第 7 章　查找

1．选择题

（1)对 n 个元素的表做顺序查找时，若查找每个元素的概率相同，则平均查找长度为(　　)。

A．(n-1)/2　　B．　n/2　　C．(n+1)/2　　D．n

答案：C

解释：总查找次数 N=1+2+3+…+n=n(n+1)/2，则平均查找长度为 N/n=(n+1)/2。

（2）适用于折半查找的表的存储方式及元素排列要求为（　　)。

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

答案：D

解释：折半查找要求线性表必须采用顺序存储结构，而且表中元素按关键字有序排列。

（3）如果要求一个线性表既能较快的查找，又能适应动态变化的要求，最好采用(　　)查找法。

A．顺序查找　　B．折半查找

C．分块查找　　D．哈希查找

答案：C

解释：分块查找的优点是：在表中插入和删除数据元素时，只要找到该元素对应的块，就可以在该块内进行插入和删除运算。由于块内是无序的，故插入和删除比较容易，无需进行大量移动。如果线性表既要快速查找又经常动态变化，则可采用分块查找。

（4）折半查找有序表（4，6，10，12，20，30，50，70，88，100)。若查找表中元素 58，则它将依次与表中（　　）比较大小，查找结果是失败。

A．20，70，30，50　　B．30，88，70，50

C．20，50　　D．30，88，50

答案：A

解释：表中共 10 个元素，第一次取⌊(1+10)/2⌋=5，与第五个元素 20 比较，58 大于 20，再取⌊(6+10)/2⌋=8，与第八个元素 70 比较，依次类推再与 30、50 比较，最终查找失败。

（5）对 22 个记录的有序表作折半查找，当查找失败时，至少需要比较（　　）次关键字。

A．3　　B．4　　C．5　　D．6

答案：B

解释：22 个记录的有序表，其折半查找的判定树深度为$\lfloor \log_2 22 \rfloor + 1 = 5$，且该判定树不是满二叉树，即查找失败时至多比较 5 次，至少比较 4 次。

（6）折半搜索与二叉排序树的时间性能（　　)。

A．相同　　B．完全不同

C．有时不相同　　D．数量级都是 $O(\log_2 n)$

答案：C

（7）分别以下列序列构造二叉排序树，与用其它三个序列所构造的结果不同的是（　　)。

A．（100，80，　90，　60，　120，110，130）

B．（100，120，110，130，80，　60，　90）

C．（100，60，　80，　90，　120，110，130）

D．(100，80，　60，　90，　120，130，110)

答案：C

解释：A、B、C、D 四个选项构造二叉排序树都以 100 为根，易知 A、B、D 三个序列中第一个比 100 小的关键字为 80，即 100 的左孩子为 80，而 C 选项中 100 的左孩子为 60，故选 C。

（8）在平衡二叉树中插入一个结点后造成了不平衡，设最低的不平衡结点为 A，并已知 A 的左孩子的平衡因子为 0 右孩子的平衡因子为 1，则应作（　　）型调整以使其平衡。

A．LL　　B．LR　　C．RL　　D．RR

答案：C

（9）下列关于 m 阶 B-树的说法错误的是（　　）。

A．根结点至多有 m 棵子树

B．所有叶子都在同一层次上

C．非叶结点至少有 m/2 (m 为偶数)或 m/2+1（m 为奇数）棵子树

D．根结点中的数据是有序的

答案：D

（10）下面关于 B-和 B+树的叙述中，不正确的是（　　）。

A．B-树和 B+树都是平衡的多叉树　　B．B-树和 B+树都可用于文件的索引结构

C．B-树和 B+树都能有效地支持顺序检索　D．B-树和 B+树都能有效地支持随机检索

答案：C

（11）m 阶 B-树是一棵（　　）。

A．m 叉排序树　　B．m 叉平衡排序树

C．m-1 叉平衡排序树　　D．m+1 叉平衡排序树

答案：B

（12）下面关于哈希查找的说法，正确的是（　　）。

A．哈希函数构造的越复杂越好，因为这样随机性好，冲突小

B．除留余数法是所有哈希函数中最好的

C．不存在特别好与坏的哈希函数，要视情况而定

D．哈希表的平均查找长度有时也和记录总数有关

答案：C

（13）下面关于哈希查找的说法，不正确的是（　　）。

A．采用链地址法处理冲突时，查找一个元素的时间是相同的

B．采用链地址法处理冲突时，若插入规定总是在链首，则插入任一个元素的时间是相同的

C．用链地址法处理冲突，不会引起二次聚集现象

D．用链地址法处理冲突，适合表长不确定的情况

答案：A

解释：在同义词构成的单链表中，查找该单链表表中不同元素，所消耗的时间不同。

（14）设哈希表长为 14，哈希函数是 H(key)=key%11，表中已有数据的关键字为 15，38，61，84 共四个，现要将关键字为 49 的元素加到表中，用二次探测法解决冲突，则放入的位置是（　　）。

