# 第 6 章 查找

## 1. 选择题

（1）B （2）A （3）C （4）C （5）D （6）B （7）C （8）D （9）D （10）D
（11）C （12）B （13）B （14）D （15）B

## 2. 判断题

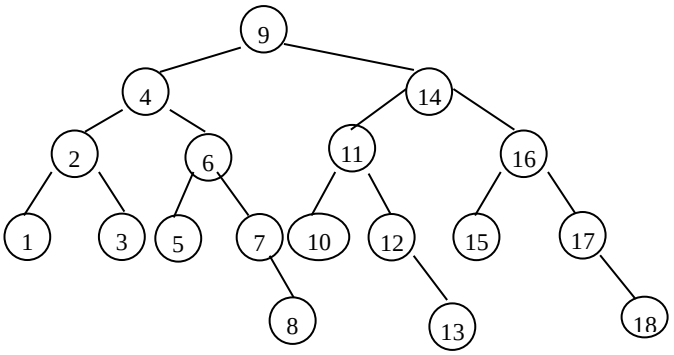
（1）X （2）√ （3）X （4）√ （5）√ （6）√ （7）√ （8）X （9）X （10）X
（11）√ （12）√ （13）√ （14）√ （15）X （16）X （17）√ （18）X （19）√ （20）√

## 3. 简答题

（1）画出对长度为 18 的有序的顺序表进行折半查找时的判定树，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。

**【解答】**

1）判定树为：




2）平均查找长度为 1/18(1+2*2+3*4+4*8+5*3)=32/9

查找最多比较 5 次。

（2）已知如下所示长度为 12 的关键字有序的表：

{Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec}

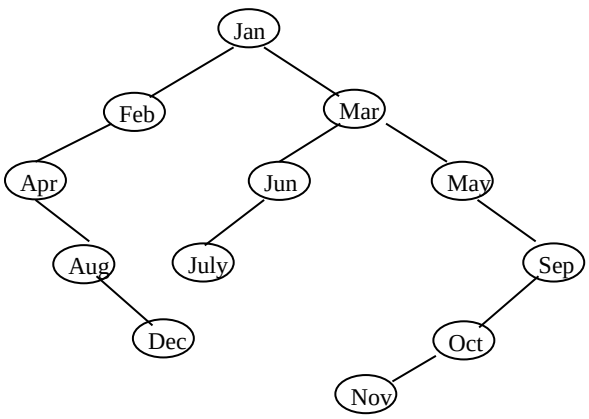
1）试按表中元素的顺序依次插入到一棵初始为空的二叉排序树，画出插入完成后的二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

