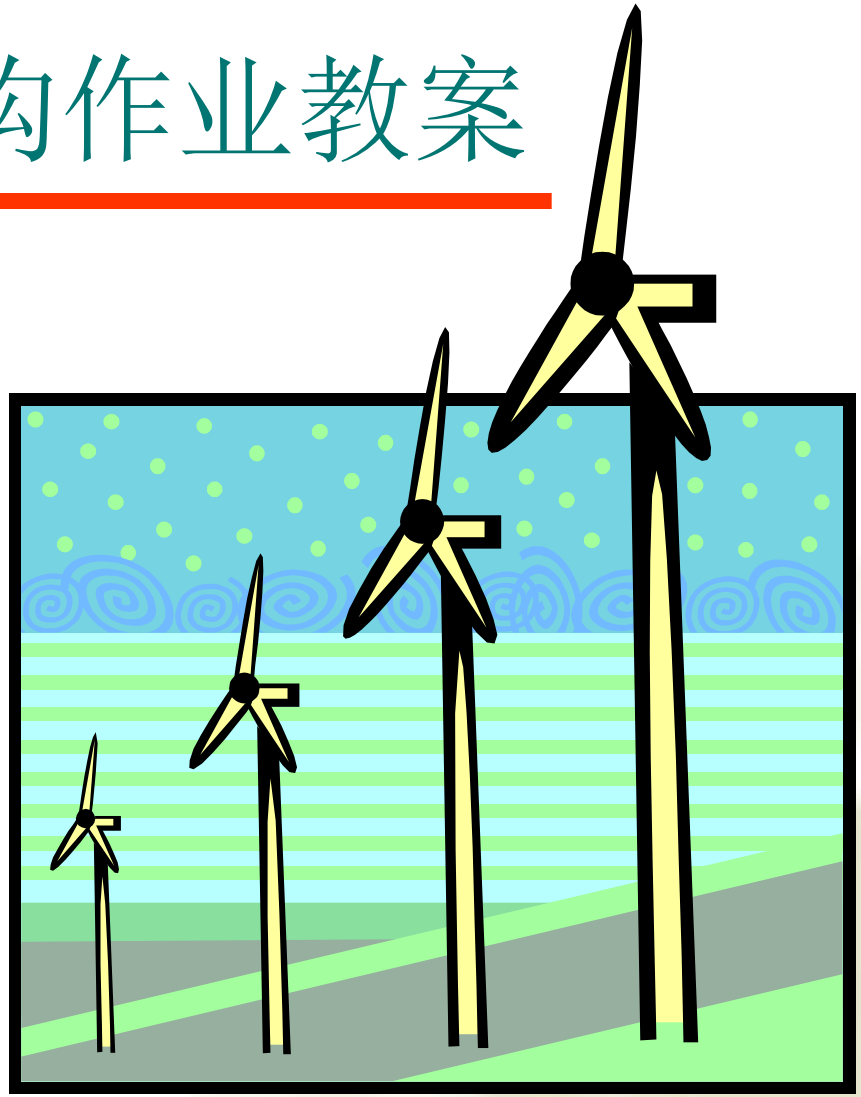
数据结构作业教案



目录

第六章 树和二叉树

第七章 图

第八章 动态存储管理

第九章 查找

第九章 内部排序

退出播放

第一章 绪论

④ 1.8 设n为正整数。试确定下列各程序段中前置以记号@的语句的频度：

（2）i=1；k=0;

```
do{
 @   k+=10*i;
     i++;
}while(i<=n-1);
```

核对本页答案

（4）k=0;

```
for (i=1;i<=n;i++){
        for( j=i;j<=n;j++)
        @   k++;
}
```

（6）i=1 ； j=0;

```
while(i+j<=n){
 @   if(i>j)  j++;
        else i++;
}
```

核对本页答案

（8）x=91; y=100;

```
while(y>0){
@ if(x>100) {x-=10;y--;}
    else x++;
 }
```

核对本页答案

③ 1.9 假设n为2的乘幂，并且n>2，试求下列算法的时间复杂度及变量count的值（以n的函数形式表示）

```
int  Time（int  n）{
      count =0；  x=2;
      while(x<n/2){
           x *=2;  count ++;
        }
       return  (count)
}//Time
```

核对本页答案

第一章 绪论答案

④ 1.8

(2) $\begin{cases} n & n=1 \\ n-1 & n>1 \end{cases}$

(4) $n+(n-1)+\cdots\cdots+1+0=n(n+1)/2$

(6) n

(8) 1100

继续答题

继续答题

继续答题

继续答题

③ 1.9

$T(n)= O(\log_2 n)$

$count=\log_2 n-2$

继续答题

第二章 线性表

② 2.6 已知L是无表头结点的单链表，且P结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。

a. 在P结点后插入S结点的语句序列是___。

b. 在P结点前插入S结点的语句序列是___。

c. 在表首插入S结点的语句序列是_____。

d. 在表尾插入S结点的语句序列是_____ 。

1. P->next = S;
2. P->next = P->next->next ;
3. P->next = S ->next ;
4. S ->next = P->next ;
5. S ->next = L;
6. S ->next = NULL;
7. Q = P;
8. while(P->next != Q) P = P->next ;
9. while(P->next != NULL) P = P->next ;
10. P = Q;
11. P = L;
12. L = S;
13. L= P;

核对本页答案

② 2.7 已知L是带表头结点的单链表，且P结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。

a. 删除P结点的直接后继结点的语句序列是_____。

b. 删除P结点的直接前驱结点的语句序列是_____。

c.删除P结点的语句序列是_____。

d.删除首元结点的语句序列是_____。

e.删除表尾结点的语句序列是_____。

1. P = P->next ;
2. P->next = P ;
3. P->next = P->next->next ;
4. P = P->next->next ;
5. while(P!= NULL) P = P->next ;
6. while(Q->next != NULL) { P = Q; Q =Q->next ;}
7. while(P->next != Q) P = P->next ;
8. while(P->next ->next != Q) P = P->next ;
9. while(P->next ->next != NULL) P = P->next ;
10. Q = P;
11. Q = P->next ;
12. P = L;
13. L= L >next ;
14. free(Q);

