Ch1 绪论

本章知识点：掌握算法时间复杂度的求解方法。算法的特点。数据结构的定义与分类（线性数据结构与非线性数据结构）。

•1.例如： 某一个语句的频度计算表达式为

$f(n)=10n^3+5n^2+18n$,该表达式中，关于变量n的数量级的最大值是n^3,所以很快得出该算法的时间复杂度为O（ n^3 ）

2. 算法的特点：有穷性、确定性、可行性、输入和输出.

Ch2 线性表

1、线性表的存储结构：

- **顺序存储**

  要求掌握顺序存储的线性表在进行表中元素插入和删除时候元素的移动特点及元素需要移动的个数

总结：向一个长度为n的线性表中的第i个元素(1≤i≤n+1)之前插入一个元素时，需向后移动 n-i+1 个元素；

向一个长度为n的向量中删除第i个元素(1≤i≤n)时，需向前移动 n-i 个元素。

Ch2 线性表——链式存储

本章知识点：掌握单链表创建、查找、插入、删除操作的算法描述及应用。单循环链表、双向链表的定义及操作特点。

- 例：在一个单链表中若p所指的结点不是最后结点，在p之后，插入s所指结点，则执行的语句为____B____。

A **s->next=p; p-> next =s**

B **s-> next =p-> next; p-> next =s;**

C **s-> next =p-> next; p=s;**

D **p-> next =s; s-> next =p**

Ch2 线性表——链式存储

- 例1：在n个结点的带头结点的单链表中，要在已知结点*p之前插入一个新结点，则其操作的时间复杂度为**B**

  A O（1） B O(n) C O (n+1) D O(n^2)

例2：非空的循环单链表head的尾结点p满足条件__**A**__。

A p->next==head
B p->next==head->next
C p=head
D p=head->next

链式存储与顺序存储的优缺点 .

答：链式存储结构时，要求内存中可用存储单元的地址连续或不连续都可以；但顺序存储的内存地址单元一定是连续的。

链式存储与顺序存储的优缺点：链式存储有利于插入删除运算，只需修改指针而不需要移动大量元素，但对于查找元素和求表长元素不如顺序存储方便；顺序存储有利于查找元素和求表长运算但做插入删除运算需要移动大量元素不方便。

简述线性数据结构与非线性数据结构的区别

答： 线性数据结构的数据元素之间的关系是唯一的，是一对一的关系，每个结点最多有一个直接前驱和直接后继；非线性数据结构的数据元素之间的关系有多种，每个结点的直接前驱和直接后继不唯一。

描述以下三个概念的区别：头指针、头结点、首元结点（第一个元素结点）。在单链表中设置头结点的作用是什么？

答：

头指针是指向链表中第一个结点（或为头结点或为首元结点）的指针；

头结点是在链表的首元结点之前附设的一个结点；数据域内只放空表标志和表长等信息.

首元结点是指链表中存储线性表中第一个数据元素a_1的结点 。

若链表中附设头结点，则不管线性表是否为空表，头指针均不为空。否则表示空表的链表的头指针为空。

习题：假设某个单向循环链表的长度大于1，且表中既无头结点也无头指针。已知s为指向链表中某个结点的指针，试编写算法在链表中删除指针s所指结点的前驱结点。

```
void Delete_Pre(CiLNode *s)
//删除单循环链表中结点s的直接前驱
{CiLNode *r;
  p=s;
//找到s的直接前驱的前驱，并用p返回；
  while(p->next->next!=s) p=p->next;
    r= p->next;
    p->next=s;
  free(r);
 }//Delete_Pre
```

对给定的单链表L，编写一个删除L中值为x的结点的直接前驱结点的算法。

```
Void DeleteX(LinkList L , DataType x)
 { // L是带头结点的单链表
  ListNode *p , *q;
  q=L ; p=L-next;  //p 指向开始结点
  while (p && p->data!=x)
    { q=p;
      p=p->next;}
   if (p= =null)
      Error(“x not in L”);
   if (q = =L )
     Error(“x is the first node ,it has no prior node”);
   q->data=x;
   q->next=p->next;
   free(p);
  }
```

算法设计题：

练习1：设A、B是两个单链表，其表中元素递增有序，写一算法将A、B归并成一个有序的单链表C，C也是递增有序的.

