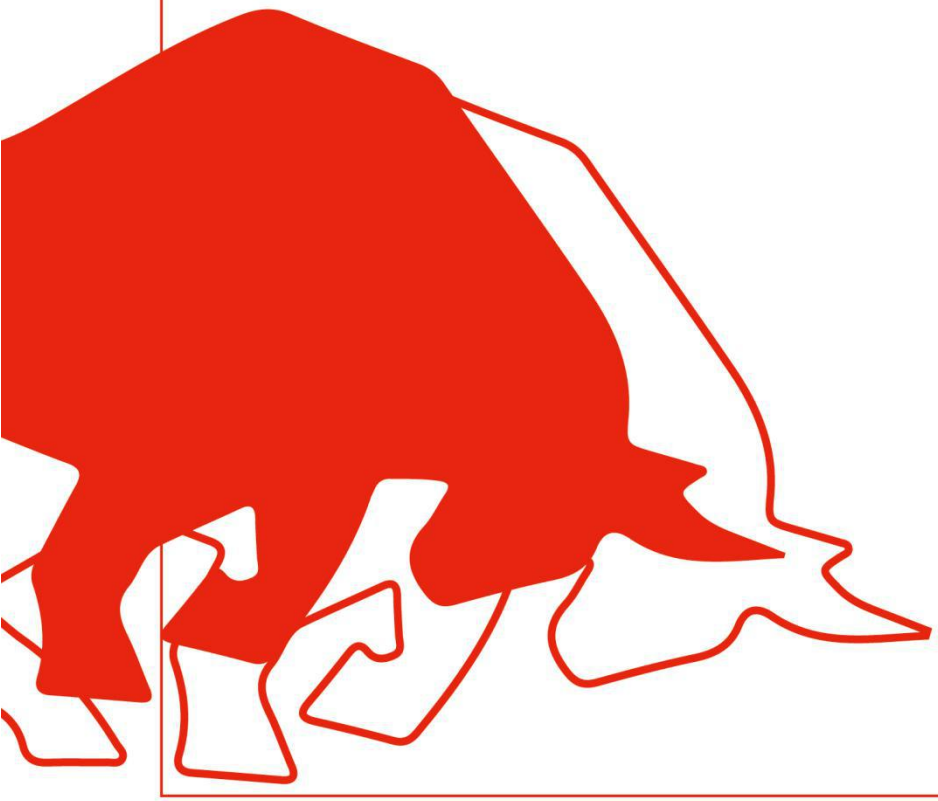
尚德机构
SUNLANDS

主观题汇总

数据结构导论



使用说明：

1. 此主观题汇总文档是按题型整理的，而题型来自于对历年真题的总结。参考 1810 考期，本科目主观题总分为 44 分，各题型分值分布为：①应用题：5*6 分=30 分；②算法设计题：2*7 分=14 分。
2. 所有知识点分高中低三个频次，以该知识点被考察次数和最新考试大纲为依据进行排序。
3. 每道题前数字表示曾经被考到的年份和考期，比如 1804，表示该题目在 2018 年 4 月份被考到。没有数字表示的为模拟题。

应用题汇总

高频知识点：

一．栈

1、（1704）设 A、B、C、D、E 五个元素依次进栈(进栈后可立即出栈),问能否得到下列序列:(1)A, B, C, D, E;(2)A,C, E, B, D

若能得到,刚给出该序列的操作过程(用 push(A)表示 A 进栈,pop(A)表示 A 出栈);若不能,则说明理由。

答案：

（1）**能**,操作过程为：**push(A)**，**pop(A)**，**push(B)**，**pop(B)**，**push(C)**，**pop(C)**，**push(D)**，**pop(D)**，**push(E)**，**pop(E)**。

（2）**不能**,不能的理由:对序列（2）中的 E,B,D 而言,E 最先出栈,此时 B 和 D 均在栈中,由于**B 先于 D 进栈,所以应有 D 先出栈**

二、队列

1、(1610)借助于队列能够将含有 n 个数据元素的栈逆置，比如栈 S 中的元素为{a，b，c}逆置后变成{c， b， a}。 试简述你的解决方案。

答案：

先将栈中元素**依次出栈并入队列**，然后使该队列元素**依次出队列并进入栈**。

三、数组

1、（1710）已知一个 7×6 的稀疏矩阵如题图所示,

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 16 & 0 & 0 & 0 & 0 & -16 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

试写出该稀疏矩阵的三元组表示。

答案：

该稀疏矩阵可表示为如下三元组表:

((0,0,16),

(0,5,-16),

(1,2,3),

(2,3,-8),

(4,0,91),

(6,2,15))