A．8　　B．3　　C．5　　D．9

答案：D

解释：关键字 15 放入位置 4，关键字 38 放入位置 5，关键字 61 放入位置 6，关键字 84 放入位置 7，再添加关键字 49，计算得到地址为 5，冲突，用二次探测法解决冲突得到新地址为 6，仍冲突，再用用二次探测法解决冲突，得到新地址为 4，仍冲突，再用用二次探测法解决冲突，得到新地址为 9，不冲突，即将关键字 49 放入位置 9。

（15）采用线性探测法处理冲突，可能要探测多个位置，在查找成功的情况下，所探测的这些位置上的关键字（　　）。

A．不一定都是同义词　　B．一定都是同义词

C．一定都不是同义词　　D．都相同

答案：A

解释：所探测的这些关键字可能是在处理其它关键字冲突过程中放入该位置的。

2．应用题

（1）假定对有序表：（3，4，5，7，24，30，42，54，63，72，87，95）进行折半查找，试回答下列问题：

① 画出描述折半查找过程的判定树；

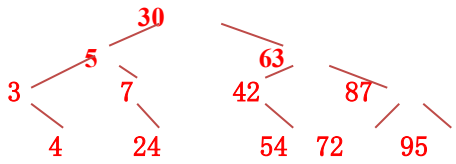
② 若查找元素 54，需依次与哪些元素比较？

③ 若查找元素 90，需依次与哪些元素比较？

④ 假定每个元素的查找概率相等，求查找成功时的平均查找长度。

答案：

①先画出判定树如下（注：mid=⌊(1+12)/2⌋=6）：



②查找元素 54，需依次与 30, 63, 42, 54 元素比较；

③查找元素 90，需依次与 30, 63,87, 95 元素比较；

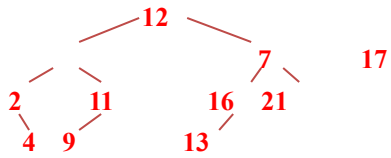
④求 ASL 之前，需要统计每个元素的查找次数。判定树的前 3 层共查找 1＋2×2＋4×3=17 次；

但最后一层未满，不能用 8×4，只能用 5×4=20 次，

所以 ASL＝1/12（17＋20）＝37/12≈3.08

（2）在一棵空的二叉排序树中依次插入关键字序列为 12，7，17，11，16，2，13，9，21，4，请画出所得到的二叉排序树。

答案：



验算方法：　用中序遍历应得到排序结果：2,4,7,9,11,12,13,16,17,21

（3）已知如下所示长度为 12 的表：（Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec）

① 试按表中元素的顺序依次插入一棵初始为空的二叉排序树，画出插入完成之后的二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

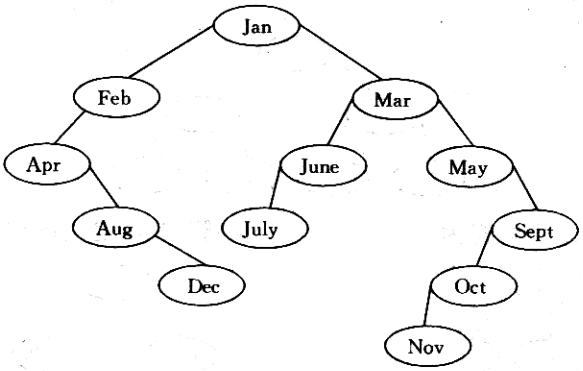
② 若对表中元素先进行排序构成有序表，求在等概率的情况下对此有序表进行折半查找时查找成功的平均查找长度。

③ 按表中元素顺序构造一棵平衡二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

答案：

（1）求得的二叉排序树如下图所示，在等概率情况下查找成功的平均查找长度为

$$ASL_{succ} = \frac{1}{12}(1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 3 + 5 \times 2 + 6 \times 1) = \frac{42}{12}$$