数据结构定义如下：

```
typedef struct node
        {int data;
          struct node *next;
        } Listnode;
typedef Listnode *Linklist;
Listnode  *p;
Linklist  A,B,C;
```

合并的思想:C=A+B

```
void  merge(Linklist A , Linklist B, Linklist C )
 {Listnode *p,*q,*r;
   C=A;
   p=A->next; q=B->next;  r=C;
   while(p!=null  && q!=null)   //尾插法创建C链表
       if(p->data<=q->data)
              {r->next= p;  r=p;  p=p->next;}
       else  {r->next= q;  r=q;  q=q->next;}
   while(p!=null )
      {r->next= p;  r=p;  p=p->next;}
   while (q!=null)
      {r->next= q;  r=q;  q=q->next;}
    free(B);
 }
```

习题：设A、B是两个单链表，其表中元素递增有序，写一算法将A、B归并成一个按照元素值递减的有序单链表C.要求辅助空间为O(1).并分析算法的时间复杂度。

```
void reverse_merge(LinkList A,LinkList B)
{ //都带头结点，返回合并后的链表C的头结点；
  Linklist C;    Lnode  *p,*q,*temp;
  p=A->next;   q=B->next;   C=A;   C->next=NULL;
 while(p&& q)
  {
    if (p>data<q->data)
      { //将A的元素插入新表
       temp=p->next; p->next=C->next; C->next=p;p=temp;       }
    else
       { //将B的元素插入新表
          temp=q->next; q->next= C->next; C->next =q; q=temp;
        }
   if(p)    while p {  temp=p->next; p->next=C->next; C->next=p;p=temp; }
     else  while q {temp=q->next; q->next= C->next; C->next =q; q=temp; }
    return C;
   }
}
```

如果A表长度为m,B表长度为n,算法的时间复杂度为O（m+n）。

习题： 编写实现将一个单链表就地逆置的算法。

数据结构定义如下：

```
typedef struct node
        {int data;
          struct node *next;
        } Listnode;
typedef Listnode *Linklist;
Listnode  *p;
Linklist  Head;
```

```
void reverse(Linklist Head)   //链表的就地逆置
 { Lnode  *pnow,*ptemp;
   pnow=Head->next;
   Head->next=null;
   while(pnow!=null)
       { ptemp=pnow->next;
         pnow->next=Head->next;
         Head->next=pnow;
         pnow=ptemp;
       }
 } //reverse
```

备注：掌握习题2.2， 2.3，2.8，2.9，2.12.

Ch3　栈和队列

1. 简述栈和队列的共同点和不同点 .

答：栈是一种先进后出的操作受限制的线性表，队列是一种先进先出的线性表。共同点：都是一种线性的数据结构，是操作受限制的线性表。栈通过栈顶指针进行元素的插入和删除操作；队列是通过队列头指针和队列尾指针进行插入和删除操作。

2.在一般的顺序队列中，什么是假溢出？怎样解决假溢出问题？

答：由于一般顺序存储队列的若干入队和出队操作，队头指针和队尾指针都后移使得想入队的元素因为没有空间而不能入队，而事实上队头中还有很多空间空闲，因此称为假溢出。解决办法是引入循环队列。

掌握栈的进栈、出栈、栈为空/满等操作的特点。读程序段写其功能。

例1：设进栈序列为a，b，c，则通过出栈和进栈操作可能得到的a，b，c的不同的排列序列是什么？共有多少个这样的排列序列？

解答：共有5个这样的序列，分别为a,b,c；b,a,c；b,c,a；a,c,b；c,b,a。不可能有的输出序列为c,a,b。因为不可能在c已经出栈了，而随后a在b的前面出栈。

例2：若一个栈的输入序列是1，2，3，…..，n，输出序列的第一个元素是n，则第i个输出元素是 n-i+1.

循环队列

（1）问题提出的原因：

因为一般的顺序队列存在“假溢出”现象，存储空间没有有效利用，特提出循环队列。

（2）循环队列的特点：

（a）表示方法：通常是少用一个元素空间来约定队列的头指针在队列尾指针的下一位置作为“满”的状态。

（b）循环队列“满”和“空”的具体表示：

一个循环队列的头指针为front，尾指针为rear。则判断队列满的条件是 front=(rear+1) % (整除) n ；判断队列为空的条件是 rear=front

（c）循环队列的入队和出队操作；

备注：掌握习题3.1-3.7.

