【考查目标】

1. 理解数据结构的基本概念；掌握数据的逻辑结构、存储结构及其差异，以及各种基本操作的实现。
2. 掌握基本的数据处理原理和方法的基础上，能够对算法进行设计与分析。
3. 能够选择合适的数据结构和方法进行问题求解。

一、线性表

大纲要求：

（一） 线性表的定义和基本操作
（二） 线性表的实现
1. 顺序存储结构
2. 链式存储结构
3. 线性表的应用

知识点：

1. 深刻理解数据结构的概念，掌握数据结构的“三要素”：逻辑结构、物理（存储）结构及在这种结构上所定义的操作“运算”。
2. 时间复杂度和空间复杂度的定义，常用计算语句频度来估算算法的时间复杂度。
以下六种计算算法时间的多项式是最常用的。其关系为：
$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$
指数时间的关系为：$O(2^n)<O(n!)<O(n^n)$
3. 线性表的逻辑结构，是指线性表的数据元素间存在着线性关系。主要是指：除第一及最后一个元素外，每个结点都只有一个前趋和只有一个后继。在顺序存储结构中，元素存储的先后位置反映出这种逻辑关系，而在链式存储结构中，是靠指针来反映这种逻辑关系的。
4. 顺序存储结构用向量（一维数组）表示，给定下标，可以存取相应元素，属于随机存取的存储结构。
5. 线性表的顺序存储方式及其在具体语言环境下的两种不同实现：表空间的静态分配和动态分配。掌握顺序表上实现插入、删除、定位等运算的算法。
6. 尽管“只要知道某结点的指针就可以存取该元素”，但因链表的存取都需要从头指针开始，顺链而行，故链表不属于随机存取结构。要理解头指针、头结点、首元结点和元素结点的差别。头结点是在插入、删除等操作时，为了算法的统一而设立的（若无头结点，则在第一元素前插入元素或删除第一元素时，链表的头指针总在变化）。对链表（不包括循环链表）的任何操作，均要从头结点开始，头结点的指针具有标记作用，故头指针往往被称为链表的名字，如链表head是指链表头结点的指针是head。理解循环链表中设置尾指针而不设置头指针的好处。链表操作中应注意不要使链意外“断开”。因此，若在某结点前插入一个元素或删除某元素，必须知道该元素的前驱结点的指针。
7. 链表是本部分学习的重点和难点。重点掌握以下几种常用链表的特点和运算：单链表、循环链表、双向链表、双向循环链表的生成、插入、删除、遍历以及链表的分解和归并等操作。并能够设计出实现线性表其它运算的算法。
8. 从时间复杂度和空间复杂度的角度综合比较线性表在顺序和链式两种存储结构下的特点，即其各自适用的场合。

小结：

顺序表和链表的比较

通过对它们的讨论可知它们各有优缺点，顺序存储有三个优点：
（1）方法简单，各种高级语言中都有数组，容易实现。
（2）不用为表示结点间的逻辑关系而增加额外的存储开销。
（3）顺序表具有按元素序号随机访问的特点。
但它也有两个缺点：
（1）在顺序表中做插入删除操作时，平均移动大约表中一半的元素，因此对n较大的顺序表效率低。
（2）需要预先分配足够大的存储空间，估计过大，可能会导致顺序表后部大量闲置；预先分配过小，又会造成溢出。

链表的优缺点恰好与顺序表相反。

在实际中怎样选取存储结构呢？

（1）基于存储的考虑
对线性表的长度或存储规模难以估计时，不宜采用顺序表；链表不用事先估计存储规模，但链表的存储密度较低，显然链式存储结构的存储密度是小于 1 的。

（2）基于运算的考虑
在顺序表中按序号访问ai的时间性能时O(1)，而链表中按序号访问的时间性能O(n)，所以如果经常做的运算是按序号访问数据元素，显然顺序表优于链表；而在顺序表中做插入、删除时平均移动表中一半的元素，当数据元素的信息量较大且表较长时，这一点是不应忽视的；在链表中作插入、删除，虽然也要找插入位置，但操作主要是比较操作，从这个角度考虑显然后者优于前者。

（3）基于环境的考虑
顺序表容易实现，任何高级语言中都有数组类型，链表的操作是基于指针的，相对来讲前者简单些，也是用户考虑的一个因素。

总之，两种存储结构各有长短，选择那一种由实际问题中的主要因素决定。通常“较稳定”的线性表选择顺序存储，而频繁做插入删除的即动态性较强的线性表宜选择链式存储。

练习题：

（一）选择题：

1．以下那一个术语与数据的存储结构无关？（ A ）
A．队列　　B．哈希表
C．线索树　　D． 双向链表

2、一个算法应该是（ B ）。
A．程序　　B．问题求解步骤的描述
C．要满足五个基本特性　　D．A 和 C.

3、数据结构中，与所使用的计算机无关的是数据的（ C ）
A．存储结构　　B．物理结构　　C．逻辑结构　　D．物理结构和存储结构

4．算法的计算量的大小称为计算的（ B ）。
A．效率　　B．复杂性　　C．现实性　　D．难度

5．下列说法，不正确的是（D）。
A．数据元素是数据的基本单位
B．数据项是数据中不可分割的最小可标识单位
C．数据可由若干个数据元素构成
D．数据项可由若干个数据元素构成

6．连续存储设计时，存储单元的地址（ A ）。
A．一定连续　　B．一定不连续
C．不一定连续　　D．部分连续，部分不连续

7． 线性表（ a1,a2,…,an）以链接方式存储时，访问第 i 位置元素的时间复杂性为（ C ）。
A．O（i）　　B．O（1）　　C．O（n）　　D．O（i-1）

8. 对于顺序存储的线性表，访问结点和增加、删除结点的时间复杂度为（ C ）。
   A. O(n) O(n)　　B. O(n) O(1)
   C. O(1) O(n)　　D. O(1) O(1)
9. 设单链表中结点的结构为（data，link)。已知指针 q 所指点是指针 p 所指结点的直接前驱，若在*q 与*p 之间插入结点*s，则应执行下列哪一个操作？（ B ）。
   A. s—>link=p—>link；p—>link=s
   B. q—>link=s；s—>link=p
   C. p—>link=s—>link；s—>link=p
   D. p—>link=s；s—>link=q
10. 在一个长度为 n 的顺序表的表尾插入一个新元素的渐进时间复杂度为（ B ）。
   A. O（n）　　B. O（1）
   C. O（n^2）　　D. O（$\log_2 n$）
11. 表长为 n 的顺序存储的线性表，当在任何位置上插入一个元素的概率相等时，插入一个元素所需移动元素的平均个数为（ B ）
   A. n　　B. n/2
   C. (n-1)/2　　D. (n+1)/2
12. 循环链表的主要优点是（ D ）
   A. 不再需要头指针了。
   B. 已知某个结点的位置后，能很容易找到它的直接前驱结点。
   C. 在进行删除操作后，能保证链表不断开。
   D. 从表中任一结点出发都能遍历整个链表。

（二）应用题

1、按增长率由小至大排列以下 7 个函数。

$(\frac{2}{3})^n$、 $(\frac{3}{2})^n$、2^{100}、$\log^{\log_2 n}_2$、$\log_2 n$、$(\log_2 n)^2$ 、n

答：$(\frac{2}{3})^n$、2^{100}、$\log^{\log_2 n}_2$、$\log_2 n$、$(\log_2 n)^2$ 、n、 $(\frac{3}{2})^n$

2、数据的存储结构由哪四种基本的存储方法实现，并做以简要说明？
答：四种表示方法
（1）顺序存储方式。数据元素顺序存放，每个存储结点只含一个元素。存储位置反映数据元素间的逻辑关系。存储密度大，但有些操作（如插入、删除）效率较差。
（2）链式存储方式。每个存储结点除包含数据元素信息外还包含一组（至少一个）指针。指针反映数据元素间的逻辑关系。这种方式不要求存储空间连续，便于动态操作（如插入、删除等），但存储空间开销大（用于指针），另外不能折半查找等。
（3）索引存储方式。除数据元素存储在一地址连续的内存空间外，尚需建立一个索引表，索引表中索引指示存储结点的存储位置（下标）或存储区间端点（下标），兼有静态和动态特性。
（4）散列存储方式。通过散列函数和解决冲突的方法，将关键字散列在连续的有限的地址空间内，并将散列函数的值解释成关键字所在元素的存储地址，这种存储方式称为散列存储。其特点是存取速度快，只能按关键字随机存取，不能顺序存取，也不能折半存取。

3. 线性表有两种存储结构：一是顺序表，二是链表。试问：
（1）如果有 n 个线性表同时并存，并且在处理过程中各表的长度会动态变化，线性表的总数也会自动地改变。在此情况下，应选用哪种存储结构？ 为什么？

（2）若线性表的总数基本稳定，且很少进行插入和删除，但要求以最快的速度存取线性表中的元素，那么应采用哪种存储结构？为什么？

答：

（1）选链式存储结构。它可动态申请内存空间，不受表长度（即表中元素个数）的影响，插入、删除时间复杂度为 0（1）。

（2）选顺序存储结构。顺序表可以随机存取，时间复杂度为 0（1）。

（三）算法设计题

1. 设计算法，求带表头的单循环链表的表长。

解：

```
int length(Linklist L)
{
  int I;
  listnode *p;
  I=0;
  P=L;
  while (p->next!=L){
     p=p->next;
     I++;
  }
  return I;
}
```

2. 已知单链表 L，写一算法，删除其重复结点。

算法思路：用指针 p 指向第一个数据结点，从它的后继结点开始到表的结束，找与其值相同的结点并删除之；p 指向下一个；依此类推，p 指向最后结点时算法结束。算法如下：