四、二叉树

1、（1810）高度为 h 的满二叉树，如果按层次自上而下，同层从左到右的次序从 1 开始编号。

（1）该树上有多少个结点？

答案：$2^h - 1$

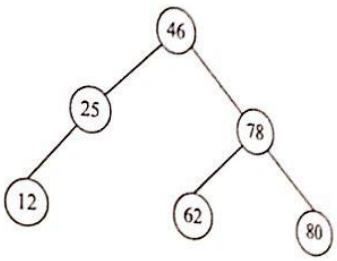
（2）编号为 i 的结点的左孩子和右孩子（若存在）的编号分别是多少？

答案：

左孩子：**2i**；右孩子：**2i+1**

2、（1810）给定数据序列{46,25,78,62,12,80}，试按元素在序列中的次序将它们依次插入一棵初始为空的二叉排序树，画出插入完成后的二叉排序树。

答案：



五、二叉树的存储结构

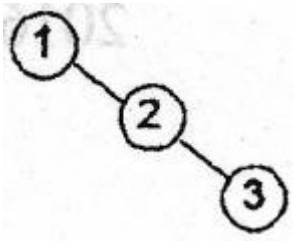
1、（1610）为便于表示二叉树的某些基本运算，

（1）则深度为 k 的二叉树的顺序存储结构中的数组的大小为多少？

答案：

数组的大小为 2k-1；

（2）画出如图所示的二叉树的顺序存储结构示意图，



答案：

顺序存储结构示意图： 

；

（3）并说明对一般形态的二叉树不太适合使用顺序存储结构来表示的原因。

答案：

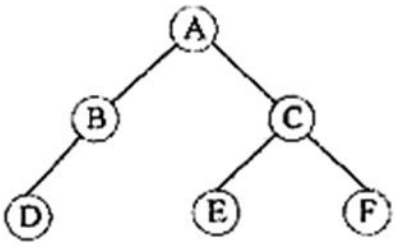
原因： **会造成存储空间的浪费现象。**

六、二叉树的遍历

1、（1704）已知一棵二叉树的先序遍历结果为 ABDCEF，中序遍历结果为 DBAECF，

（1）试画出这棵二叉树，

答案：

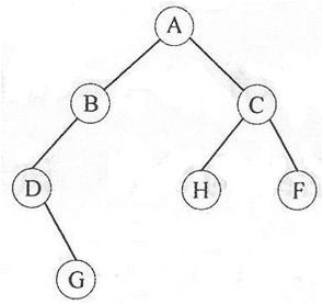


（2）写出这棵二叉树的后序遍历序列。

答案：

该二叉树的**后序**遍历序列为**:DBEFCA**

2、（1710）已知一棵二叉树如题图所示,试求该二叉树的先序遍历序列、后序遍历序列和层次遍历序列。



答案：

先序遍历序列为:**ABDGCHF**

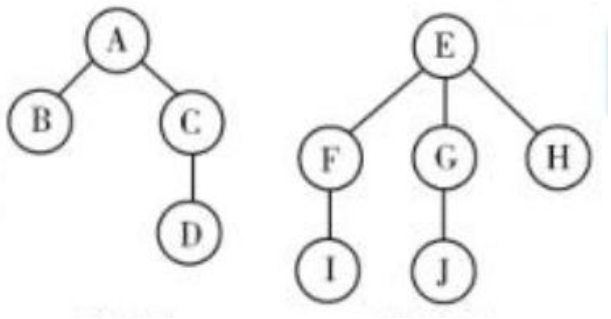
后序遍历序列为:**GDBHFCA**

层次遍历序列为:**ABCDHFG**

3、（1610）先序遍历、 中序遍历一个森林分别等同于先序、 中序遍历该森林所对应的二叉树。现已知一个森林的先序序列和中序序列分别为 ABCDEFIGJH 和 BDCAIFJGHE， 试画出该森林。