例1：读程序写结果

```
Void  Demo(SeqStack *S, int m)
{ SeqStack T; int i;
   InitStack(&T);
   While(!StackEmpty(S))
     { If ((i=Pop(S))!=m   Push(&T,i);)}
   While (!StackEmpty(&T))
     { i=Pop(&T);  Push(S,i) }
  }
```

该程序段的功能：利用栈T做辅助，过滤栈S中值为m的元素。

例2：读程序写结果

```
void algo3(Queue &Q)
{
  Stack S; int d;
  InitStack (S);
  While (!QueueEmpty (Q))
    { DeQueue (Q,d) ;   Push (S,d);}
  While (!StackEmpty (S))
    { Pop(S,d);  EnQueue (Q,d); }
}
```

该程序段的功能是：利用栈做辅助，将队列中的数据元素进行逆置。

Ch 4 串

- 1、串的定义：零个或者多个字符组成的有限长的序列。
- 串的相等：两个串的长度相等并且对应的字符相等。

- 2、串的运算

（1）求子串Substring(Sub,s,pos,len)

例如：设串s1='ABCDEFG'，subs(s, i, j)返回串s的从序号i开始的j个字符组成的子串，

subs(s1, 2, 5) 的结果串是：“BCDEF”。

(2)串/字符的定位；（3）串的比较

掌握：习题4.1，串的比较，定位算法的应用.

Ch5 数组和广义表

- 1、掌握二维数组的存储方法：

按行存储；按列存储。注意数组的第一个元素的行列下标是否从0还是。

2、特殊矩阵（对称矩阵）的压缩存储，稀疏矩阵的三元组表的存储方式（行、列和值）；

3、掌握广义表的取表头GetHead()和表尾GetTail()运算

备注：掌握习题5.1，5.11.

Ch5 数组和广义表---练习题

1. 二维数组M的成员是6个字符（每个字符占一个存储单元）组成的串，行下标i的范围从0到8，列下标j的范围从1到10，则存放M至少需要（ 1 ）个存储单元；M的第8列和第5行共占（ 2 ）个存储单元；若M按行优先方式存储，元素M[8][5]的起始地址与当M按列优先方式存储时的（ 3 ）元素的起始地址一致。

（1） A.90 B.180 C.240 D.540

（2） A.108 B.114 C.54 D.60

（3） A.M[8][5] B.M[3][10] C.M[5][8] D.M[0][9]

Ch6 树和二叉树

（1）简答：

1.一棵度为2的树与一棵二叉树有何区别？

答：度为2的树从形式上看与二叉树很相似，但它的子树是无序的，而二叉树是有序的。即，在一般树中若某结点只有一个孩子，就无需区分其左右次序，而在二叉树中即使是一个孩子也有左右之分。

2.简述二叉树的特点。

3.用二叉链表法（lchild-rchild）存储包含n个结点的二叉树，结点的2n个指针区域中有n+1个为空指针，n-1个为非空指针。

Ch6 树和二叉树

（2）掌握二叉树的5个基本性质

- 例1：具有35个结点的完全二叉树的深度为___6___。（性质4的应用）

（用$\lfloor \log_2 n \rfloor+1=\lfloor 5.xx \rfloor+1=6$ ）

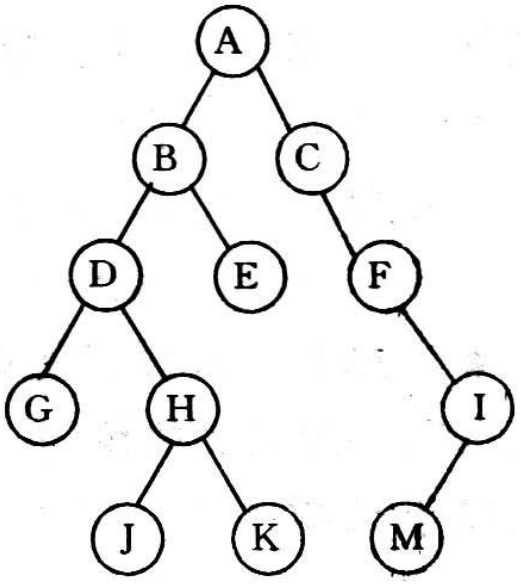
同理，一棵具有257个结点的完全二叉树，它的深度为___9___。

例2：已知二叉树有50个叶子结点，则二叉树的总结点数至少是___99___。

（注意：由$n=n_0+n_1+n_2$,而$n_0=n_2+1$,所以结点最少的情况就是缺少n_1结点，即$n=n_0+n_2=2n_0-1=99$）

(3) 二叉树的遍历

- (a)给定一棵二叉树，会写出其前序、中序、后序序列。
- (b)给定先序+中序/中序+后序序列能还原一棵二叉树；
- (c)树、森林与二叉树的相互转换;