核对本页答案

② 2.8 已知P结点是某双向链表的中间结点，试从下列提供的答案中选择合适的语句序列。

a. 在P结点后插入S结点的语句序列是____。

b. 在P结点前插入S结点的语句序列是____。

c. 删除P结点的直接后继结点的语句序列是_____。

d. 删除P结点的直接前驱结点的语句序列是_____。

e.删除P结点的语句序列是_____。

```
1.  P->next = P->next->next  ;
2.  P->priou= P-> priou -> priou  ;
3.  P->next = S;
4.  P->priou = S;
5.  S->next = P;
6.  S-> priou = P;
7.  S->next = P->next ;
8.  S-> priou = P->priou ;
9.  P->priou->next  = P->next ;
10. P->priou->next  = P;
11. P-> next ->priou  = P ;
12. P-> next ->priou  = S ;
13. P-> priou -> next  = S ;
14. P-> next ->priou  = P->priou ;
15. Q = P->next  ;
16. Q = P->priou ;
17. free(P);
18. free(Q);
```

核对本页答案

② 2.9 简述一下算法的功能。

```
LinkList Demo(LinkList L){ // L 是无头结点单链表
    ListNode *Q,*P;
    if(L&&L->next){
        Q=L; L=L->next;  P=L;
        while (P->next) P=P->next;
        P->next=Q;  Q->next=NULL;
    }
    return L;
}// Demo
```

② 2.9 简述一下算法的功能。

```
void BB(LNode *s, LNode *q){
    p = s;
    while (p->next !=q) p =p ->next;
    p->next = s;
}//BB
void AA(LNode *pa, LNode *pb){
    //pa和pb分别指向单循环链表中的两个结点
    BB(pa,pb);
    BB(pb,pa);
}//AA
```

第二章 线性表答案

② 2.6

a. 4 1

b. 7 11 8 4 1

c. 5 12

d. 11 9 1 6

继续答题

② 2.7

a. 11 3 14

b. 10 12 8 11 3 14

c. 10 12 7 3 14

d. 12 11 3 14

e. 12 9 11 3 14

继续答题

② 2.8

a. 7 12 3 6

b. 5 13 8 4 或 13 8 5 4

c. 15 1 11 18

d. 16 2 10 18

e. 9 14 17

继续答题

② 2.9

(1) 如果L的长度不小于2，则第1个结点删去插到表尾

（2）构造2个单循环链表

继续答题

② 2.11

```
Status Insert_SqList(SqList &va,int x)
                        //把x插入递增顺序表va中
{
  if  (va.length+1>va.listsize) return ERROR;
        va.length++;
  for (i=va.length-1;va.elem[i]>x&&i>=0;i--)
    va.elem[i+1]=va.elem[i];
  va.elem[i+1]=x;
  return OK;
}//Insert_SqList
```

继续答题

③ 2.21

- 算法一：

```
void reverse(SqList &A ){
    //顺序表的就地逆置
    for(i=0,j=A.length-1; i<j; i++,j--)
        A.elem[i]<->A.elem[j];
        return OK;
}//reverse
```

- 算法二：

```
Status  reverse(SqList &A ){
                //顺序表的就地逆置
    for(i=0;i<=A.length/2-1; i++)
        A.elem[i]<->A.elem[A.length-i-1];
    return OK;
}//reverse
```

继续答题

② 2.13(模仿p26算法2.6)

```
Status Locate(LinkList &L,ElemType x){
    //返回单链表L中第1个与X满足函数compare()
    //判定关系的元素的位置。L为带头结点的单链
   //表，若不存在返回NULL
    P=L->next;//p指向第一个结点或表空p=NULL
    while(p&&!(*compare)(p->data,x))
        p=p->next;
    return p;
}//Locate
```

继续答题

③ 2.23

```
void MergeList(LinkList &A,LinkList &B,LinkList &C){
    //把链表A和B合并为C,A和B的元素间隔排列,且使用原存
    //储空间,ha、hb、hc分别指向头结点
      pa=ha->next; pb=hb->next;pc=hc=ha;
      while(pa&&pb){
            pc->next=pa;pc=pa;pa=pa->next;
           pc->next=pb;pc=pb;pb=pb->next;
      }//while
      while(!pa)  pc->next=pb;
     while(!pb)  pc->next=pa;
     free(hb);
     return  hc;
}//MergeList
```

