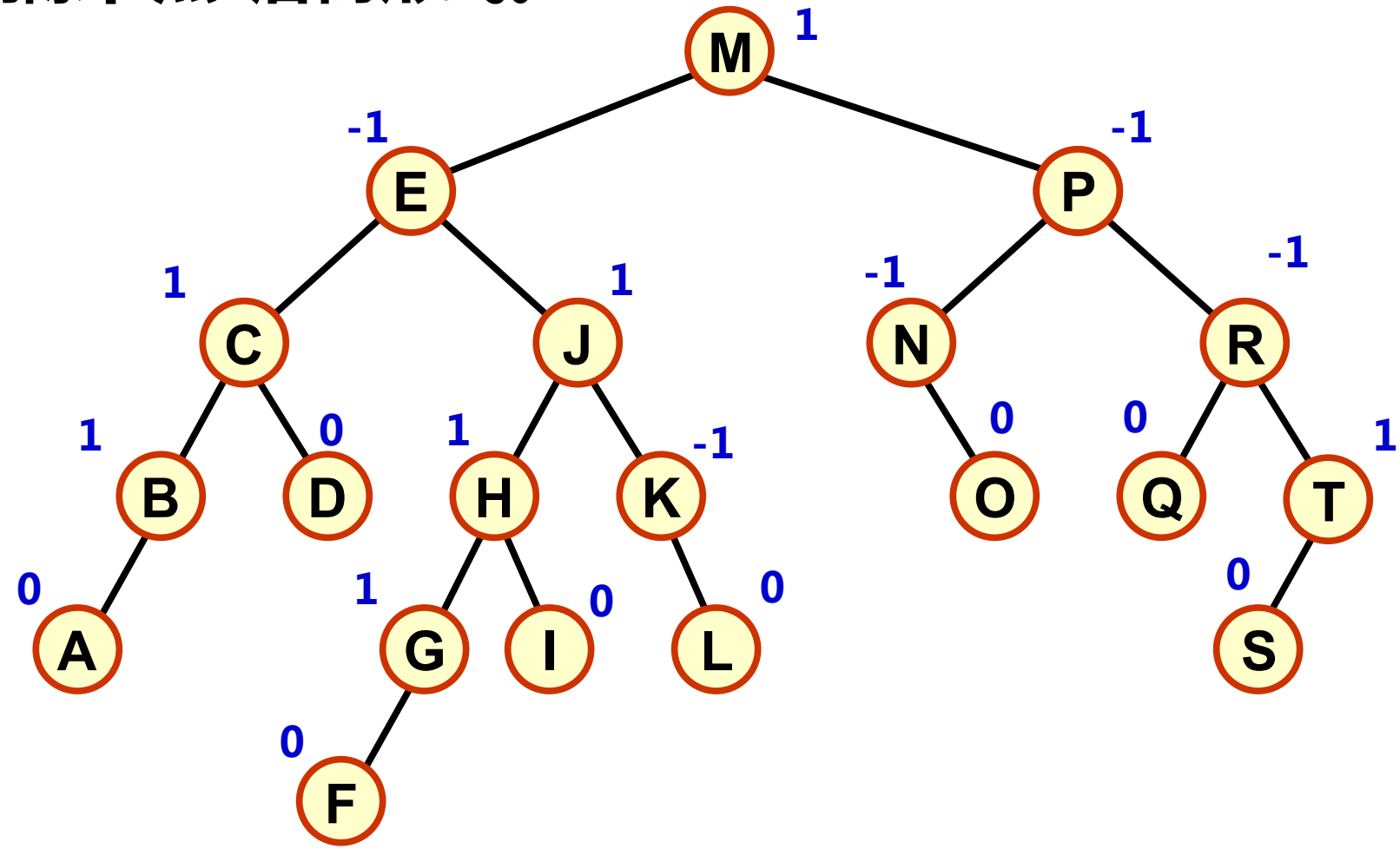
第六讲

7. 给定一个关键字序列100，90，80，70，60，50，40，55，58，57，85，63，62，56，20，68试按照顺序构造平衡二叉搜索树，平衡规则按照教材所给，试写出最后的平衡二叉搜索树形式及每个节点的关键字.



