第 9 章　查找

教材中练习题及参考答案

1. 设有5个数据*do*、*for*、*if*、*repeat*、*while*，它们排在一个有序表中，其查找概率分别是p_1=0.2，p_2=0.15，p_3=0.1，p_4=0.03，p_5=0.01。而查找它们之间不存在数据的概率分别为q_0=0.2，q_1=0.15，q_2=0.1，q_3=0.03，q_4=0.02，q_5=0.01，该有序表如下：

do	*for*	*if*	*repeat*	*while*

q_0　p_1　q_1　p_2　q_2　p_3　q_3　p_4　q_4　p_5　q_5

（1）试画出对该有序表分别采用顺序查找和折半查找时的判定树。

（2）分别计算顺序查找的查找成功和不成功的平均查找长度。

（3）分别计算折半查找的查找成功和不成功的平均查找长度。

答：（1）对该有序表分别采用顺序查找和折半查找时的判定树分别如图9.2和9.3所示。

（2）对于顺序查找，成功查找到第 *i* 个元素需要 *i* 次比较，不成功查找需要比较的次数为对应外部结点的层次减 1：

$ASL_{成功}=(1p_1+2p_2+3p_3+4p_4+5p_5)=0.97$。

$ASL_{不成功}=(1q_0+2q_1+3q_2+4q_3+5q_4+5q_5)=1.07$。

（3）对于折半查找，成功查找需要比较的次数为对应内部结点的层次，不成功查找需要比较的次数为对应外部结点的层次减 1：

$ASL_{成功}=(1p_3+2(p_1+p_4)+3(p_2+p_5))=1.04$。

$ASL_{不成功}=(2q_0+3q_1+3q_2+2q_3+3q_4+3q_5)=1.3$。

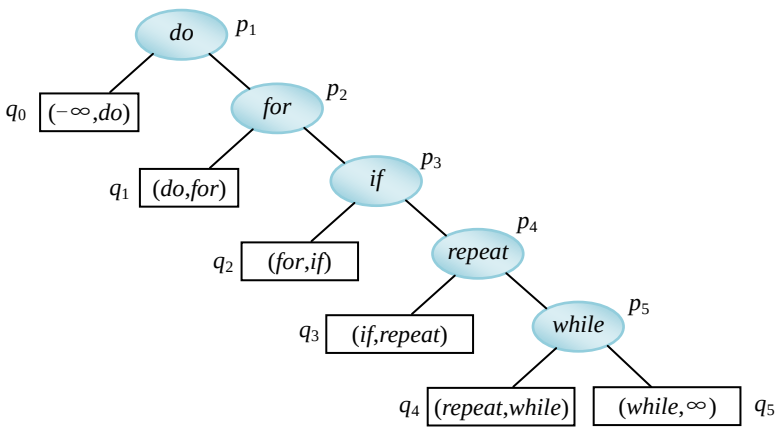


图 9.2 有序表上顺序查找的判定树

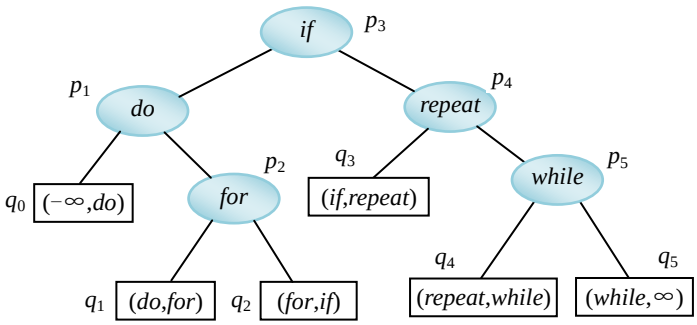


图 9.3 有序表上折半查找的判定树

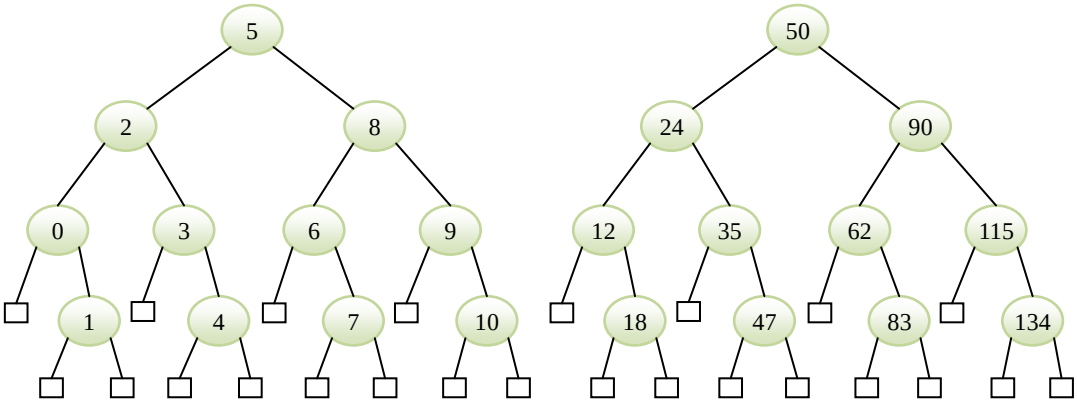
2. 对于 $A[0..10]$有序表，在等概率的情况下，求采用折半查找法时成功和不成功的平均查找长度。对于有序表（12，18，24，35，47，50，62，83，90，115，134），当用折半查找法查找 90 时，需进行多少次查找可确定成功；查找 47 时需进行多少次查找可确定成功；查找 100 时，需进行多少次查找才能确定不成功。

答：对于 $A[0..10]$有序表构造的判定树如图 9.4（a）所示。因此有：

$$ASL_{成功}=\frac{1\times1+2\times2+4\times3+4\times4}{11}=3$$

$$ASL_{不成功}=\frac{4\times3+8\times4}{12}=3.67$$

对于题中给定的有序表构造的判定树如图 9.4（b）所示。查找 90 时，关键字比较次序是 50、90，比较 2 次。查找 47 时，关键字比较次序是 50、24、35、47，比较 4 次。查找 100 时，关键字比较次序是 50、90、115，比较 3 次。



（a） （b）

图 9.4 两棵判定树

3. 有以下查找算法：

```
int fun(int a[],int n,int k)
```

```
{    int i;
     for (i=0;i<n;i+=2)
          if (a[i]==k)
               return i;
     for (i=1;i<n;i+=2)
          if (a[i]==k)
               return i;
     return -1;
}
```

（1）指出 fun(*a*，*n*，*k*)算法的功能。

（2）当*a*[]={2，6，3，8，1，7，4，9}时，执行fun(*a*，*n*，1)后的返回结果是什么？一共进行了几次比较。

（3）当*a*[]={2，6，3，8，1，7，4，9}时，执行fun(*a*，*n*，5)后的返回结果是什么？一共进行了几次比较。

答：（1）fun(*a*，*n*，*k*)算法的功能是在数组*a*[0..*n*−1]中查找元素值为*k*的元素。若找到了返回*k*对应元素的下标；否则返回−1。算法先在奇数序号的元素中查找，如没有找到，再在偶数序号的元素中查找。