继续答题

② 3.4

(1)利用辅助数组A把栈S的各个数据元素逆置

（2）利用辅助栈T，删除栈中为e的数据元素

② 3.12

char

错误(1)：c, h, a, r

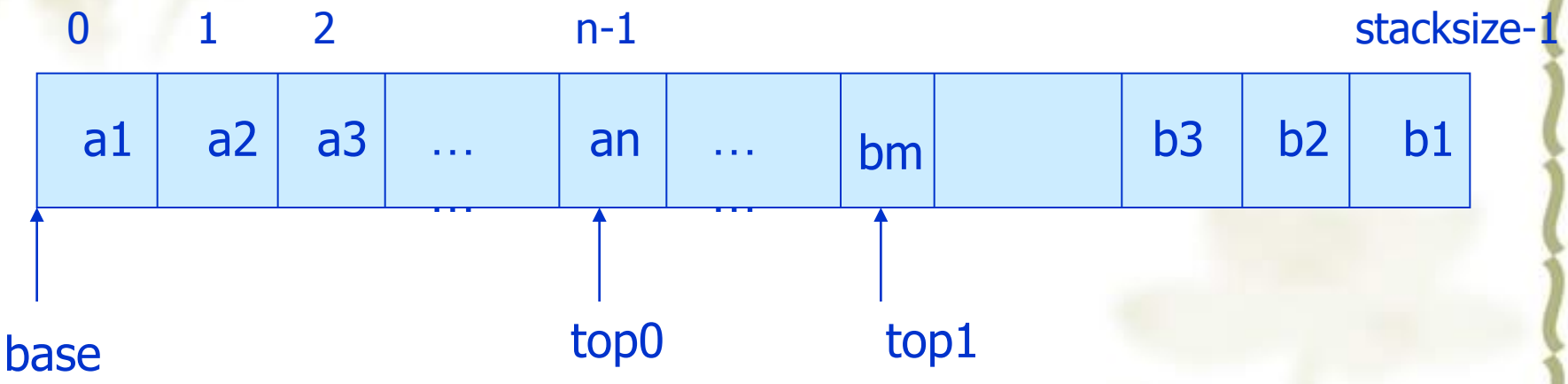
错误(2)：c

h

a

r

③ 3.15



双向栈

```
#define StackSize 100;
typedef struct {
      SElemType  *base;
      SElemType  *top0;
      SElemType  *top1;
}TSqStack;

Status InitStack(TSqStack &tws){
   //构造一个空的双向栈
   if(!(tws.base=(SElemType*)malloc(StackSize*sizeof
                    (SElemType))))
    exit(OVERFLOW);
    tws.top0=tws.base;
    tws.top1=tws.base[StackSize-1];
    return OK;
} //InitStack
```

```
Status  Push(TSqStack &tws,  int I , SElemType x){
    //将元素x压入双向栈tws的第i个栈中，i=0，1
    if (tws.top0==tws.top1+1)  exit(OVERFLOW);
    if (i==0) *tws.top0++=x;
    else if (i==1) *tws.top1--=x;
    else return ERROR;
  return OK;
}//Push
```

```
SElemType Pop(TSqStack &tws, int i){
//若双向栈tws的第i个栈不空，则删除第i个栈的栈顶元素
    并返回其值，否则返回空元素
  if (i!=0||i!=1) return NULL;
  if (i==0&&tws.top0) return *--tws.top0;
  if (i==1&&tws.top1!=tws.base[StackSize-1]) return
    *++tws.top1;
  return NULL;
} //Pop
```

② 3.28

```
typedef struct CQNode {
        CQElemType      data;
        struct CQNode  *next;
}CQNode, *CQueuePtr;

typedef struct {
         CQueuePtr  rear;
}CLinkQueue;
```

```
Status DeCQueue(CLinkQueue &L, CQElemType &e){
  //若队列不空，则删除L的队头元素，用e返回其值
  //并返回OK，否则返回ERROR
  if (L.rear=L.rear->next) return ERROR;
  p=L.erar->next->next;
  e=p->data;  L.rear->next->next=p->next;
  if (L.rear==p) L.rear=L.rear->next;
  free(p);
  return OK;
} //DeCQueue
```

```
Status InitCQueue( CLinkQueue &L) {
 //初始化一个只有尾指针的带头结点的循环链表表示的队列
if (!(L.rear=(CQueuePtr)malloc(sizeof(CQNode)))
    exit(OVERFLOW);
L.rear->next=L.rear;
return OK;
} // InitCQueue

Status EnCQueue(CLinkQueue &L, CQElemType e){
//插入元素e为cq的新的队尾元素
if (!(p=(CQueuePtr)malloc(sizeof(CQNode)))) exit(OVERFLOW);
p->data=e;  p->next=L.rear->next;
L.rear->next=p; L.rear=p;
return OK;
}//EnCQueu
```

第四章　串

① 4.5

t=THESE　ARE　BOOKS

v=YXY

u=XWXWXW

② 4.6

```
SubString(a,s,3,1);      //Y
SubString(b,s,6,1);      //+
SubString(c,s,7,1);      //*
StrAssign(t,replace(s,a,b));
replace(t,Concat(u,b,c),Concat(v,c,a));
```

第5章 数组与广义表

① 5.1

（1）6*8*6=288

（2）1000+（5*8+7）*6=1282

（3）1000+（1*8+4）*6=1072

（4）1000+（7*6+4）*6=1276

③ 5.5

$$\begin{pmatrix} a_{11} & a_{12} & \cdots\cdots & a_{1n} \\ & a_{22} & \cdots\cdots & a_{2n} \\ & & a_{ij}\cdots\cdots a_{ij}\cdots\cdots & \\ 0 & & & \\ & & & a_{nn} \end{pmatrix}_{n*n}$$

如图所示

在第i行前共需存放存放

n+（n-1）+（n-2）+……+[n-(i-2)]

=[n+n-(i-2)(i-1)](i-1)/2个

另外，在第i行除aij外需要存放j-1个

因此aij在一维数组的存放位置的下标为：

k= [n+n-(i-2)(i-1)](i-1)/2+j-i-1

=(n+1/2)*i-i^2/2+j-n-1

=$f_1(i)+f_2(j)+C$

(k=0,……，n(n+1)/2-1)

所以 f1(i)= (n+1/2)*i-i2/2

f2(j)=j C=-(n-1)

② 5.6

解答一：特殊情形

数组B中的三行分别对应（a_{ij}）$_{n*n}$的三条对角线上的元素

$$\begin{pmatrix} \text{无} & a12 & a23 & \cdots\cdots & a(n-2)(n-1) & a(n-1)n \\ a11 & a22 & a33 & \cdots\cdots & ann & \\ a21 & a32 & & \cdots\cdots & an(n-1) & \text{无} \end{pmatrix}_{3*n}$$

可得 $\begin{cases} u=i-j+1 \ (u=0,1,2) \\ v=j-1 \end{cases}$

解答二：

矩阵$(a_{ij})_{n*n}$下标从1开始，若把非0元存储到一个下标从0开始的一维数组A中，则a_{ij}在这个一维数组的下标为：

$k=3*(i-1)+2+j-i+1=2*i+j-3$

若$a_{ij}=A[k]=B[u][v]$（B数组从0开始）

则　$2*i+j-3=u*n+v$

所以　$u= [2*i+j-3/n]$　　$u=0,1,2$

　　$v=(2*i+j-3)\%n$

② 5.10

(1) p

(2) (k,p,h)

(3) (a,b)

(4) ((c,d))

(5) (c,d)

(6) (b)

(7) b

(8) (d)