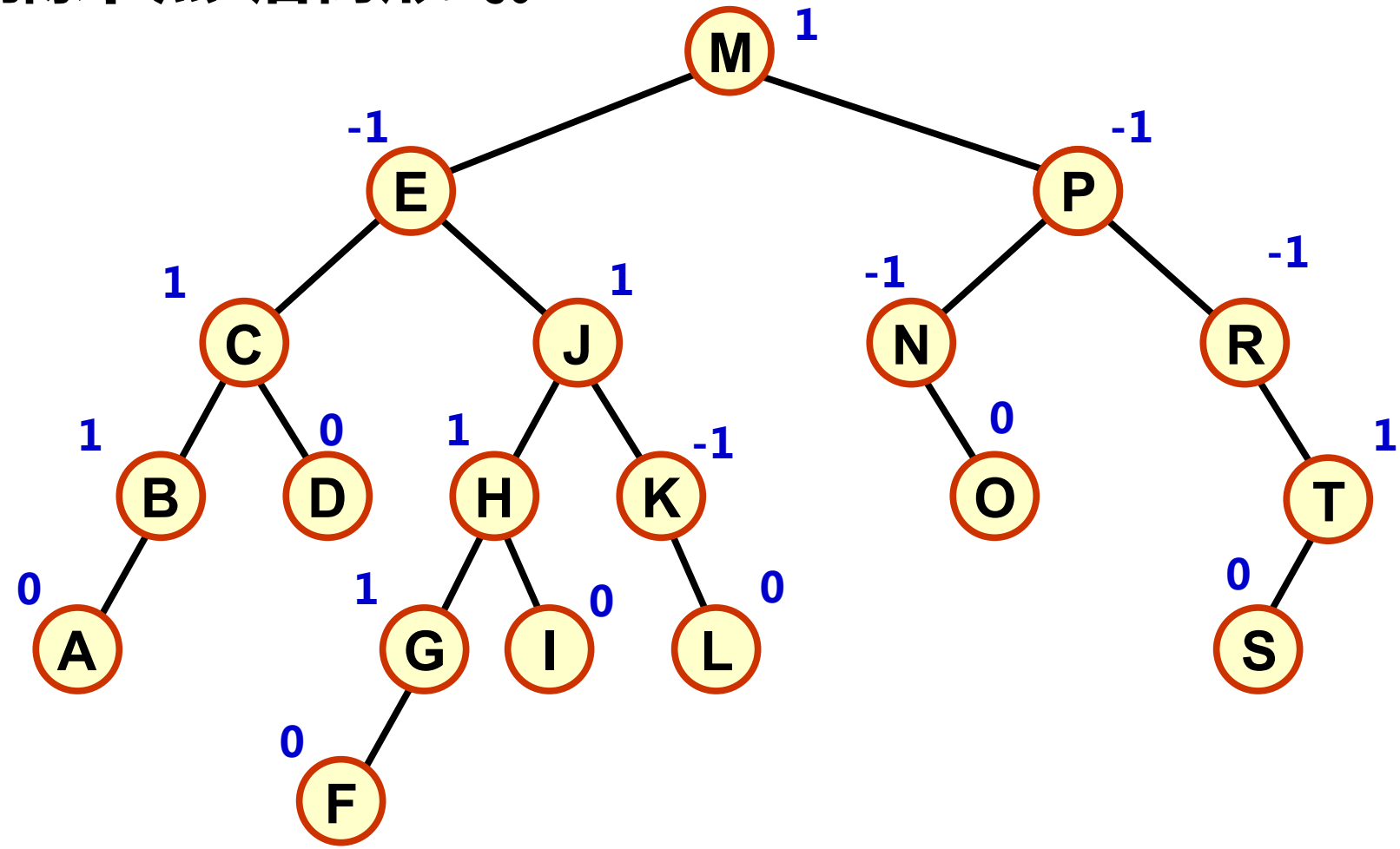
第六讲

8. 如图AVL树，画出删除节点Q后树形式，进一步画出删除节点J后树形式。



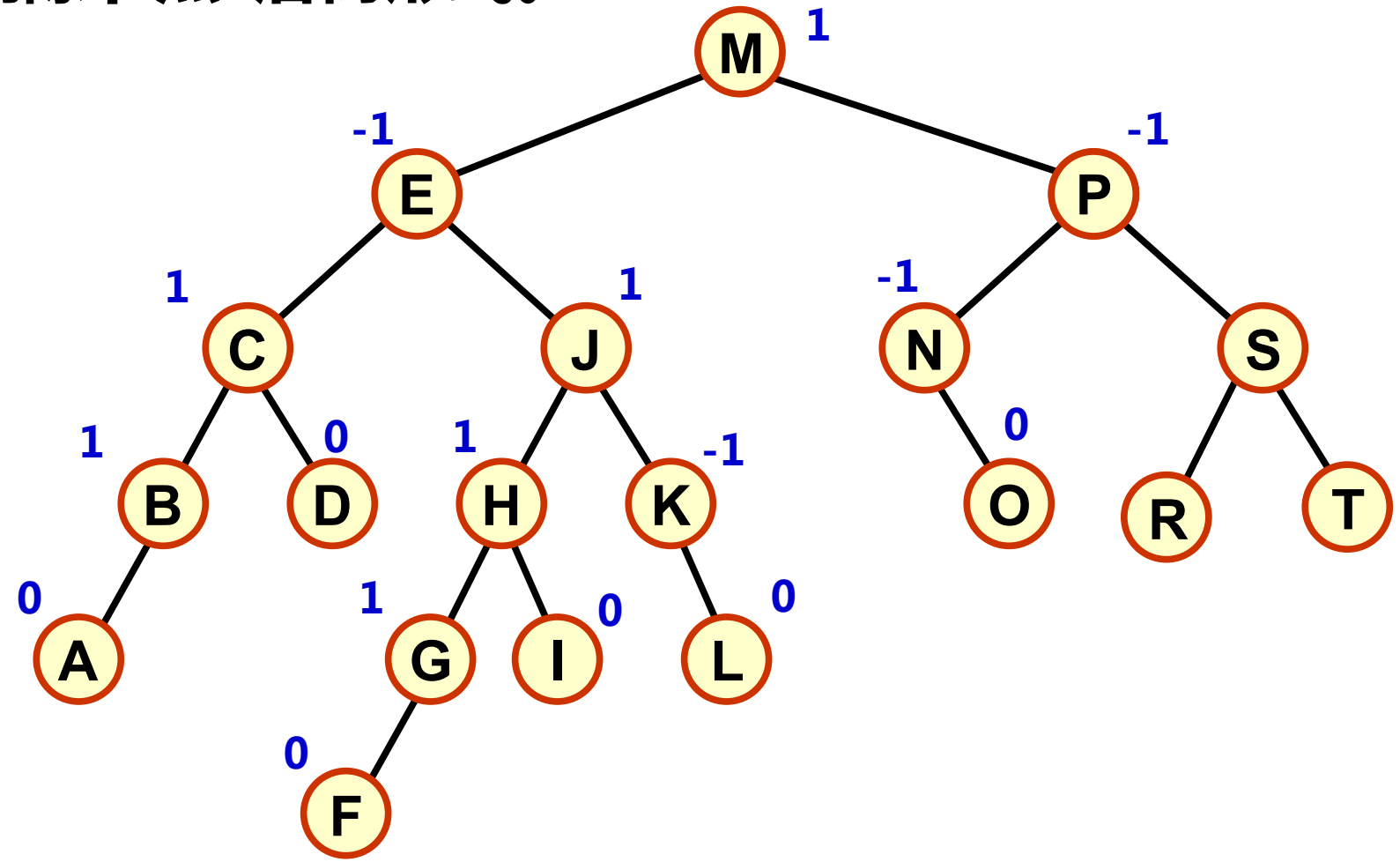
第六讲

8. 如图AVL树，画出删除节点Q后树形式，进一步画出删除节点J后树形式。



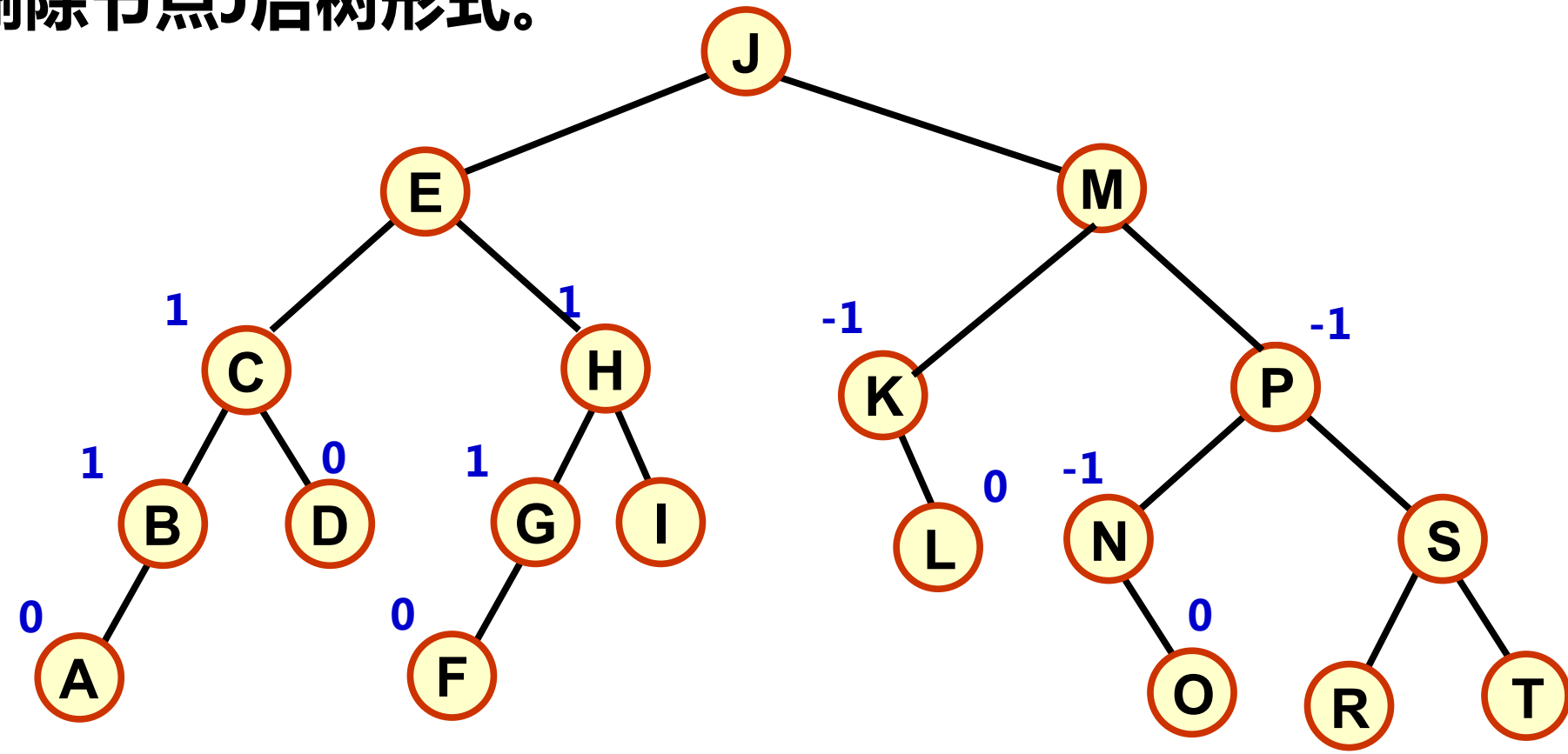