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查找成功的平均查找长度。

3）按表中元素的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。

**【解答】**

1）按关键字的顺序构造的二叉排序树：



2）根据构造的二叉排序树，求查找成功时的平均查找长度：

$ASL_{SUCC}=(1*1+2*2+3*3+4*3+5*2+6*1)/12=3.5$

3）若对表中元素先进行排序构成有序表再构造二叉排序树，则构造的二叉排序树是一棵单支树，在等概率的情况下查找成功的平均查找长度则为：

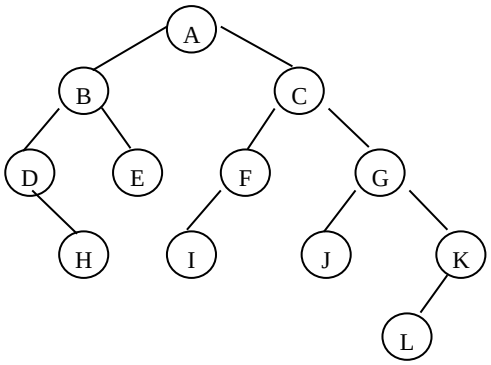
$ASL_{SUCC}=(1+2+\ldots+12)/12=6.5$

这种情况就退化成为顺序查找。

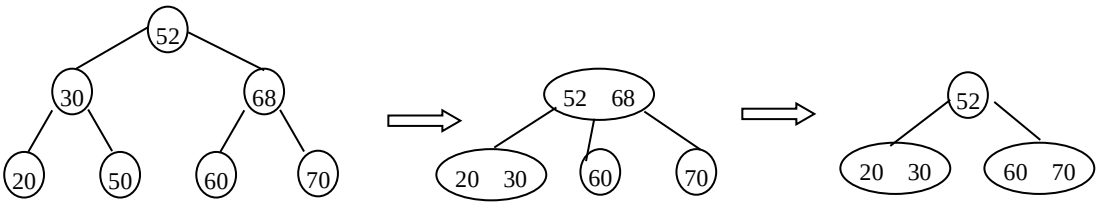
（3）试推导含有 12 个结点的平衡二叉树的最大深度，并画出一棵这样的树。

令 $F_k$ 表示含有最少结点的深度为 k 的平衡二叉树的结点数目。那么,可知道 $F_1=1,F_2=2,.....F_n=F_{n-2}+F_{n-1}+1$.

含有 12 个结点的平衡二叉树的最大深度为 5 ． 例如：




（5）试从空树开始，画出按以下次序向 3 阶 B 树中插入关键码的建树过程：20,30,50,52,60,68,70。如果此后删除 50 和 68，画出每一步执行后 3 阶 B 树的状态。




（6）在地址空间为 0～16 的散列区中，对以下关键字序列构造两个散列表：

{Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec}

1）用线性探测开放定址法处理冲突；

2）用链地址法处理冲突。

并分别求这两个散列表在等概率情况下查找成功和不成功的平均查找长度。设散列函数为 H(key)=i/2, 其中 i 为关键字中第一个字母在字母表中的序号。

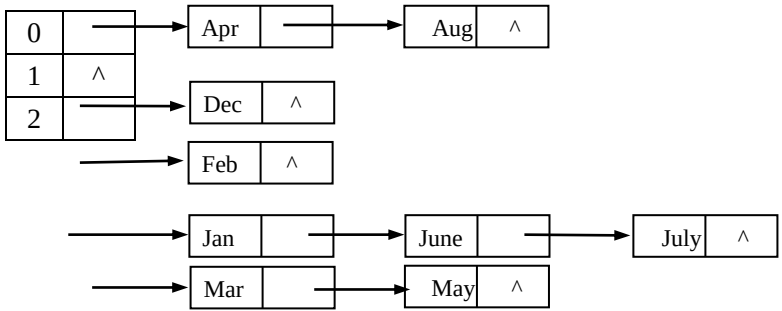
**【解答】**

1）

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Apr | Aug | Dec | Feb | | Jan | Mar | May | June | July | Sep | Oct | Nov | | | | |

平均查找长度为:31/12

2）



| | |
|---|---|
| 3 | |
| 4 | ^ |
| 5 | |
| 6 | |
| 7 | |
| 8 | ^ |
| 9 | |
| 10 | ^ |
| 11 | ^ |
| 12 | ^ |
| 13 | ^ |
| 14 | ^ |
| 15 | ^ |
| 16 | ^ |

平均查找长度为:3/2

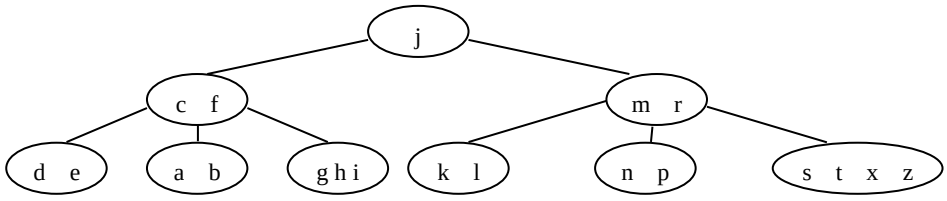
（7）设散列函数 H(key)=(3×key) % 11，用开放定址法处理冲突，探测序列为：$d_i$=i× ((7×key) %10+1)，i=1,2,3,…。试在 0～10 的散列地址空间中对关键字序列（22,41,53,46,30,13,01,67）构造散列表，并求等概率情况下查找成功时的平均查找长度。