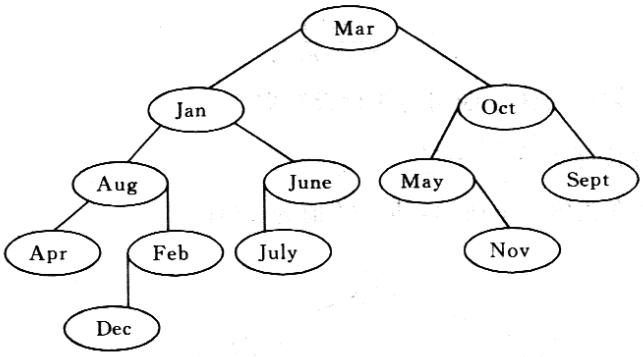
（2）经排序后的表及在折半查找时找到表中元素所经比较的次数对照如下：

Apr	Aug	Dec	Feb	Jan	July	June	Mar	May	Nov	Oct	Sept
3	4	2	3	4	1	3	4	2	4	3	4

等概率情况下查找成功时的平均查找长度为

$$ASL_{succ} = \frac{1}{12}(1 \times 1 + 2 \times 2 + 3 \times 4 + 4 \times 5) = \frac{37}{12}$$

（3）　　平衡二叉树为



它在等概率情况下的平均查找长度为

$$ASL = \frac{1}{12}(1 \times 1 + 2 \times 2 + 3 \times 4 + 4 \times 4 + 5 \times 1) = \frac{38}{12}$$

（4）对图 7.31 所示的 3 阶 B-树，依次执行下列操作，画出各步操作的结果。

① 插入 90　② 插入 25　③ 插入 45　④ 删除 60

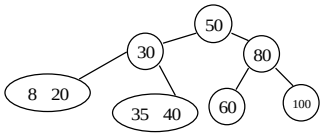
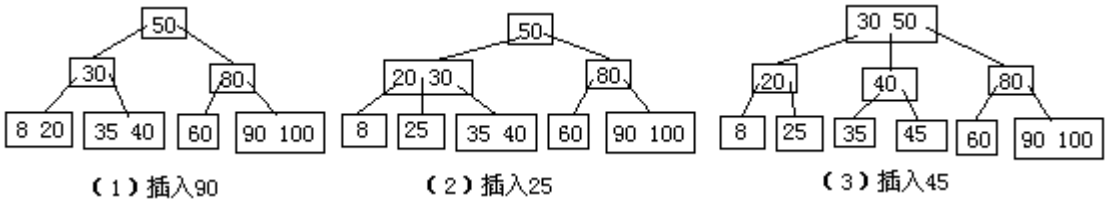
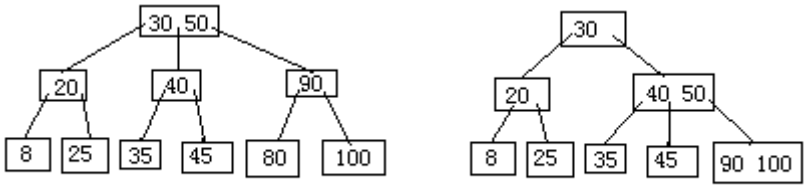


图 7.31　3 阶 B-树



（1）插入90　　（2）插入25　　（3）插入45



（4）删除60　　（5）删除80

（5）设哈希表的地址范围为 0～17，哈希函数为：H（key）=key%16。用线性探测法处理冲突，输入关键字序列：（10，24，32，17，31，30，46，47，40，63，49），构造哈希表，试回答下列问题：

① 画出哈希表的示意图；

② 若查找关键字 63，需要依次与哪些关键字进行比较？

③ 若查找关键字 60，需要依次与哪些关键字比较？

④ 假定每个关键字的查找概率相等，求查找成功时的平均查找长度。

答案：

①画表如下：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
32	17	63	49					24	40	10				30	31	46	47

②查找 63,首先要与 H(63)=63%16=15 号单元内容比较，即 63 与 31 比较 ,不匹配;
然后顺移，与 46,47,32,17,63 相比，一共比较了 6 次!