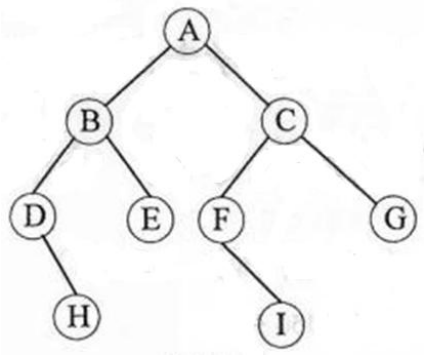
答案：

先根据给定的两个序列**构造出相应的二叉树**， 然后再将其**转成森林**：

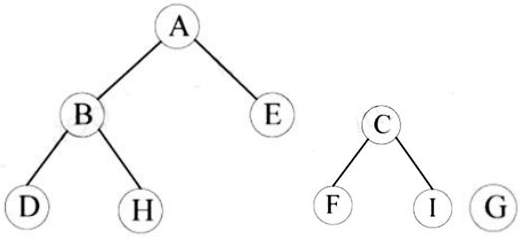


七、树和森林

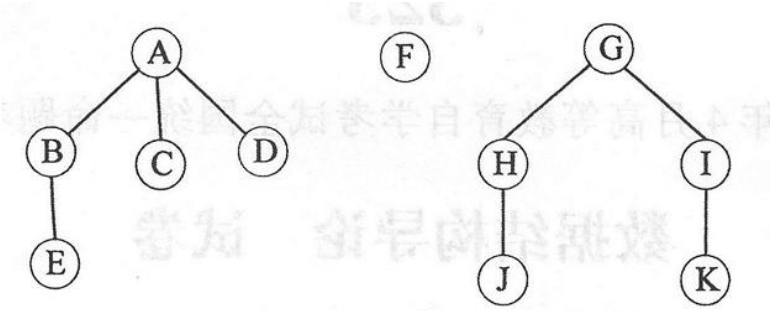
1、（1804）将题图所示的二叉树转换为对应的树或森林。



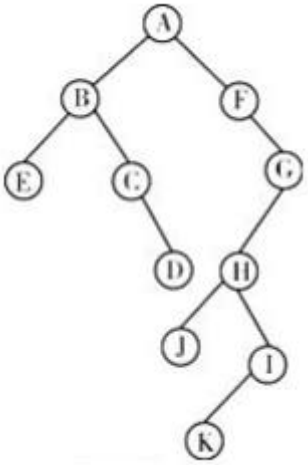
答案：



2、（1704）画出题图所示森林转换后所对应的二叉树。



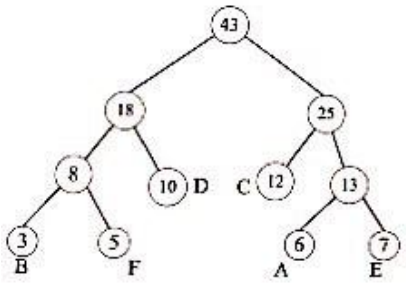
答案：



八、判定树和哈夫曼树

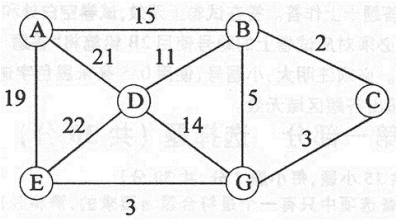
1、（1810）假设用于通讯的电文仅由 6 个字母 A,B,C,D,E,F 组成，各个字母在电文中出现的频率分别为：6，3，12，10，7，5，试为这 6 个字母设计哈夫曼树。（构建新二叉树时，要求新二叉树的左子树根的权值小于等于右子树根的权值。）

答案：



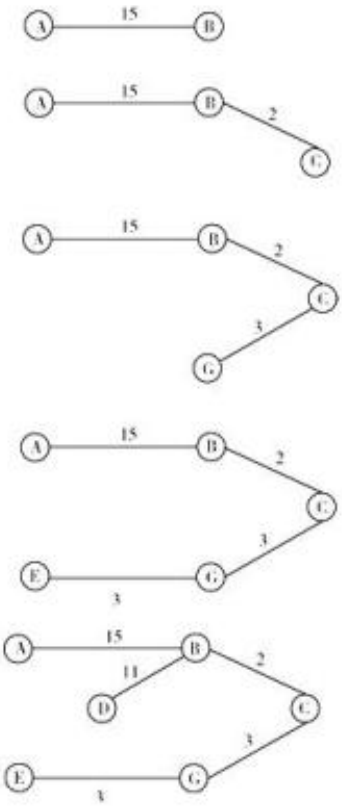
九、图的应用

1、（1704）已知如题图所示的无向带权图，

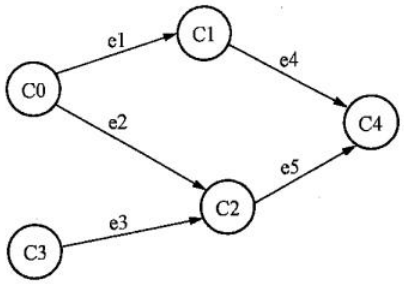


请从结点 A 出发，用普里姆（Prim）算法求其最小生成树，并画出过程示意图。

答案：



2、求图 a 所示有向图顶点的拓扑序列，图 b 是它的邻接表，在表头结点中增加一个数据域 in 表示相应顶点的入度。



a)