（2）当*a*[]={2，6，3，8，1，7，4，9}时，执行fun(*a*，*n*，1)后的返回结果是4，表示查找成功。一共进行了3次比较。

（3）当*a*[]={2，6，3，8，1，7，4，9}时，执行fun(*a*，*n*，5)后的返回结果是−1，表示查找不成功。一共进行了8次比较。

4. 假设一棵二叉排序树的关键字为单个字母，其后序遍历序列为 *ACDBFIJHGE*，回答以下问题：

（1）画出该二叉排序树；

（2）求在等概率下的查找成功的平均查找长度。

（3）求在等概率下的查找不成功的平均查找长度。

答：（1）该二叉排序树的后序遍历序列为*ACDBFIJHGE*，则中序遍历序列为*ABCDEFGHIJ*，由后序序列和中序序列构造的二叉排序树如图9.5所示。

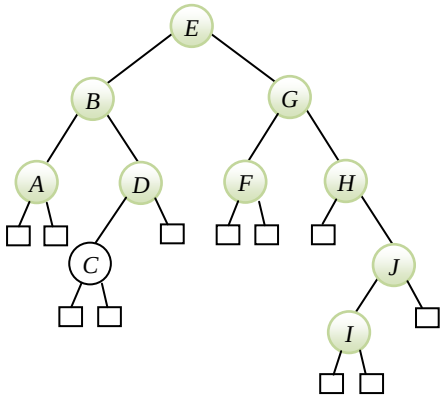


图 9.5　一棵二叉排序树

（2）$ASL_{成功}=(1\times1+2\times2+4\times3+2\times4+1\times5)/10=3$。

（3）$ASL_{不成功}=(6\times3+3\times4+2\times5)/11=3.64$。

5. 证明如果一棵非空二叉树（所有结点值均不相同）的中序遍历序列是从小到大有序的，则该二叉树是一棵二叉排序树。

证明：对于关键字为k的任一结点a，由中序遍历过程可知，在中序遍历序列中，它的左子树的所有结点的关键字排在k的左边，它的右子树的所有结点的关键字排在k的右边，由于中序序列是从小到大排列的，所以结点a的左子树中所有结点的关键字小于k，结点a的右子树中所有结点的关键字大于k，这满足二叉排序树的性质，所以该二叉树是一棵二叉排序树。