第六章 树和二叉树

② 6.14

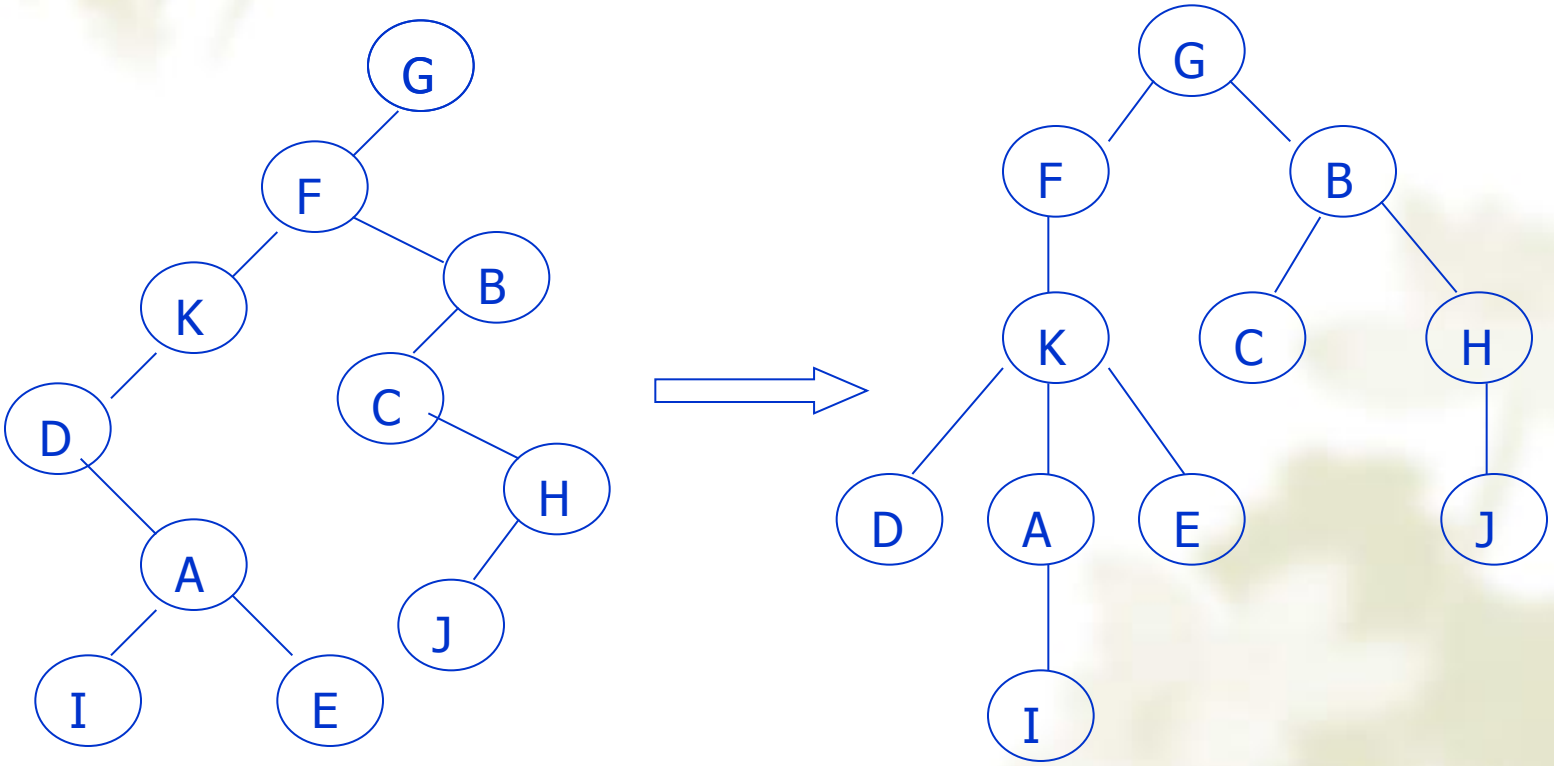
（a）空树，任一结点均无左孩子的非空二叉树

（b）空树，任一结点均无右孩子的非空二叉树

（c）空树，仅有一个结点的二叉树

② 6.23 原理： 树的先根次序==二叉树的先序

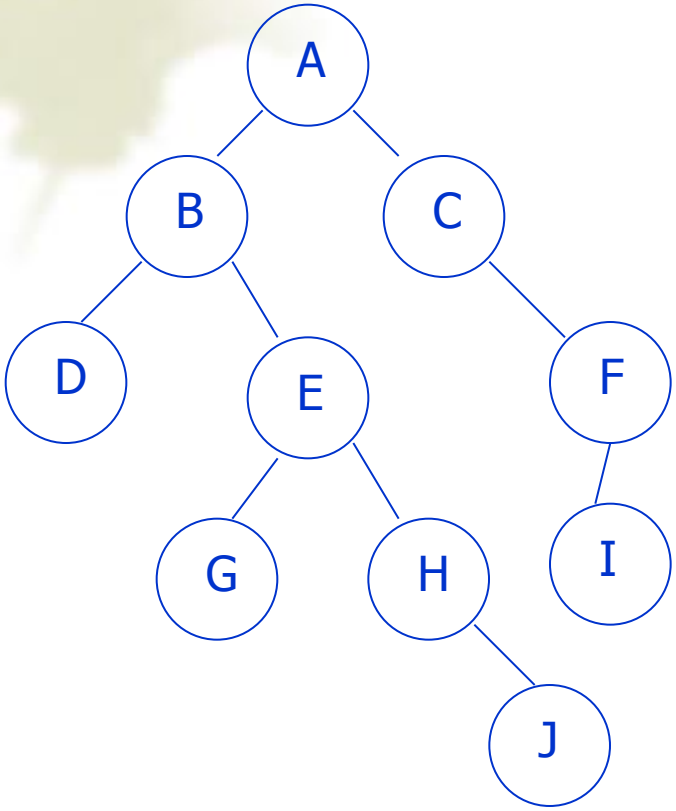
树的后根次序==二叉树的中序



相应二叉树

树

③ 6.29



③ 6.42

```
int LeafCount_BiTree(Bitree T){
  //求二叉树中叶子结点的数目
  if(!T) return 0; //空树没有叶子
  else if(!T->lchild&&!T->rchild) return 1; //叶子结点
  else return Leaf_Count(T->lchild)+Leaf_Count(T->rchild);
                                        //左子树的叶子数加上右子树的叶子数
}//LeafCount_BiTree
```

③ 6.43

```
void Bitree_Swap(Bitree T){
  //交换所有结点的左右子树
  T->lchild<->T->rchild; //交换左右子树
  if(T->lchild) Bitree_Swap(T->lchild);
  if(T->rchild) Bitree_Swap(T->rchild);
                          //左右子树再分别交换各自的左右子树
}//Bitree_Swap
```