第六讲

8. 如图AVL树，画出删除节点Q后树形式，进一步画出删除节点J后树形式。



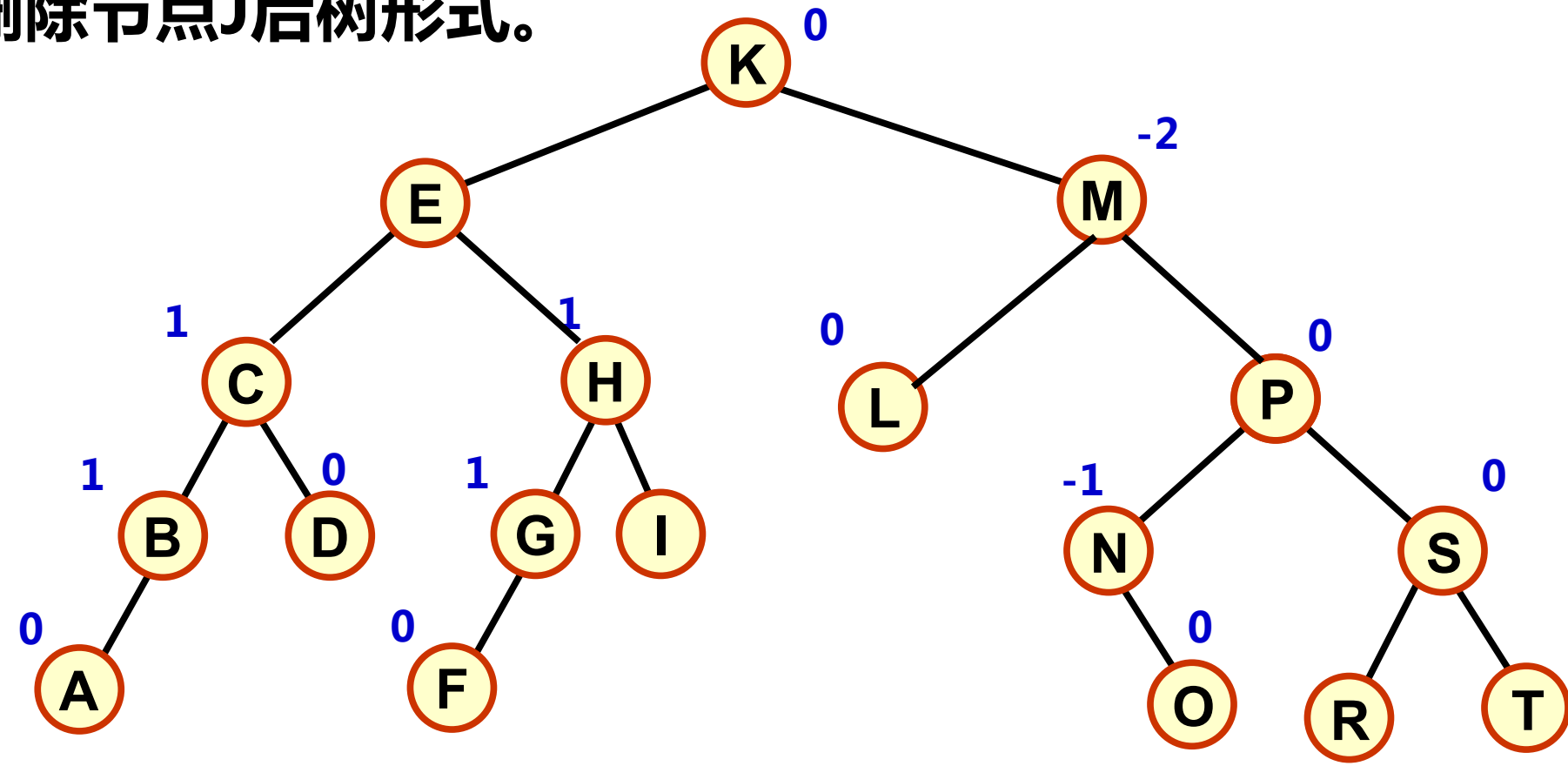
回顾：随堂测试

8. 如图AVL树，画出删除节点Q后树形式，进一步画出删除节点J后树形式。



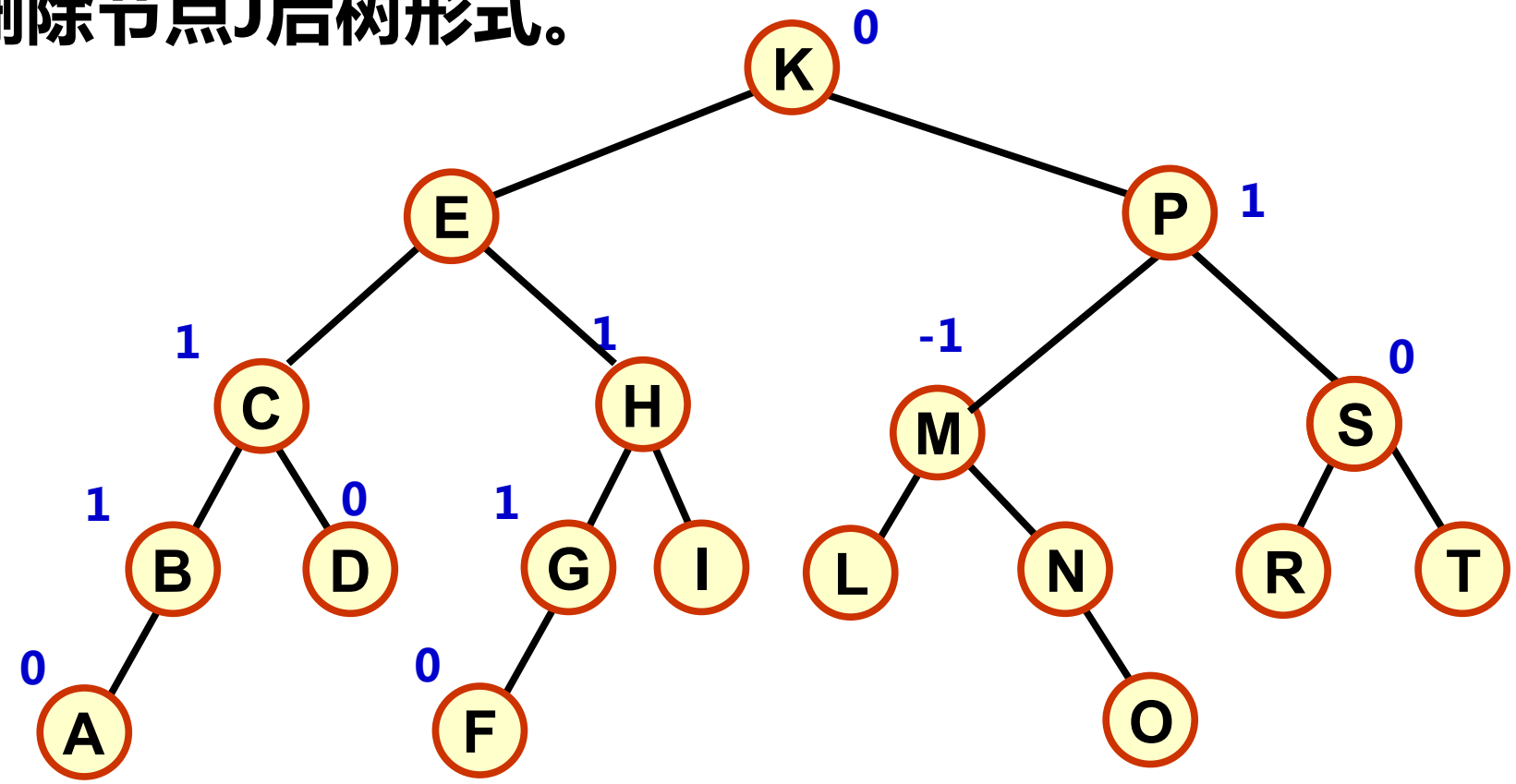
第六讲

8. 如图AVL树，画出删除节点Q后树形式，进一步画出删除节点J后树形式。