序号	vertex	in	firstarc
0	C0	0	→ 1 → 2 ∧
1	C1	1	→ 4 ∧
2	C2	2	→ 4 ∧
3	C3	0	→ 2 ∧
4	C4	2	∧

b)

答案：

首先 C0、C3 的入度都为 0,选 C0,删除 C0 及其边 e1、e2,调整 C1 的入度为 0, C2 的入度为 1，此时 C1、C3 的入度为 0,选 C3,删除 C3 及边 e3，调整 C2 的入度为 0,从 C1、C2 中选 C1,删除 C1 及边 e4,调整 C4 的入度为 1,选择 C2,删除 C2 及边 e5,调整 C4 的入度为 0,输出 C4,至此拓扑排序完成，**拓扑序列为 C0, C3, C1，C2, C4。**

十、交换排序

1、（1810）对键值序列（61，87，12，3，8，70）以位于最左位置的键值为基准进行由小到大的快速排序，请写出第一趟排序后的结果，并给出快速排序算法在平均情况和最坏情况下的时间复杂度。

答案：

初始关键字： 61 87 12 3 8 70

一次交换后： 8 87 12 3 61 70

二次交换后： 8 61 12 3 87 70

第一趟排序结果：[8 3 12] 61 [87 70]

平均情况下的**时间复杂度：** $O(nlog_2n)$

最坏情况下的**时间复杂度：** $O(n^2)$

2、（1710）给出一组关键字(20,29,11,74,35,3,8,56),写出冒泡排序前两趟的排序结果,并说明冒泡排序算法的稳定性如何?

答案：

第一趟:(20,11,29,35,3,8,56),74

第二趟:(11,20,29,3,8,35),56,74

冒泡排序算法是**稳定**的排序算法

十一、归并排序

1、（1704、1610）若采用二路归并排序方法对关键字序列{25， 9， 78， 6， 65， 15，58， 18， 45， 20}进行升序排序， 写出其每趟排序结束后的关键字序列。

答案：

初始态： [25] [9] [78] [6] [65] [15] [58] [18] [45] [20]

第一趟： [9 25] [6 78] [15 65] [18 58] [20 45]

第二趟： [6 9 25 78] [15 18 58 65] [20 45]

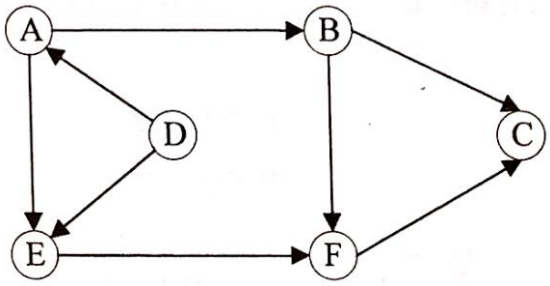
第三趟： [6 9 15 18 25 58 65 78] [20 45]

第四趟： [6 9 15 18 20 25 45 58 65 78]

中频知识点：

一、图的存储结构

1、（1810）写出下图所示的的有向图邻接矩阵表示和所有拓扑排序序列。

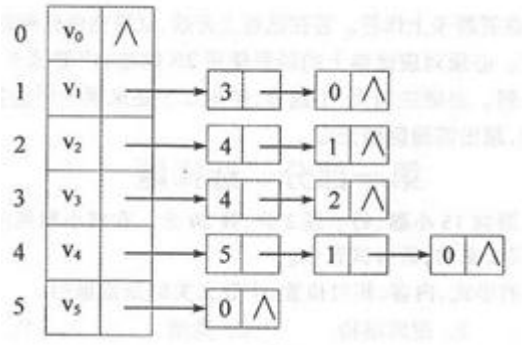


答案：

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

拓扑序列：DABEFC;DAEBFC

2、（1710）设有向图的邻按表表示如题图所示,



请给出每个顶点的入度和出度。

答案：

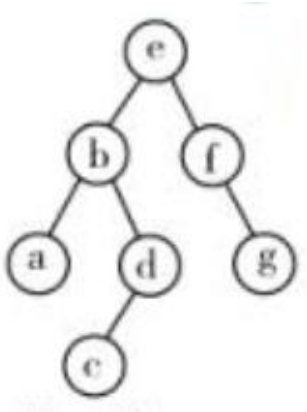
顶点	V_0	V_1	V_2	V_3	V_4	V_5
入度	3	2	1	1	2	1
出度	0	2	2	2	3	1