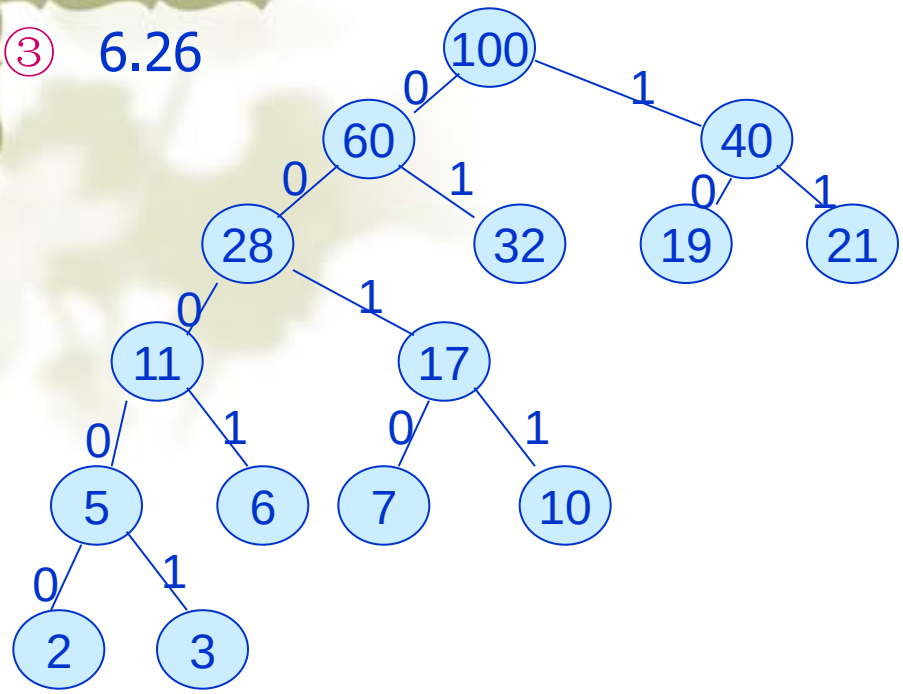
③ 6.60

```
int LeafCount_Tree(CSTree T){
//求孩子兄弟链表表示的树T的叶子数目
  if(!T->firstchild) return 1; //叶子结点
  else
  {
    count=0;
    for(child=T->firstchild;child;child=child->nextsibling)
      count+=LeafCount_Tree(child);
    return count; //各子树的叶子数之和
  }
}//LeafCount_Tree
```

④ 6.47

```
void LayerOrder_BiTree(Bitree T){
  //层序遍历二叉树
  if(T)
  {   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_BiTree
```

③ 6.26



频数	7	19	2	6	32	3	21	10
哈夫曼编码	0010	10	00000	0001	01	00001	11	011
等长编码	000	001	010	011	100	101	110	111

WRLHF=（4*7+2*19+5*2+4*6+2*32+5*3+2*21+3*10）*0.01/8=2.61

WRLEQ=(7+19+2+6+32+3+21+10)*0.01*3/8=3

结论：哈夫曼编码方式比等长编码方式提高了通信信道的利用率，提高了报文发送速度，节省了存储空间。

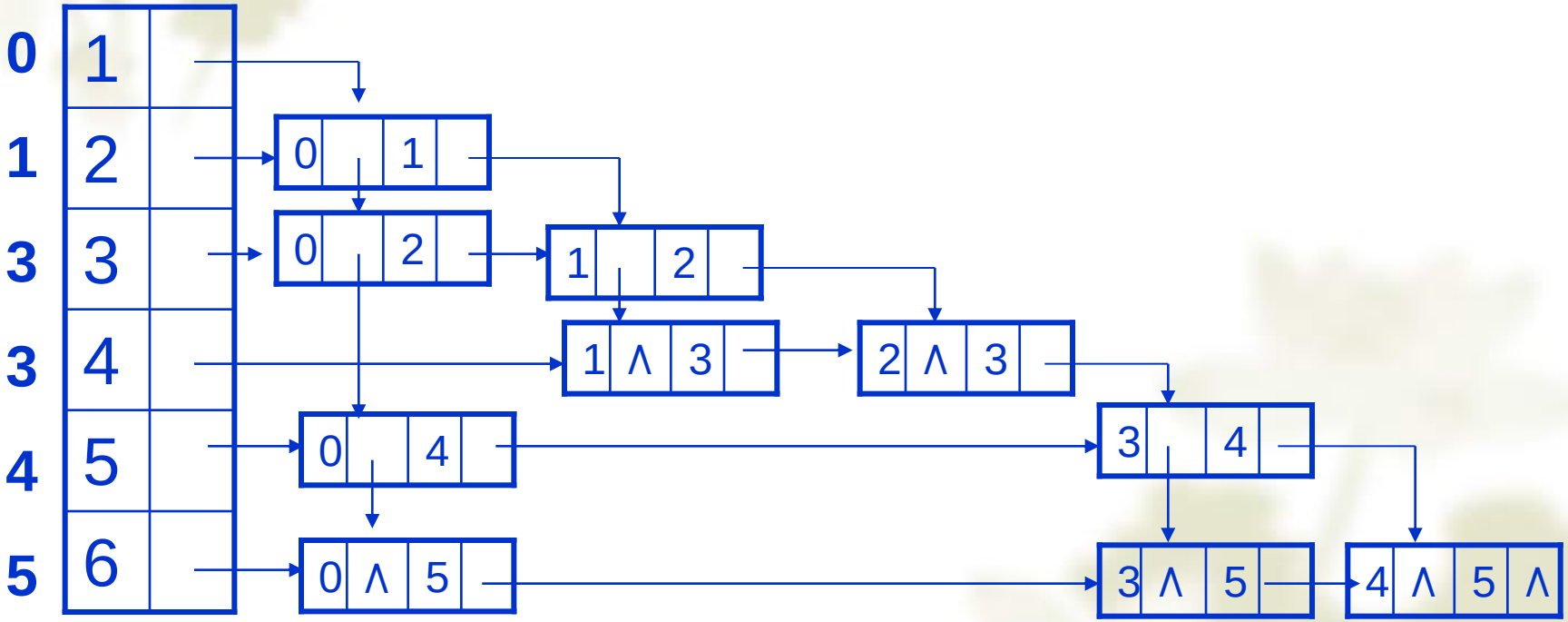
第七章 图

③ 7.14

```
Status Build_AdjList(ALGraph &G){
    //输入有向图的顶点数,边数,顶点信息和边的信息建立邻接表
    InitALGraph(G);

    scanf(“%d”,&v);
    if(v<0) return ERROR; //顶点数不能为负
   G.vexnum=v;

    scanf("%d",&a);
    if(a<0) return ERROR; //边数不能为负
    G.arcnum=a;

   for(m=0;m<v;m++)
       G.vertices[m].data=getchar(); //输入各顶点的符号
```

```
for(m=1;m<=a;m++)
{
    t=getchar();h=getchar(); //t为弧尾,h为弧头
    if((i=LocateVex(G,t))<0) return ERROR;
    if((j=LocateVex(G,h))<0) return ERROR;
                                        //顶点未找到
    p=(ArcNode*)malloc(sizeof(ArcNode));
    if(!G.vertices.[i].firstarc) G.vertices[i].firstarc=p;
    else
    {
       for(q=G.vertices[i].firstarc;q->nextarc;q=q->nextarc);
            q->nextarc=p;
    }
    p->adjvex=j;p->nextarc=NULL;
  }//for
 return OK;

}//Build_AdjList
```

