



# 数据结构与算法设计 习题课-II





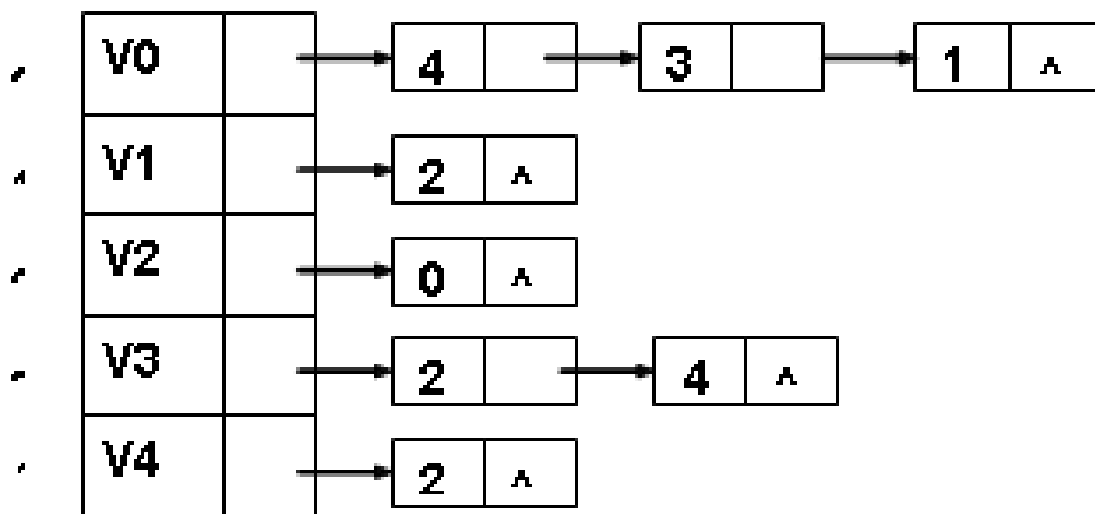
# 内容

- ◆ 第6章 图
- ◆ 第7章 查找
- ◆ 第8章 排序



## 第6章 图

- ◆ 1、有向图的邻接表（出边表）如图所示。请完成下列小题：
- ◆ （1）画出该图；
- ◆ （2）给出以V0为起始结点的深度优先遍历序列和广度优先遍历序列；
- ◆ （3）简述在邻接表存储结构下，求图中某顶点i的入度的方法；





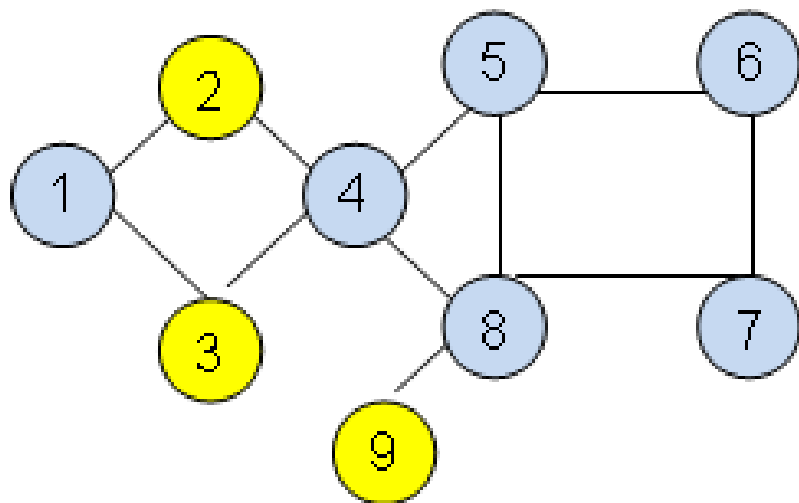
# 思考题

- ◆ 2、假设图用邻接表来表示，请编写对图的深度优先“非递归”遍历算法。



# 思考题

- ◆ 3、已知无向图G，写一个算法判断是否存在一条从v0出发且包含所有结点的简单路径？





```
Status _DFSSearch(Graph G, int v, SqList &PATH int num) {  
    //深度优先搜索的递归程序， 回溯法  
    // PATH当前路径; //num 当前路径包含节点数;  
    visited[v] = TRUE; // 访问第 v 个顶点  
    num ++;  
    ListAppend_Sq(PATH, G.vertices[v].data); //v点添加到路径  
    if (num == G.n ) return TRUE;  
    for(w = FirstAdjVex(G, v); w != -1; w = NextAdjVex(G, v, w))  
        if (!visited[w])  
            if (_DFSSearch (G, w, PATH, num)) return TRUE;  
    ListDelete_Sq(PATH,G.vertices[v]. data);  
    visited[v] = FALSE; // 取消顶点v的访问标志  
    return FALSE;  
}// _DFSSearch
```





# 思考题

- ◆ 4、已知无向图 $G$ ，写一个算法判断 $v_0$ 和 $v_1$ 之间是否存在一条长度小于等于 $k$ 的路径。
- ◆ 广度优先遍历





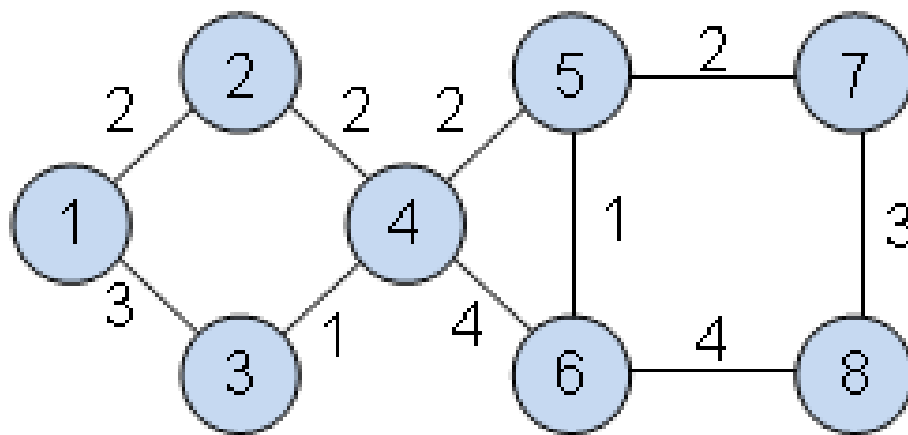
- ◆ 5、试扩充深度优先搜索算法，在遍历图的过程中建立生成森林的子女-兄弟链表。
- ◆ 提示：在继续按深度方向从根结点 $v$ 的某一未访问过的邻接顶点 $w$ 向下遍历之前，建立子女结点。但是需要判断是作为根的第一个子女还是第二个以后的子女。





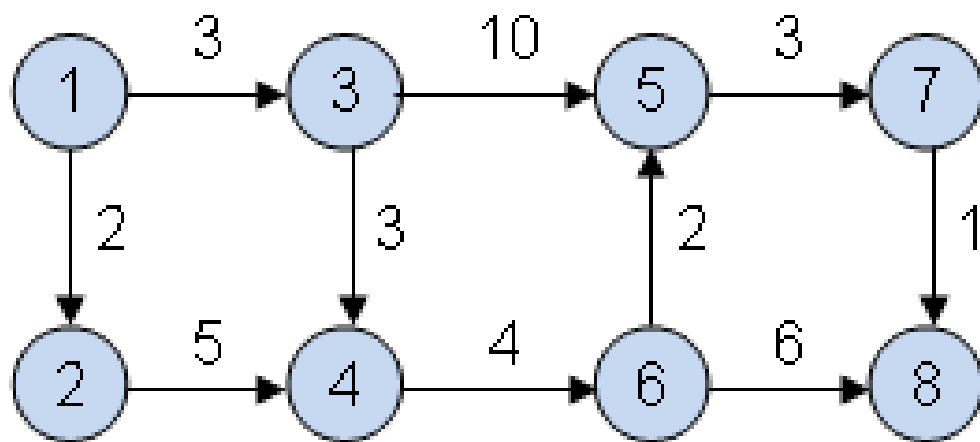
# 思考题

- ◆ 7、对于如图所示的带权无向图，
- ◆ 按照Prim算法求其最小生成树，并给出在构造最小生成树算法过程中辅助数组各个分量的值。



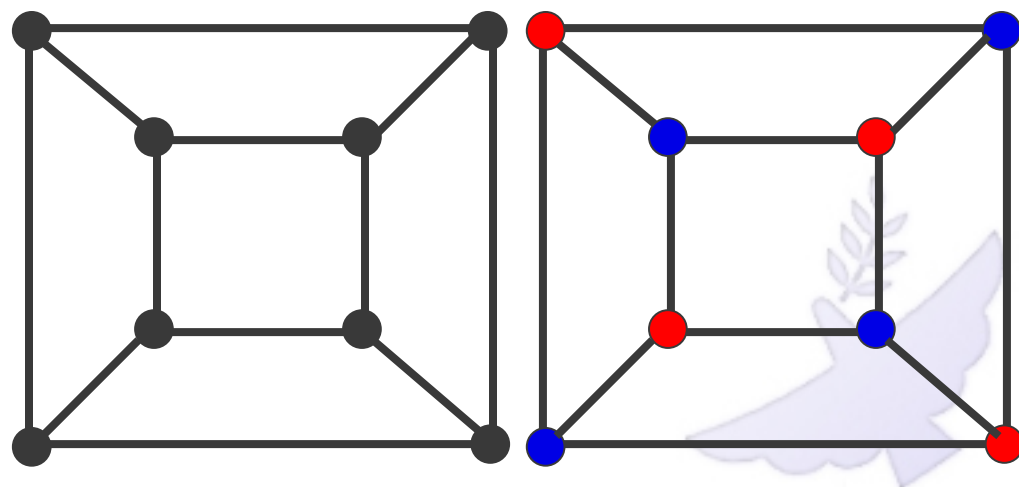
# 思考题

- ◆ 10、对于右图所示的带权有向图
- ◆ (1) 求解从v1到其它各点的最短路径。
- ◆ (2) 给出所有的拓扑排序。
- ◆ (3) 以v1为源点，以v8为终点，给出所有时间允许发生的最早时间和最晚时间，并给出关键路径。