③查找 60,首先要与 H(60)=60%16=12 号单元内容比较，但因为 12 号单元为空（应当有空标记），所以应当只比较这一次即可。

④对于黑色数据元素，各比较 1 次；共 6 次；

对红色元素则各不相同，要统计移位的位数。“63”需要 6 次，“49”需要 3 次，“40”需要 2 次，“46”需要 3 次，“47”需要 3 次，

所以 ASL=1/11（6＋2＋3×3+6）＝23/11

（6）设有一组关键字（9，01，23，14，55，20，84，27），采用哈希函数：H（key）=key %7 ，表长为 10，用开放地址法的二次探测法处理冲突。要求：对该关键字序列构造哈希表，并计算查找成功的平均查找长度。

答案：

散列地址	0	1	2	3	4	5	6	7	8	9
关键字	14	1	9	23	84	27	55	20		
比较次数	1	1	1	2	3	4	1	2		

平均查找长度：ASL_{succ}=（1+1+1+2+3+4+1+2）/8=15/8

以关键字 27 为例：H（27）=27%7=6（冲突）　H_1=（6+1）%10=7（冲突）

H_2=（$6+2^2$）%10=0（冲突）　H_3=（$6+3^3$）%10=5　所以比较了 4 次。

（7）设哈希函数 H（K）=3 K mod 11，哈希地址空间为 0～10，对关键字序列（32，13，49，24，38，21，4，12），按下述两种解决冲突的方法构造哈希表，并分别求出等概率下查找成功时和查找失败时的平均查找长度 ASL_{succ} 和 ASL_{unsucc}。

① 线性探测法；

② 链地址法。

答案：

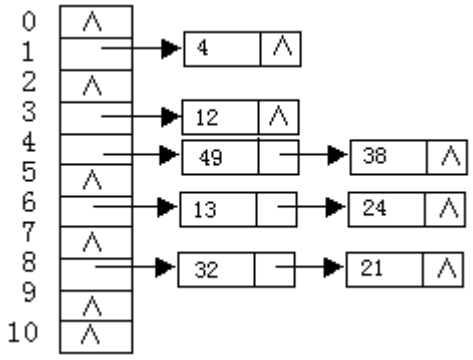
①

散列地址	0	1	2	3	4	5	6	7	8	9	10
关键字		4		12	49	38	13	24	32	21	
比较次数		1		1	1	2	1	2	1	2	

ASL_{succ} =（1+1+1+2+1+2+1+2）/8=11/8

ASL_{unsucc}=（1+2+1+8+7+6+5+4+3+2+1）/11=40/11

②



ASL_{succ} =（1*5+2*3）/8=11/8

ASL_{unsucc}=（1+2+1+2+3+1+3+1+3+1+1）/11=19/11

3．算法设计题

（1）试写出折半查找的递归算法。

[算法描述]

```
int  BinSrch（rectype r[ ]，int  k，low，high）
//在长为 n 的有序表中查找关键字 k，若查找成功，返回 k 所在位置，查找失败返回 0。
{if（low≤high）  //low 和 high 分别是有序表的下界和上界
  {mid=（low+high）/2;
   if（r[mid].key==k）return  （mid）;
   else if（r[mid].key>k）return  （BinSrch（r,k，mid+1，high））;
        else return  （BinSrch（r,k，low，mid-1））;
   }
 else  return  （0）; //查找失败。
}//算法结束
```

（2）试写一个判别给定二叉树是否为二叉排序树的算法。

[题目分析] 根据二叉排序树中序遍历所得结点值为增序的性质，在遍历中将当前遍历结点与其前驱结点值比较，即可得出结论，为此设全局指针变量 pre（初值为 null）和全局变量 flag，初值为 true。若非二叉排序树，则置 flag 为 false。

[算法描述]

```
#define true 1
#define false 0
typedef struct node
{datatype data; struct node *lchild,*rchild;} *BTree;
void  JudgeBST（BTree T,int  flag）
// 判断二叉树是否是二叉排序树，本算法结束后，在调用程序中由 flag 得出结论。
{ if（T!=null && flag）
     { Judgebst（T->lchild,flag）；// 中序遍历左子树
       if（pre==null）pre=T；// 中序遍历的第一个结点不必判断
       else  if（pre->data<T->data）pre=T；//前驱指针指向当前结点
               else{flag=flase；}   //不是完全二叉树
       Judgebst （T->rchild,flag）；// 中序遍历右子树
}//JudgeBST 算法结束
```

（3）已知二叉排序树采用二叉链表存储结构，根结点的指针为 T，链结点的结构为（lchild,data,rchild），其中 lchild，rchild 分别指向该结点左、右孩子的指针，data 域存放结点的数据信息。请写出递归算法，从小到大输出二叉排序树中所有数据值>=x 的结点的数据。要求先找到第一个满足条件的结点后，再依次输出其他满足条件的结点。