解：

```
void pur_LinkList(LinkList H)
 { LNode  *p,*q,*r;  p=H->next;  /*p 指向第一个结点*/
  if(p==NULL) return;
  while (p->next)
   {q=p;
     while (q->next)   /* 从*p 的后继开始找重复结点*/
       { if (q->next->data==p->data)
          { r=q->next;  /*找到重复结点，用 r 指向，删除*r */
            q->next=r->next;
            free(r);
          }  /*if*/
        else q=q->next;
        } /*while(q->next)*/
     p=p->next;       /*p 指向下一个，继续*/
   }  /*while(p->next)*/
}
```

该算法的时间性能为 0(n2)。

3. 已知指针 la 和 lb 分别指向两个无头结点的单链表中的首结点。请编写函数完成从表 la 中删除自第 i 个元素开始的共 len 个元素并将它们插入到表 lb 中第 j 个元素之前，若 lb 中只有 j-1 个元素，则插在表尾。函数原型如下：

```
int DeleteAndInsertSub(LinkList &la,LinkList &lb,int i,int j,int len);
```

答：

```
int DeleteAndInsertSub(LinkList &la,LinkList &lb,int i,int j,int len)
  {
    int k;
    LinkList p,q,prev,s;
    if(i<0||j<0||len<0)
        return -1;
    p=la;
    k=1;
    prev=NULL;
    while(p&&k<i)
    {
        prev=p;
        p=p->next;
        k++;
    }
    if(!p)
        return -1;
    q=p;k=1;
    while(q&&k<len)
    {
        q=q->next;
        k++;
    }
    if(!q)
        return -1;
    if(!prev)
        la=q->next;
    else
        prev->next=q->next;
    if(j==1)
    {
        q->next=lb;
        lb=q;
    }
    else
    {
        s=lb;k=1;
        while(s&&k<j-1)
        {
            s=s->next;
            k++;
        }
        if(!s)
            return -1;
        q->next=s->next;
        s->next=p;
```

```
        return 1;
    }
}
```

4．写一算法，将一带有头结点的单链表就地逆置，即要求逆置在原链表上进行，不允许重新构造新链表。

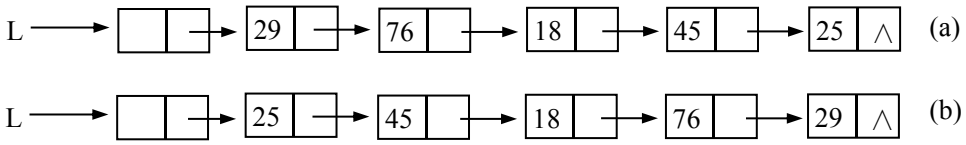


图　单链表的倒置

函数原型如下：

```
void LinkList_reverse(LinkList &L);
```

答：

```
void LinkList_reverse(LinkList &L)
{
    LinkList p,q,s;
    p=L->next;q=p->next;s=q->next;p->next=NULL;
    while(s->next)
    {
        q->next=p;p=q;
        q=s;s=s->next;
    }
    q->next=p;s->next=q;L->next=s;
}
```

5．写一算法，将带有头结点的非空单链表中数据域值最小的那个结点移到链表的最前面。要求：不得额外申请新的链结点。函数原型如下：

```
void delinsert(LinkList &L);
```

答：

```
void delinsert(LinkList &L)
{
    p=L->next;  //p 是链表的工作指针
    pre=L;  //pre 指向链表中数据域最小值结点的前驱
    q=p;        //q 指向数据域最小值结点，初始假定是第一结点
    while(p->next!=NULL)
    {
        if(p->next->data<q->data)    //找到新的最小值结点
        {   pre=p;  q=p->next; }
        p=p->next;
    }
    if(q!=L->next)  //若最小值是第一元素结点，则不需再操作
    {
        pre->next=q->next;  //将最小值结点从链表上摘下
        q->next=L->next;          //将 q 结点插到链表最前面
        L->next=q;
    }
}
```

6．编写一个算法来交换单链表中指针 P 所指结点与其后继结点，HEAD 是该链表的头指针，P 指向该链表中某一结点。

答：单链表中查找任何结点，都必须从头指针开始。本题要求将指针 p 所指结点与其后继结点交换，这不仅要求知道 p 结点，还应知道 p 的前驱结点。这样才能在 p 与其后继结点交换后，由原 p 结点的前驱来指向原 p 结点的后继结点。

```
LinkedList Exchange（LinkedList HEAD，p）
 //HEAD 是单链表头结点的指针，p 是链表中的一个结点。本算法将 p 所指结点与其后继结点交换。
{q=head->next;        //q 是工作指针，指向链表中当前待处理结点。
pre=head;             //pre 是前驱结点指针，指向 q 的前驱。
while（q!=null && q!=p）{pre=q；q=q->next；}     //未找到 p 结点，后移指针。
if（p->next==null）printf（“p 无后继结点\n”）；   //p 是链表中最后一个结点，无后继。
Else                                              //处理 p 和后继结点交换
   {q=p->next;         //暂存 p 的后继。
    pre->next=q;       //p 前驱结点的后继指向 p 的后继。
    p->next=q->next;  //p 的后继指向原 p 后继的后继。
    q->next=p       ；//原 p 后继的后继指针指向 p。
    }
}//算法结束。
```

7．已知线性链表第一个链结点指针为 list，请写一算法，将该链表分解为两个带有头结点的循环链表，并将两个循环链表的长度分别存放在各自头结点的数据域中。其中,线性表中序号为偶数的元素分解到第一个循环链表中,序号为奇数的元素分解到第二个循环链表中。

答：算法如下：

```
void split(ListNode *List, ListNode *&list1, ListNode *&list2)
{list1=(ListNode *)malloc(sizeof(ListNode ));
list2=(ListNode *)malloc(sizeof(ListNode ));
p=list;;
q=list1;
r=list2;
len1=0;
len2=0;
mark=1;
while (p!=null)
  {if(mark=1)
    {q->next=p;q=q->next;len1++;mark=2;}
  else
    {r->next=p;r=r->next;len2++;mark=1;}
  }
list1->data=len1;
list2->data=len2;
q->next=list1;
r->next=list2;
}
```

8．设 A 和 B 是两个单链表，其表中元素递增有序。

试写一算法将 A 和 B 归并成一个按元素值递减有序的单链表 C，并要求辅助空间为 O(1)。

答：Linklist merge(Linklist A,Linklist B)

```
{
Linklist C;
Listnode *p;
C=null;
while (A&&B)
  if(A->data<=B->data)
  {p=A->next;A->next=C;C=A;A=p;}
  else
  {p=B->next;B->next=C;C=B;B=p;}
  if (A)
    while(A) { p=A->next;A->next=C;C=A;A=p;}
  else
    while(B) {p=B->next;B->next=C;C=B;B=p;}
return C;
}
```

二、栈、队列和数组

大纲要求：

（一） 栈和队列的基本概念
（二） 栈和队列的顺序存储结构
（三） 栈和队列的链式存储结构
（四） 栈和队列的应用
（五） 特殊矩阵的压缩存储

知识点：

1. 栈、队列的定义及其相关数据结构的概念，包括：顺序栈、链栈、循环队列、链队列等。栈与队列存取数据（请注意包括：存和取两部分）的特点。
2. 掌握顺序栈和链栈上的进栈和退栈的算法，并弄清栈空和栈满的条件。注意因栈在一端操作，故通常链栈不设头结点。
3. 如何将中缀表达式转换成前缀、后缀表达式，了解对两种表达式求值的方法。
4. 栈与递归的关系。用递归解决的几类问题：问题的定义是递归的，数据结构是递归的，以及问题的解法是递归的。掌握典型问题的算法以及将递归算法转换为非递归算法，如n!阶乘问题，fib数列问题，hanoi问题。了解在数值表达式的求解、括号的配对等问题中应用栈的工作原理。
5. 掌握在链队列上实现入队和出队的算法。注意对仅剩一个元素的链队列删除元素时的处理（令队尾指针指向队头）。还需特别注意仅设尾指针的循环链队列的各种操作的实现。
6. 循环队列队空及队满的条件。队空定义为队头指针等于队尾指针，队满则可用牺牲一个单元或是设标记的方法，这里特别注意取模运算。掌握循环队列中入队与出队算法。
7. 在后续章节中多处有栈和队列的应用，如二叉树遍历的递归和非递归算法、图的深度优先遍历等都用到栈，而树的层次遍历、图的广度优先遍历等则用到队列。这些方面的应用应重点掌握。
8. 数组在机器（内存）级上采用顺序存储结构。掌握数组（主要是二维）在以行序为主和列序为主的存储中的地址计算方法。
9. 特殊矩阵（对称矩阵、对角矩阵、三角矩阵）在压缩存储是的下标变换公式。

练习题：

（一）选择题：

1. 一个栈的输入序列为 1 2 3 4，则（ D ）不可能是其出栈序列。
   A. 1 2 4 3　　B. 2 1 3 4　　C. 1 4 3 2　　D. 4 3 1 2
2. 一个递归算法必须包括（ B ）。
   A. 递归部分　　B. 终止条件和递归部分
   C. 迭代部分　　D. 终止条件和迭代部分
3. 一个递归的定义可以用递归过程求解，也可以用非递归过程求解，但单从运行时间来看，通常递归过程比非递归过程（B）。
   A. 较快　　B. 较慢
   C. 相同　　D. 以上答案都不对
4. 栈和队列都是（C）
   A. 顺序存储的线性表　　B. 链式存储的线性表
   C. 限制存储的线性表　　D. 限制存储的非线性结构
5. 二维数组 N 的元素是 4 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 4，列下标 j 的范围从 0 到 5，N 按行存储时元素 N[3][5]的起始地址与 N 按列存储时元素（ B ）的起始地址相同。
   A. N[2][4]　　B. N[3][4]
   C. N[3][5]　　D. N[4][4]
6. 设有数组 A[i,j]，数组的每个元素长度为 3 字节，i 的值为 1 到 8，j 的值为 1 到 10，数组从内存首地址 BA 开始顺序存放，当以列为主序存放时，元素 A[5,8]的存储首地址是（ B ）
   A. BA+141　　B. BA+180
   C. BA+222　　D. BA+225
7. 递归过程或函数调用时，处理参数及返回地址，要用一种称为（ C ）的数据结构。
   A. 队列　　B. 多维数组
   C. 栈　　D. 线性表
8. 对于单链表形式的队列，队空的条件是（ A ）
   A. F＝R＝nil　　B. F＝R
   C. F≠nil 且 R＝nil　　D. R－F＝1
9. 若循环队列以数组 Q[0..m-1]作为其存储结构，变量 rear 表示循环队列中的队尾元素的实际位置，其移动按 rear=(rear+1) Mod m 进行，变量 length 表示当前循环队列中的元素个数，则循环队列的队首元素的实际位置是（ C ）
   A. rear-length
   B. (rear-length+m) Mod m
   C. (1+rear+m-length) Mod m
   D. M-length