（4）哈夫曼树

- **1.** 用哈夫曼算法创建哈夫曼树。

  n个叶子结点创建的哈夫曼树共有**2n-1**个结点

- **2.** 写哈夫曼编码和计算**WPL**。

例：假设字符a、b、c、d、e、f在电文中出现的概率分别为0.09，0.07，0.12,0.22，0.23，0.27,求出这些字符的哈夫曼编码

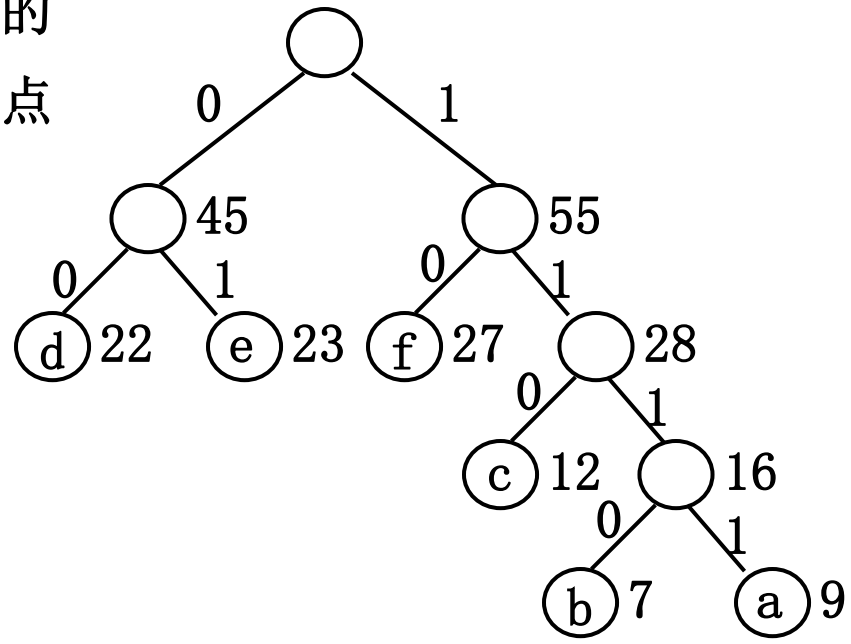
第一步：构造哈夫曼树(将概率视为a,b,c,d,e,f的权)

按左子树根结点的权小于等于右子树根结点的次序构造；

W_a=9，W_b=7,W_c=12,W_d=22，W_e=23，W_f=27,

第二步对路径编码：对一切非叶结点，其左分支标以0,右分支标以1；

第三步字母编码：从根到叶将各路径上标记的0和1收集起来就得到叶结点对应字符的具体编码；



a: 1 1 1 1

b: 1 1 1 0

c: 1 1 0

d: 0 0

e: 0 1

f: 1 0

该编码方案中的任意一个编码均不是另一个编码的前缀

Ch6 树和二叉树 --练习题

1、已知二叉树有50个叶子结点，则二叉树的总结点数至少是多少？

求解过程：由n0=n2+1,可得n_2=49。由N=n0+n1+n2,
N_{min} =n0+n1+n2，当n1=0时， N_{min} =99。

2、已知二叉树有50个叶子结点，且仅有一个孩子的结点数为30，则总结点数为多少？

求解过程：由n0=n2+1,可得n_2=49。由N=n0+n1+n2,
N =n0+n1+n2，当n1=30时， 则N =50+49+30=129。

3、已知完全二叉树的第7层有10个叶子结点，则整个二叉树的结点数最多是多少？

求解过程：因为第7层有10个叶子结点，则第7层还有非叶子结点，这些非叶子结点会产生第8层的结点。要使得该完全二叉树的总结点数最多，则第7层的非叶子结点所产生的第8层的结点个数为最多，即：第7层的每个非叶子结点会产生2个第8层的叶子结点。则总的结点个数==前7层的结点个数 + 第8层的叶子结点个数。