二、二叉排序树

1、（1610）设有一组关键字值序列{e， b， d， f， a， g， c}。 现要求：

（1）根据二叉排序树的创建方法构造出相应的二叉排序树(关键字值的大小按字母表顺序计)；

答案：



（2）计算等概率情况下在该二叉排序树上查找成功的平均查找长度 ASL。

答案：

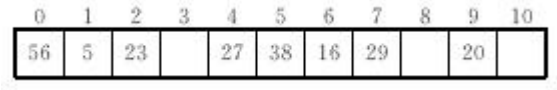
ASL=(1×1+2×2+3×3+1×4)/7=18/7。

三、散列表

1、（1710）已知散列表的地址空间为 0~10,散列函数为 H(key)= key mod 11(mod 表示求余运算)，采用二次探测法解决冲突：

（1）试用键值序列 20,38,16,27,5,23,56,29 建立散列表。

答案：



（2）计算出等概率情况下查找成功的平均查找长度。

答案：

等概率情况下查找成功的**平均查找长度=（1+1+2+3+5+2+3+1）/11=18/11**

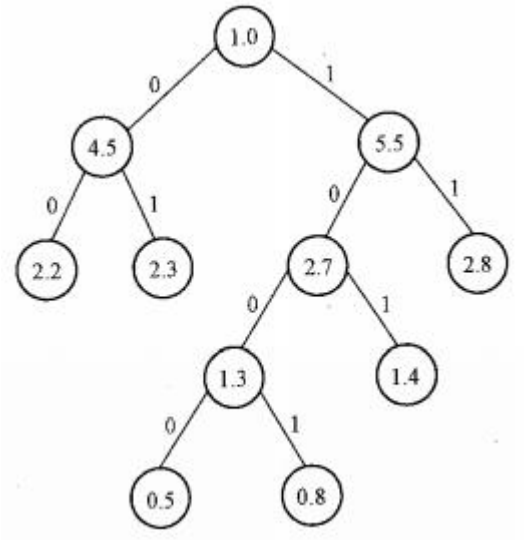
低频知识点：

一、判定树和哈夫曼树

1、设某通信系统中一个待传输的文本有 6 个不同字符，它们的出现频率分别是 0.5，0.8，1.4, 2.2，2.3，2.8，试设计哈夫曼编码。

答案：

以这些频率作为权值，构造一棵哈夫曼树，并对其进行哈夫曼编码，



出现频率为 **0.5** 的字符编码为 **1000**

出现频率为 **0.8** 的字符编码为 **1001**

出现频率为 **1.4** 的字符编码为 **101**

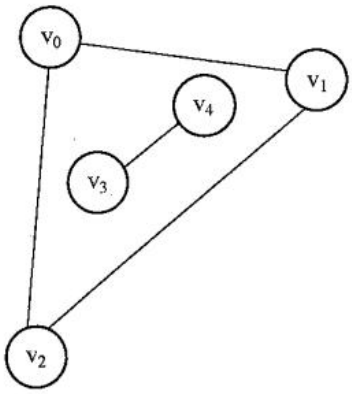
出现频率为 **2.2** 的字符编码为 **00**

出现频率为 **2.3** 的字符编码为 **01**

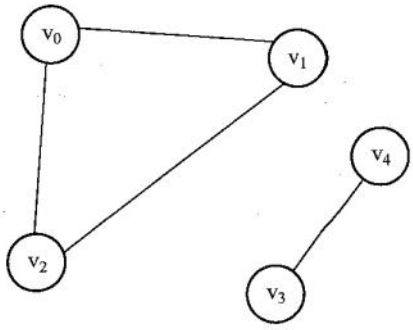
出现频率为 **2.8** 的字符编码为 **11**

二、图的遍历

1、利用连通分量计算算法求出图中非连通图连通分量的个数。



答案：



连通分量 1 包含以下顶点：**012**

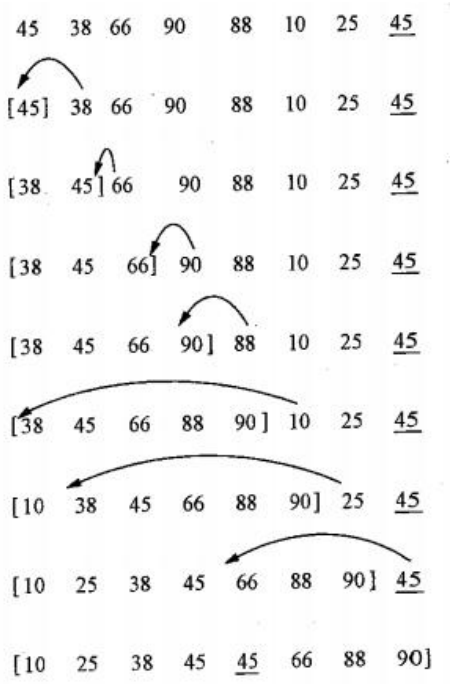
连通分量 2 包含以下顶点：**34**

共有 2 个连通分量。

三、插入排序

1、应用直接插入排序方法，对序列{45, 38, 66, 90, 88, 10, 25, 45}进行排序。

答案：



四、交换排序

1、用快速排序方法对 45 38 66 90 88 10 25 45 进行排序。

答案：

初始关键字	[45	38	66	90	88	10	25	45]
第一趟排序之后	[25	38	10]	**45**	[88	90	66	45]
分别对子序列排序	[10]	**25**	[38]	45	[45	66]	**88**	[90]
	[10]	**25**	[38]	45	**45**	[66]	**88**	[90]
有序序列	10	25	38	45	45	66	88	90

2、对键值 45 38 66 90 88 10 25 45 冒泡排序。

答案：

初始键值序列	45	38	66	90	88	10	25	45
第一趟排序后	38	45	66	88	10	25	45	**90**
第二趟排序后	38	45	66	10	25	45	**88**	**90**
第三趟排序后	38	45	10	25	45	**66**	**88**	**90**
第四趟排序后	38	10	25	45	**45**	**66**	**88**	**90**
第五趟排序后	10	25	38	**45**	**45**	**66**	**88**	**90**
第六趟排序后	10	25	**38**	**45**	**45**	**66**	**88**	**90**
第七趟排序后	10	**25**	**38**	**45**	**45**	**66**	**88**	**90**

算法设计题汇总

高频知识点：

一、线性表的顺序存储

1、（1810）假定线性表的数据元素的类型为 Data Type，顺序表的结构定义如下：

```
const int Maxsize=100;
typedef struct
{       Data Type data[Maxsize];
        int length;
}SeqList;
SeqList L;
```