（二）应用题

1、（10 分）假设一个准对角矩阵

$$\begin{bmatrix} a_{11} & a_{12} & & & & & & & \\ a_{21} & a_{22} & & & & & & & \\ & & a_{33} & a_{34} & & & & & \\ & & a_{43} & a_{44} & & & & & \\ & & & & \cdots\cdots & & & & \\ & & & & & a_{i,j} & & & \\ & & & & & & \cdots\cdots & & \\ & & & & & & & a_{2m-1,2m-1} & a_{2m-1,2m} \\ & & & & & & & a_{2m,2m-1} & a_{2m,2m} \end{bmatrix}$$

按以下方式存储于一维数组 B[4m]中：

0	1	2	3	4		k		4m－2	4m－1
$a_{1,1}$	$a_{1,2}$	$a_{2,1}$	$a_{2,2}$	$a_{3,3}$	.....	$a_{i,j}$	.....	$a_{2m,2m-1}$	$a_{2m,2m}$

写出由一对下标（i，j）表示的 k 的转换公式。

答：

i 为奇数时 k=i+j-2

i 为偶数时 k=i+j-1

合并后可写成 k=i+j-(i%2)-1　或　k=2(i/2)+j-1

2、特殊矩阵和稀疏矩阵哪一种压缩存储后失去随机存取的功能？为什么？

答：

特殊矩阵指值相同的元素或零元素在矩阵中的分布有一定规律，因此可以对非零元素分配单元（对值相同元素只分配一个单元），将非零元素存储在向量中，元素的下标 i 和 j 和该元素在向量中的下标有一定规律，可以用简单公式表示，仍具有随机存取功能。

而稀疏矩阵是指非零元素和矩阵容量相比很小（t<<m*n），且分布没有规律。用十字链表作存储结构自然失去了随机存取的功能。即使用三元组表的顺序存储结构，存取下标为 i 和 j 的元素时，要扫描三元组表，下标不同的元素，存取时间也不同，最好情况下存取时间为 0(1)，最差情况下是 0(n)，因此也失去了随机存取的功能。

3、有人说，采用循环链表作为存储结构的队列就是循环队列，你认为这种说法对吗？说明你的理由。

答：

这种说法是错误的。队列（包括循环队列）是一个逻辑概念，而链表是一个存储概念，一个队列是否是循环队列。不取决于它将采用何种存储结构。根据实际的需要，循环队列可以采用顺序存储结构，也可以采用链式存储结构，包括采用循环链表作为存储结构。

4、指出下列程序段的功能是什么？

(1) void demo1(seqstack *s)

```
  {
     int I;arr[64];n=0;
     while (!stackempty(s)) arr[n++]=pop(s);
     for(I=0;<n;I++) push(s,arr[I]);
   }
```

(2) void demo2(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);
  }
}
```

（三）算法设计题

1. 试利用循环队列编写求 k 阶斐波那契序列中前 n+1 项（$f_0, f_1, \ldots\ldots f_n$）的算法，要求满足 $f_n \leq \max$ 且 $f_{n+1} > \max$，其中 max 为某个约定的常数。循环队列的容量为 k，因此，在算法执行结束时，留在循环队列中的元素应是所求 k 阶斐波那契序列中的最后 k 项 $f_{n-k+1}, \ldots\ldots f_n$。

答：

```
void GetFib(int k,int n)
{
    InitQueue(Q);
    for(i=0;k<k-1;i++)
        Q.base[i]=0;
    Q.base[k-1]=1;
    for(i=0;i<k;i++)
        printf(“%d”,Q.base[i]);
    for(i=k;i<=n;i++)
    {
        m=i%k;
        sum=0;
        for(j=0;j<k;j++)
            sum+=Q.base[(m+j)%k];
        Q.base[m]=sum;
        printf(“%d”,sum);
    }
}
```

2. 已知 num 为无符号十进制整数，请写一非递归算法，该算法输出 num 对应的 r 进制的各位数字。要求算法中用到的栈采用线性链表存储结构（1<r<10）。

解：

```
typedef struct node{
int data;
struct node *next;
}link;
void trans(int num,int r)
{ link *head=NULL,*s;
   int n;
  while (num>0)
    {
```

```
        n=num%r;
        s=(link *)malloc(sizeof(link));
        s->data=n;
        s->next=head;head=s;
        num=num/r;
      }
    printf(“输出 r 进制的各位数字：”);
    s=head;
    while (s!=NULL)
    {
        printf(“%d”,s->data);
        s=s->next;
    }
}
```

三、树与二叉树

大纲要求：

（一） 树的概念
（二） 二叉树
1. 二叉树的定义及其主要特征
2. 二叉树的顺序存储结构和链式存储结构
3. 二叉树的遍历
4. 线索二叉树的基本概念和构造
5. 二叉排序树
6. 平衡二叉树

（三） 树、森林
1. 树的存储结构
2. 森林与二叉树的转换
3. 树和森林的遍历
（四） 树的应用
1. 等价类问题
2. 哈夫曼（Huffman）树和哈夫曼编码

知识点：

1. 二叉树的概念、性质
（1）掌握树和二叉树的定义。
（2）理解二叉树与普通双分支树的区别。二叉树是一种特殊的树，这种特殊不仅仅在于其分支最多为2以及其它特征，一个最重要的特殊之处是在于：二叉树是有序的。即二叉树的左右孩子是不可交换的，如果交换了就成了另外一棵二叉树，这样交换之后的二叉树与原二叉树是不相同的两棵二叉树。但是，对于普通的双分支树而言，不具有这种性质。
（3）满二叉树和完全二叉树的概念。
（4）重点掌握二叉树的五个性质及证明方法，并把这种方法推广到K叉树。普通二叉树的五个性质：第i层的最多结点数，深度为k的二叉树的最多结点数，n0=n2+1的性质，n个结点的完全二叉树的深度，顺序存储二叉树时孩子结点与父结点之间的换算关系（序号i结点的左孩子为：2*i，

右孩子为：2*i+1）。

2. 掌握二叉树的顺序存储结构和二叉链表、三叉链表存储结构的各自优缺点及适用场合，以及二叉树的顺序存储结构和二叉链表存储结构的相互转换的算法。
3. 熟练掌握二叉树的先序，中序和后序遍历算法以及按层次遍历

   二叉树的先序、中序和后序三种遍历算法，划分的依据是视其每个算法中对根结点数据的访问顺序而定。不仅要熟练掌握这三种遍历的递归算法，理解其执行的实际步骤，并且应该熟练掌握三种遍历的非递归算法。

   ●按层次遍历二叉树