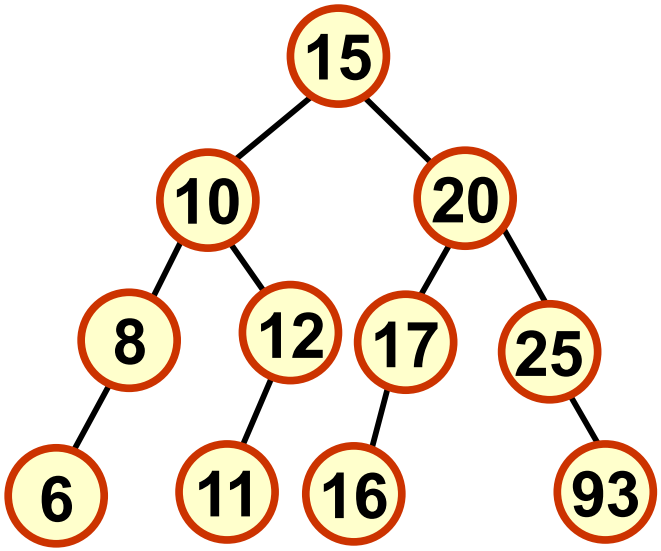
第六讲

8. 如图AVL树，画出删除节点Q后树形式，进一步画出删除节点J后树形式。



第六讲

9. 给定二叉搜索树根节点BSTNode* root及某个节点指针BSTNode* current, 写一算法（代码）求解树所有节点中大于current->data的最小值。（树中节点无指向父节点指针，并且要求算法复杂度不大于O(h)）