6. 由 23、12、45 关键字构成的二叉排序树有多少棵，其中属于平衡二叉树的有多少棵？

答：这里n=3，构成的二叉排序树的个数=$\frac{1}{n+1}C_{2n}^{n}$=5，如图9.6所示。

其中的平衡二叉树有1棵，为图中第3棵。

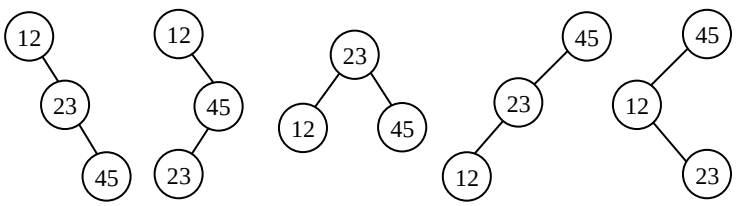


图 9.6　5 棵二叉排序树

7. 将整数序列（4，5，7，2，1，3，6）中的元素依次插入到一棵空的二叉排序树中，试构造相应的二叉排序树，要求用图形给出构造过程。

答：构造一棵二叉排序树过程如图9.7所示。

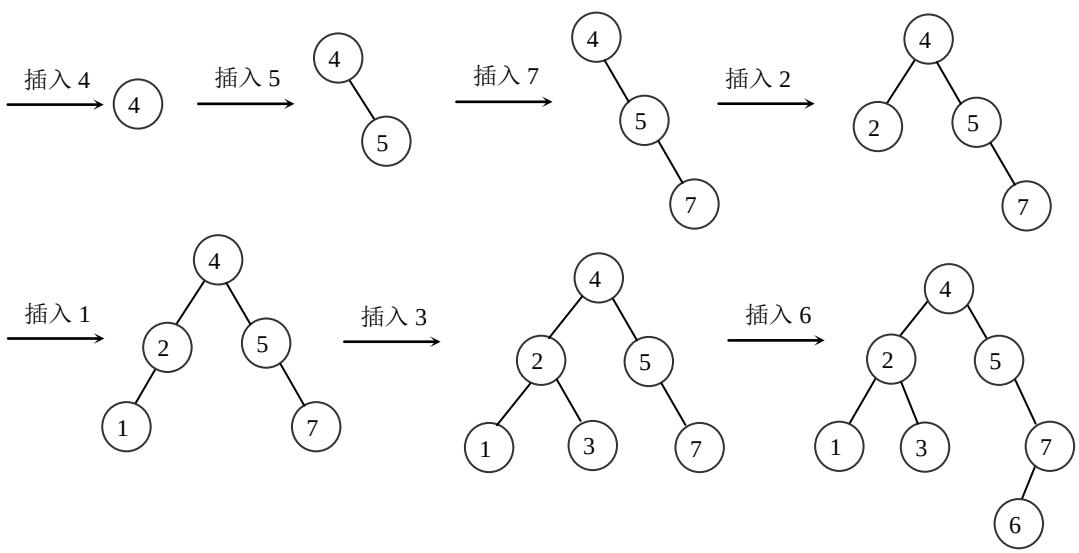