```
void LayerOrder(Bitree T)//层序遍历二叉树
{
    InitQueue(Q); //建立工作队列
    EnQueue(Q,T);
    while(!QueueEmpty(Q))
    {
    DeQueue(Q,p);
    visit(p);
    if(p->lchild) EnQueue(Q,p->lchild);
    if(p->rchild) EnQueue(Q,p->rchild);
    }
 }//LayerOrder
```

4. 遍历是基础，重点掌握在三种基本遍历算法的基础上实现二叉树的其它算法

   如求二叉树叶子结点总数，求二叉树结点总数，求度为1或度为2的结点总数，复制二叉树，建立二叉树，交换左右子树，查找值为n的某个指定结点，删除值为n的某个指定结点等等。
5. 线索二叉树的引出，是为避免如二叉树遍历时的递归求解。递归虽然形式上比较好理解，但是消耗了大量的内存资源，如果递归层次一多，势必带来资源耗尽的危险。二叉树线索化的实质是建立结点在相应序列中与其前驱和后继之间的直接联系。

   对于线索二叉树，应该掌握：线索化的实质，三种线索化的算法，线索化后二叉树的遍历算法，基本线索二叉树的其它算法问题（如：查找某一类线索二叉树中指定结点的前驱或后继结点）。
6. 二叉排序树的中序遍历结果是一个递增的有序序列。二叉排序树的形态取决于元素的输入顺序，二叉排序树在最差情况下形成单支树。熟练掌握其建立、查找、插入和删除算法，以及判断某棵二叉树是否二叉排序树这一问题的递归与非递归算法。
7. 平衡二叉树是二叉排序树的优化，平衡二叉树对左右子树的深度有了限定：深度之差的绝对值不得大于1。掌握平衡二叉树的四种调整算法。
8. 树与森林的遍历，只有两种遍历算法：先根与后根（对于森林而言称作：先序与中序遍历）。二者的先根与后根遍历与二叉树中的遍历算法是有对应关系的：先根遍历对应二叉树的先序遍历，而后根遍历对应二叉树的中序遍历。二叉树使用二叉链表分别存放它的左右孩子，树利用二叉链表存储孩子及兄弟（称孩子兄弟链表），而森林也是利用二叉链表存储孩子及兄弟。掌握树、森林和二叉树间的相互转换。
9. 简单掌握等价类的生成算法。
10. 哈夫曼树为了解决特定问题引出的特殊二叉树结构，它的前提是给二叉树的每条边赋予了权值，这样形成的二叉树按权相加之和是最小的，一般来说，哈夫曼树的形态不是唯一的。理解哈夫曼编码的基本原理，掌握基于哈夫曼树生成哈夫曼编码的方法。

练习题：

（一）选择题：

1. 一棵二叉树的前序遍历结果为 ABCDEF，中序遍历结果为 CBAEDF，则后序遍历结果为（ A ）

   A. CBEFDA　　B. FEDCBA

C. CBEDFA　　D. 不确定

2. 某二叉树的后序遍历序列为 dabec，中序遍历序列为 debac，则前序遍历序列为（D）。
   A. acbed　　B. decab
   C. deabc　　D. cedba
3. 具有 10 个叶子结点的二叉树中有（ B）个度为 2 的结点。
   A. 8　　B. 9
   C. 10　　D. 11
4. 树中所有结点的度等于所有结点数加（ C　）。
   A. 0　　B. 1
   C. －1　　D. 2
5. 设 n，m 为一棵二叉树的两个结点，在中序遍历时，n 在 m 前的条件是（ C ）
   A. n 在 m 的右方　　B. n 是 m 的祖先
   C. n 在 m 的左方　　D. n 是 m 的子孙
6. 利用逐点插入建立序列（50，72，43，85，75，20，35，45，65，30）对应的二叉排序树以后，要查找元素 30 要进行（ B ）次元素间的比较。
   A. 4　　B. 5
   C. 6　　D. 7
7. 在平衡二叉树中，（ C ）。
   A. 任意结点的左、右子树结点数目相同
   B. 任意结点的左、右子树高度相同
   C. 任意结点的左右子树高度之差的绝对值不大于 1
   D. 不存在度为 1 的结点
8. 由元素序列（27，16，75，38，51）构造平衡二叉树，则首次出现的最小不平衡子树的根（即离插入结点最近且平衡因子的绝对值为 2 的结点）为（ D ）
   A. 27　　B. 38
   C. 51　　D. 75
9. 在二叉树的顺序存储中，每个结点的存储位置与其父结点、左右子树结点的位置都存在一个简单的映射关系，因此可与三叉链表对应。若某二叉树共有 n 个结点，采用三叉链表存储时，每个结点的数据域需要 d 个字节，每个指针域占用 4 个字节，若采用顺序存储，则最后一个结点下标为 k（起始下标为 1），那么（ A ）时采用顺序存储更节省空间。
   A. d<12n/(k-n)　　B. d>12n/(k-n)
   C. d<12n/(k+n)　　D. d>12n/(k+n)
10. 在常用的描述二叉排序树的存储结构中，关键字值最大的结点（ B ）
   A. 左指针一定为空　　B. 右指针一定为空
   C. 左右指针均为空　　D. 左右指针均不为空

（二）应用题

1、一棵满 k 叉树，按层次遍历（从 1 开始对全部结点进行编号）存储在一维数组中，试计算编号为 u 的结点的第 i 个孩子（若存在）的下标以及编号为 v 的结点的双亲结点（若存在）的下标。

答：

结点下标为 u 的结点的第 i 个孩子的下标：$k(u-1)+1+i$

结点下标为 v 的结点的父母结点的下标：$\lfloor (v-2)/k \rfloor +1$

2、试求有 n 个叶结点的非满的完全二叉树的高度.

答：

完全二叉树中叶子结点数为 n，则根据完全二叉树的性质，度为 2 的结点数是 n-1，而完全二叉树中，度为 1 的结点数至多为 1，

所以具有 n 个叶子结点的完全二叉树结点数是 n+(n-1)+1=2n 或 2n-1（有或无度为 1 的结点）。

由于具有 2n(或 2n-1)个结点的完全二叉树的深度是⌊log2(2n)⌋+1(或⌊log2(2n-1)⌋+1)，即⌈log2n⌉+1,故 n 个叶结点的非满的完全二叉树的高度是⌈log2n⌉+1。(最下层结点数>=2)。

3、已知一棵度为 m 的树中有 N1 个度为 1 的结点，N2 个度为 2 的结点，……Nm 个度为 m 的结点，问该树中有多少个叶子结点。请写出推导过程。

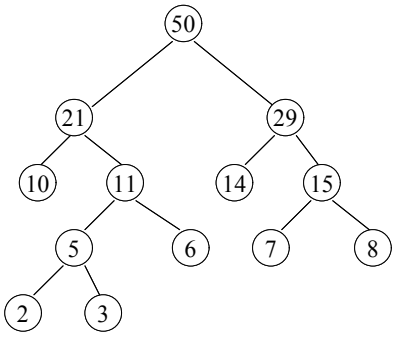
答：设 N 为总结点数，N0 为叶子结点数则：N=N0+N1+N2+……+Nm

又有：N-1=度的总数，则：N-1=N1*1+N2*2+……Nm*m

则有：N0=1+N2+2N3+……+(m-1)Nm

4. 有七个带权结点，其权值分别为 3,7,8,2,6,10,14，试以它们为叶子结点构造一棵哈夫曼树，并计算出带权路径长度 WPL。

答：

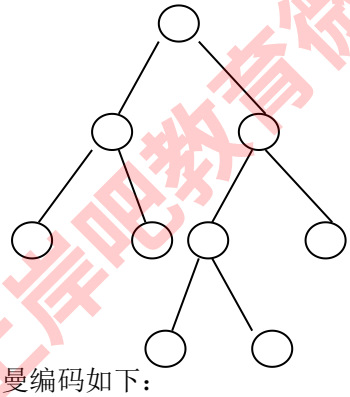


WPL = 3*4+7*3+8*3+2*4+6*3+10*2+14*2 = 131

5、给定字母 a,b,c,d,e 的使用频率为 0.09,0.17,0.2,0.23,0.31。设计以该权值为基础的哈夫曼树，并给出哈夫曼编码？平均长度是多少？

答：

构造的哈夫曼树如下：



哈夫曼编码如下：

c(00)

d(01)

a(100)

b(101)

e(11)

平均长度=0.09*3+0.17*3+0.2*2+0.23*2+0.31*2=2.26

6. 从概念上讲，树，森林和二叉树是三种不同的数据结构，将树，森林转化为二叉树的基本目的是什么，并指出树和二叉树的主要区别。

解：

树的孩子兄弟链表表示法和二叉树二叉链表表示法，本质是一样的，只是解释不同，也就是说树（树是森林的特例，即森林中只有一棵树的特殊情况）可用二叉树唯一表示，并可使用二叉树的一些算法去解决树和森林中的问题。

树和二叉树的区别有：一是二叉树的度至多为 2，树无此限制；二是二叉树有左右子树之分，即使在只有一个分枝的情况下， 也必须指出是左子树还是右子树，树无此限制；

7. 如果给出了一个二叉树结点的前序序列和中序序列，能否构造出此二叉树?若能，请证明之。若不能，请给出反例。如果给出了一个二叉树结点的前序序列和后序序列，能否构造出此二叉树?若能，请证明之。若不能，请给出反例。

解：

给定二叉树前序序列和中序序列，可以唯一确定该二叉树。因为前序序列的第一个元素是根结点，该元素将二叉树中序序列分成两部分，左边（设 l 个元素）表示左子树，若左边无元素，则说明左子树为空；右边（设 r 个元素）是右子树，若为空，则右子树为空。根据前序遍历中“根—左子树—右子树”的顺序，则由从第二元素开始的 l 个结点序列和中序序列根左边的 l 个结点序列构造左子树，由前序序列最后 r 个元素序列与中序序列根右边的 r 个元素序列构造右子树。

由二叉树的前序序列和后序序列不能唯一确定一棵二叉树，因无法确定左右子树两部分。例如，任何结点只有左子树的二叉树和任何结点只有右子树的二叉树，其前序序列相同，后序序列相同，但却是两棵不同的二叉树。

（三）算法设计题

1. 请利用栈的基本操作写出先序遍历二叉树的非递归形式的算法。要求以二叉链表作为二叉树的存储结构。函数原型如下：

```
void PreOrder(Bitree T);
```

答：

```
void PreOrder(Bitree T)
{
    InitStack(S);
    Push(S,T);
    while(!StackEmpty(S))
    {
        while(Gettop(S,p)&&p)
        {
            visit(p->data);
            push(S,p->lchild);
        }
        pop(S,p);
        if(!StackEmpty(S))
        {
            pop(S,p);
            push(S,p->rchlid);
        }
    }
}
```

2. 试写一个判别给定二叉树是否为二叉排序树的递归算法，设此二叉树以二叉链表作存储结构，且树中结点的关键字均不同。函数原型如下：

```
int Is_BSTree (BiTree T);
```

答：int last=0,flag=1;

```
int Is_BSTree (BiTree T)
{
    if(T->lchild&&flag)     Is_BSTree(T->lchild);
    if(T->data<last)        flag=0;
    last=T->data;
    if(T->rchild&&flag)     Is_BSTree(T->rchild);
    return flag;
}
```

3．假设一个仅包含二元运算符的算术表达式以链表形式存储在二叉树 BT 中，写出计算该算术表达式值的算法。

答：以二叉树表示算术表达式，根结点用于存储运算符。若能先分别求出左子树和右子树表示的子表达式的值，最后就可以根据根结点的运算符的要求，计算出表达式的最后结果。