**【解答】**

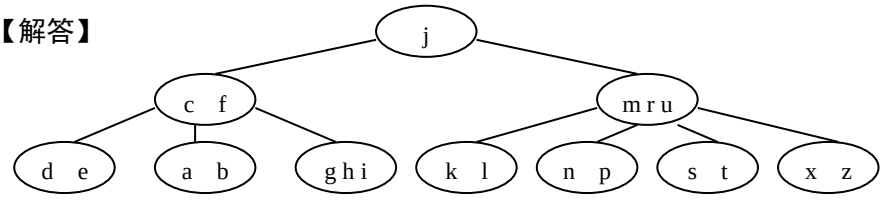
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | 67 | 41 | 30 | | 53 | 46 | | 13 | | 01 |

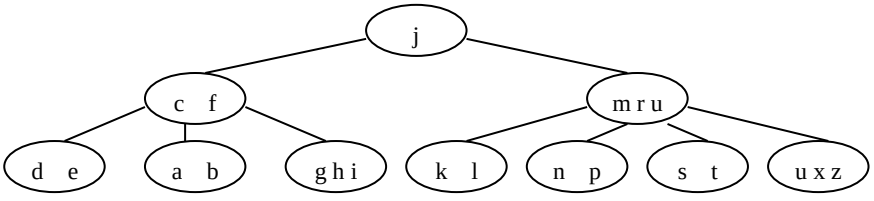
平均查找长度为:17/8

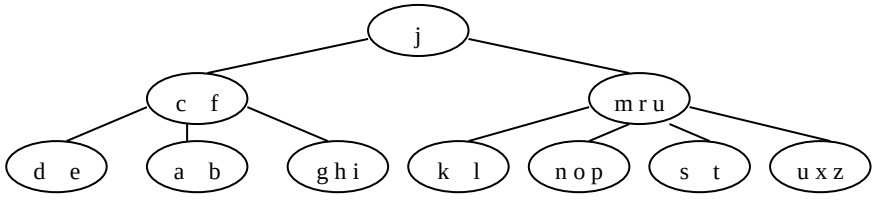
（8）画出依次插入 z,v,o,q,w,y 到下图所示的 5 阶 B 树的过程。

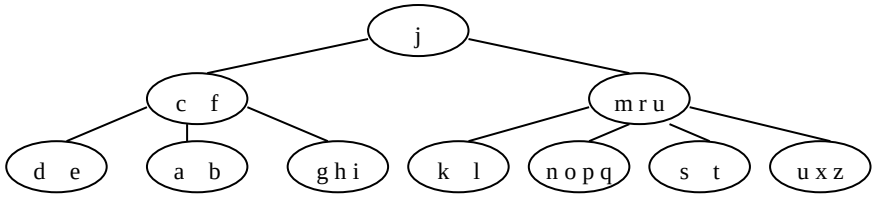


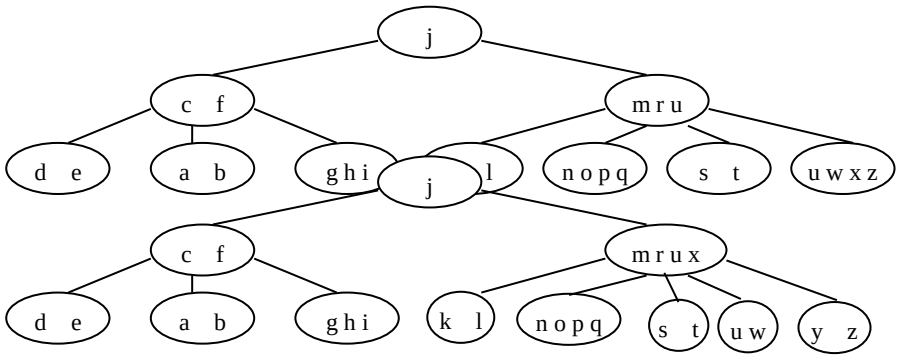

**【解答】**






## 2. 算法设计题

1．将监测哨设在高下标端，改写教科书上给出的顺序查找算法。并分别求出等概率情况下查找成功和查找失败的平均查找长度。

```
typedef struct
  {datatype *data;                    /*查找表的存储空间基址*/
   int    length;                     /*查找表的长度*/
  }SSTable;
int search(SSTable st, keytype kx)
   {st.data[st.length+1]=kx ;
    i=1;
    while (kx!=st.data [i])      i++ ;
    return ( i%( st.length +1)) ;
   }
```