解答：分两情况。

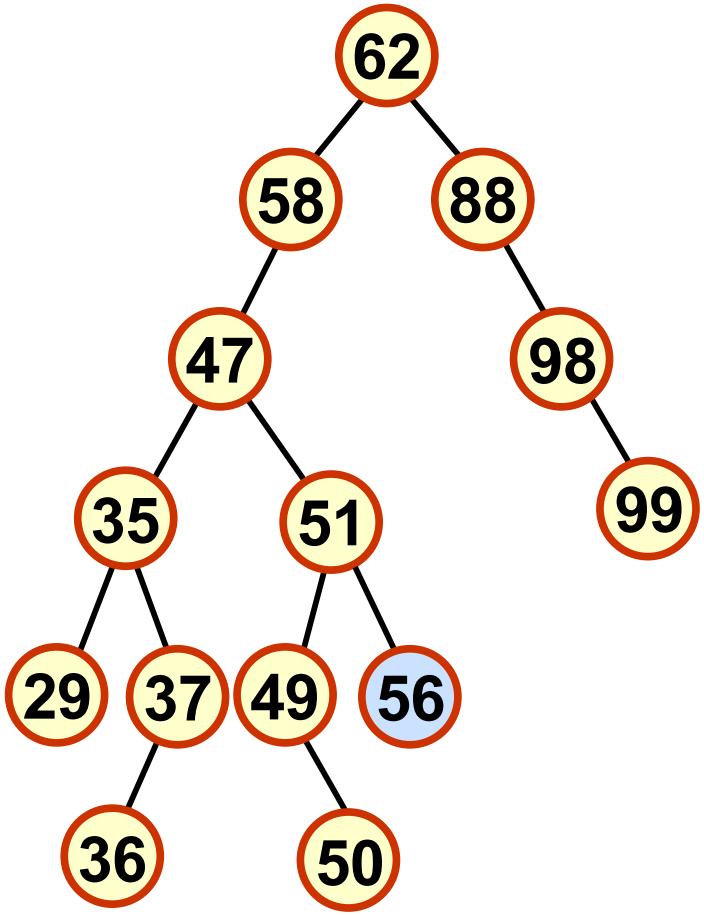
情况1：为该节点存在右孩子，则只需从右孩子子树找到中序遍历下的第一个节点，即尽可能左的子孙节点（如图节点10，其后继为11）

情况2：若给定节点无右孩子，则要求的节点为给定节点的最近祖先，并且处在该祖先的左子树中。（如节点12，所求为15,15为12的最近祖先，并且满足12在15的左子树中）