② 7.3

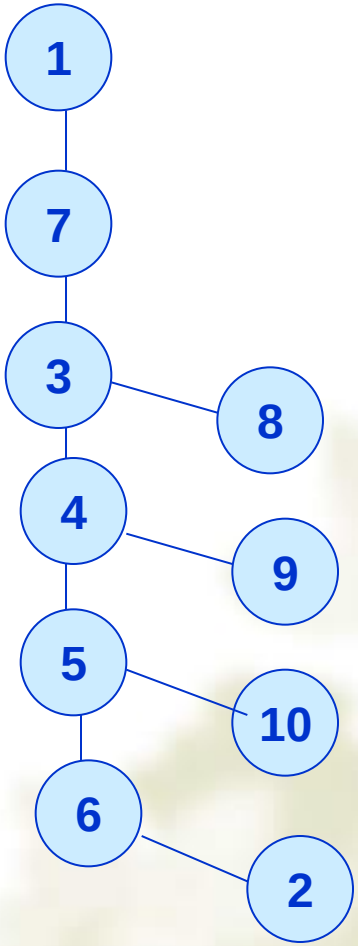


深度优先顶点序列：**1-2-3-4-5-6**

广度优先顶点序列：**1-2-3-5-6-4**

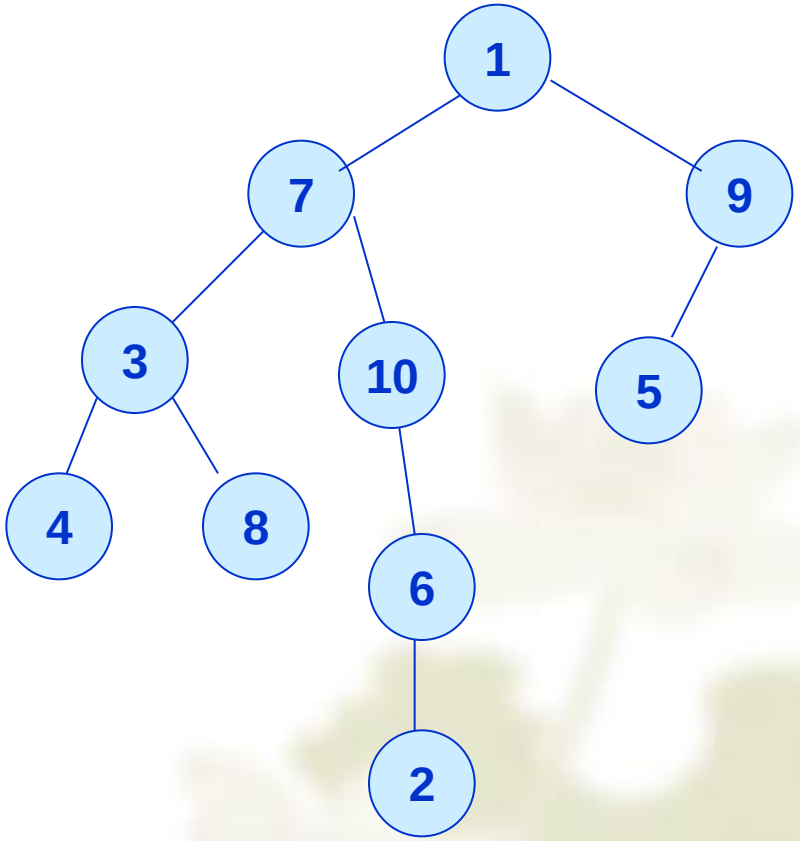
② 7.4

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



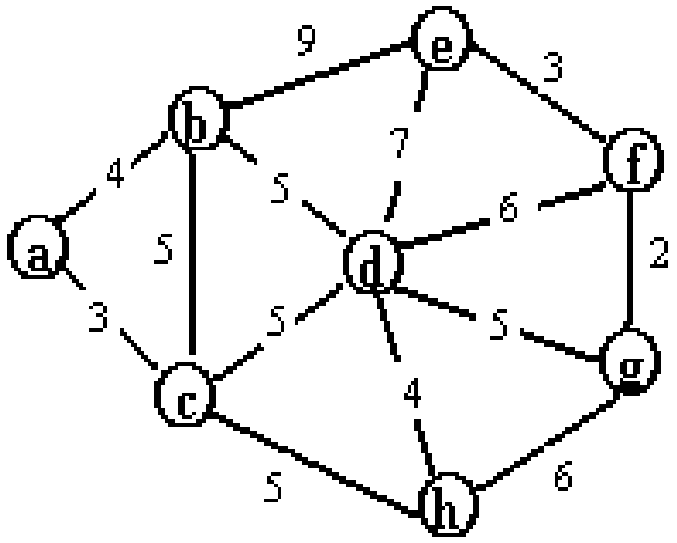
DFS

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



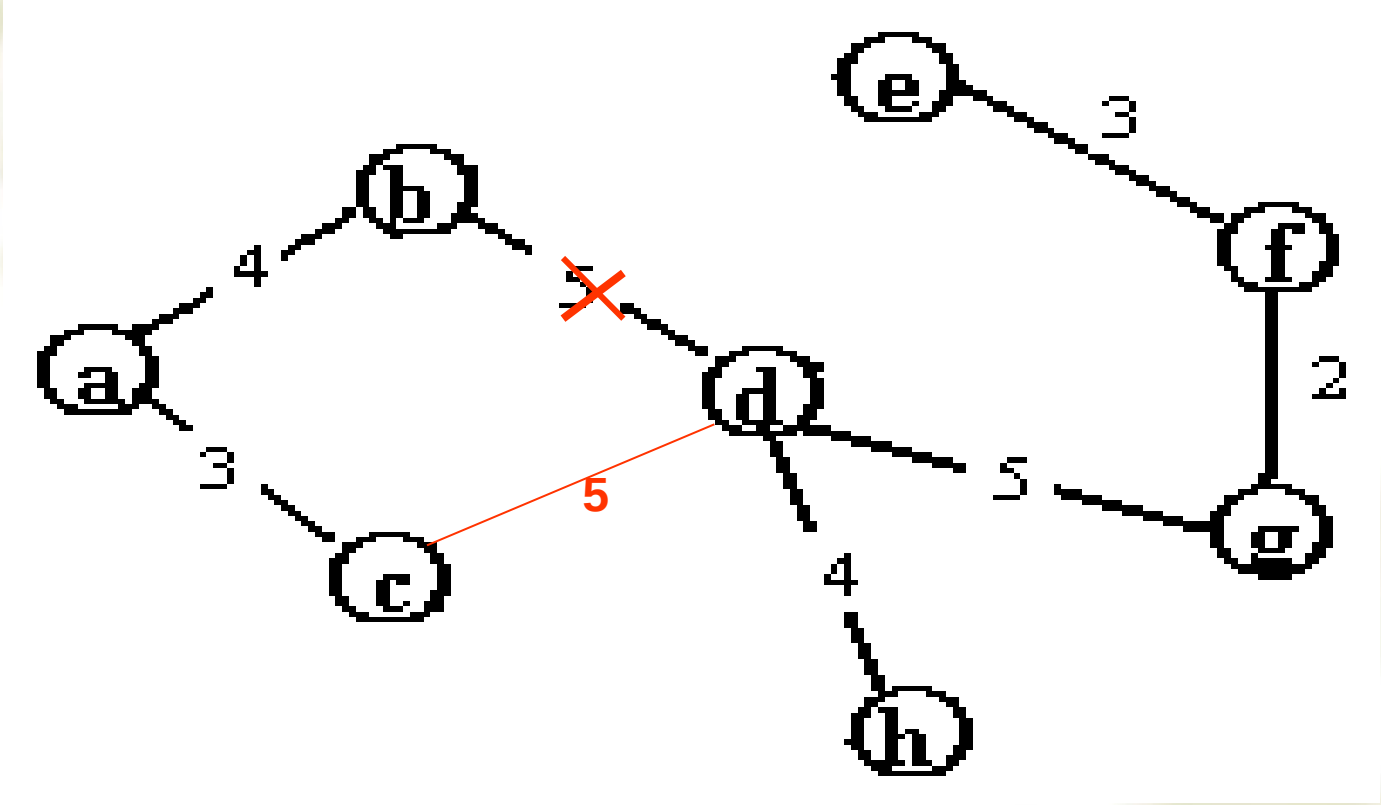
BFS

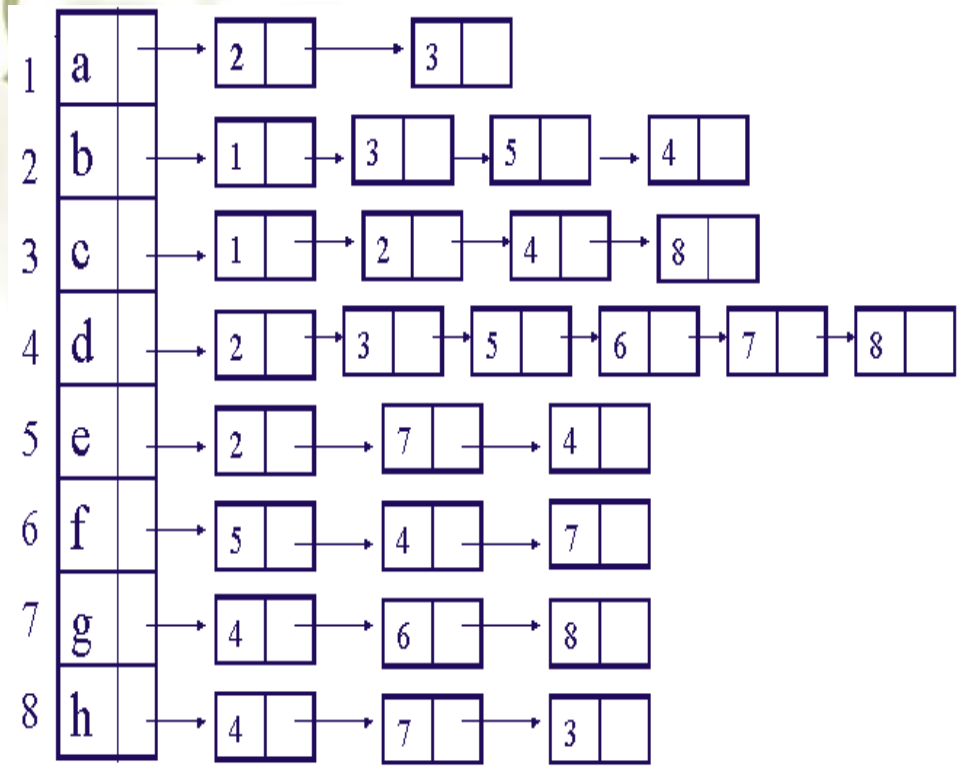
② 7.7



	a	b	c	d	e	f	g	h
a	∞	4	3	∞	∞	∞	∞	∞
b	4	∞	5	5	9	∞	∞	∞
c	3	5	∞	5	∞	∞	∞	5
d	∞	5	5	∞	7	6	5	4
e	∞	9	∞	7	∞	3	∞	∞
f	∞	∞	∞	6	3	∞	2	∞
g	∞	∞	∞	5	∞	2	∞	6
h	∞	∞	5	4	∞	∞	6	∞

	0 1 2 3 4 5 6 7	u	v-u	k
adjvex lowcost	a a a a a a a a 0 4 3 ∞ ∞ ∞ ∞ ∞	{a}	{c, b, d, h, g, f, e}	2
adjvex lowcost	a a a c a a a c 0 4 0 5 ∞ ∞ ∞ 5	{a, c}	{b, d, h, g, f, e}	1
adjvex lowcost	a a a c b a a c 0 0 0 5 9 ∞ ∞ 5	{a, c, b}	{d, h, g, f, e}	3
adjvex lowcost	a a a c d d d d 0 0 0 0 7 6 5 4	{a, c, b, d}	{h, g, f, e}	7
adjvex lowcost	a a a c d d d d 0 0 0 0 7 6 5 0	{a, c, b, d, h}	{g, f, e}	6
adjvex lowcost	a a a c d g d d 0 0 0 0 7 2 0 0	{a, c, b, d, h, g}	{f, e}	5
adjvex lowcost	a a a c f g d d 0 0 0 0 3 0 0 0	{a, c, b, d, h, g, f}	{e}	4
adjvex lowcost	a a a c f g d d 0 0 0 0 0 0 0 0	{a, c, b, d, h, g, f, e}	{ }	