```
typedef struct node
   {
    ElemType  data;  float val;
    char optr;  //只取 '+',  '-',  '*', '/'
    struct node *lchild, *rchild
    }BiNode,*BiTree;
float PostEval(BiTree bt)   // 以后序遍历算法求以二叉树表示的算术表达式的值
{
   float lv,rv;
   if(bt!=null)
   {
     lv=PostEval(bt->lchild); // 求左子树表示的子表达式的值
     rv=PostEval(bt->rchild); // 求右子树表示的子表达式的值
     switch(bt->optr)
      {
        case  '+' : value=lv+rv; break;
        case  '-' : value=lv-rv;break;
        case  '*' : value=lv*rv;break;
        case  '/' : value=lv/rv;
      }
   }
   return(value);
}
```

4．设计算法， 已知一棵以二叉链表存储的二叉树，root 指向根结点，p 指向二叉树中任一结点，编写算法求从根结点到 p 所指结点之间的路径（要求输出该路径上每个结点的数据）。

答：

```
void path(Bintree  T, Bintree p)
{Bintree stack[max],q;
int tag[max],top=0,find=0;
q=T;
while ((q||top)&& find==0)
  {while (q)
    {stack[top]=q; tag[top++]=0;
     q=q->lchild;
    }
  if (top>0)
    {q=stack[top-1];
      if (tag[top-1]==1)
         {if (q==p)
             {for (i=0;i<top;i++) printf(“%d”,stack[i]->data);
        find=1;
        }
             else top--;
      }
        if (top>0&&!find)
           {q=q->rchild;
             tag[top-1]=1;
      }
    }
  }
}
```

5．已知二叉树中的结点类型用 BinTreeNode 表示，被定义为：

```
struct BinTreeNode{
      char data;
      BinTreeNode  *leftChild, *rightChild;
      };
```

其中 data 为结点值域；leftChild 和 rightChild 分别为指向左、右孩子结点的指针域，根据下面函数声明编写出求一棵二叉树高度的算法，该高度由函数返回。参数 BT 初始指向这棵二叉树的根点。

```
int BtreeHeight (BinTreeNode *BT);
```

答：算法如下

```
int BtreeHeight(BinTreeNode * BT)
{ int h1,h2,h;
  if (BT==NULL) h=0;
  else{
      h1 = BTreeHeight (BT－>leftChild);
      h2 = BTreeHeight (BT－>rightChild);
      if (h1>h2) h=h1＋1;
      else h=h2＋1;
```

```
      }
  return h;
}
```

6.设一棵二叉树以二叉链表为存储结构，结点结构为(lchild, data,rchild)，设计一个算法将二叉树中所有结点的左，右子树相互交换。

解：

```
void exchange(BiTree bt)
{BiTree s;
if(bt)
{s=bt->lchild;
 bt->lchild=bt->rchild;
 bt->rchild=s;
 exchange(bt->lchild);
 exchange(bt->rchild);}
}
```

7.试写出复制一棵二叉树的算法。

解：

```
BiTree  Copy(BiTree  t)//复制二叉树 t
{BiTree bt;
    if (t==null) bt=null;
    else{
         bt=(BiTree)malloc(sizeof(BiNode));
         bt->data=t->data;
         bt->lchild=Copy(t->lchild);
         bt->rchild=Copy(t->rchild);
         }
return(bt);
 }//结束 Copy
```

8. 假设二叉树采用二叉链存储结构存储，试设计一个算法，输出从每个叶子结点到根结点的路径。

解：

题目解析：采用 path 数组存放路径， pathlen 整数存放路径长度。递归模型如下：

```
f(b,path,pathlen)：输出 path 值                          当 b 为叶子结点
f(b,path,pathlen)：将 b->data 放入 path，pathlen++;      其他情况
                   f(b->lchild,path,pathlen);
                   f(b->rchild,path,pathlen);
```

具体算法如下：

```
void Allpath(BTNode *b,ElemType path[],int pathlen)
//初始调用时 path 为空，pathlen 为 0
{
  int i;
  if (b!=NULL)
  { if (b->lchild==NULL&& b->rchild==NULL)  //*b 为叶子结点
    { printf(“%c 到根结点路径：%c”,b->data, b->data);
      for (i=pathlen-1;i>=0;i--)
          printf (“%c”,path[i]);
      printf (“\n”);
```

```
    }
    else
    { path[pathlen]= b->data;                    //将当前结点放入路径中
      pathlen++;                                 //路径长度增 1
      Allpath(b->lchild,path,pathlen);           //递归扫描左子树
      Allpath(b->rchild,path,pathlen);           //递归扫描右子树
      pathlen--;                                 //环境恢复
    }
  }
}
```

9. 假设二叉树采用二叉链存储结构存储，试设计一个算法，输出该二叉树中第一条最长的路径长度，并输出此路径上个结点的值。

解：

题目解析：采用 path 数组保存扫描到当前结点的路径， pathlen 保存扫描到当前结点的路径长度，longpath 数组保存最长的路径， longpathlen 保存最长路径长度。当 b 为空时，表示当前扫描的一个分支已扫描完毕，将 pathlen 与 longpathlen 进行比较，将较长路径及路径长度分别保存在 longpath 和 longpathlen 中。

具体算法如下：

```
void Longpath(BTNode *b,ElemType path[],int pathlen,ElemType longpath[],int longpathlen)
{
  int i;
  if (b==NULL)
  { if (pathlen>longpathlen)                   //若当前路径更长，将路径保存在 longpath 中
    { for (i=pathlen-1;i>=0;i--)
        longpath[i]=path[i];
      longpathlen=pathlen;
    }
  }
  else
  { path[pathlen]= b->data;                                    //将当前结点放入路径中
    pathlen++;                                                 //路径长度增 1
    Longpath(b->lchild,path,pathlen,longpath,longpathlen);     //递归扫描左子树
    Longpath(b->rchild,path,pathlen,longpath,longpathlen);     //递归扫描右子树
    pathlen--;                                                 //环境恢复
  }
}
```

四、图

大纲要求：

（一） 图的概念
（二） 图的存储及基本操作
  1. 邻接矩阵法
  2. 邻接表法
（三） 图的遍历
  1. 深度优先搜索

2. 广度优先搜索

（四） 图的基本应用及其复杂度分析

1. 最小（代价）生成树
2. 最短路径
3. 拓扑排序
4. 关键路径

知识点：

1. 图的基本概念，包括：图的定义和特点、无向图、有向图、入度、出度、完全图、生成树、路径长度、回路、（强）连通图、（强）连通分量等概念。掌握与这些概念相联系的相关计算题。在基本概念中，完全图、连通分量、生成树和邻接点是重点。
2. 图的存储形式。图是复杂的数据结构，有顺序和链式两种存储结构：数组表示法（重点是邻接矩阵），邻接表与逆邻接表，这两种存储结构对无向图和有向图均使用。
3. 熟练掌握图的两种遍历算法：深度遍历和广度遍历。深度遍历和广度遍历是图的两种基本的遍历算法，这两个算法对图一章的重要性等同于“先序、中序、后序遍历”对于二叉树一章的重要性。掌握图的两种遍历算法的应用，图一章的算法设计题常常是基于这两种基本的遍历算法而设计的。例如，在（强）连通图中，主过程一次调用深（广）度优先遍历过程（DFS/BFS），即可遍历全部顶点，故可以用此方法求出连通分量的个数，要会画出遍历中形成的深（广）度优先生成树和生成森林。又如，“求最长的最短路径问题”和“判断两顶点间是否存在长为K的简单路径问题”，就用到了广度遍历和深度遍历算法。
4. 最小生成树的概念。连通图的最小生成树通常是不唯一的，但最小生成树边上的权值之和是唯一的。掌握最小生成树的构造方法：PRIM算法和KRUSKAL算法，根据这两种算法思想用图示法表示出求给定网的一棵最小生成树的过程。
5. 拓扑排序是在有向图上对入度（先、后）为零的顶点的一种排序，通常结果不唯一。拓扑排序有两种方法，一是无前趋的顶点优先算法，二是无后继的顶点优先算法。换句话说，一种是“从前向后”的排序，一种是“从后向前”排。后一种排序出来的结果是“逆拓扑有序”的。用拓扑排序和深度优先遍历都可判断图是否存在环路。
6. 关键路径问题是图一章的难点问题。理解关键路径的关键有三个方面：一是何谓关键路径，二是最早时间的含义及求解方法，三是最晚时间的含义及求解方法。简单地说，最早时间是通过“从前向后”的方法求的，而最晚时间是通过“从后向前”的方法求解的，并且，要想求最晚时间必须是在所有的最早时间都已经求出来之后才能进行。熟练掌握求解的过程和步骤。关键路径问题是工程进度控制的重要方法，具有很强的实用性。理解“减少关键活动时间可以缩短工期”是指该活动为所有关键路径所共有，且减少到尚未改变关键路径的前提下有效。
7. 最短路径问题也是为图一章的难点问题。最短路径问题分为两种：一是求从某一点出发到其余各点的最短路径；二是求图中每一对顶点之间的最短路径。解决第一个问题用DIJSKTRA算法，解决第二个问题用FLOYD算法，注意区分。掌握这两个算法，并能手工熟练模拟。掌握用求最短路径问题来解决的应用问题（如旅游景点及旅游路线的选择问题）。

练习题：

（一）选择题：

1. 下列哪一种图的邻接矩阵是对称矩阵？（ B ）

   A. 有向图　　B. 无向图

   C. AOV 网　　D. AOE 网

2. 当各边上的权值（ A ）时，广度优先遍历算法可用来解决单源最短路径问题。

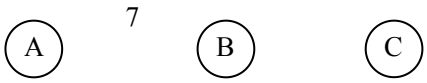
   A. 均相等　　B. 均不相等

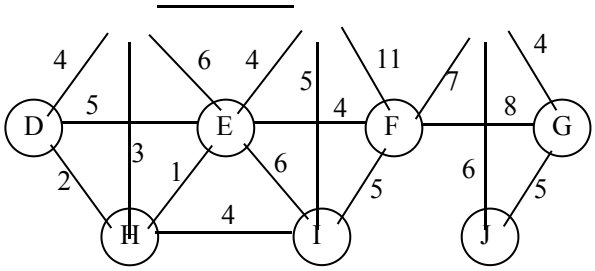
   C. 至少一多半相等　　D. 至少一少半相等

3. 有 n 个结点的无向图的边数最多为（ B ）
   A. n+1　　B. n(n-1)/2
   C. n(n+1)　　D. 2n(n+1)
4. 在一个图中，所有顶点的度数之和与图的边数的比是（ C ）
   A. 1:2　　B. 1:1
   C. 2:1　　D. 4:1
5. 已知有向图 G=(V,E)，其中 V={v1,v2,v3,v4,v5,v6,v7}，E={<v1,v2>,<v1,v3>,<v1,v4>,<v2,v5>,<v3,v5>,<v3,v6>,<v4,v6>,<v3,v7>,<v6,v7>}，G 的拓扑序列是（ A ）。
   A. v1,v3,v4,v6,v2,v5,v7　　B. v1,v3,v2,v6,v4,v5,v7
   C. v1,v3,v4,v5,v2,v6,v7　　D. v1,v2,v5,v3,v4,v6,v7
6. 在一个具有 n 个顶点的无向图中，要连通全部顶点至少需要（ C ）条边。
   A. n　　B. n+1
   C. n-1　　D. n/2
7. 简单无向图的邻接矩阵是对称的，可以对其进行压缩存储。若无向图 G 有 n 个结点，其邻接矩阵为 A[1…n,1…n]，且压缩存储在 B[1…k]，则 k 的值至少为（ D ）。
   A. n(n+1)/2　　B. $n^2/2$
   C. (n-1)(n+1)/2　　D. n(n-1)/2
   分析：简单无向图的邻接矩阵是对称的，且对角线元素均是 0，故压缩存储只须存储下三角或上三角（均不包括对角线）即可。
8. 无向图中一个顶点的度是指图中（ C ）
   A. 通过该顶点的简单路径数
   B. 通过该顶点的回路数
   C. 与该顶点相邻接的顶点数
   D. 与该顶点连通的顶点数
9. 一个含有 n 个顶点和 e 条边的简单无向图，在其邻接矩阵存储结构中共有（ D ）个零元素。
   A. e
   B. 2e
   C. n^2-e
   D. n^2-2e
10. 若采用邻接矩阵来存储简单有向图，则其某一个顶点 i 的入度等于该矩阵（ D ）。
   A. 第 i 行中值为 1 的元素个数
   B. 所有值为 1 的元素个数
   C. 第 i 行及第 i 列中值为 1 的元素总个数
   D. 第 i 列中值为 1 的元素个数
11. 若一个具有 n 个结点、k 条边的非连通无向图是一个森林（n>k），则该森林中必有（ C ）棵树。
   A. k　　B. n
   C. n-k　　D. n+k
12. 若 G 是一个具有 36 条边的非连通无向图（不含自回路和多重边），则图 G 至少有（ B ）个顶点。
   A. 11　　B. 10
   C. 9　　D. 8

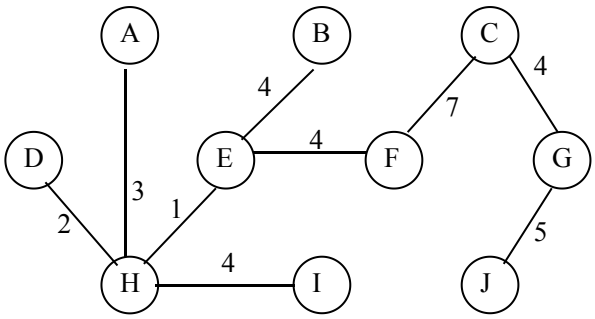
（二）应用题

1、用深度优先搜索遍历如下图所示的无向图，试给出以 A 为起点的顶点访问序列（同一个顶点的多个邻点，按字母顺序访问），并给出一个最小生成树。





答：
最小生成树：



深度优先搜索顶点访问序列：A　　B　E　D　H　I　F　C　G　J

2. 下面是求无向连通图最小生成树的一种方法。

将图中所有边按权重从大到小排序为（$e_1, e_2, \cdots, e_m$）