图 9.7 构造二叉排序树过程

8. 将整数序列（4，5，7，2，1，3，6）中的元素依次插入到一棵空的平衡二叉树中，试构造相应的平衡二叉树，要求用图形给出构造过程。

答：构造一棵平衡二叉树过程如图9.8所示。

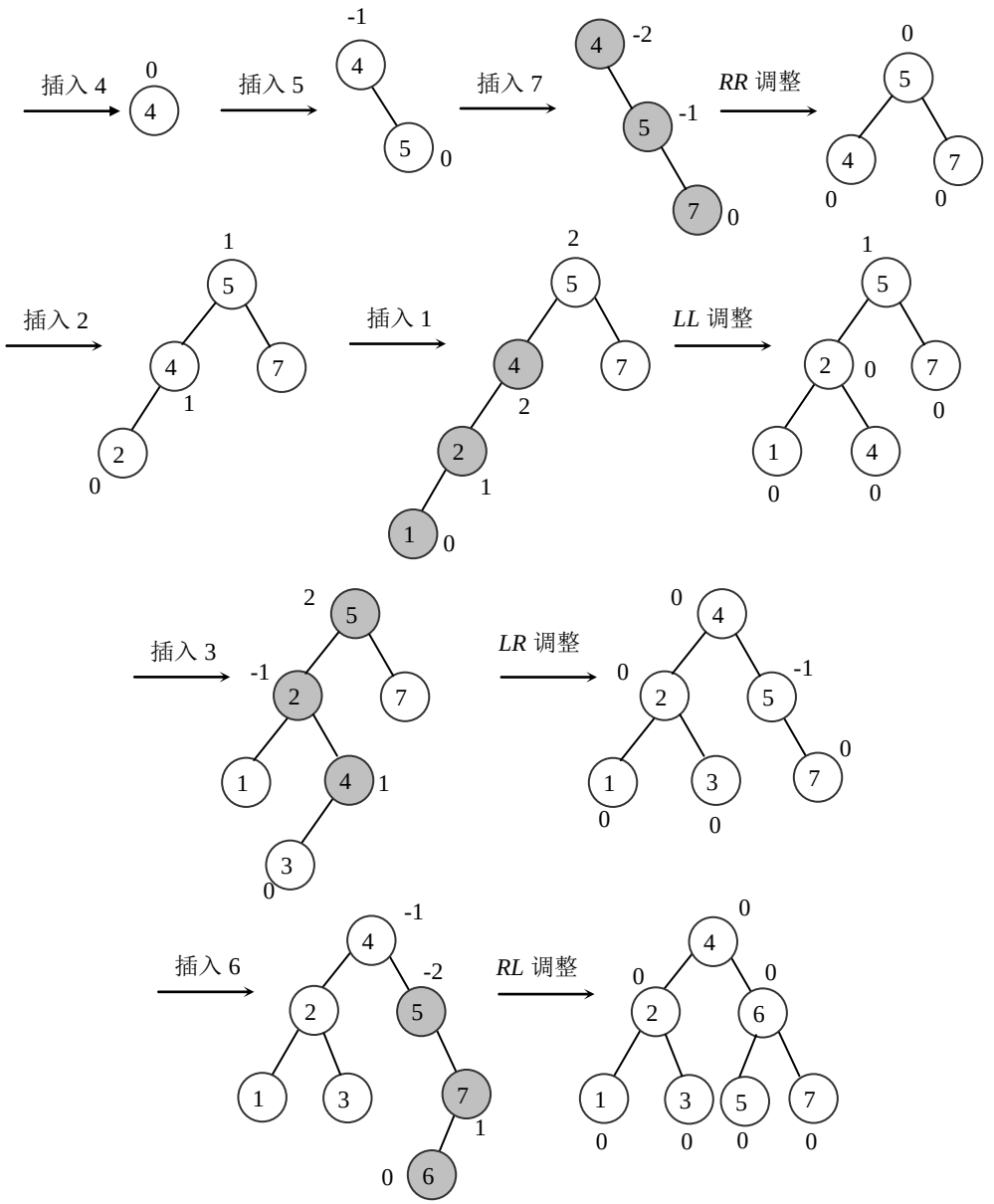


图 9.8 构造平衡二叉树过程

9. 已知一棵5阶B−树中有53个关键字，则树的最大高度是多少？

答：当每个结点的关键字个数都最少时，该B−树的高度最大。根结点最少有1个关键字、2棵子树，第1层至少有1个结点。除根结点外每个结点至少有⌈5/2⌉−1=2个关键字、3棵子树，则第2层至少有2个结点，共2×2=4个关键字。第3层至少有2×3个结点，共2×3×2=12