第7层的非叶子结点个数为：$2^{(7-1)}-10=54$，则产生的第8层的最多的叶子结点个数为$54\times2=108$；

前面7层总共的结点个数为：$2^7-1=127$；

所以，整个二叉树的结点数最多为N=127+108=235

（5）掌握如何编写一个递归算法

例：编写一个递归算法求二叉树的叶子数。

```
int num=0; /存放叶子数的全局变量
Void cout_leaf(bittree T)  / *T为树根指针
{ if(T!=null)
   { if (T->lchild==null  && T->rchild==null)
          num++;  /叶子计数加1;
     count_leaf(T->lchild); /递归求左子树的叶子数
     count_leaf(T->rchild); /递归求右子树的叶子树
   }
}
```

掌握本章习题：6.1，6.4，6.5，6.9，6.11，6.14.

Ch7 图

- **1、图的定义和术语**

完全图：有n(n-1)/2条边；

有向完全图：有n(n-1)条边

例1：有8个结点的无向图最多有__28__条边。

出度/ 入度

图中边（弧）的关系：边（弧）为图中所有顶 点度之和的一半；

强连通图 /连通图

例2：有8个结点的无向连通图最少有__7__条边。

（注意：n个顶点的连通图至少有n-1条边；n个顶点的强连通图至少有n条边。）

2、图的存储结构

- （1）邻接矩阵表示
- （2）邻接表的表示

n个顶点e条边的图，若采用邻接矩阵存储，则时间复杂度为 $O(n^2)$ 。

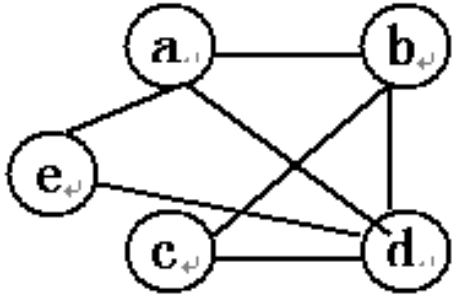
n个顶点e条边的图，若采用邻接表存储，则时间复杂度为 $O(n+e)$

例：已知如图所示的无向图，请给出该图的：

（1） 每个顶点的度；

（2） 邻接矩阵表示；

（3） 邻接表表示；



（4）写出从顶点a出发的深度优先搜索序列和广度优先搜索序列

解答如下：

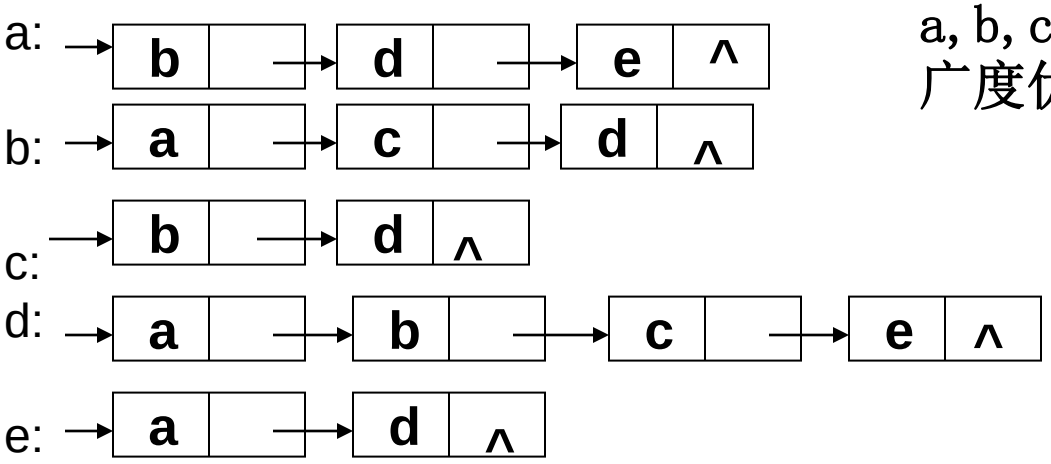
1. 结点　　度

a	3
b	3
c	2
d	4
e	2

2.邻接矩阵表示：

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{pmatrix} 0&1&0&1&1 \\ 1&0&1&1&0 \\ 0&1&0&1&0 \\ 1&1&1&0&1 \\ 1&0&0&1&0 \end{pmatrix}$$