```
i:=1
WHILE（所剩边数 >=顶点数）
    BEGIN
        从图中删去 ei
        若图不再连通，则恢复 ei
        i:=i+1
    END.
```

试证明这个算法所得的图是原图的最小代价生成树。

答：无向连通图的生成树包含图中全部 n 个顶点，以及足以使图连通的 n-1 条边。而最小生成树则是各边权值之和最小的生成树。从算法中 WHILE（所剩边数>=顶点数）来看，循环到边数比顶点数少 1（即 n-1）停止，这符合 n 个顶点的连通图的生成树有 n-1 条边的定义；由于边是按权值从大到小排序，删去的边是权值大的边，结果的生成树必是最小生成树；算法中“若图不再连通，则恢复 ei”，含义是必须保留使图连通的边，这就保证了是生成树，否则或者是有回路，或者成了连通分量，均不再是生成树。

3. 对一个图进行遍历可以得到不同的遍历序列，那么导致得到的遍历序列不唯一的因素有哪些？
答：遍历不唯一的因素有：开始遍历的顶点不同；存储结构不同；在邻接表情况下邻接点的顺序不同。

4、一个带权连通图的最小生成树是否唯一？说明在什么情况下最小生成树有可能不唯一？
答：一个带权连通图的最小生成树有可能不唯一。当图中依附于某个顶点的多条边出现权值相同的边时，就有可能得到的最小生成树不唯一。这里所说的最小生成树不唯一，是指生成树的形状不唯一，这些生成树的权值之和应该是相同的。

5. 已知加权有向图 G 的邻接矩阵如下：

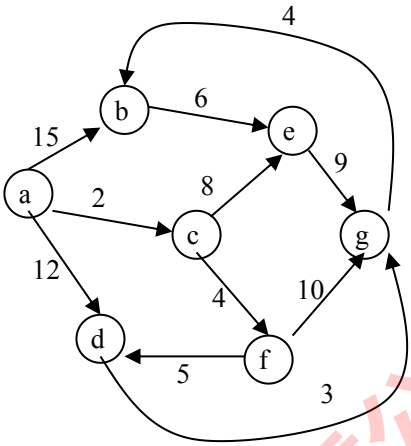
$$\begin{bmatrix} \infty & 15 & 2 & 12 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 6 & \infty & \infty \\ \infty & \infty & \infty & \infty & 8 & 4 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & \infty & \infty & 5 & \infty & 10 \\ \infty & 4 & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

（1） 画出该有向图 G

（2） 试利用 Dijkstra 算法求 G 中从顶点 a 到其他各顶点间的最短路径，并给出求解过程。

答：

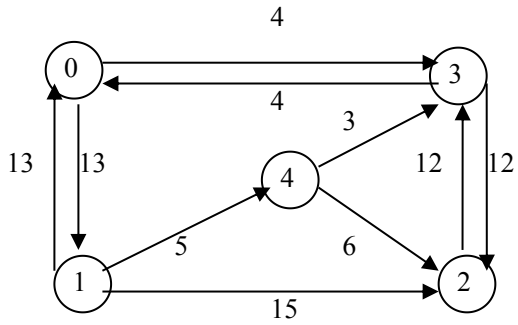
（1）



（2）

终点 / Dist	b	c	d	e	f	g	S
k=1	15 (a, b)	2 (a, c)	12 (a, d)				{a, c}
k=2	15 (a, b)		12 (a, d)	10 (a, c, e)	6 (a, c, f)		{a, c, f}
k=3	15 (a, b)		11 (a, c, f, d)	10 (a, c, e)		16 (a, c, f, g)	{a, c, f, e}
k=4	15 (a, b)		11 (a, c, f, d)			16 (a, c, f, g)	{a, c, f, e, d}
k=5	15 (a, b)					14 (a, c, f, d, g)	{a, c, f, e, d, g}
k=6	15 (a, b)						{a, c, f, e, d, g, b}

6. 下图中的顶点表示村庄，有向边代表交通路线，若要建立一家医院，试问建在哪一个村庄能使各村庄总体交通代价最小？



解：
该图的邻接矩阵如下：

$$A=\begin{bmatrix} 0 & 13 & \infty & 4 & \infty \\ 13 & 0 & 15 & \infty & 5 \\ \infty & \infty & 0 & 12 & \infty \\ 4 & \infty & 12 & 0 & \infty \\ \infty & \infty & 6 & 3 & 0 \end{bmatrix}$$

利用 Floyd 算法可求得两顶点之间最短路径长度。最后求得：

$$A_4=\begin{bmatrix} 0 & 13 & 16 & 4 & 18 \\ 12 & 0 & 11 & 8 & 5 \\ 16 & 29 & 0 & 12 & 34 \\ 4 & 17 & 12 & 0 & 22 \\ 7 & 20 & 6 & 3 & 0 \end{bmatrix}$$

从 A_4 中可求得每对村庄之间的最少交通代价。假设医院建在 i 村庄时，其他各村庄往返总的交通代价如下所示：
医院建在村庄 0 时，各村庄往返总的交通代价为 12+16+4+7+13+16+4+18=90；
医院建在村庄 1 时，各村庄往返总的交通代价为 13+29+17+20+12+11+8+5=115；
医院建在村庄 2 时，各村庄往返总的交通代价为 16+11+12+6+16+29+12+34=136；
医院建在村庄 3 时，各村庄往返总的交通代价为 4+8+12+3+4+17+12+22=82；
医院建在村庄 4 时，各村庄往返总的交通代价为 18+5+34+22+7+20+6+3=115。
显然，把医院建在村庄 3 时总体交通代价最少。

（三）算法设计题

1.设计一个算法，求无向图 G（采用邻接表存储）的连通分量个数。
解法一：采用深度优先遍历方法。算法如下：