第六讲

节点的直接后继

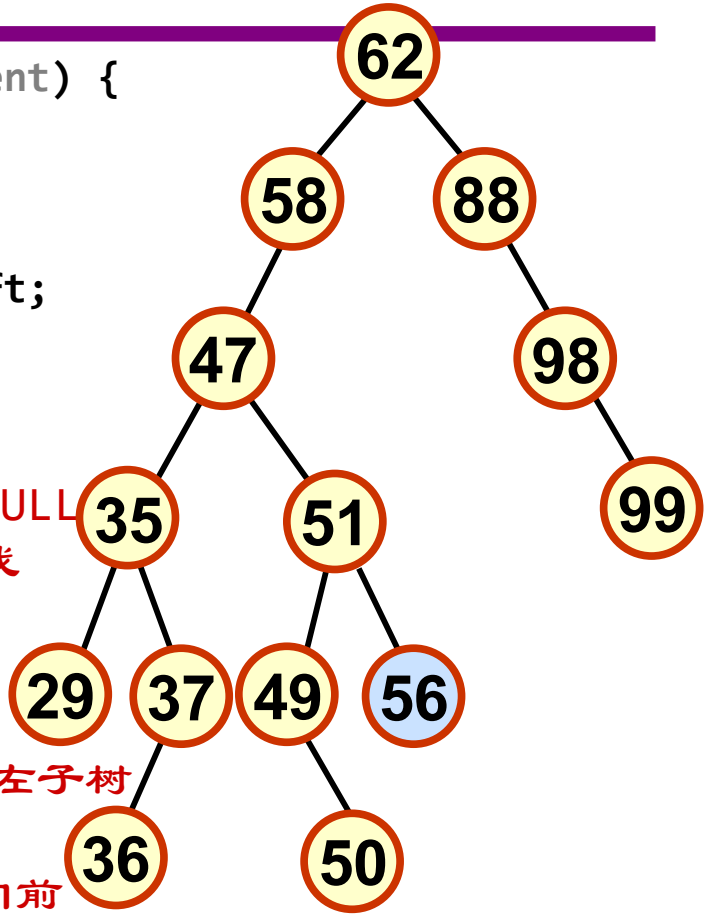
```
template <typename T> BinNodePosi(T)
BinNode<T>::succ() { //定位节点v直接后继
   BinNodePosi(T) s = this; //记录后继的临时变量
      if ( rc ) {
   //若有右孩子，则直接后继必在右子树中，具体地就是
            s = rc; //右子树中
            while ( HasLChild ( *s ) )
               s = s->lc; //最靠左（最小）的节点
      }
      else {
//否则，直接后继应是“将当前节点包含于其左子树中的
最低祖先”，具体地就是
          while ( IsRChild ( *s ) )
             s = s->parent;
              //逆向地沿右向分支不断朝左上方移动
          s = s->parent;
//最后再朝右上方移动一步，即抵达直接后继（如果存在）
      }
      return s;
}
```



56的直接后继为58，因58为祖先中，56位于其左子树，并且是最低的祖先

第六讲

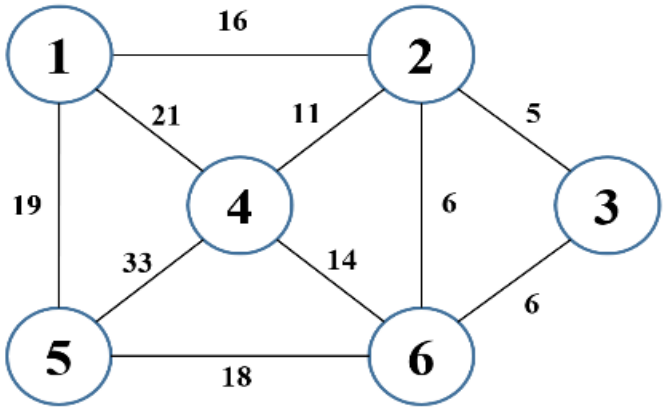
```
BSTNode* GetSuccessor(BSTNode* root, BSTNode* current) {
    //case1: Node has right subtree 存在右孩子
    if (current->right != NULL){
        BSTNode* temp = current->right;
        while (temp->left != NULL) temp = temp->left;
        return temp;
    }
    else { //case2: 不存在右孩子
        BSTNode* successor = NULL; //结果赋值初值NULL
        BSTNode* ancestor = root;  //从根节点开始找
        while (ancestor != current) {
             // 直到到达给定节点为止
            if (current->data < ancestor->data)
            { // 判断进入左子树还是右子树，若进入左子树
                successor = ancestor; //更新结果
                ancestor = ancestor->left; //继续向前
            }
            else // 继续向前
                ancestor = ancestor->right;
        }
        return successor;
    }
}
```