个关键字。第4层至少有6×2个结点，共6×3×2=36个关键字。而1+4+12+36=53，加上外部结点层，该B−树中最大高度是5层。

10. 设有一组关键字（19，1，23，14，55，20，84，27，68，11，10，77），其哈希函数为$h(key)=key \% 13$。采用开放地址法的线性探测法解决冲突，试在0～18的哈希表中对该关键字序列构造哈希表，并求在成功和不成功情况下的平均查找长度。

答：依题意，m=19，利用线性探测法计算下一地址的计算公式为：

$d_0=h(key)$

$d_{j+1}=(d_j+1) \% m \quad j=0，1，2，\cdots$

计算各关键字存储地址的过程如下：

$h(19)=19 \% 13=6$，$h(1)=1 \% 13=1$，$h(23)=23 \% 13=10$

$h(14)=14 \% 13=1$（冲突），$h(14)=(1+1) \% 19=2$

$h(55)=55 \% 13=3$，$h(20)=20 \% 13=7$

$h(84)=84 \% 13=6$（冲突），$h(84)=(6+1) \% 19=7$（仍冲突），$h(84)=(7+1) \% 19=8$

$h(27)=27 \% 13=1$（冲突），$h(27)=(1+1) \% 19=2$（仍冲突），$h(27)=(2+1) \% 19=3$（仍冲突），$h(27)=(3+1) \% 19=4$

$h(68)=68 \% 13=3$（冲突），$h(68)=(3+1) \% 19=4$（仍冲突），$h(68)=(4+1) \% 19=5$

$h(11)=11 \% 13=11$

$h(10)=10 \% 13=10$（冲突），$h(10)=(10+1) \% 19=11$（仍冲突），$h(10)=(11+1) \% 19=12$

$h(77)=77 \% 13=12$（冲突），$h(77)=(12+1) \% 19=13$

因此，构建的哈希表如表 9.1 所示。

表 9.1 哈希表

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
key		1	14	55	27	68	19	20	84		23	11	10	77					
探测次数		1	2	1	4	3	1	1	3		1	1	3	2					

表中探测次数即为相应关键字成功查找时所需比较关键字的次数，因此：

$ASL_{成功}=(1+2+1+4+3+1+1+3+1+1+3+2)/12=1.92$

查找不成功表示在表中未找到指定关键字的记录。以哈希地址是 0 的关键字为例，由于此处关键字为空，只需比较 1 次便可确定本次查找不成功；以哈希地址是 1 的关键字为例，若该关键字不在哈希表中，需要将它与从 1～9 地址的关键字相比较，由于地址 9 的关键字为空，所以不再向后比较，共比较 9 次，其他的依次类推，所以得到如表 9.2 所示结果。

表 9.2 不成功查找的探测次数

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
key		1	14	55	27	68	19	20	84		23	11	10	77					
探测次数	1	9	8	7	6	5	4	3	2	1	5	4	3	2	1	1	1	1	1

而哈希函数为 $h(key)=key \% 13$，所以只需考虑 $h(key)=0 \sim 12$ 的情况，即：

$ASL_{不成功}=(1+9+8+7+6+5+4+3+2+1+5+4+3)/13=58/13=4.46$

11. 设计一个折半查找算法，求查找到关键字为 *k* 的记录所需关键字的比较次数。假设 *k* 与 *R*[*i*].*key* 的比较得到 3 种情况，即 *k*==*R*[*i*].*key*，*k*<*R*[*i*].*key* 或者 *k*>*R*[*i*].*key*，计为 1 次比较（在教材中讨论关键字比较次数时都是这样假设的）。