3. 邻接表表示：



4.顶点a出发的深度优先搜索序列：
a, b, c, d, e;
广度优先搜索序列： **a,b,d,e,c**

Ch7 图

3、图的两种遍历方法及其特点。

（ 1）图的深度优先遍历

类似树的先序遍历。借用的辅助存储结构为栈；

（2）图的广度优先遍历

类似树的层次遍历。借用的辅助存储结构为队列；

4、运用Prim算法和Kruscal算法求解连通图的最小生成树及其时间复杂度分析。

比较两种算法时间复杂度

算法名	普里姆算法	克鲁斯卡尔算法
时间复杂度	$O(n^2)$	O(eloge)
适应范围	稠密图	稀疏图

5.拓扑排序的方法与特点

拓扑排序的算法思想：

（1）从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它。

（2）从图中删去该顶点，并且删去从该顶点发出的全部有向边。

（3）重复上述两步，直到剩余的图中不再存在没有前驱的顶点为止。

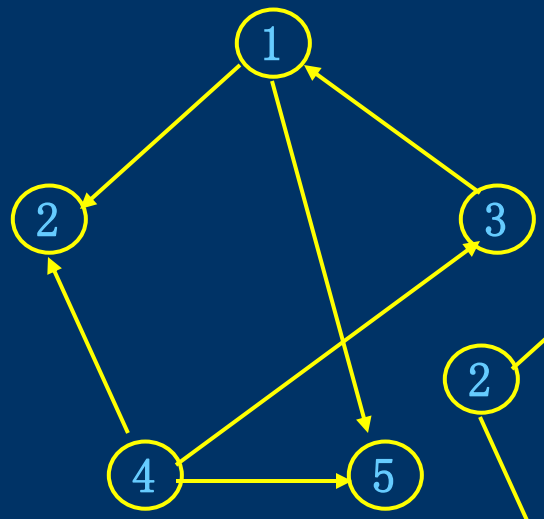
拓扑排序的特点：拓扑排序输出的顶点数小于图中的总顶点数，说明DAG图中存在环或者回路。

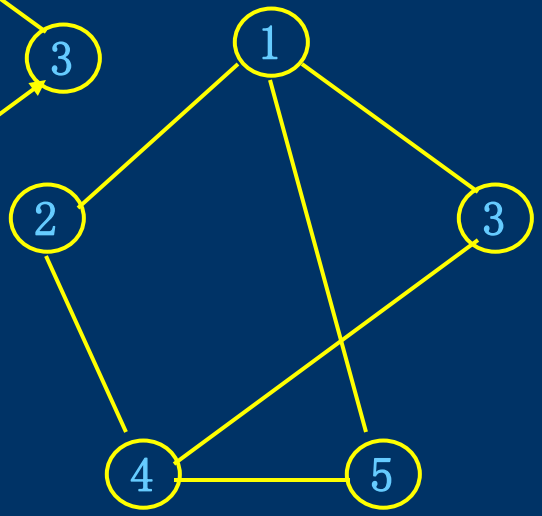
6. 最短路径

- 思想：按照路径长度递增的顺序产生最终的最短路径。
- 单源最短路径：Dijstrala算法的求解，时间复杂度为$O(n^2)$;
- 多源最短路径问题： Floyd算法的时间复杂度$O(n^3)$;

练习：

1. 求出下图各顶点的出度和入度；
2. 用邻接矩阵分别表示其存储结构；
3. 用邻接表分别表示其存储结构；





4. 已知以二维数组表示的图的邻接矩阵如下所示，自顶点1出发，试分别求出按深度优先搜索遍历得到的顶点序列和按广度优先搜索遍历得到的顶点序列。

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	1	0	1	0
2	0	0	1	0	0	0	1	0	0	0
3	0	0	0	1	0	0	0	1	0	0
4	0	0	0	0	1	0	0	0	1	0
5	0	0	0	0	0	1	0	0	0	1
6	1	1	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	1
8	1	0	0	1	0	0	0	0	1	0
9	0	0	0	0	1	0	1	0	0	1
10	1	0	0	0	0	1	0	0	0	0