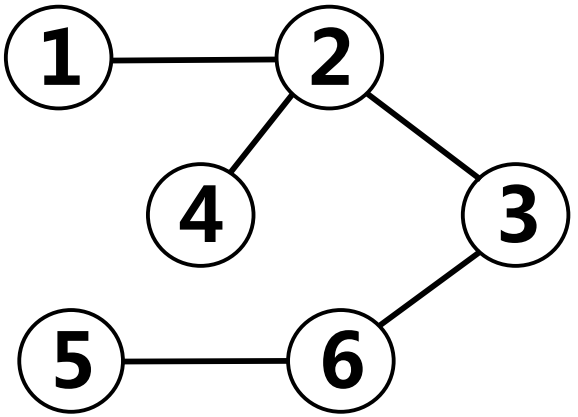


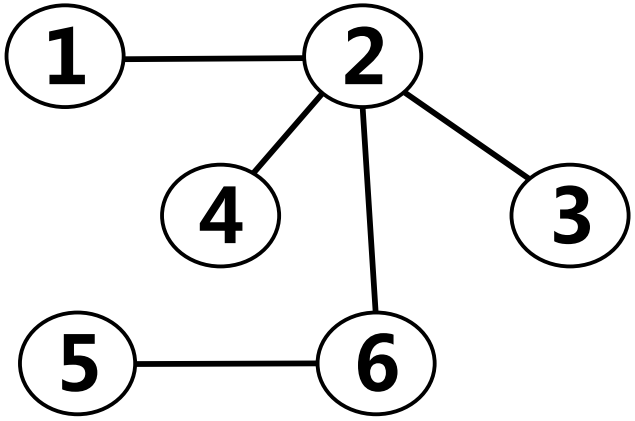
56的直接后继为58，因58为祖先中，56位于其左子树，并且是最低的祖先

第九讲

2．如下图表示一个地区的通讯网，边表示城市间的通讯线路，边上的权重表示架设线路花费的代价，如何选择能沟通每个城市且总代价最省的n-1条线路，画出所有可能的选择





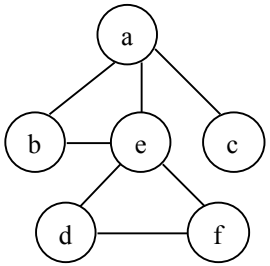


第七讲

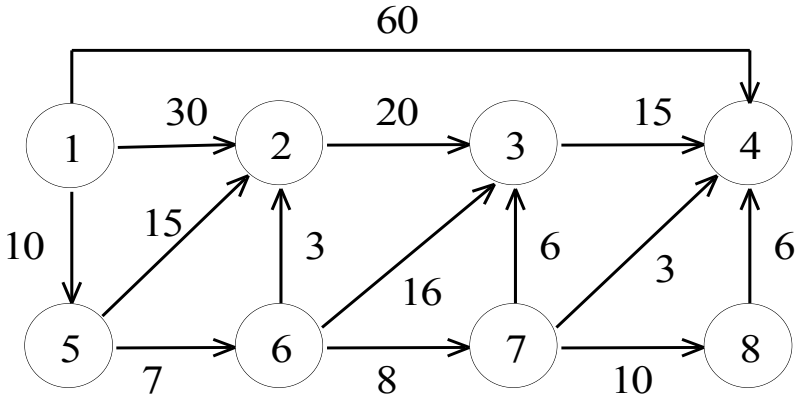
4. 设图所示，在下面的5个序列中，符合深度优先遍历的序列有多少？（D）

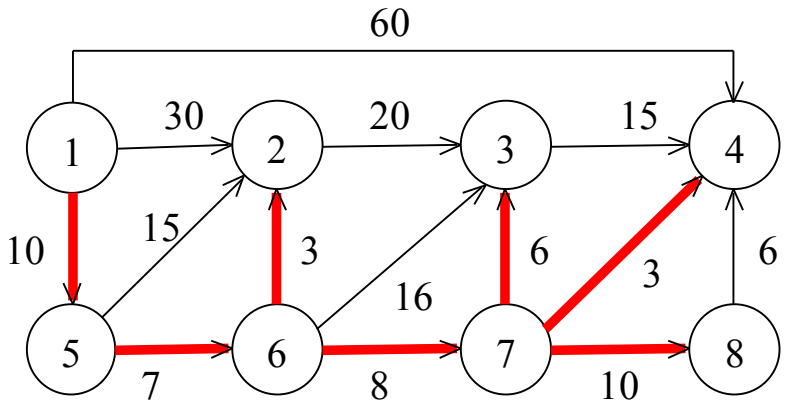
a e b d f c √　a c f d e b X　a e d f c b X　a e f d c b X　a e f d b c √

A．5个　　B．4个　　C．3个　　D．2个



5.求出下图中顶点1到其余各顶点的最短路径





第八讲

1. 对序列{50，10，90，30，70，40，80，60, 20}进行堆排序(大顶堆)，求第一步骤进行堆重建后，向量为？90 70 80 60 10 40 50 30 20

2. 堆重建后，若插入元素100，进行上滤后，请列出向量情况
   100 90 80 60 70 40 50 30 20 10

3. 在进行2之后，对向量进行堆排序，需进行置换+下滤的迭代， 请列出每次迭代向量中各元素值情况

90 70 80 60 10 40 50 30 20 100
70 60 50 30 10 40 20 80 90 100
50 30 40 20 10 60 70 80 90 100
30 20 10 40 50 60 70 80 90 100
10 20 30 40 50 60 70 80 90 100

80 70 50 60 10 40 20 30 90 100
60 30 50 20 10 40 70 80 90 100
40 30 10 20 50 60 70 80 90 100
20 10 30 40 50 60 70 80 90 100

4. 若在现有堆中修改某一元素关键码值，为维护堆序性应如何修改？时间复杂度是？若修改值增大，尝试上滤；若修改值减小，尝试下滤

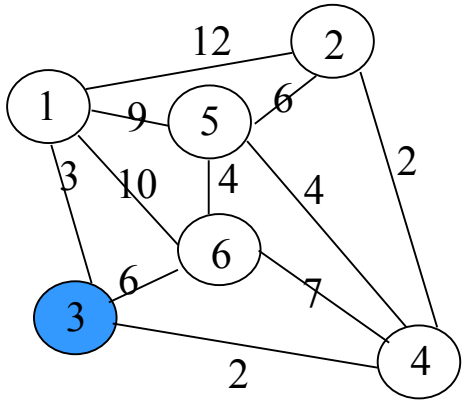
第九讲

1．n个顶点的有向图用邻接矩阵array表示，下面是其拓扑排序算法，试补充完整。
注：（1）．图的顶点号从 0开始计；（2）．indegree 是有n个分量的一维数组，放顶点的入度；（3）．函数 crein 用于算顶点入度；（4）．有三个函数push(data),pop( ),check( )其含义为数据 data进栈，退栈和测试栈是否空（不空返回1，否则0）。

```
crein( array ,indegree,n) {
      for (i=0;i<n;i++)  indegree[i]= ((1)__0__)
      for (i=0,i<n;i++)
            for (j=0;j<n; j++)  indegree[i]+=array[(2)_j_][(3)_i_];
 }
 topsort (array,indegree,n) {
      count=  ((4)__0__)
       for (i=0;i<n;i++)  if ((5)__indegree[i]==0__)  push(i)
       while (check( )) {
            vex=pop(  );  printf(vex);  count++;
             for (i=0;i<n;i++) {
                  k=array(6)_[vex][i]_
                  if ((7)_k==1__ )  { indegree[i]--;  if ((8)_indegree[i]==0 )  push(i); }
              }
        }
       if( count<n)   printf(“图有回路”);
}
```