解：用 *cnum* 累计关键字的比较次数，最后返回其值。由于题目中的假设，实际上 *cnum* 是求在判定树中比较结束时的结点层次（首先与根结点比较，所以 *cnum* 初始化为 1）。对应的算法如下：

```
int BinSearch1(RecType R[],int n,KeyType k)
{   int low=0,high=n-1,mid;
    int cnum=1;                 //成功查找需要 1 次比较
    while (low<=high)
    {   mid=(low+high)/2;
        if (R[mid].key==k)
            return cnum;
        else if (k<R[mid].key)
            high=mid-1;
        else
            low=mid+1;
        cnum++;
    }
    cnum--;                     //不成功查找比较次数需要减 1
    return cnum;
}
```

12. 设计一个算法，判断给定的二叉树是否是二叉排序树。假设二叉树中结点关键字均为正整数且均不相同。

解：对二叉排序树来说，其中序遍历序列为一个递增有序序列。因此，对给定的二叉树进行中序遍历，如果始终能保持前一个值比后一个值小，则说明该二叉树是一棵二叉排序树。对应的算法如下：

```
KeyType predt=-32768; //predt 为全局变量,保存当前结点中序前驱的值,初值为-∞
bool JudgeBST(BSTNode *bt)
{   bool b1,b2;
    if (bt==NULL)
        return true;
    else
    {   b1=JudgeBST(bt->lchild);    //判断左子树
        if (b1==false)              //左子树不是 BST，返回假
            return false;
        if (bt->key<predt)          //当前结点违反 BST 性质，返回假
            return false;
        predt=bt->key;
        b2=JudgeBST(bt->rchild);    //判断右子树
        return b2;
    }
}
```

13. 设计一个算法，在一棵非空二叉排序树 *bt* 中求出指定关键字为 *k* 结点的层次。

解：采用循环语句边查找边累计层次 *lv*。当找到关键字为 *k* 的结点时返回 *lv*；否则返回 0。对应的算法如下：

```
int Level(BSTNode *bt,KeyType k)
{   int lv=1;                          //层次 lv 置初值 1
    BSTNode *p=bt;
    while (p!=NULL && p->key!=k)       //二叉排序树未找完或未找到则循环
    {   if (k<p->key)
            p=p->lchild;               //在左子树中查找
        else
            p=p->rchild;               //在右子树中查找
        lv++;                          //层次增 1
    }
    if (p!=NULL)                       //找到后返回其层次
        return lv;
    else
        return(0);                     //表示未找到
}
```

14. 设计一个哈希表 *ha*[0..*m*−1]存放 *n* 个元素，哈希函数采用除留余数法 *H*(*key*)=*key* % *p*（*p*≤*m*），解决冲突的方法采用开放定址法中的平方探测法。

（1）设计哈希表的类型。

（2）设计在哈希表中查找指定关键字的算法。

解：哈希表为*ha*[0..*m*−1]，存放*n*个元素，哈希函数为*H*(*key*)=*key* % *p*（*p*≤*m*）。平方探测法：$H_i=(H(key)+d_i) \bmod m$（1≤*i*≤*m*−1），其中，$d_i=1^2、-1^2、2^2、-2^2、\cdots$。

（1）设计哈希表的类型如下：

```
#define MaxSize 100              //定义最大哈希表长度
#define NULLKEY -1               //定义空关键字值
#define DELKEY  -2               //定义被删关键字值
typedef int KeyType;             //关键字类型
typedef char * InfoType;         //其他数据类型
typedef struct
{   KeyType key;                 //关键字域
    InfoType data;               //其他数据域
    int count;                   //探测次数域
} HashTable[MaxSize];            //哈希表类型
```

（2）对应的算法如下：

```
int SearchHT1(HashTable ha,int p,int m,KeyType k)  //在哈希表中查找关键字 k
{   int adr,adr1,i=1,sign;
    adr=adr1=k % p;              //求哈希函数值
    sign=1;
    while (ha[adr].key!=NULLKEY && ha[adr].key!=k) //找到的位置不空
    {   adr=(adr1+sign*i*i) % m;
        if (sign==1)
            sign=-1;
        else                     //sign==-1
```

```
            {    sign=1;
                 i++;
            }
        }
        if (ha[adr].key==k)          //查找成功
            return adr;
        else                         //查找失败
            return -1;
    }
```