2．将折半查找的算法改写为递归算法。

**【提示】**对于每次查找的思想是相同的，只是查找区间发生了变化，因此写为递归算法时要将查找区间的上下限作为参数，递归调用时查找区间的上下限是变化的。

```
int BinSearch(datatype R[ ], int low, int hig, keytype K)
    {int mid;
     if (low<hig)    return(0);
     else{mid=(low+hig)/2;
          if(K==R[mid].key)      return(mid);
          else if (K> R[mid].key)      return(Binsearch(R, mid+1,hig,K));
               else    return( Binsrch (R,low,mid-1,K));
         }
    }
```

3．编写算法，利用折半查找算法在一个有序表中插入一个元素 x，并保持表的有序性。

**【提示】**先在有序表 R 中利用折半查找法查找关键字值小于或等于 x 的结点，mid 指向关键字正好等于 x 的结点或 low 指向关键字正好大于 x 的结点，然后采用移动法插入 x 结点即可。

```
void BinInsert (datatype R, KeyType x)
    {int low=1,hig=n,mid,inplace,i,find=0;
     while(low<=hig&&!find)
          {mid=(low+hig)/2;
```

```
            if(x.key< R[mid].key)    hig=mid-1;
            else if(x.key> R[mid].key)    low=mid+1;
                 else{i=mid;
                      find=1;
                     }

           }
        if(find) inspace=mid;
        else inspace=low;
        for(i=n;i>=inspace;i--)       R[i+1]=R[i];
        R[inspace]=x;
      }
```

4．假设二叉排序树 T 的各个元素值均不相同，设计一个算法按递减次序打印各元素的值。

**【提示】**调整中序递归遍历算法，先遍历右子树，然后输出根结点的值，再遍历左子树。

```
void PrintNode (BiSTree T)
  {if(T)
    {PrintNode(T->rchild);
     printf( T->data) ;
     PrintNode(T->lchild);
    }
  }
```

5．已知一棵二叉排序树上所有关键字中的最小值为-max，最大值为 max，又知-max<x<max。编写递归算法，求该二叉排序树上的小于 x 且最靠近 x 的值 a 和大于 x 且最靠近 x 的值 b。

**【提示】**解决本题的关键是递归函数的参数设置，采用逐渐缩小查找区间的方法来实现。

```
void fun(BiSTree T, int x, int *a, int *b)
  {if(T)
    if(x<T->data.key)                    /*当 x 小于根结点时，修改 b，在左子树上继续查找*/
       {*b=T->data.key ;
        fun(T->lchild, x,a,b);}
    else if (x>T->data.key)              /*当 x 大于根结点时，修改 a，在右子树上继续查找*/
         {*a=T->data.key ;
          fun(T->rchild,x,a,b);}
         else                            /*当 x 等于根结点时，a 是其左子树的最右下结点，b*/
             {if(T->lchild)              /*是其右子树的最左下结点*/
                 {p=T->lchild;
                  while (p->rchild)    p=p->rchild;
                      *a=p->data.key;
                  }
              if(T->rchild)
                 { p=T->rchild;
                  while (p->lchild)    p=p->lchild;
                      *b=p->data.key;
                 }
             }
  }
```

6．在平衡二叉排序树的每个结点中增设一个 lsize 域，其值为它的左子树中的结点数加 1。试写时间复杂度为 $O(\log_2 n)$的算法，确定树中第 k 小的结点的位置。