```
void DFS(AGraph *G, int v)
{
  ArcNode *p;
  visited[v]=1;                    //置已访问标记
  prinf(”%d”,v);                   //输出被访问顶点的编号
  p=G->adjlist[v].firstarc;        //p 指向顶点 v 的第一条边的终结点
  while (p!=NULL)
  {
    if (visited[p->adjvex]==0)     //若 p->adjvex 顶点未访问，递归访问它
        DFS(G,p->adjvex);
    p=p->nextarc;                  //p 指向顶点 v 的下一条边的终结点
```

```
  }
}
int ConnNum1(AGraph *G)            //求图 G 的连通分量
{
  int i, num=0;
  for (i=0; i<G->n; i++)
      visited[i]=0;
  for (i=0; i<G->n; i++)
      if (visited[i]==0)
      {
        DFS(G,i);                  //调用 DFS 算法
        num++;
      }
  return(num);
}
```

解法二：采用广度优先遍历方法。算法如下：

```
void BFS(AGraph *G, int v)
{
  ArcNode *p;
  int Qu[MAXV], front=0, rear=0;          //定义循环队列并初始化
  int w,i;
  for (i=0; i<G->n; i++)  visited[i]=0;   //访问标志数组初始化
  prinf(" 2%d",v);                        //输出被访问顶点的编号
  visited[v]=1;                           //置已访问标记
  rear=(rear+1)%MAXV;
  Qu[rear]=v;                             //v 入队
  while (front!=rear)                     //若队列不空时循环
  {
    front=(front+1)%MAXV;
    w=Qu[front];                          //出队并赋予 w
    p=G->adjlist[w].firstarc;             //找与顶点 w 邻接的第一个顶点
    while (p!=NULL)
    {
      if (visited[p->adjvex]==0)          //若当前邻接顶点未被访问
      {
        printf(" %2d", p->adjvex);        //访问相邻顶点
        visited[p->adjvex]=1;             //置该顶点已被访问的标志
        rear=(rear+1)%MAXV;               //该顶点入队
        Qu[rear]= p->adjvex;
      }
      p=p->nextarc;                       //找下一个邻接顶点
    }
  }
  printf(" \n");
}
int ConnNum2(AGraph *G)            //求图 G 的连通分量
{
  int i, num=0;
```

```
  for (i=0; i<G->n; i++)
      visited[i]=0;
  for (i=0; i<G->n; i++)
      if (visited[i]==0)
      {
        BFS(G,i);                    //调用 BFS 算法
        num++;
      }
  return(num);
}
```

五、查找

大纲要求：

（一） 查找的基本概念
（二） 顺序查找法
（三） 折半查找法
（四） B-树
（五） 散列（Hash）表及其查找
（六） 查找算法的分析及应用

知识点：

1. 线性表上的查找。对于顺序表采用顺序查找方法，逐个比较，顺序表设置了监视哨使查找效率大大提高。对于有序顺序表采用折半查找法，其判定树是唯一的。对于索引结构，采用索引顺序查找算法，此算法综合了上述两者的优点，既能较快速地查找，又能适应动态变化的要求。注意这三种查找的平均查找长度。掌握顺序查找和折半查找算法的实现，其中，折半查找还要特别注意适用条件以及其递归实现方法。
2. B-树是多路平衡外查找树，用于文件系统。要能手工模拟 B-树插入和删除关键字使 B-树增高和降低，会推导 B-树的平均查找长度。
3. 散列表的查找算法。基本思想是：根据当前待查找数据的特征，以记录关键字为自变量，设计一个散列函数，该函数对关键字进行转换后，其解释结果为待查的地址。熟练掌握散列函数的设计，冲突解决方法的选择及冲突处理过程的描述。散列表中关键字的查找只能用散列函数来计算，不能顺序查找，也不能折半查找。在闭散列法解决冲突的情况下，元素删除也只能做标记，不能物理地删除。理想情况下，散列表的平均查找长度是 O(1)，优于其他查找方法。

练习题：

（一）选择题：

1. 某顺序存储的表格中有 90000 个元素，已按关键字值额定升序排列，假定对每个元素进行查找的概率是相同的，且每个元素的关键字的值皆不相同。用顺序查找法查找时，平均比较次数约为（ C ）
   A. 25000　　B. 30000
   C. 45000　　D. 90000
2. 适用于折半查找的表的存储方式及元素排列要求为（ D ）
   A. 链接方式存储，元素无序　　B. 链接方式存储，元素有序

C．顺序方式存储，元素无序　　D．顺序方式存储，元素有序

3. 散列文件使用散列函数将记录的关键字值计算转化为记录的存放地址，因为散列函数是一对一的关系，则选择好的（　D　）方法是散列文件的关键。

A．散列函数　　B．除余法中的质数

C．冲突处理　　D．散列函数和冲突处理

4. 每个存储结点只含有一个数据元素，存储结点均匀地存放在连续的存储空间，使用函数值对应结点的存储位置，该存储方式是（　D　）存储方式

A．顺序　　B．链接

C．索引　　D．散列

5. 如果要求一个线性表既能较快地查找，又能适应动态变化的要求，则可以采用（A ）查找法。

A．分块　　B．顺序

C．二分　　D．散列

（二）应用题

1.设散列表的长度为 13，散列函数为 H(K)=K%13，给定的关键字序列为：19，14，23，01，68，20，84，27，55，11，10，79。试画出用线性探测再散列解决冲突时所构成的散列表。并求等概率情况下这两种方法查找成功和查找不成功时的平均查找长度。

答：

线性探测再散列的散列表：

0	1	2	3	4	5	6	7	8	9	10	11	12
	14	1	68	27	55	19	20	84	79	23	11	10
	1	2	1	4	3	1	1	3	9	1	1	3

查找成功的平均长度为 ASL=1/12（1*6+2*1+3*3+4*1+9）=2.5

查找不成功的平均长度为 ASL=1/13（1+2+3+4…….+13）=7

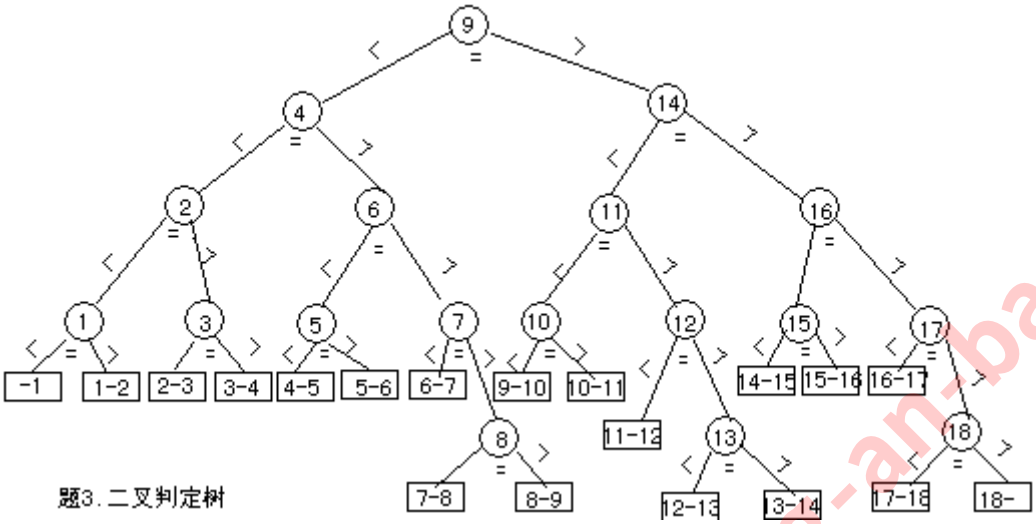
2．为什么说当装填因子非常接近 1 时，线性探查类似于顺序查找?为什么说当装填因子比较小(比如 α=0.7 左右)时，散列查找的平均查找时间为 O(1)?

答：

当 α 非常接近 1 时，整个散列表几乎被装满。由于线性探查法在关键字同义时解决冲突的办法是线性地向后查找，当整个表几乎装满时，它就很类似于顺序查找了。

当 α 比较小时，关键字碰撞的几率比较小，一般情况下只要按照散列函数计算出的结果能够 1 次性就找到相应结点，因此它的平均查找时间接近于 1.

3．画出对长度为 18 的有序的顺序表进行二分查找的判定树，并指出在等概率时查找成功的平均查找长度，以及查找失败时所需的最多的关键字比较次数。



题3.二叉判定树

答：如图：

答：请看题图。

等概率情况下，查找成功的平均查找长度为：

ASL=(1+2*2+3*4+4*8+5*3)/18=3.556

也可以用公式代，大约为：ASL=(18+1)lg(18+1)/18-1=3.346

查找失败时，最多的关键字比较次树不超过判定树的深度，此处为 5.

六、内部排序

大纲要求：

（一） 排序的基本概念
（二） 插入排序
    1. 直接插入排序
    2. 折半插入排序
（三） 起泡排序（bubble sort）
（四） 简单选择排序
（五） 希尔排序（shell sort）
（六） 快速排序
（七） 堆排序
（八） 二路归并排序（merge sort）
（九） 基数排序
（十） 各种内部排序算法的比较
（十一） 内部排序算法的应用

知识点：

1. 插入类排序的基本思想是假定待排序文件第一个记录有序，然后从第二个记录起，依次插入到排好序的有序子文件中，直到整个文件有序。从减少比较次数和移动次数进行了各种改进，在插入排序中有直接插入、折半插入、希尔排序。直接插入是依次寻找，折半插入是折半寻找，希尔排序是通过控制每次参与排序的数的总范围“由小到大”的增量来实现排序效率提高的目的。
2. 交换类排序基于相邻记录比较，若逆序则进行交换。起泡排序和快速排序是交换排序的例子，在起换排序的基础上改进得到快速排序，快速排序是目前最好的内部排序法。快速排序的思想：用

中间数将待排数据组一分为二。快速排序，在处理的“问题规模”这个概念上，与希尔有点相反，快速排序，是先处理一个较大规模，然后逐渐把处理的规模降低，最终达到排序的目的。

3. 选择类排序，可以分为：简单选择排序、堆排序。这两种方法的不同点是，根据什么规则选取最小的数。简单选择，是通过简单的数组遍历方案确定最小数；堆排序，是利用堆这种数据结构的性质，通过堆元素的删除、调整等一系列操作将最小数选出放在堆顶。堆排序较为重要，其最差性能比快速排序的最差性能好。
4. 归并排序是通过“归并”这种操作完成排序的目的，既然是归并就必须是两者以上的数据集合才可能实现归并，算法思想比较简单。
5. 基数排序，是一种特殊的排序方法，分为两种：多关键字的排序（扑克牌排序），链式排序（整数排序）。基数排序的核心思想也是利用“基数空间”这个概念将问题规模规范、变小，在排序的过程中，只要按照基数排序的思想，是不用进行关键字比较的，这样得出的最终序列就是一个有序序列。
6. 掌握各种排序方法的算法思想以及算法实现。掌握在最好、最坏、平均情况下各种排序方法的性能分析。归并排序、基数排序及时间复杂度为 $O(n^2)$ 的排序是稳定排序，而希尔排序、快速排序、堆排序等时间性能好的排序方法是不稳定排序（但特别注意，简单选择排序是不稳定排序）。

各种排序方法的综合比较

（1）时间性能

●按平均的时间性能来分，有三类排序方法：

时间复杂度为 O(nlogn)的方法有：快速排序、堆排序和归并排序，其中以快速排序为最好；

时间复杂度为 $O(n^2)$ 的有：直接插入排序、起泡排序和简单选择排序，其中以直接插入为最好，特别是对那些对关键字近似有序的记录序列尤为如此；

时间复杂度为 O(n)的排序方法只有，基数排序。

●当待排记录序列按关键字顺序有序时，直接插入排序和起泡排序能达到 O(n)的时间复杂度;而对于快速排序而言，这是最不好的情况，此时的时间性能蜕化为 $O(n^2)$，因此是应该尽量避免的情况。

●简单选择排序、堆排序和归并排序的时间性能不随记录序列中关键字的分布而改变。

（2）空间性能：指的是排序过程中所需的辅助空间大小。

●所有的简单排序方法(包括：直接插入、起泡和简单选择)和堆排序的空间复杂度为 O(1)；

●快速排序为 O(log$n$)，为栈所需的辅助空间;

●归并排序所需辅助空间最多，其空间复杂度为 O(n)；

●链式基数排序需附设队列首尾指针，则空间复杂度为 O(rd)。

（3）排序方法的稳定性能

稳定的排序方法指的是，对于两个关键字相等的记录，它们在序列中的相对位置，在排序之前和经过排序之后，没有改变。

当对多关键字的记录序列进行 LSD 方法排序时，必须采用稳定的排序方法。

对于不稳定的排序方法，只要能举出一个实例说明即可。

快速排序和堆排序是不稳定的排序方法。

练习题：

（一）选择题：

1. 下列四个序列中，哪一个是堆（ C ）

   A. 75,65,30,15,25,45,20,10　　B. 75,65,45,10,30,25,20,15

   C. 75,45,65,30,15,25,20,10　　D. 75,45,65,10,25,30,20,15

2. 下列排序算法中，在最好情况下，时间复杂度为 O(n)的算法是（ D ）。

   A. 选择排序　　B. 归并排序

   C. 快速排序　　D. 冒泡排序

3. 下述排序算法中，稳定的是（ B ）。

   A. 直接选择排序　　B. 基数排序

C．快速排序　　D．堆排序

4. 下列排序算法中，第一趟排序完毕后，其最大或最小元素一定在其最终位置的算法是（ D ）

A．归并排序　　B．直接插入排序

C．快速排序　　D．冒泡排序

5. 如果只想得到 1024 个元素组成的序列中的前 5 个最小元素，那么用（ D ）方法最快。

A．起泡排序　　B．快速排序

C．堆排序　　D．直接选择排序

6. 下面给出的 4 种排序方法中，排序过程中的比较次数与初始排序次序无关的是（A）。

A．选择排序法　　B．插入排序法

C．快速排序法　　D．堆排序法

（二）应用题

1. 一个堆积可以表示为一棵完全二叉树，请分别叙述堆积与二叉排序树的区别。

答：

若将堆积表示为一棵完全二叉树，则该二叉树中任意分支结点的值都大于或者等于其孩子结点（以大顶堆积为例），并且根结点具有最大值。而在二叉排序树中，要求所有左子树中的结点的值均小于根结点的值，所有右子树中的结点的值均大于或等于根结点的值，根结点不一定具有最大值。因此，堆积与二叉排序树不是同一回事。

（三）算法设计题

1.荷兰国旗问题：设有一个仅由红、白、蓝三种颜色的条块组成的条块序列，请编写一个时间复杂度为 O(n)的算法，使得这些条块按红、白、蓝的顺序排好，即排成荷兰国旗图案。函数原型如下：