# 思考题

- ◆ 11、二部图判定问题：对于无向图 $G=(V, E)$ ，若能将其结点集合 $V$ 分为两个不相交的子集 $V_1$ 和 $V_2 = V - V_1$ ，使得 $V_1$ 中的任何两个结点在图 $G$ 中均不相邻， $V_2$ 中的任何两个结点在图 $G$ 中也均不相邻，则 $G$ 为二部图。请写一个算法判断 $G$ 是否是二部图。





**Status bBigraph(Graph G, VexIndex v )**

{//从第v个顶点出发，广度优先遍历G,对点进行2-着色。

InitQueue(Q); color[ ] == NULL;

c1 = 0; c2 = 1; ////0 **RED** 1**BLUE**;

color[v]= c1; EnQueue(Q,v) // v着红色，入队

while(!QueueEmpty(Q))

{ DeQueue(Q,u); //队头元素出队,并赋值给u

c1 = color[u]; c2 = ~c1;

for(w=FirstAdjVex(G,u); w; w=NextAdjVex(G,u,w))

if(color[w] == c1) return false;

else if(color[w] == NULL){

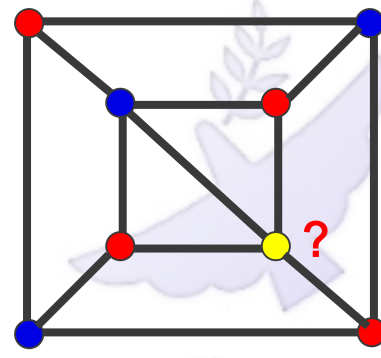
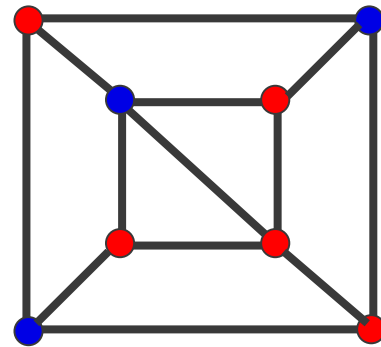
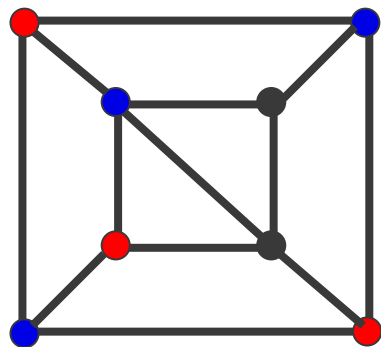
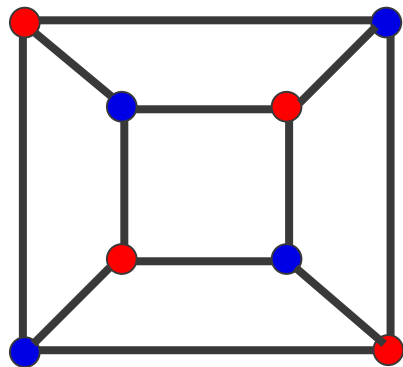
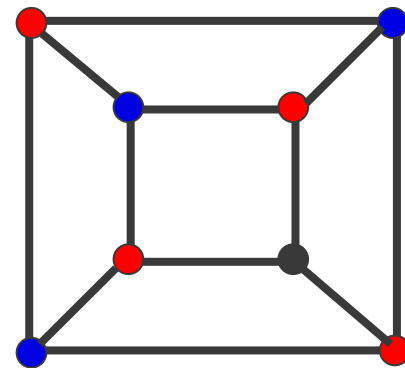
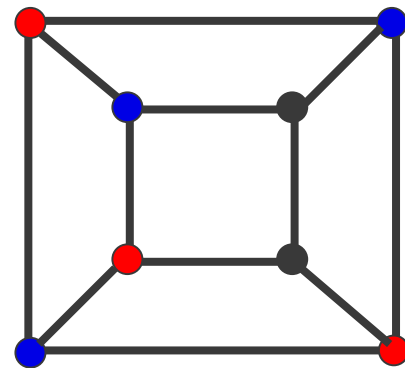
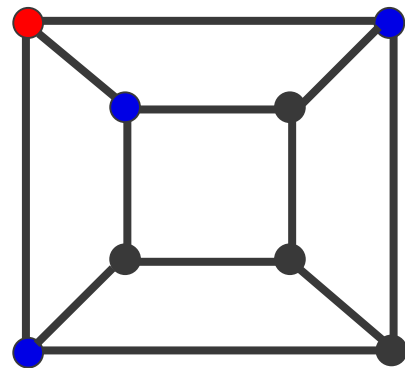
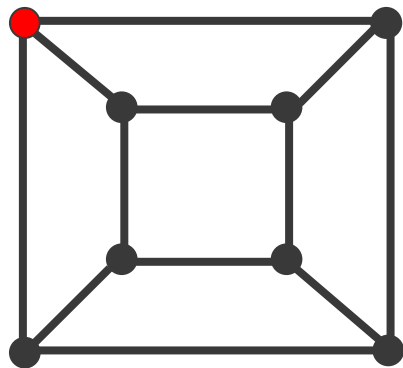
color[w]= c2; EnQueue(Q,w);}

}//while(!QueueEmpty(Q))

return true;

}// bBigraph







# 思考题

- ◆ 12、如何判断图G中是否存在回路？请给出至少两种算法。
- ◆ 注意：对于无向图和有向图采用的算法是不同的。
- ◆ 方法一：拓扑排序（有向图）
- ◆ 方法二：强连通分量（有向图）
- ◆ 方法三：DFS（有向图）
- ◆ 方法四：DFS'（无向图）





## ◆ 方法四：DFS'（无向图）

- 🔑 深度优先遍历图，如果在遍历的过程中，发现某个结点有一条边指向已访问过的结点，并且这个已访问过的结点不是上一步访问的结点，则表示存在环。
- 🔑 规定每个结点都拥有三种状态，白、灰、黑。
- 🔑 开始时所有结点都是白色，当访问过某个结点后，该结点变为灰色，当该结点的所有邻接点都访问完，该节点变为黑色。
- 🔑 那么我们的算法可以表示为：如果在遍历的过程中，发现某个结点有一条边指向灰色节点，并且这个灰色结点不是上一步访问的结点，那么存在环。



- ◆ 13、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用于计算顶点入度；
- ◆ （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, int 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, int 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((7))**

**// for**

**// while(check())**

**if(count<n) printf(“图有回路”);**

## 第七章 查找

- ◆ 用分块查找法，对于有2000个数据项的表分成多少块最理想？每块的理想长度是多少？在你的分块方式下平均查找长度是多少？
- ◆ 假设将2000个数据项分为d组，每组y个数据项，
- ◆  $d * y = 2000$ 。
- ◆ （1）若对分块内部采用顺序查找，对索引表也采取顺序查找，则
- ◆  $ASL = (y+1)/2 + (d+1)/2$
- ◆ 令  $t = 2 * ASL - 2$ ，则  $t = y + d = y + 2000/y$
- ◆ 求t的极大值：  $t' = 1 - 2000/y^2$ ，
- ◆ 令  $t' = 0$ ,  $y^2 = 2000$ . 所以  $y=44.72$ ,  $y = 45$ ,  $d = 45$ .