设计算法实现顺序表的插入运算 InsertSeqlist(SeqList L,Data Type x, int i)。该算法是指在顺序表的第 I（1≤i≤n+1）个元素之前，插入一个新元素 x。使长度为 n 的线性表

$(a_1, a_2, \ldots, a_{i-1}, a_i, \ldots, a_n)$ 变为长度为 n+1 的线性表 $(a_1, a_2, \ldots, a_{i-1}, x, a_i, \ldots, a_n)$。

答案：

```
void InsertSeqlist(SeqList L,DataType x, int i)
{   //将元素 x 插入到顺序表 L 的第 i 个数据元素之前
    if (L.length==Maxsize)   exit("表已满");
    if (i<1||i>L.length+1)   exit("位置错");//检查插入位置是否合法
    for(j=L.length;j>=i;j--)                  //初始 i=L.length
      L.data[j]=L.data[j-1];                  //依次后移
    L.data[i-1]=x;                            //元素 x 置入到下标为 i-1 的位置
    L.length++;                               //表长度加 1
}
```

二、循环链表

1、（1610）某电商有关手机的库存信息， 按其价格从低到高存储在一个带有头结点的单循环链表中， 链表中的结点由品牌型号(nametype)、 价格(price)、 数量(quantity)和指针(next)四个域组成。 现新到 m 台、 价格为 c、 品牌型号为 x 的新款手机需入库：

（1）写出相应的存储结构

答案：

存储结构为：

```
typedef struct node {
    char * nametype; float price; int quantity;
    struct node * next; } Node, * LinkedList;
```

（2）写出实现该存储结构的算法。

答案：

实现算法为：

```
void InsertData(LinkedList head, char *x, int m, float c)
{ new=(Node *)malloc(sizeof(Node));
  new->nametype=x;new->price=c;new->quantity=m;new->next=Null;

  q=head;p=head->next;  //假设指针 q 指向 p 所指结点的前驱
  while (p! =head && p->price<c) {q=q->next;p=p->next;}
  new->next=p;q->next=new;return;}
```

三、数组

1、（1710）设有一 n 阶方阵 A,设计算法实现对该矩阵的转置。

答案：

```
void MM(int A[n][n])
{    int i,j,temp;
     for(i=0;i<n;i++)
     for(j=0;j<i;j++)
        {  temp=A[i][j];
           A[i][j]=A[j][i];
           A[j][i]=temp;
        }
}
```

四、二叉树的存储结构

1、（1810）已知二叉链表的类型定义如下：

```
typedef struct btnode
{        DataType data;
          struct btnode *lchild,*rchild;
} *BinTree;
```

以二叉链表作存储结构 ,试编写求二叉树叶子结点个数的算法 leafnode_num(BinTree bt)。

答案：

```
int leafnode_num(BinTree bt)
{
    if(bt == NULL) return 0;
    else
        if(bt->lchild == NULL) && (bt->rchild == NULL)
            return 1;
        else
            return leafnode_num(bt->lchild)+leafnode_num(bt->rchild
}
```