```
typedef enum{RED,WHITE,BLUE} color;
void Flag(color a[],int n);
```

答：

```
typedef enum{RED,WHITE,BLUE} color;
void Flag(color a[],int n)
{
    i=0;j=0;k=n-1;
    while(j<=k)
        switch(a[j])
        {
            case RED:
                a[i]<->a[j];
                i++;j++;break;
            case WHITE:
                j++;break;
            case BLUE:
                a[j]<->a[k];
                k--;
        }
}
```

2. 可按如下所述实现归并排序：假设序列中有 k 个长度为小于等于 n 的有序子序列。利用过程 merge 对它们进行两两归并，得到$\lceil k/2 \rceil$个长度小于等于 2n 的有序子序列（$\lceil\ \rceil$表示取整），称为一趟归并

排序。反复调用一趟归并排序过程，使有序子序列的长度自 n=1 开始成倍地增加，直至使整个序列成为一个有序序列。试采用链表存储结构实现上述归并排序的非递归算法。函数原型如下：

```
void Linked_Mergesort(LinkedList &L);            //链表结构上的归并排序非递归算法
void Linked_Merge(LinkedList &L, LNode *p, Lnode *e1,Lnode *e2);
//对链表上的子序列进行归并，第一个子序列是从 p->next 到 e1,第二个是从 e1->next 到 e2
```

答：

```
void Linked_Mergesort(LinkedList &L);
{
    for(l=1;l<L.length;l*=2)
        for(p=L->next,e2=p; p->next; p=e2)
        {
            for(i=1,q=p; i<=l && q->next; i++,q=q->next)
                e1=q;
            for(i=1; i<=l && q->next; i++,q=q->next)
                e2=q;  //求两个待归并子序列的尾指针
            if(e1!=e2)
                Linked_Merge(L,p,e1,e2);
        }
}

void Linked_Merge(LinkedList &L, LNode *p, Lnode *e1,Lnode *e2);
{
    q=p->next;                    //q 和 r 为两个子序列的起始位置
    r=e1->next;
    while(q!=e1->next && r!=e2->next)
    {
        if(q->data < r->data)
        {
          p->next=q;
          p=q;
          q=q->next;
        }
        else
        {
          p->next=r;
          p=r;
          r=r->next;
        }
    }
    while(q!=e1->next)
      {
        p->next=q;
        p=q;
        q=q->next;
        }
    while(r!=e2->next)
      {
        p->next=r;
```

```
        p=r;
        r=r->next;
      }
}
```

3.有一种简单的排序算法,叫做计数排序。这种排序算法对一个待排序的表(用数组表示)进行排序,并将排序结果存放到另一个新的表中。必须注意的是,表中所有待排序的关键字互不相同,计数排序算法针对表中的每个记录,扫描待排序的表一趟,统计表中有多少个记录的关键字比该记录的关键字小。假设对某一个记录,统计出数值为 c,那么这个记录在新的有序表中的合适的存放位置即为 c。

(1)给出适用于计数排序的数据表定义。

(2)编写实现计数排序的算法。

(3)对于有 n 个记录的表,比较次数是多少?

(4)与直接选择排序相比,这种方法是否更好?为什么?

解:

```
(1)  typedef struct
{
ElemType data;
KeyType key;
}listtype;
(2)  void countsort(listtype a[],listtype b[],int n)
{
int i,j,count;
for(i=0;i<n;i++)
 {
   count=0;
   for(j=0;j<n;j++)
   if(a[j].key<a[i].key) count++;
   b[count]=a[i];
  }
}
```

(3) 对于有 n 个记录的表，关键字比较的次数是 n^2.

(4)直接选择排序比这种计数排序好，因为直接选择排序的比较次数为 n*(n-1)/2，且可在原地进行排序（稳定排序），而计数排序为不稳定排序，需要辅助空间多，为 0(n).

4．快速排序算法中，如何选取一个界值（又称为轴元素），影响着快速排序的效率，而且界值也并不一定是被排序序列中的一个元素。例如，我们可以用被排序序列中所有元素的平均值作为界值。编写算法实现以平均值为界值的快速排序方法。

解：

题目解析：保存划分的第一个元素。以平均值作为枢轴，进行普通的快速排序，最后枢轴的位置存入已保存的第一个元素，若此关键字小于平均值，则它属于左半部，否则属于右半部。

```
int partition (RecType r[],int l,h)
{
  int i=l,j=h,avg=0;
  for(;i<=h;i++)  avg+=R[i].key;
  i=l;
  avg=avg/(h-l+1);
  while (i<j)
    {
      while (i<j &&R[j].key>=avg) j--;
      if (i<j) R[i]=R[j];
      while (i<j &&R[i].key<=avg) i++;
      if (i<j) R[j]=R[i];
     }
  if(R[i].key<=avg) return  i;
  else return  i-1;
 }
void quicksort (RecType R[],int S,T);
{
  if (S<T)
   {
      k=partition (R,S,T);
      quicksart (R,S,k);
      quicksart (R,k+1,T);
    }
}
```