◆ 写一个算法判断给定的关键字序列 $k_1, k_2, \dots, k_n$ 是否为有序表中进行折半查找过程中可能出现的关键字比较序列。例：

‖ 10, 20, 15, 12 (正确)

‖ 10, 20, 15, 8 (错误)

基本思想：设置两个参数左界和右界。

1) 初始时，左界设为负无穷，右界设为正无穷。

2) 从 $k_2$ 开始依次对当前元素 $k_i$ 进行如下判断，直到处理完所有关键字。

如果 $k_i$ 比 $k_{i-1}$ 大则将左界更新为 $k_{i-1}$ ，否则将右界更新为 $k_{i-1}$ 。

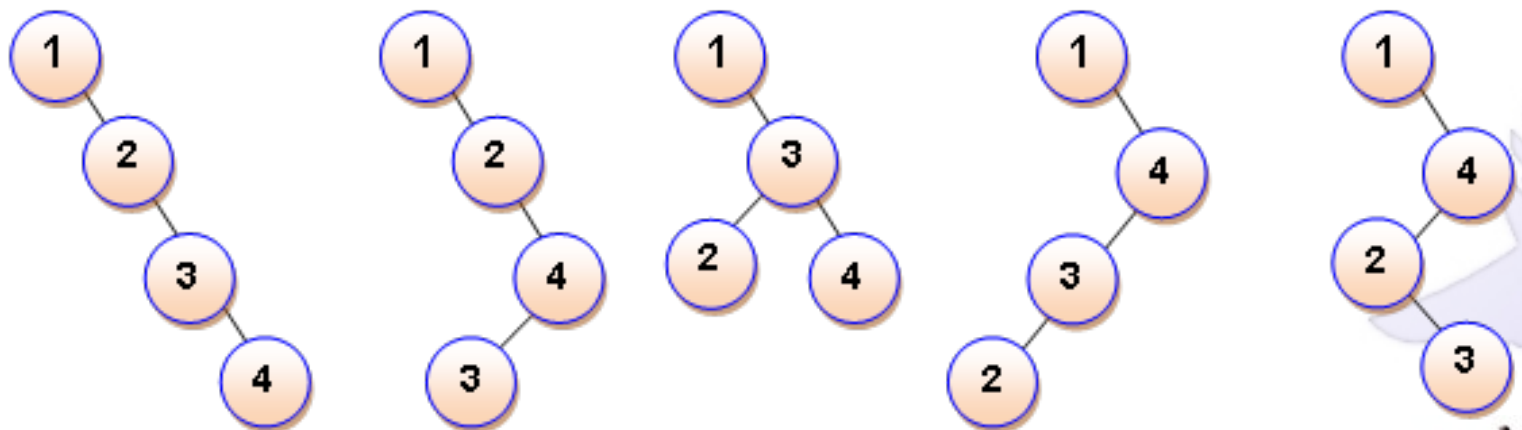
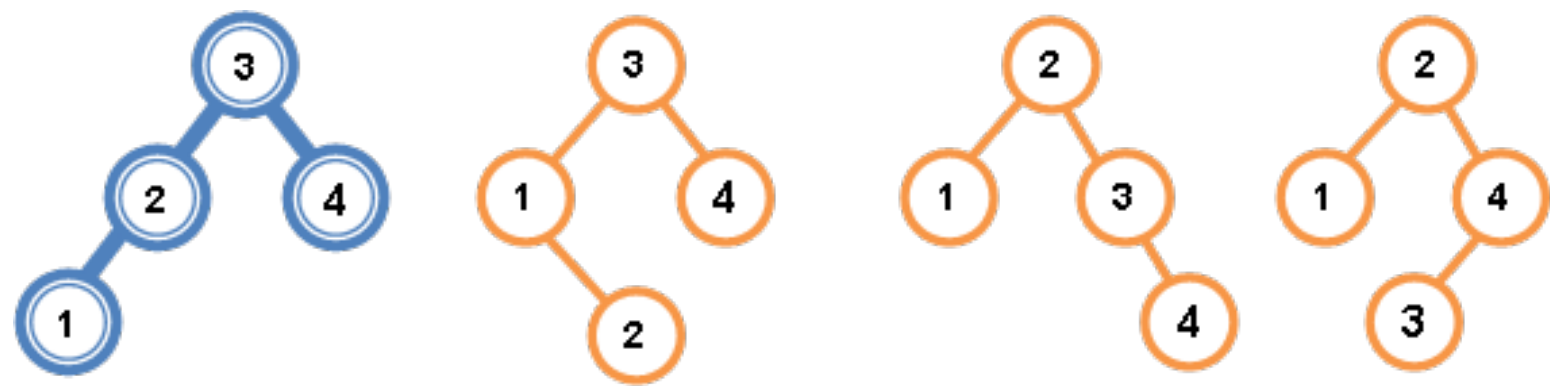
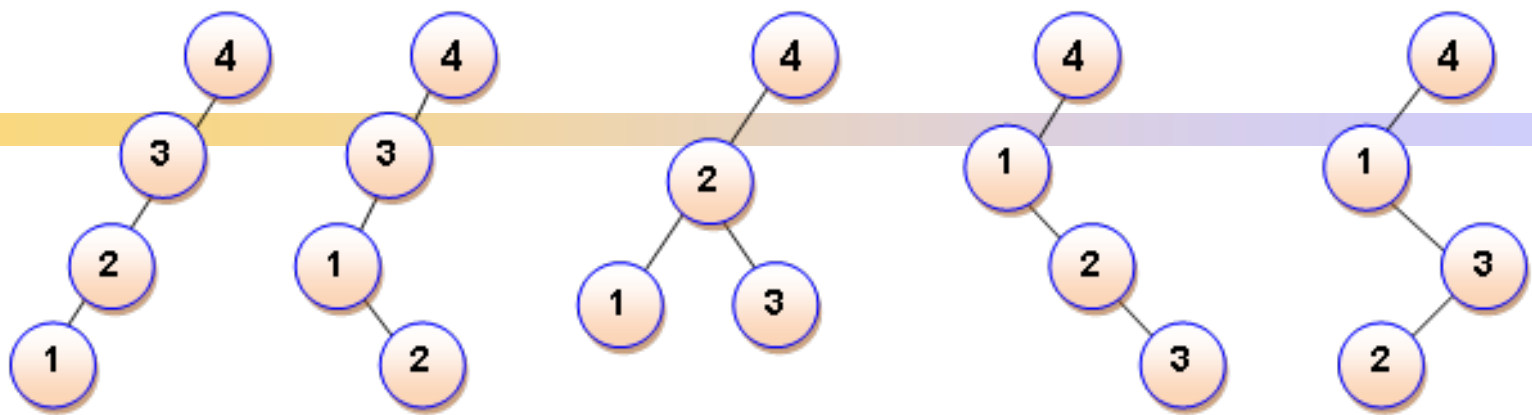
如果 $k_i$ 不在左右界之内，则说明给定的序列不是折半查找的序列，返回false。

3) 返回true。



- ◆ 已知关键字为1、2、3、4的四个节点，是回答下列问题：
- ◆ （1）能构造出几种不同的二叉排序树？其中哪些是最优查找树（假设每个节点查找的概率相同）？
- ◆ （2）能构造出几种不同的AVL树？
- ◆ (1)  $b_0=1, b_1=1, b_2=2$
- ◆  $b_3=b_2*b_0+b_1*b_1+b_0*b_2=5$
- ◆  $b_4=b_0*b_3+b_1*b_2+b_2*b_1+b_3*b_0 = 14$







- ◆ 写一个算法将一棵二叉排序树分裂为两棵二叉排序树，使得其中一棵上所有节点的关键字都小于或等于 $x$ ，而另一棵中树所有节点的关键字都大于 $x$ 。假设分裂算法的定义如下：
- ◆ `int BSTree_Split( BST T, BST & T1, BST & T2, int x);`
  - 🔧 方法一：递归先序遍历，将所有大于 $x$ 的节点插入到 $T2$ 中，小于等于 $x$ 的节点插入到 $T1$ 中。
  - 🔧 方法二：中序遍历 $T$ ，找到最后一个小于等于 $x$ 的节点，然后继续中序遍历，将后面遍历到的节点都从 $T$ 中删除，插入到 $T2$ 中。





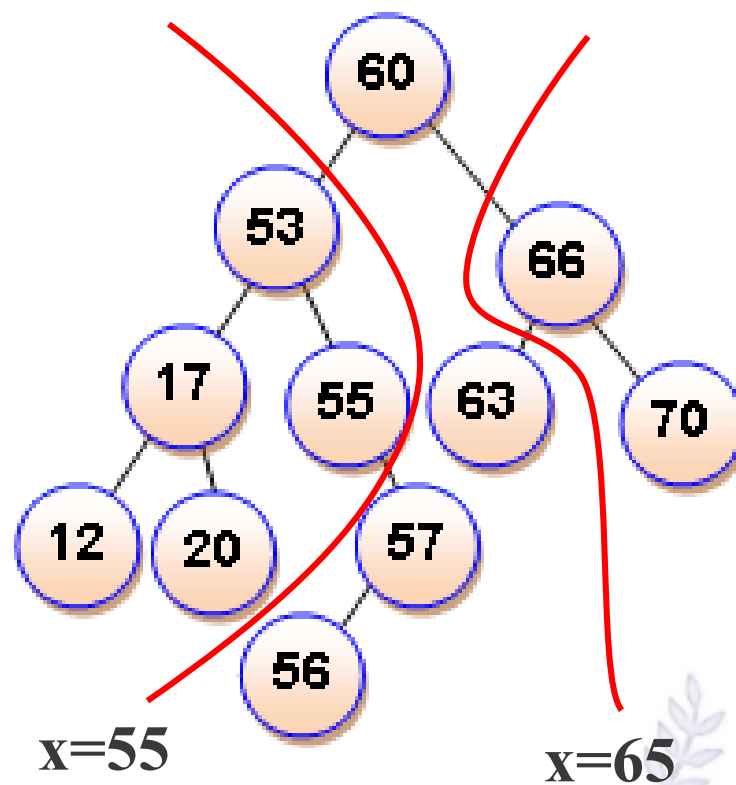
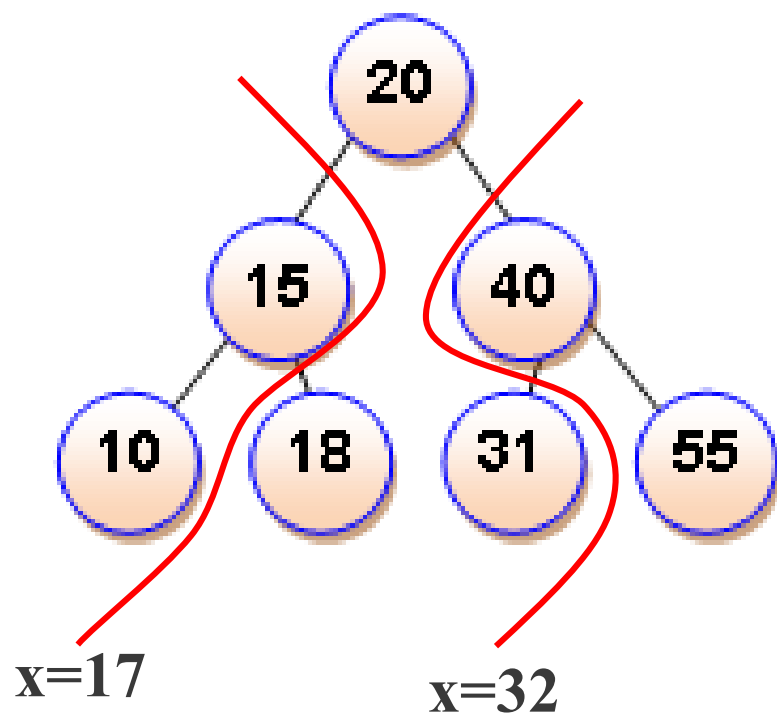
**/\*方法一：递归先序遍历，将所有大于x的节点插入到T2中，小于等于x的节点插入到T1中。\*/**

```
void split(BSTree T, ElemType e, BSTree &T1, BSTree &T2){  
    if(!T) return;  
    if(T->data > e)  
        InsertBT (T2, T->data);  
    else  
        InsertBT(T1, T->data);  
    split(T->lchild, e, T1, T2);  
    split(T->rchild, e, T1, T2);  
}
```