第九讲

3．给定n个村庄之间的交通图，若村庄i和j之间有道路，则将顶点i和j用边连接，边上的W_{ij}表示这条道路的长度，现在要从这n个村庄中选择一个村庄建一所医院，问这所医院应建在哪个村庄，才能使离医院最远的村庄到医院的路程最短? 试设计一个解答上述问题的算法，并应用该算法解答如图所示实例。



可用求每对顶点间最短路径的FLOYD算法求解。求出每一顶点（村庄）到其它顶点（村庄）的最短路径。在每个顶点到其它顶点的最短路径中，选出最长的一条。因为有n个顶点，所以有n条，在这n条最长路径中找出最短一条，它的出发点（村庄）就是医院应建立的村庄。

```
void Hospital(AdjMatrix w,int n)
  {for (k=1;k<=n;k++)   //求任意两顶点间的最短路径
     for (i=1;i<=n;i++)
       for (j=1;j<=n;j++)
         if(w[i][k]+w[k][j]<w[i][j])  w[i][j]=w[i][k]+w[k][j];
    m=MAXINT;              //设定m为机器内最大整数。
    for (i=1;i<=n;i++)     //求最长路径中最短的一条。
      {s=0;
       for (j=1;j<=n;j++) //求从某村庄i（1<=i<=n）到其它村庄的最长路径。
         if (w[i][j]>s) s=w[i][j];
       if (s<=m) {m=s; k=i;}//在最长路径中，取最短的一条。m记最长路径，k记出发顶点的下标。
       Printf(“医院应建在%d村庄，到医院距离为%d\n”,i,m);
       }//for
  }//算法结束
```

对以上实例模拟的过程略。在$A^{(6)}$矩阵中,各行中最大数依次是9，9，6，7，9，9。这几个最大数中最小者为6，故医院应建在第三个村庄中,离医院最远的村庄到医院的距离是6。

第九讲

5. 自由树(即无环连通图)T=(V,E)的直径是树中所有点对间最短路径长度的最大值,即T的直径定义为MAX D(u,v),这里D(u,v)（u,v∈V）表示顶点u到顶点v的最短路径长度(路径长度为路径中所包含的边数)。写一算法求T的直径,并分析算法的时间复杂度。(时间复杂度越小越好)

依次删去树叶(度为1的结点)，将与树叶相连的结点度数减1。设在第一轮删去原树T的所有树叶后，所得树为T_1；再依次做第二轮删除，即删除所有T_1的叶子；如此反复，若最后剩下一个结点，则树直径应为删除的轮数×2，

具体算法如下：

```
int SZJ() {
    int m=0;
    for(i=0; i≤n; i++)
        if(degree(v)==1){
            enqueue(Q, v[i]); // 叶子vi入队
            m=m+1; // m记录当前一轮叶子数
        }
    int r=0; // 表示删除叶子轮数
    while(m>=2){ // 终止条件，仍有足够的叶节点
        j=0; // j计算新一轮叶子数目
        for(i=1; i<=m; i++){
            dequeue(Q, v); // 出队
            if(edge(v,w)) degree(w)--;
            // 删去叶子v将与v相邻的结点w的度数减1
            if(degree(w)==1){ // w是新一轮的叶子
                j=j+1;
                enqueue(Q, w); // w入队
            }
        }
        r=r+1; // 轮数加1
        m=j; // 新一轮叶子总数
    }
    if(m==0) return 2*r-1 // m=0, 直径为轮数*2-1
    else return 2*r; // m=1, 直径为轮数*2
}
```

复杂度O(n)

第十二讲

1. 平均复杂度为O(nlogn)，但最坏情况复杂度为O(n^2)的排序算法为快速排序，其最坏情况复杂度在初值基本有序，采用首元素作为轴点的情况下取得。
2. 当排序元素个数较少时（如小于20个），插入排序算法最合适；冒泡和插入排序算法最优情况下的复杂度为O(n)。
3. 选择排序算法在不同具体实现下具有不一样的排序稳定性；归并排序算法达到O(nlogn)的复杂度，并且一定是稳定的排序算法。
4. 希尔排序算法基于插入排序改进，其主要核心思想是由粗到细的分组策略，提高单个交换所去除的逆序对数目。
5. 快速排序比归并排序更实用的原因为 归并排序需要额外的空间，并且每次归并需要进行额外的数据拷贝，而快速排序是就地排序，无需额外的拷贝
6. 快速排序比堆排序更实用的原因为_虽然复杂度量级相当，快速排序一般比堆排序更快，因堆排序在进行堆调整时效率低，每次删除堆顶元素后的置换调整产生新的无序对，做无用功，并且堆元素的访问总是跳变，寻址速度低。

第十二讲

有1000个无序的元素，找出其前十个最小的元素的最好方法是C

A. 冒泡排序； B.快速排序； C.堆排序； D.选择排序；

以下不属于快速排序优势的说法的是_D_

A．快速排序属于就地排序，无需内存搬移，速度快

B．快速排序寻址访问较为连续，速度快

C. 相比于冒泡排序，快速排序每次交换减少逆序对更多，速度快

D. 快速排序在数据基本有序的输入下能降低排序复杂度