7. 二叉排序树

二叉排序树（简称BST）又称二叉查找（搜索）树，其定义为：二叉排序树或者是空树，或者是满足如下性质（BST性质）的二叉树：

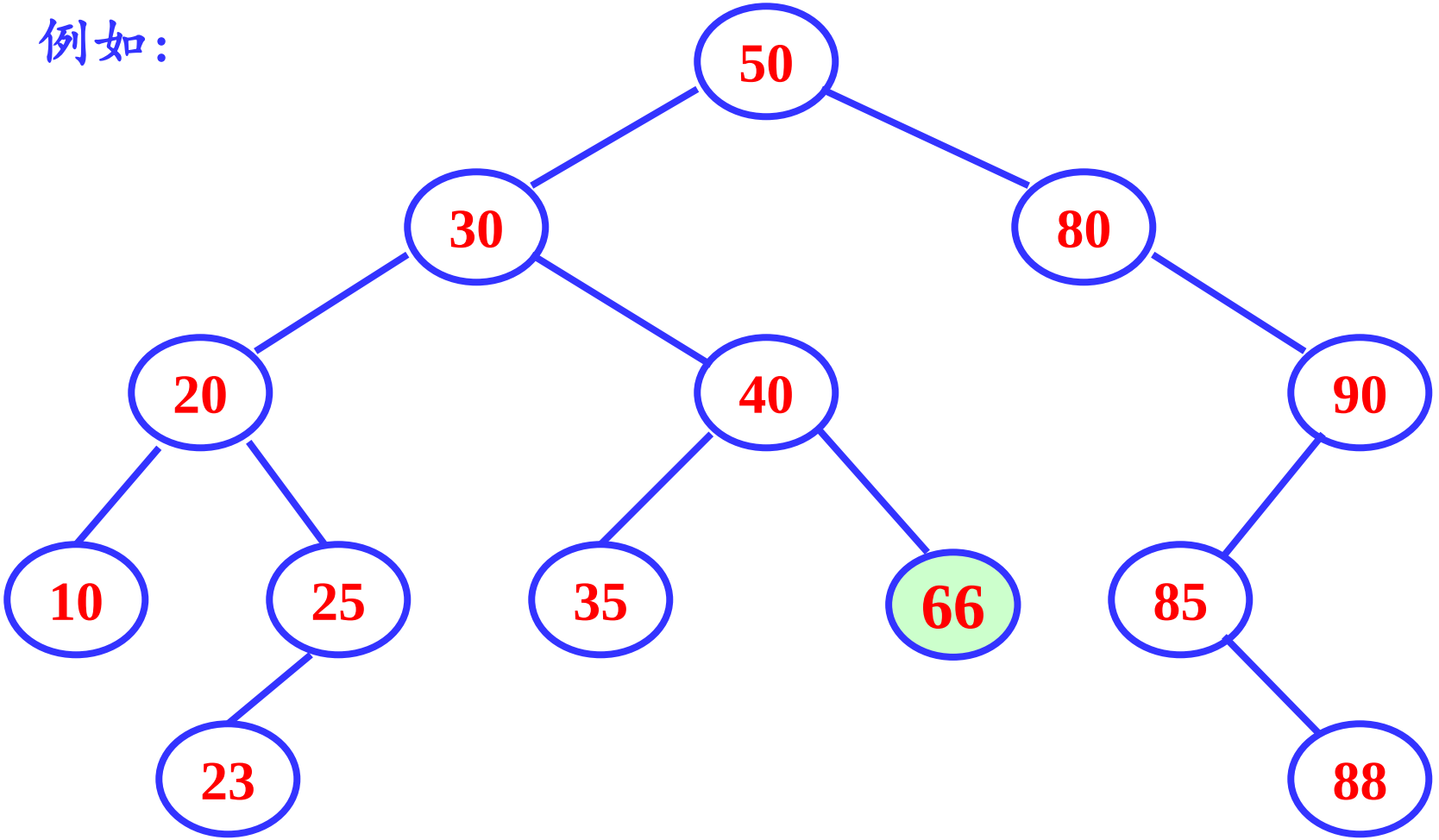
❶ 若它的左子树非空，则左子树上所有结点值（指关键字值）均小于根结点值；

❷ 若它的右子树非空，则右子树上所有结点值均大于根结点值；

❸ 左、右子树本身又各是一棵二叉排序树。

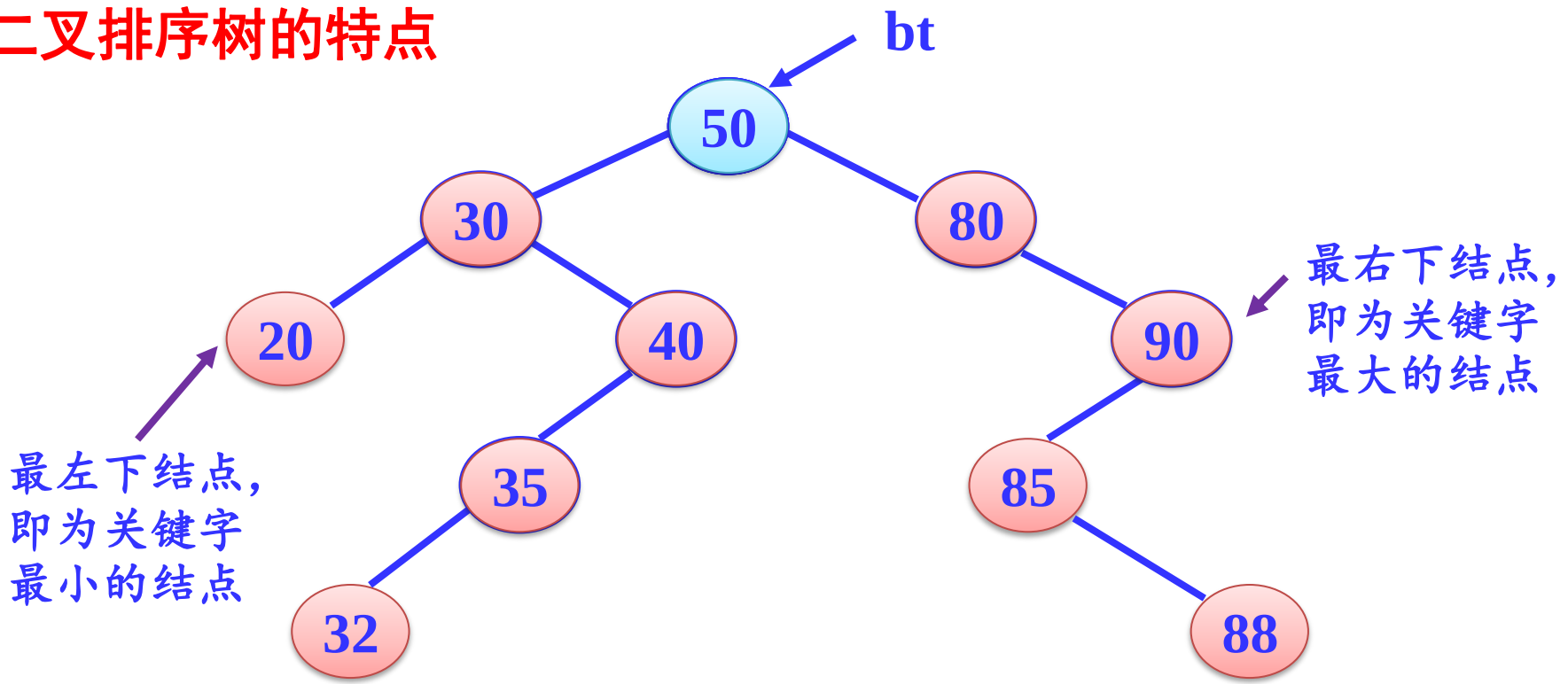
注意：二叉排序树中没有相同关键字的结点。

例如：



不是二叉排序树。

二叉排序树的特点



中序序列： 20，30，32，35，40，50，80，85，88，90

- 二叉排序树的中序序列是一个递增有序序列
- 根结点的最左下结点是关键字最小的结点
- 根结点的最右下结点是关键字最大的结点

递归查找算法SearchBST()如下（在二叉排序树bt上查找关键字为k的记录，成功时返回该结点指针，否则返回NULL）：

```
BSTNode *SearchBST(BSTNode *bt，KeyType k)
{
    if (bt==NULL || bt->key==k)          //递归出口
        return bt;
    if (k<bt->key)
        return SearchBST(bt->lchild，k);  //在左子树中递归查找
    else
        return SearchBST(bt->rchild，k);  //在右子树中递归查找
}
```