[题目分析]本题算法之一是如上题一样，中序遍历二叉树，在“访问根结点”处判断结点值是否≥x，如是则输出，并记住第一个≥x 值结点的指针。这里给出另一个算法，利用二叉排序树的性质，如果根结点的值>=x,则除左分枝中可能有<x 的结点外都应输出。所以从根结点开始查找，找到结点值<x 的结点后，将其与双亲断开输出整棵二叉排序树。如果根结点的值<x,则沿右子树查找第一个≥x 的结点，找到后，与上面同样处理。

```
    void  Print（BSTree  t）
       // 中序输出以 t 为根的二叉排序树的结点
     {if（t）{Print（t->lchild）;
              Cout<<t-data;
              Print（t->rchild）;
             }
     }
    void  PrintAllx(BSTree bst, datatype x）
       //在二叉排序树 bst 中，查找值≥x 的结点并输出
{p=bst;
 if（p）
     {while（p && p->data<x）p=p->rchild; //沿右分枝找第一个值≥x 的结点
      bst=p; //bst 所指结点是值≥x 的结点的树的根
      if（p）
        {f=p; p=p->lchild ; //找第一个值<x 的结点
         while（p && p->data≥x）//沿左分枝向下，找第一个值<x 的结点
           {f=p; p=p->lchild ; } //f 是 p 的双亲结点的指针，指向第一个值≥x 的结点
            if(p) f->lchild=null; //双亲与找到的第一个值<x 的结点断开
         Print(bst); //输出以 bst 为根的子树
           }//while
```

```
        }//内层 if（p）
      }//第一层 if（p）
  }//PrintAllx
```

（4）已知二叉树 T 的结点形式为（lling,data,count,rlink），在树中查找值为 X 的结点，若找到，则记数（count）加 1，否则，作为一个新结点插入树中，插入后仍为二叉排序树，写出其非递归算法。

[算法描述]

```
void SearchBST(BiTree &T,int target){
  BiTree s,q,f;   //以数据值 target,新建结点 s
  s=new BiTNode;
  s->data.x=target;
  s->data.count=0;
  s->lchild=s->rchild=NULL;

  if(!T){
      T=s;
      return ;
  }    //如果该树为空则跳出该函数
  f=NULL;
  q=T;
  while (q){
      if (q->data.x==target){
          q->data.count++;
          return ;
      }  //如果找到该值则计数加一
      f=q;
      if (q->data.x>target)     //如果查找值比目标值大，则为该树左孩子
          q=q->lchild;
      else                          //否则为右孩子
          q=q->rchild;
  }  //将新结点插入树中
  if(f->data.x>target)
      f->lchild=s;
  else
      f->rchild=s;
}
```

（5）假设一棵平衡二叉树的每个结点都表明了平衡因子 b，试设计一个算法，求平衡二叉树的高度。

［题目分析］ 因为二叉树各结点已标明了平衡因子 b，故从根结点开始记树的层次。根结点的层次为 1，每下一层，层次加 1，直到层数最大的叶子结点，这就是平衡二叉树的高度。当结点的平衡因子 b 为 0 时，任选左右一分枝向下查找，若 b 不为 0，则沿左（当 b=1 时）或右（当 b=-1 时）向下查找。

［算法描述］

```
int   Height（BSTree t）
  // 求平衡二叉树 t 的高度
  {level=0；p=t；
   while（p）
    {level++；  // 树的高度增 1
  if（p->bf<0）p=p->rchild；//bf=-1 沿右分枝向下
  //bf 是平衡因子，是二叉树 t 结点的一个域，因篇幅所限，没有写出其存储定义
     else p=p->lchild；          //bf>=0 沿左分枝向下
  }//while
   return   （level）；//平衡二叉树的高度
} //算法结束
```

（6）分别写出在散列表中插入和删除关键字为 K 的一个记录的算法，设散列函数为 H，解决冲突的方法为链地址法。

[算法描述]

```
bool insert(){
  int data;
  cin>>data;
  int ant=hash(data);
  LinkList p=HT[ant];    //初始化散列表
  while (p->next){
        if(p->next->data==data)
              return false;
        p=p->next;
  }    //找到插入位置
  LinkList s;
  s=new LNode;
  s->data=data;
  s->next=p->next;
  p->next=s;   //插入该结点
  return true;
}

bool deletes(){
  int data;
  cin>>data;
  int ant=hash(data);
  LinkList p=HT[ant];   //初始化散列表
  while (p->next){
        if(p->next->data==data){
              LinkList s=p->next;
              p->next=s->next;
              delete s;   //删除该结点
              return true;
```

```
        }   //找到删除位置
        p=p->next;   //遍历下一个结点
    }
    return false;
}
```