### 方法三：直接进行分裂







方法四：从根节点T开始，判断当前结点与x的大小关系

1)  $x == T \rightarrow \text{data}$ :

    Insert(T2, T->rchild);

    T->rchild = NULL;

    Insert(T1, T);

2)  $x < T \rightarrow \text{data}$ :

    lp = T->lchild;

    T->lchild = NULL;

    Insert(T2, T);

    Split(lp, T1, T2);

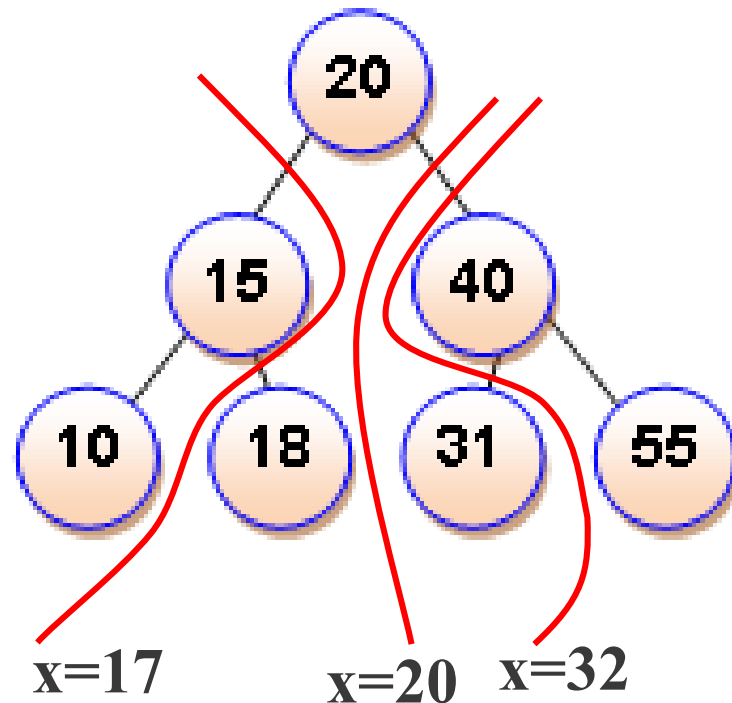
3)  $x > T \rightarrow \text{data}$ :

    rp = T->rchild;

    T->rchild = NULL;

    Insert(T1, T);

    Split(rp, T1, T2);





## ◆ 写一个算法判断给定的二叉树是否是平衡二叉树。

```
int isBalanced2(BiTree T, int & depth){
    if(T == null){    depth = 0;        return true;    }
    int left = 0, right = 0;
    if(isBalanced2(T->lchild, left) && isBalanced2(T->rchild, right)){
        int diff = left-right;
        if(diff <= 1 && diff >= -1){
            depth = 1+(left > right?left : right);
            return true;
        } // if(diff <= 1 && diff >= -1)
    } //if(isBalanced2(T->lchild, left)
    return false;
} //isBalanced2
int isBalancedTree(BiTree T){ // 判断T是否是平衡树
    int depth = 0;
    return isBalanced2(T, depth);
}
```



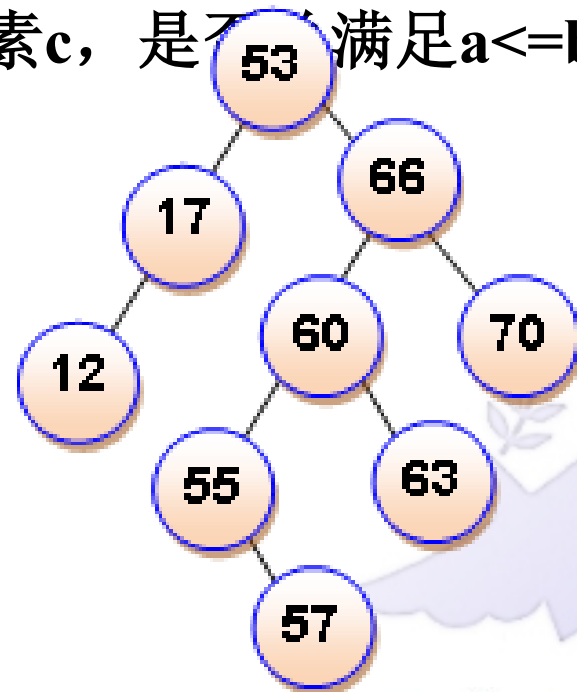
- ◆ 写一个算法判断给定的关键字序列 $k_1, k_2, \dots, k_n$ 是否为二叉排序树上查找过程中可能出现的关键字比较序列。

- ◆ 设两个边界(lbound, rbound);
- ◆ 初始时lbound = 负无穷, rbound = 正无穷;
- ◆ 从第一个关键字开始依次扫描所有关键字, 当前关键字为 $k_i$ :
  - 否则 若 $k_i > k_{i-1}$ , 则lbound =  $k_{i-1}$ ;
  - 否则rbound =  $k_{i-1}$ ;
  - 若 $k_i$ 不在(lbound, rbound)范围内, 则 return 0;return 1;



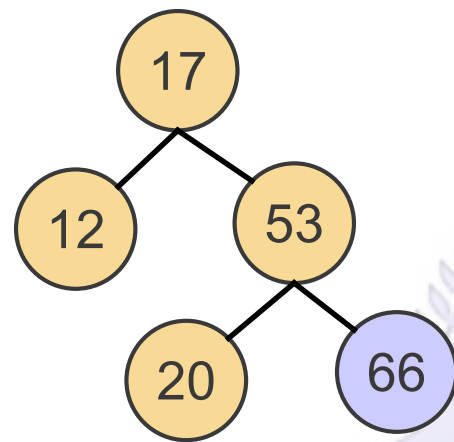
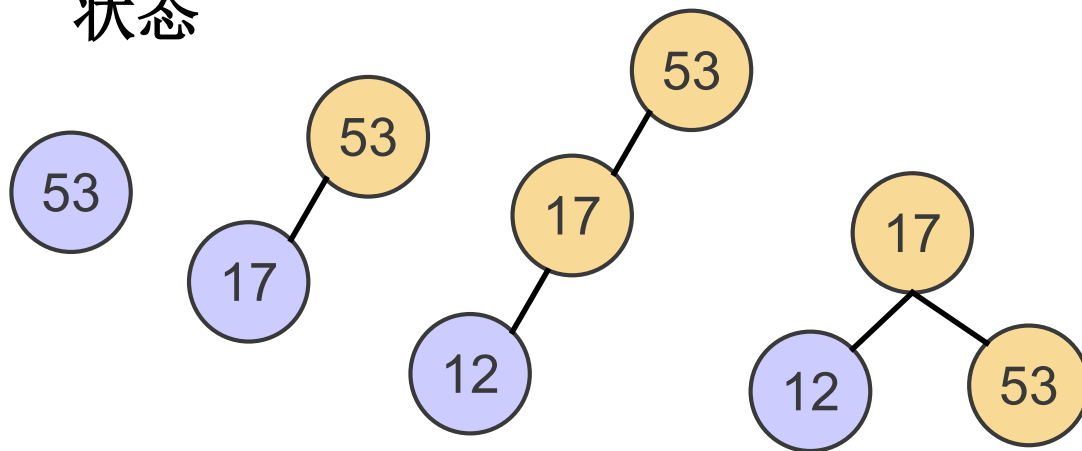