五、二叉树的遍历

1、（1710）已知二叉链表的类型定义如下：

```
typedef struct btnode
{   DataType data;
    struct btnode *lchild, *rchild;
} *BinTree;
```

假定 visit(bt)是一个已定义的过程,其功能是访问指针 bt 所指结点,设计递归算法 preorder(BinTree bt)实现在二叉链表上的先序遍历。

答案：

```
void preorder(BinTree bt)
{   if(bt! =NULL)
    {  visit(bt);
       preorder(bt->lchild);
       preorder(bt->rchild);
    }
}
```

2、（1704）已知二叉链表的类型定义如下：

typedef struct btnode

{DataType data；

struct btnode*lchild，*rchild；

}*BinTree；

利用二叉树遍历的递归算法，设计求二叉树的高度的算法 Height(BinTree bt)。

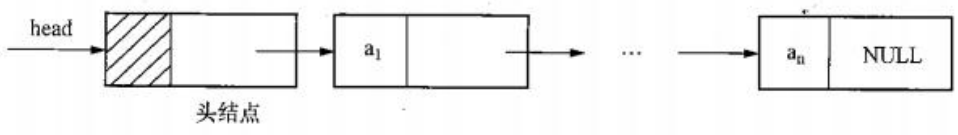
答案：

```
int  Height(BinTree bt)
{
    int  lh,rh;                                  //初始左右子树的高度为 0
    if (bt==NULL)  return 0;
    else
       { lh=Height(bt->lchild);                  //左子树的高度
         rh=Height(bt->rchild);                  //右子树的高度
         return  1+(lh>rh ? lh : rh);
       }
}
```

中频知识点：

一、栈

1、写一个算法，借助栈将图所示的带头结点的单链表逆置。



答案：

```
void  ReverseList(LkStk  *head)
 {    LkStk  *S;
      DataType x;
      InitStack(S);                          //初始化链栈
      p=head->next;
      while(p!=NULL)                         //扫描链表
      {   Push(S,p->data);                   //元素进栈
          p=p->next;
      }
      p=head->next;
      while (!EmptyStack(S))                 //栈不为空时
      {   p->data=Gettop(S);                 //元素填入单链表中
          Pop(S);                            //出栈
          p=p->next;
      }
 }
```

二、图的存储结构

1、（1610）写出向存储结构为邻接矩阵的无向图 G 中插入一条边(x， y)的算法。 算法的头函数为： void AddEdgetoGraph(Graph *G， VertexType x， VertexType y)，无向图 G 的存储结构为：

```
#define MaxVertex num
typedef char VertexType;
typedef int EdgeType;
typedef struct graph {
    int n, e; //图的实际顶点数和边数
    EdgeType edge [MaxVertex][MaxVertex]; //邻接矩阵
    VertexType vertex[MaxVertex]; //顶点表
} Graph;
```

答案：

```
void AddEdgetoGraph(Graph *G, VertexType x, VertexType y)
  { i=-1;j=-1;
    for (k=0;k<G->n;k++)   //查找 x,y 的编号
      { if(G->vertex[k]==x)i=k;
        if (G->vertex[k]==y)j=k;}
    if (i==-1||j==-1)Error("结点不存在");
    else {   //插入边(x,y)
        G->edge[i][j]=1;G->edge[j][i]=1;G->e++;}
  }
```

三、静态查找表

1、（1704）设计一个算法实现以下功能：在整型数组 A[n]中查找值为 k 的元素，若找到，贝!1 输出其位置 i(0≤i≤n-1)，否则输出-1 作为标志。

答案：

```
int search(int A[], int n, int k)
{     int i;
      i = 0;
      while(i <= n-1)
      if(A[i] != k) i++;
      else break;
      if(i <= n-1) return i;
      else return -1;
}
```

低频知识点：

一、算法分析

1、设计算法在整型数组 A[n]中查找值为 K 的元素，若找到，则输出其位置 i(0≤i ≤n-1),否则输出-1 作为标志，并分析算法的时间复杂度。

答案：

```
int search(int A[], int n, int k)
  {  int i;
     i=0;
     while (i<=n-1)
     if  (A[i]!=k) i++;
     else  break;
     if (i<=n-1)  return  i;
     else  return -1;
  }
```

当查找成功时，A[i]与 k 比较次数≤n;当查找不成功时，A[i]与 k 比较 n 次，所以，算法时间复杂复杂度 T(n)＝O(n)。

二、线性表的顺序存储

1、编制函数求 1!+2! +…+n!。

答案：

```
int fact2(int n)
{  int  i,j,temp,s;
    s=0;  temp=1;
    for (i=1;i<=n;i++)
    {  temp=temp*i;
       s=s+temp;
    }
    return  s;
}
```