e
3
b
f
4
5
a
d
2
3
5
5
c
4
g
h



(f,g) 2

(a,c) 3

(e,f) 3

(a,b) 4

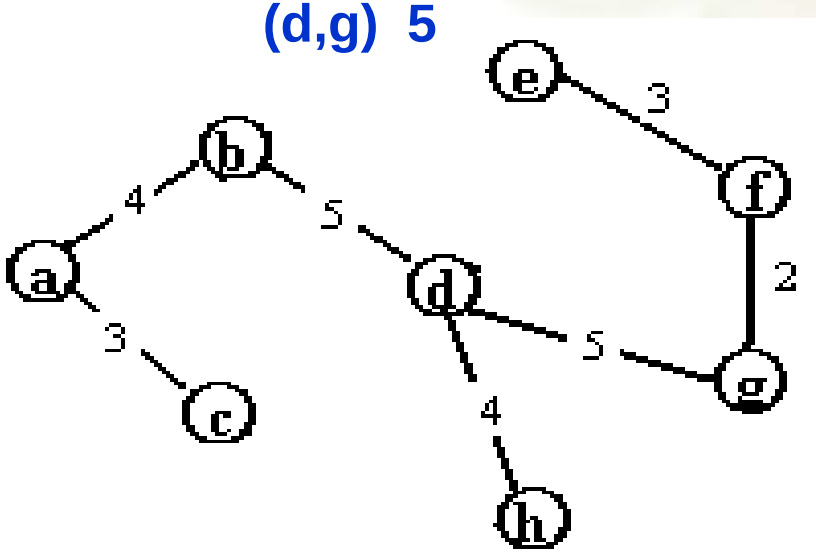
(d,h) 4

(b,c) 5

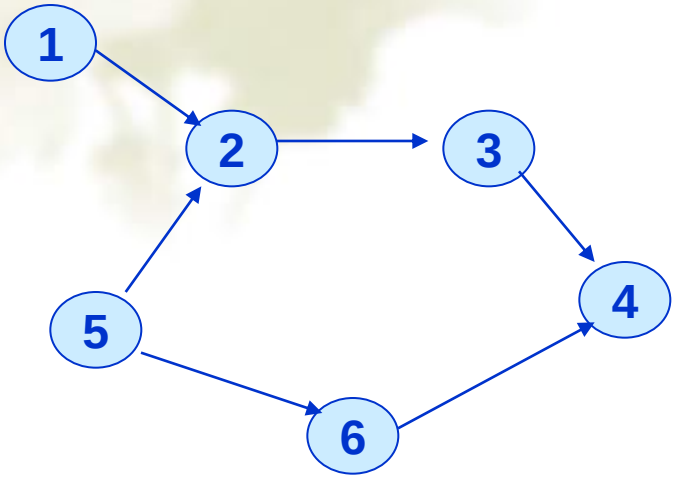
(b,d) 5

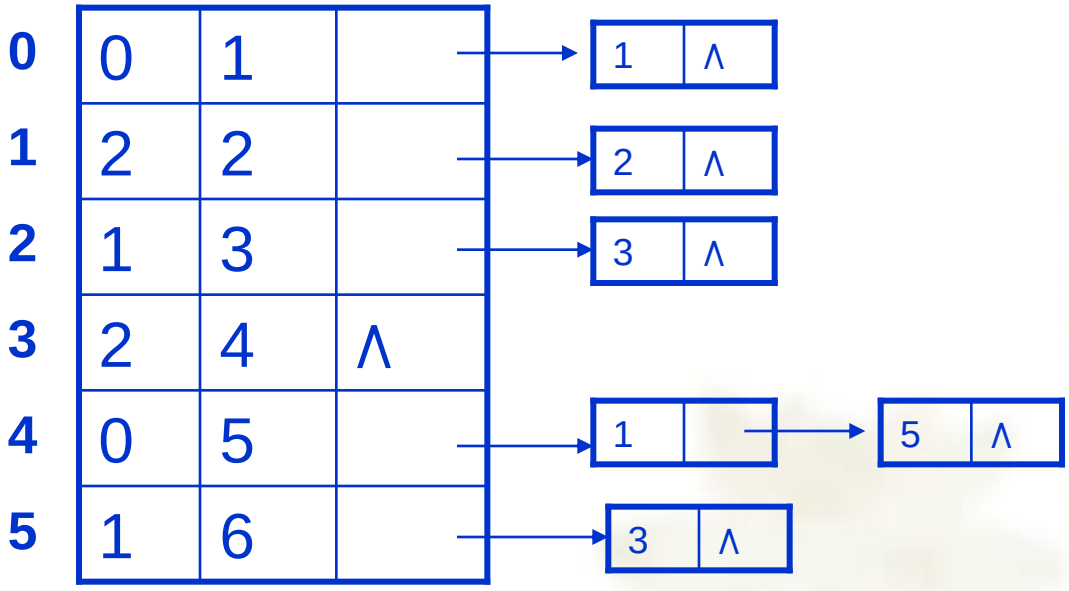
(c,d) 5

(d,g) 5



② 7.9





算法所得次序：5—6—1—2—3—4

第九章 查找

② 9.1 （1） 不相同

（2） 一样

（3） 不相同

② 9.26

```
int Search_Bin(SSTable ST,KeyType Key,KeyType low,KeyType high)
    //折半查找第归算法
    if(low>high)  return 0;
    else{
          mid=(low+high)/2;
          if(ST.elem[mid].key)==key) return mid;
          else if( ST.elem[mid]. key> key)
                    return Search_Bin( ST,key,low,mid-1);
               else return  Search_Bin( ST,key,mid+1,high);
       }//if
}//Search_Bin
```

③ 9.19 选取哈希函数为H（K）＝（3K） mod 11，用开放定址处理冲突di=i((7k)MOD10+1)(i=1,2,3)。试在0-10的散列地址空间对关键字序列（22，41，53，46，30，13，01，67）造哈希表。并求出在等概率的情况下查找成功时的平均搜索长度。

答：设散列表的长度m=11；散列函数为H（K）＝（3K） mod 11，给定的关键码序列为（22，41，53，46，30，13，01，67），则有：

H（22）=0，成功；　　H（41）=2，成功；

H（53）=5，成功；　　H（46）=6，成功；

H（30）=2，冲突，=3，成功；　　H（13）=6，冲突，=8，成功；

H（01）=3，冲突，=0，冲突，=8，冲突，=5，冲突，=2，冲突，=10，成功；

H（67）=3，冲突，=2，冲突，=1，成功；

0	1	2	3	4	5	6	7	8	9	10
22	67	41	30		53	46		13		01
1	3	1	2		1	1		2		6（查找成功时的比较次数）

在等概率的情况下，搜索成功时的平均搜索长度：

ASL=（1*4+2*2+3+6）/8=17/8