- ◆ 在一棵二叉排序树S中，任意一条从根节点到叶子节点的路径path将S中的所有节点划分为三个集合
- ◆ 在path左边的节点组成集合S1，
- ◆ 在path上的节点组成集合S2，
- ◆ 在path右边的节点组成集合S3。对于任意属于S1的元素a，任意属于S2的元素b和任意属于S3的元素c，是否满足 $a \leq b \leq c$ ？请证明你的结论。
- ◆ 考察下列路径：
- ◆  $53 \rightarrow 57$
- ◆  $53 \rightarrow 63$



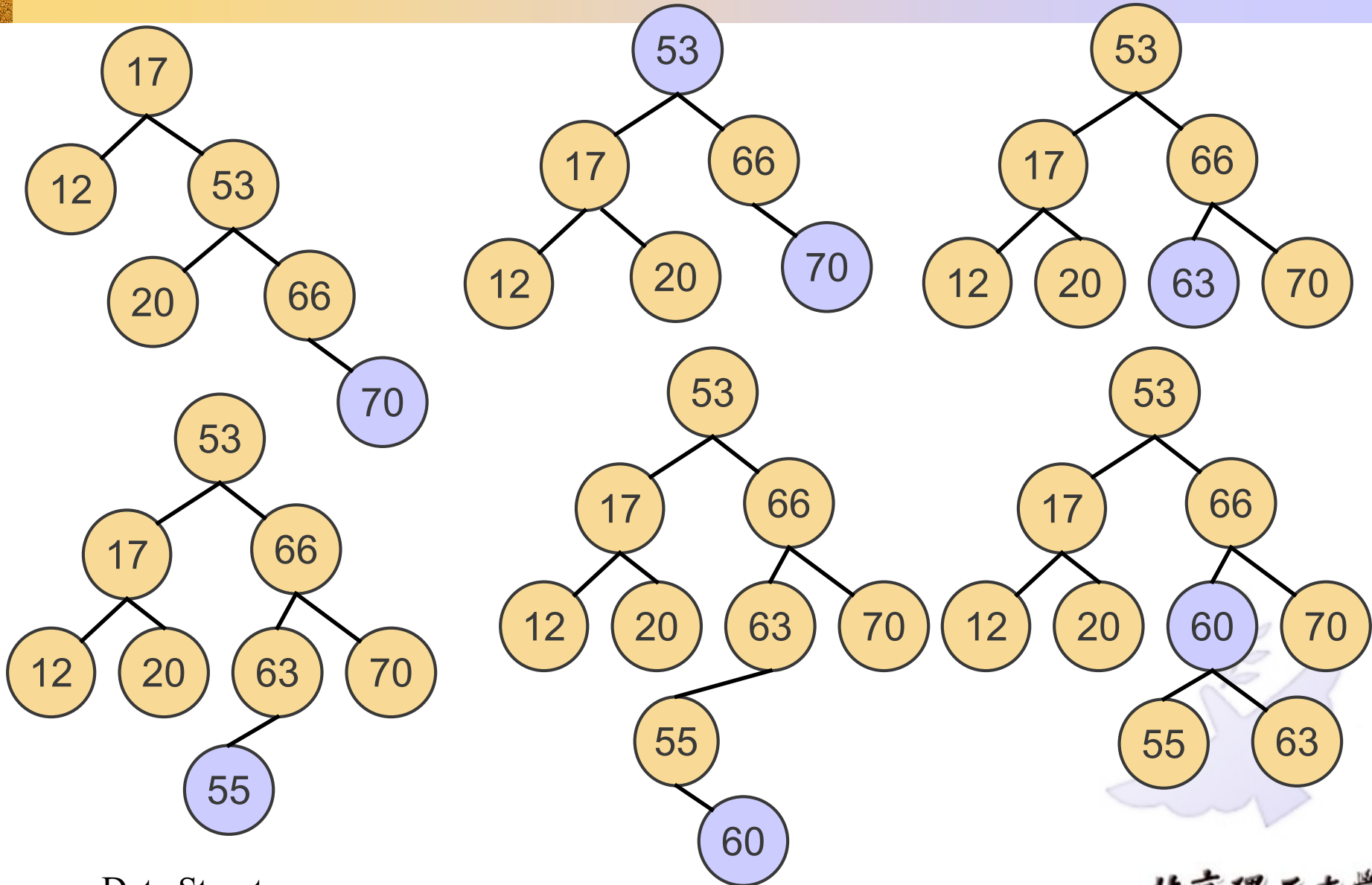


- ◆ 设有一个关键字序列{53, 17, 12, 66, 20, 70, 63, 55, 60, 57, 56}, 请完成下列各个小题:
- ◆ 按照上述顺序, 通过依次插入关键字, 构造一棵平衡二叉树. 请画出每插入一个关键字后树的状态.
- ◆ 假设依次删除关键字66、63, 请画出每插入一个关键字后树的状态





{53, 17, 12, 66, 20, 70, 63, 55, 60, 57, 56}

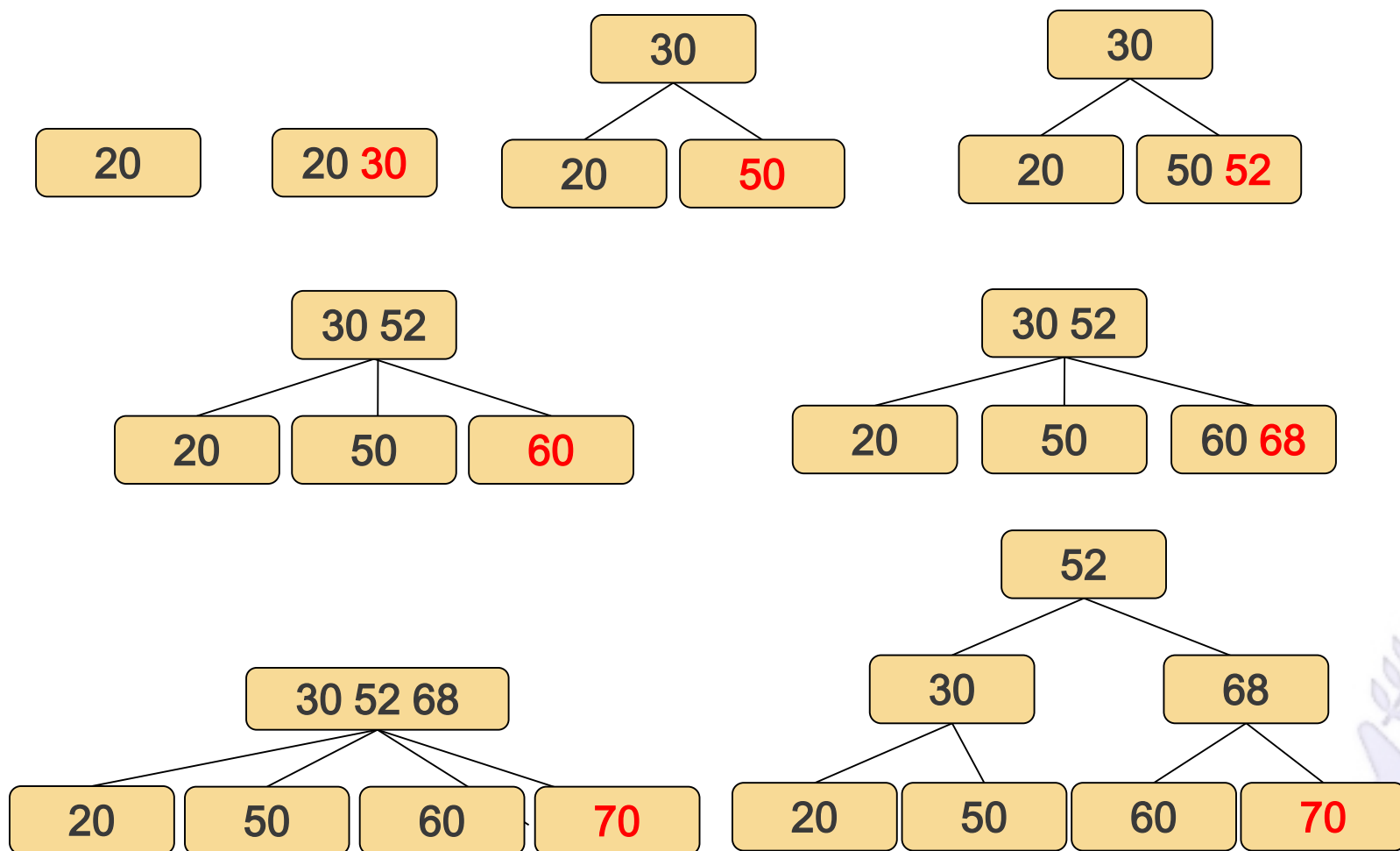




- ◆ 请按照下列顺序依次向一棵空的3阶B-树中插入关键字，逐步建立一棵树。
- ◆ 20,30,50,52,60,68,70
- ◆ (1) 请画出该树一步步的创建过程。
- ◆ (2) 树建立后，依次删除50和68，请依次画出树变化后的状态