2、读入 n= 100 个整数到一个数组中，写出实现将该组数进行逆置的算法，并分析算法的空间复杂度。

答案：

```
void  f1( int a[],int n)
{   int  i,temp;
    for(i=0;i<=n/2-1;i++)
     {   temp=a[i]
         a[i]=a[n-1-i];
         a[n-1-i]=temp
     }
}
```

f1 所需要的辅助变量为 2 个整型变量 **i 和 temp**,与问题的规模无关 ,其**空间复杂度为 0(1)。**

3、试以顺序表作存储结构，写一个实现线性表的就地(即使用尽可能少的附加空间)逆置的算法，在原表的存储空间内将线性表(a1,a2,...,an)逆置为(an,an-1,...,a1)。

答案：

```
void inverse_sqlist(SeqList L)
{
   int m,n,k;
   DataType temp;
   m=0; n=L.length-1;
   while (m<n)
   { temp=L.data[m];
     L.data[m]=L.data[n];
     L.data[n]=temp;
     m++;
     n--;
   }
}
```

三、线性表的链接存储

1、已知单链表 L 是一个递增有序表，试写一高效算法，删除表中值大于 min 且小于 max 的结点，同时释放被删结点的空间，这里 min 和 max 是两个给定的参数。

答案：

```
void  LinkList_Del_ between_MinMax(LinkList  L, int  min, int  max)
{
     p=L->next;
     q=L;
     while (p!=NULL)
      { if  ( (p->data>min ) && (p->data<max ) )
        { q->next=p->next ;
          free(p);
          p=q->next ;
        }
       else
        {  q=p;
           p=p->next;
        }
      }
}
```

四、图的存储结构

1、给出建立有向图的邻接表的算法。

答案：

```
CreateAdjlist(Graph *g)
 {   int  n,e,i,j,k;
     ArcNode   *p;
     scanf("%d  %d",n,e);                          //读入顶点数和边数
     g->vexnum=n;g->arcnum=e;
     for (i=0;i<n;i++)
      {   g->adjlis[i].vertex=i;                   //初始化顶点 vi 的信息
          g->adjlis[i].firstarc=NULL;              //初始化 i 的第一个邻接点为 NULL
      }
     for (k=0;k<e;k++)                            //输入 e 条弧
      { scanf("%d  %d",&i,&j);
        p=(ArcNode*)malloc(sizeof(ArcNode));//生成 j 的表结点
        p->adjvex=j;
        p->nextarc=g->adjlis[i].firstarc;  //将结点 j 链到 i 的单链表中
        g->adjlis[i].firstarc=p;
      }
 }
```

五、图的遍历

1、若图的存储结构为邻接表，队采用链队列，给出相应的广度优先搜索算法。

答案：

```
Bfs(Graph g,int v)
 {   LkQue  Q;                                  //Q 为链队列
     ArcNode  *p;
     InitQueue(&Q);
     printf("%d",v);
     visited[v]=1;                              //置已访问标志
     EnQueue(&Q,v);                             //访问过的顶点入队列
     while (!EmptyQueue(Q))
      {  v=Gethead(&Q);
         OutQueue(&Q);                          //顶点出队列
         p=g.adjlist[v].firstarc;               //找到 v 的第一个邻接点
         while(p!=NULL )                        //判断邻接点是否存在
          {  if (!visited[p->adjvex])           //邻接点存在未被访问
               {  printf("%d",p->adjvex);
                  visited[p->adjvex]=1;         //置已访问标志
                  EnQueue(&Q, p->adjvex); //邻接点入队列
               }
             p=p->nextarc;                      //沿着 v 的邻接点链表顺序搜索
          }
      }
 }
```

六、图的应用

1、给出拓扑排序算法。

答案：

```
Tp_Sort(Graph g)
{   建立图 g 中入度为 0 的顶点的栈 S;
    m=0;                                        //m 记录输出的顶点个数
    while (!EmptyStack(S))                      //当栈非空
     {    Pop(S,v) ;                            //弹出栈顶元素,赋给 v
          输出 v;
          m ++;
          w=Firstvex(g,v);                      //图 g 中顶点 v 的第一个邻接点
          while (w存在)
           {  w 的入度减 1;
              if (w 的入度==0)
                  push(s,w);
              Nextvex(g,v,w);                   //图 g 中顶点 v 的下一个邻接点
           }
     }
    if (m < n )  printf("图中有环\n");
}
```