**【提示】**由二叉排序树的性质和 lsize 域的定义可知，第 k 小的结点即 lsize 域的值为 k 的结点；因此，可以采用递归算法在二叉排序树中查找这样的结点。

```
typedef struct BiTNode
    {datatype data;
     int lsize;
     struct BiTNode *lchild, *rchild;
     }BiTNode ,*BiTree;                          /*增设 lsize 域的二叉树的类型定义*/
BiTree Index (BiTree T, int k)
    {if(T==NULL)    return(NULL);                /*若根结点 t 为空，则查找失败*/
     if(T->lsize==k)    return(T);                /*找到第 k 小结点，返回其地址*/
     else if(T->lsize>k)     return (T->lchild,k);    /*若 k 小于根结点，在左子树继续查找*/
          else return(T->rchild,k);               /*若 k 大于根结点，在右子树继续查找*/
    }
```

7．假设一棵平衡二叉树的每个结点都标明了平衡因子 bf,设计算法求平衡二叉树的高度。

**【提示】** 因为二叉树各结点已标明了平衡因子 bf，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 bf 为 0 时，任选左右一分支向下查找，若 bf 不为 0，则沿左（当 bf=1 时）或右（当 bf=-1 时）向下查找。

```
typedef struct bsnode
   {datatype     data;
    struct bsnode *lchild,*rchild;
    int bf;                                      /*结点的平衡因子*/
   }BBiST Node, BBiSTree;                        /*带平衡因子域的平衡二叉树的类型定义*/
int Height (BBiSTree 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，沿右分支向下*/
        else p=p->lchild;                        /*bf>=0，沿左分支向下*/
        }
    return (level);
   }
```

8．设计在有序顺序表上进行斐波那契查找的算法，并画出长度为 20 的有序表进行斐波那契查找的判定树，求出在等概率下查找成功的平均查找长度。

**【提示】**采用类似折半查找的算法，按照斐波那契数列分割查找表，实现查找。

```
int Fibo_Search(SSTable L, keytype x, int fibo[])
    {/*向量 fibo 中为斐波那契分割序列*/
      low=1;
      high=L.length;                 /*设 L.length=fibo[K]-1*/
      while(low<=high)
          {mid=low+fibo[K-1];
           if(x<L.elem[mid]) high=mid-1;
           else if(x>L.elem[mid]) low=mid+1;
                else return(mid);
```

```
        }
    return(0);
  }
```

9．试编写一个判定二叉树是否为二叉排序树的算法，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。

**【提示】**判断一棵二叉树是否是二叉排序树，可以采用递归算法。对于树中所有结点，检查其左子树上所有结点的关键字是否都小于它的关键字，检查其右子树上所有结点的关键字是否都大于它的关键字。

```
int BinSortTree(BiSTree T)
/*判别由二叉链表存储的二叉树是否为二叉排序树，是则返回 1，否则返回 0*/
  {int l,r;
   if (!T)    return 1;                                    /*空树是二叉排序树*/
   else if (T->lchild==NULL&&T->rchild=NULL )
          return 1;                                        /*只有一个根结点是二叉排序树*/
      else {l=BinSortTree(T->lchild);
            r=BinSortTree(T->rchild);
            if(T->lchild&&T->rchild)
               if (T->key>T->lchild->key&&T->key<T->rchild->key&&l&&r)
                     return 1;
               else    ruturn 0;
            else if (T->lchild==NULL )                     /*只有一个根结点是二叉排序树*/
                  return (T->key < T->rchild->key && r)
                else return(T->key>T->lchild->key && l)/*只有一个根结点是二叉排序树*/
           }
  }
```

10．假设散列表表长为 m ,散列函数为 H(key)，用链地址法处理冲突。试编写输入一组关键字并建立散列表的算法。

**【提示】**输入关键字后，应先确定其散列地址，再将其插入到相应的单链表中去。

```
typedef struct HNode
   {datatype data;
    struct HNode *next;
   }HNode ,*Hlist;                          /*链地址法处理冲突的散列表的类型定义*/
void CreateHList(Hlist L[m])
   {for(i=0;i<m;i++ )
      L[i]=NULL;                            /*表头指针初始化*/
    scanf(x);
    while(x!='#')
      {h=Hash(x);                           /*计算 x 的散列地址*/
       s=(Hnode *) malloc (sizeof(HNode)) ;
       s->data.key=x;                       /*申请、填装结点*/
       s->next=L[h] ;
       L[h]=s;                              /*插入到散列表中去*/
       scanf(x);
      }
 }
```