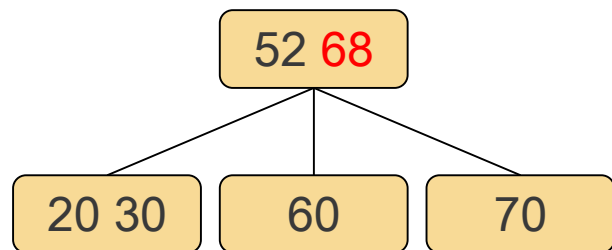
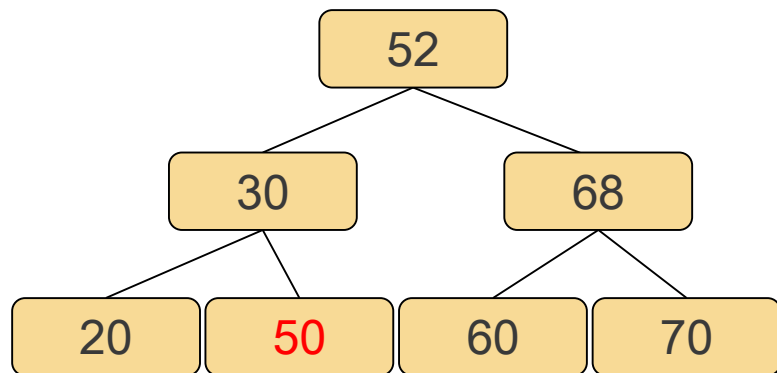
20,30,50,52,60,68,70



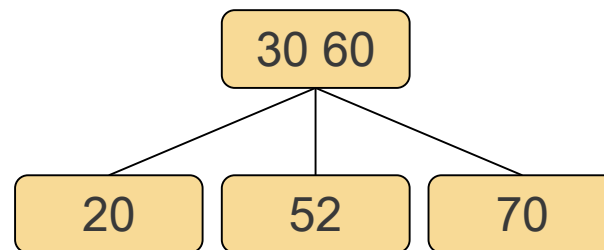




# 20,30,50,52,60,68,70



删除50



删除68



- ◆ 请回答下列有关B-树的问题：
- ◆ （1）高度为5（除叶子层之外）的4阶B-树至少包含多少个关键字？至少有多少个节点？
- ◆ （2）高度为5（除叶子层之外）的4阶B-树最多包含多少个关键字？最多有多少个节点？
- ◆ （3）含9个叶子结点的3阶B-树中至少有多少个非叶子结点？含10个叶子结点的3阶B-树中至多有多少个非叶子结点？

- **m阶B-树性质3：**

- 树中所有叶子结点均在树中的同一层次上；
- 根结点或为空, 或至少含有两棵子树；
- 其余所有结点均至少含有 $\lceil m/2 \rceil$ 棵子树, 至多含有  $m$  棵子树；





- ◆ 4阶B-树每个节点至少含有2棵子树，至多含有 4 棵子树.
- ◆ 所以高度为5的4阶B-树，
- ◆ 至少含有：  $1 + 2^1 + 2^2 + 2^3 + 2^4 = 31$  个节点； 31个关键字
- ◆ 至多含有：  $1 + 4^1 + 4^2 + 4^3 + 4^4 = 341$  个节点；  $341 \times 3$  个关键字

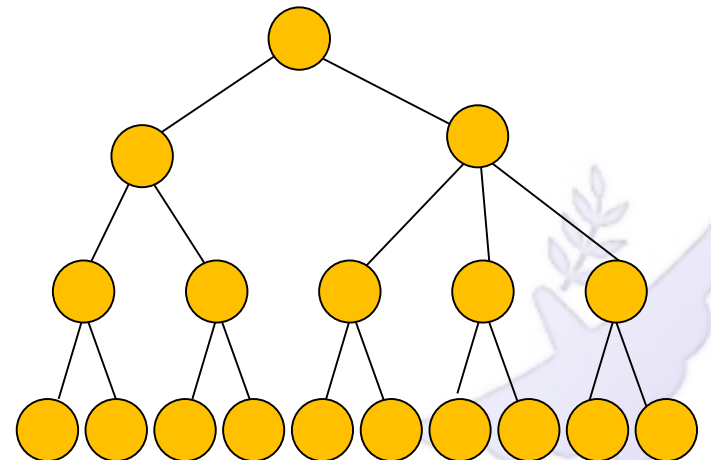
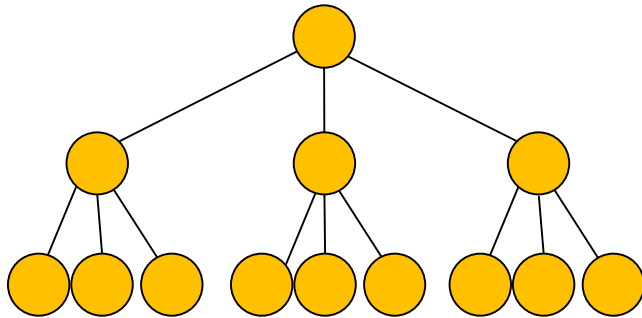
- **m阶B-树性质3:**

- 树中所有叶子结点均在树中的同一层次上；
- 根结点或为空, 或至少含有两棵子树；
- 其余所有结点均至少含有  $\lceil m/2 \rceil$  棵子树, 至多含有  $m$  棵子树；





- ◆ (3) 含9个叶子结点的3阶B-树中至少有多少个非叶子结点?  
含10个叶子结点的3阶B-树中至多有多少个非叶子结点?
- ◆ 3阶B-树每个节点至少含有2棵子树, 至多含有 3棵子树.
- ◆ 9个叶子结点, 满3叉树, 非叶子结点最少, 4个
- ◆ 10个叶子结点, 非叶子结点最多8个





- ◆ 已知一组记录的关键字为{25,40,33,47,12,66,72,87,94,22,5,58}，他们存储在散列表中，利用双散列函数解决冲突。要求向表中插入新数据的平均查找次数不超过3次。
- ◆ (1) 该散列表的大小应该设计为多大？
- ◆ (2) 设计散列函数（用除留余数法），并设计出现冲突时所需的再散列函数。
- ◆ (3) 应用你设计的双散列函数将上述关键字存储到散列表中。





◆ (1) 根据查找成功时的平均查找次数

$$S_{nl} \approx \frac{1}{2} \left( 1 + \frac{1}{1-\alpha} \right) \leq 3$$

所以  $\alpha \leq 4/5$ ,  $\alpha = 12/m$ ,  $m = 16$ , 最近的质数为 17

$$(2) H(\text{key}) = (\text{key} + d_i) \text{ MOD } 17$$

	25	40	33	47	12	66	72	87	94	22	5	58
mod 17	8	6	16	13	12	15	4	2	9	5	5	7

再散列函数:  $H_2(\text{key}) = \text{key} * 5 \text{ MOD } 13 + 1$   
 $d_i = i * H_2(\text{key})$





	25	40	33	47	12	66	72	87	94	22	5	58
mod 17	8	6	16	13	12	15	4	2	9	5	5	7

再散列函数:  $H_2(\text{key}) = \text{key} * 5 \text{ MOD } 13 + 1$   
 $d_i = i * H_2(\text{key})$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	5	87		72	22	40		25	94			12	47		66	33
	2	1		1	1	1		1	1			1	1		1	1

处理冲突5:  $d_1 = H_2(5) = 5 * 5 \text{ MOD } 13 + 1 = 13$

$H(5) = (5 + 13) \text{ MOD } 17 = 1$





## 排序相关练习题








- ◆ 假设一组数据存在于**单链表**中，请编写一个算法用**直接插入排序**的思想实现对数据的排序。
- ◆ 假设一组数据存在于**单链表**中，请编写一个算法用**简单选择排序**的思想实现对数据的排序。





```
void selectsort(LinkList head){
    p=head->next;// p指向无序区的第一个记录
    while (p!=null){
        q = p; //初始化指针q，令其指向无序区的第一个记录
        r = p->next; //r指向无序区第二个记录
        while(r!=null){//在无序区中寻找最小的记录
            if (r->data < q->data) //判断r指向节点的值是否更小
                q = r; //q指向当前最小值节点
            r= r->next; //指针r后移
        }//end of while(r!=null)
        q->data $\leftrightarrow$ q->data; //交换p和q所指节点值
        p= p->next ; //指针p后移
    }//end of while (p!=null)
}// end of selectsort
```



◆ 请写出快速排序的非递归算法。

```
void Qsort(SqList &L, int low, int high)
{ //对顺序表L中的子序列L.r[low.. high]作快速排序
  if (low < high)
  {
    pivotloc = Partition(L, low, high);
    Qsort(L, low, pivotloc-1);
    Qsort(L, pivotloc+1, high);
  }
}
```



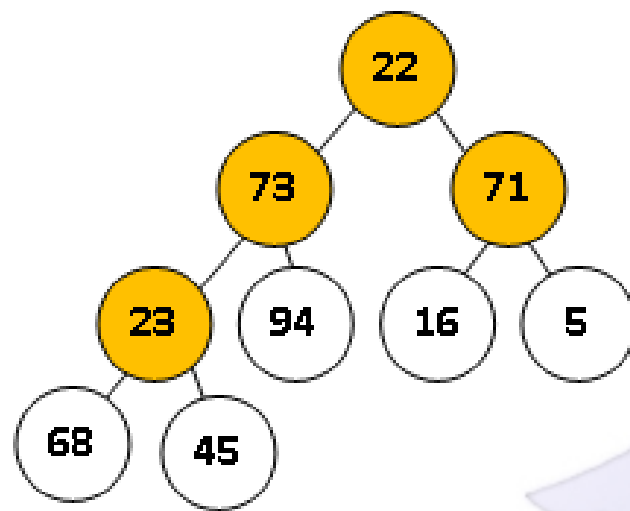


- ◆ 若在 $10^8$ 个记录中找最小的两个记录，采取哪种排序算法所需用的关键字比较次数最少？
- ◆ 用该算法需用比较多少次？
- ◆ 答：简单选择排序中找最小和次小的值。



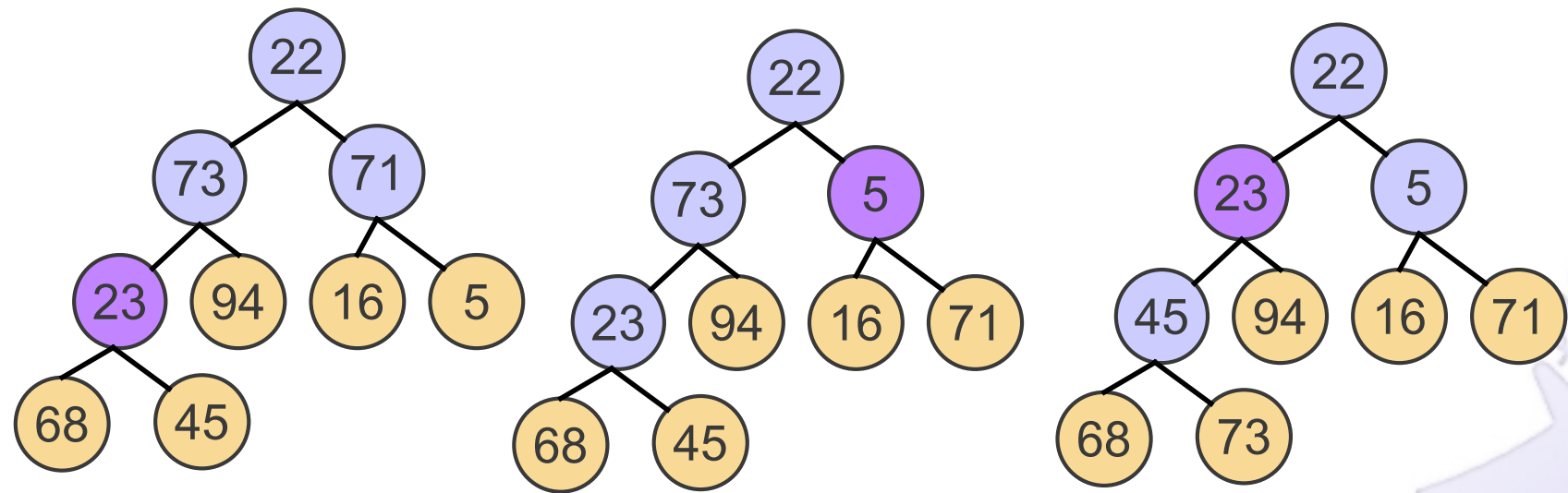


- ◆ 请回答下列关于堆的问题：
- ◆ （1）对于序列：22，73，71，23，94，16，05，68，45，建立小顶堆，需要进行多少次比较。
- ◆ （2）对于一个小堆，其具有最大值的元素可能在什么位置？
- ◆ （3）请写出一个算法，检查一个给定的顺序表L是否是一个堆





- ◆ (1) 共9个关键字，所以建立小顶堆需要对4棵子树做调整，
- ◆ 调整子树23需2次比较，调整子树71需2次比较，
- ◆ 调整子树73需4次比较，调整子树22需4次比较，
- ◆ 所以共需要12次比较。
- ◆ (2) 对于一个小堆，其具有最大值的元素会在叶子节点上。





- ◆ (3) 先序遍历该完全二叉树的非叶子节点
- ◆ 假设遍历的当前节点为 $i$ ,
- ◆ 判断 $L[i]$ 的值是否大于其叶子节点 $L[2i]$ 和 $L[2i+1]$ , 如果不满足则返回**false**。
- ◆ 如果所有非叶子节点都满足上述条件, 则 $L$ 是一个大顶堆。





- ◆ 荷兰国旗问题：已知一个含有 $n$ 个记录的序列，其关键字的取值为{red, white, blue}。请给出一个时间复杂度为 $O(n)$ 的算法，将这个序列按{red- white- blue}的顺序排好。
- ◆ 答：
  - || 快排
  - || 基数排序
  - || 计数排序
  - || 桶排序







- 指针**R**指向白色区域第一个元素，即下一个红色记录放置位置；
- 指针**W**指向未处理区域第一个元素，即下一个白色记录放置位置；
- 指针**B**指向未处理区域最后一个元素，即下一个蓝色记录放置位置；

根据**A[W]**的颜色将未处理的记录逐渐分类到红、白、蓝三个子序列中。

- (1) 若**A[W]**是白色的，则**W++**；
  - (2) 若**A[W]**是红色的，则**A[W]**与**A[R]**进行交换，然后**R++**，**W++**；
  - (3) 若**A[W]**是蓝色的，则**A[W]**与 **A[B]**进行交换，然后**B--**；
- 初始时，所有记录都未处理，所以设**R=W=0**； **B=n-1**。

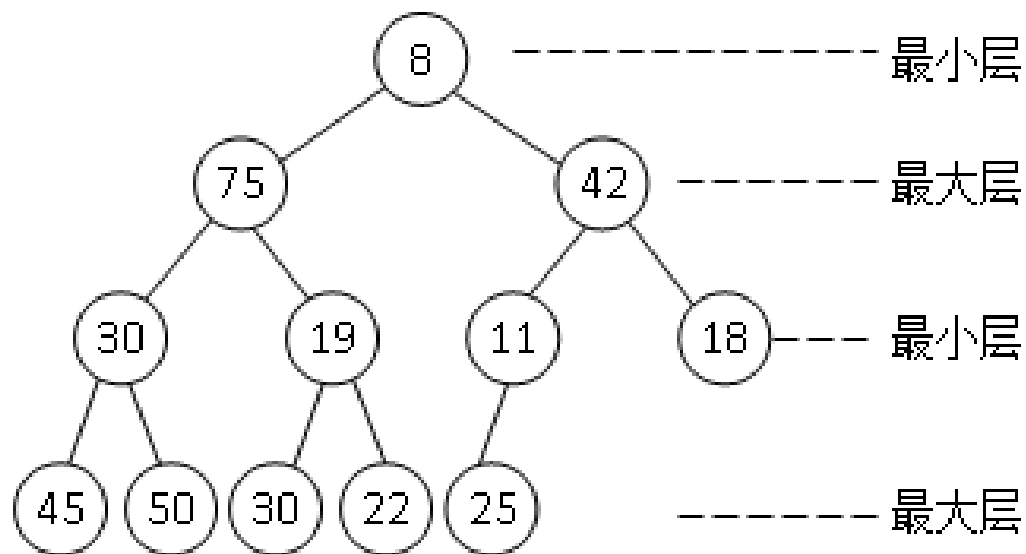


- ◆ 一个最小最大堆（Min-Max Heap）是一种特定的堆，其最小层和最大层交替出现，根节点总是处于最小。它具有以下性质：
- ◆ 对于堆中处于最大（小）层的任一节点，其关键字的值总是在以之为根的子树中的最大（小）关键字；
- ◆ 下图是一个最小最大堆的例子，其存储结构为{8, 75, 42, 30, 19, 11, 18, 45, 50, 30, 22, 25}。
- ◆ （1）请画出在上图堆后面依次插入关键字为5和80的节点后的最小最大堆。
- ◆ （2）请说明插入操作的基本过程和插入操作的时间复杂度。

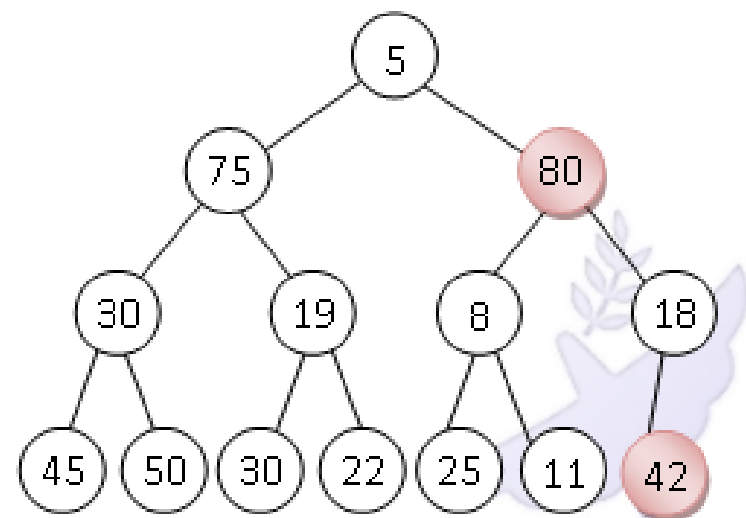
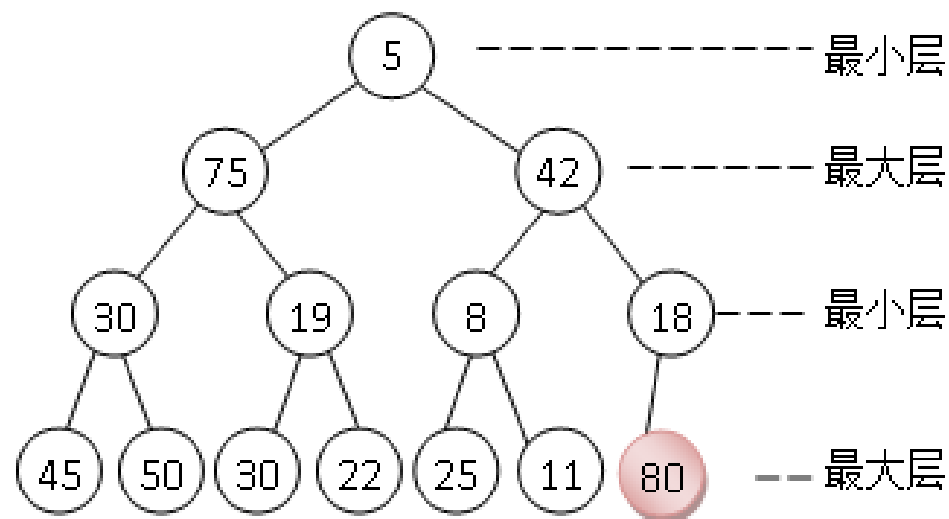
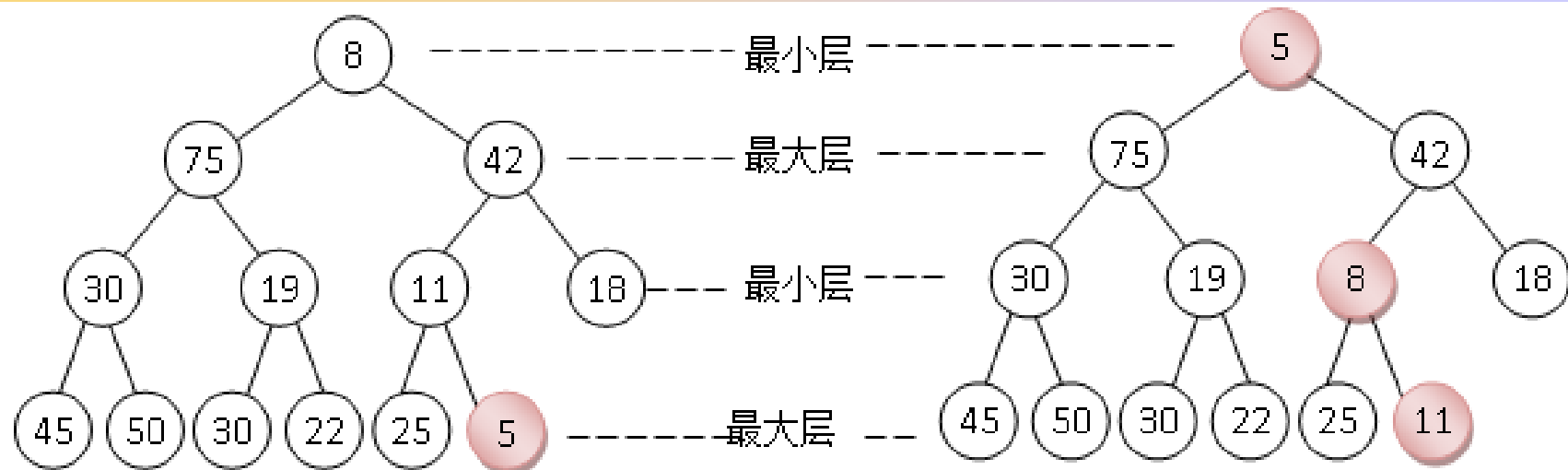




{8, 75, 42, 30, 19, 11, 18, 45, 50, 30, 22, 25}



## 依次插入关键字5和80





- ◆ (2) 基本思想：从插入的位置开始，从下向上依次进行调整。
- ◆ 1、根据元素的位置求出其所在的层数，若为奇数层则在最小层，若为偶数则在最大层。
- ◆ 2、若插入元素在最大层，则先比较其关键字是否比其双亲小，如果是则交换。
- ◆ 然后将已经形成的小堆与其祖先进行同样的调整，直到满足小堆定义或者到达根节点；若插入元素不小于其双亲，则调整大堆，直到满足大堆定义。
- ◆ 若插入元素在最小层，则反之。
- ◆ 复杂度  $O(\log n)$





# 外部排序

- ◆ 对于输入文件

{**101,51,19,61,3,71**,31,17,19,100,55,20,9,30,50,6,90}, 当 $k=6$ 时, 适用置换-选择排序的方法, 写出建立的初始败者树。

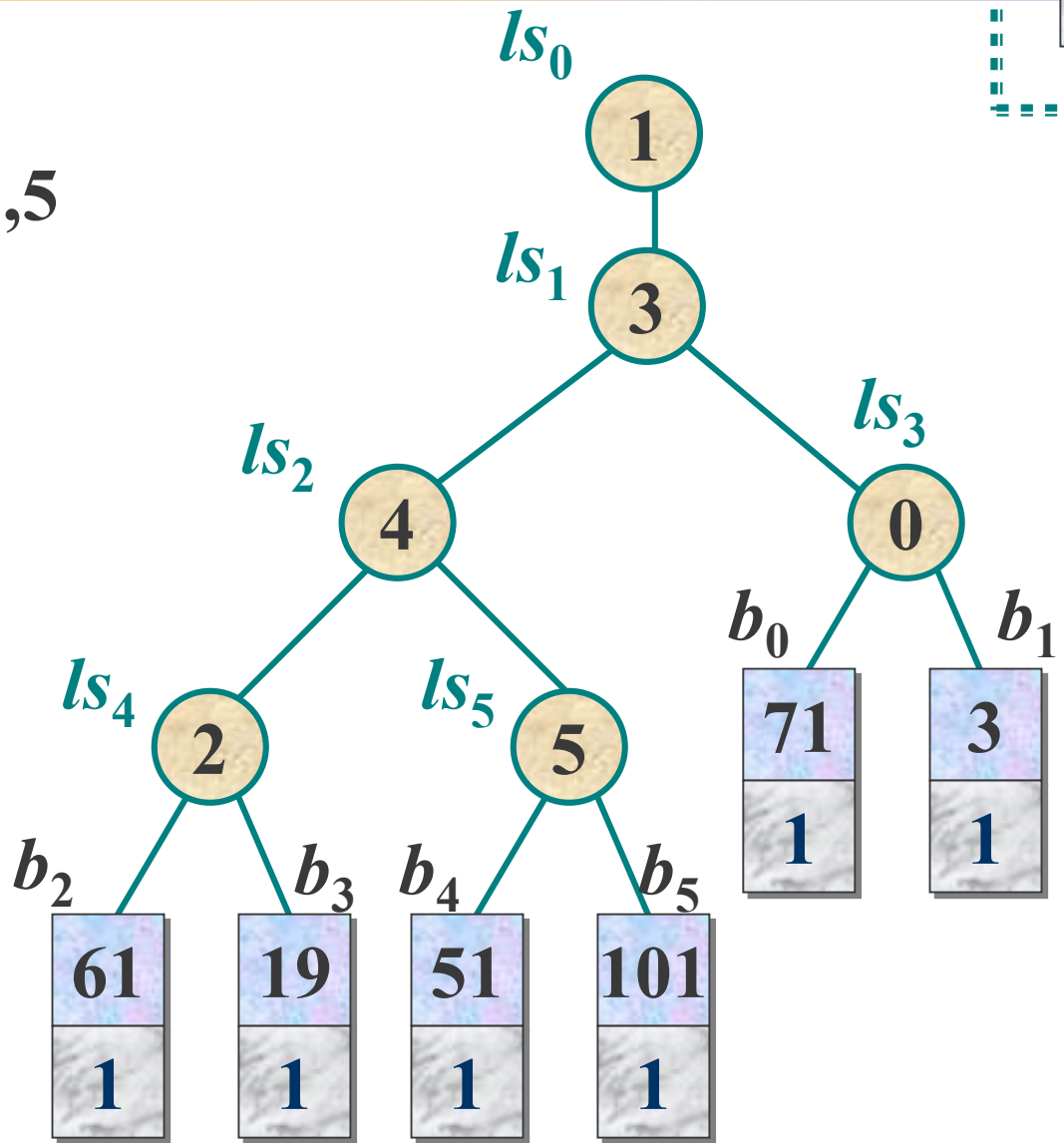




101,51,19,61,3,71

0	关键码
1	段号

1,3,4,0,2,5





- ◆ 设有11个长度不同的初始归并段，他们所包含的记录个数分别是25,40,16,38,77,64,53,88,9,48,98。试对它们做四路平衡归并，要求：
- ◆ (1) 构造最佳归并树。
- ◆ (2) 根据最佳归并树计算记录读写的总次数。
- ◆ (1)  $m = 11, k = 4, (11-1) \% (4-1) = 1$
- ◆ 需补充 $(k-1)-1=2$ 个空段。
- ◆ 0, 0, 9, 16, 25, 38, 40, 48, 53, 64, 77, 88, 98

$$u = \begin{cases} 0, & \text{if } (m-1)\%(k-1) = 0 \\ k-1 - (m-1)\%(k-1), & \text{else} \end{cases}$$







读写的总次数:  $876 * 2 = 1752$





**END**

