第5章 递归

5.1 本章知识体系

1. 知识结构图

本章的知识结构如图 5.1 所示。

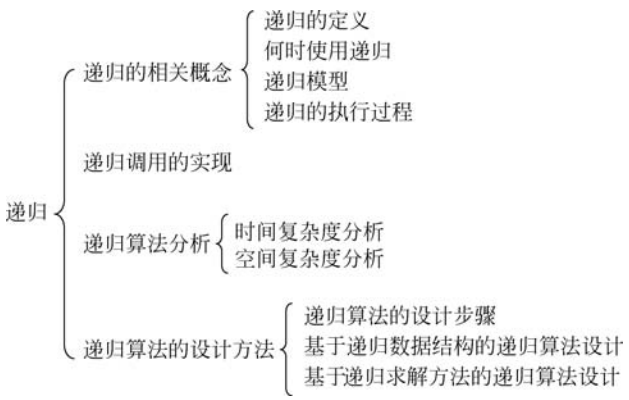

图 5.1　第 5 章知识结构图

2. 基本知识点

(1) 何时使用递归。

(2) 如何从递归角度提取求解问题的递归模型。

(3) 递归算法的执行过程。

(4) 递归调用的实现原理。

(5) 递归算法的时间复杂度分析方法是先由递归算法推导出执行时间的递推式,然后计算 $T(n)$,再用 O 表示。

(6) 递归算法的空间复杂度分析方法是先由递归算法推导出占用空间的递推式,然后计算 $S(n)$,再用 O 表示。

(7) 设计递归算法的一般步骤。

(8) 理解递归数据结构的特征。

(9) 利用递归思想求解复杂的应用问题。

3. 要点归纳

(1) 递归分为直接递归和间接递归,而间接递归算法都可以转换为直接递归算法来实现。

(2) 尾递归是指递归调用语句是最后一条执行语句且把当前运算结果放在参数里传给下层函数。单向递归是指求值过程总是朝着一个方向进行的递归。

(3) 如果求解问题的定义是递归的、存放数据的数据结构是递归的或者问题的求解方法是递归的,一般使用递归算法来求解。

(4) 递归模型是递归算法的抽象,它反映一个递归问题的递归结构。在设计递归算法时首先获取求解问题的递归模型,然后转换为相应的递归算法。

(5) 递归模型由递归出口和递归体两部分构成。递归出口确定递归到何时结束,而递归体确定递归求解时的递推关系。

(6) 递归思路是把一个不能或不好直接求解的“大问题”转化成一个或几个“小问题”来解决。

(7) 函数调用是通过一个栈来实现的,用于保存返回地址、函数实参和局部变量值等。

(8) 一般情况下,尾递归和单向递归算法可以通过循环或者迭代方式转换为等价的非递归算法。其他复杂递归算法可以用栈来模拟递归的执行过程,从而将其转换为等价的非递归算法。

(9) 获取递归模型通常分为 3 个步骤,即分析问题、提取递归体和提取递归出口。在实际中需要根据求解问题来操作。

5.2 教材中的练习题及参考答案

1. 有以下递归函数:

```
void fun(int n)
{   if (n == 1)
        printf("a: %d\n",n);
    else
    {   printf("b: %d\n",n);
        fun(n - 1);
        printf("c: %d\n",n);
    }
}
```

分析调用 fun(5)的输出结果。

解：在调用递归函数 fun(5)时先递推到递归出口，然后求值。这里的递归出口语句是 printf("a:%d\n",n);，递推时执行的语句是 printf("b:%d\n",n);，求值时执行的语句是 printf("c:%d\n",n);。调用 fun(5)的输出结果如下：

```
b:5
b:4
b:3
b:2
a:1
c:2
c:3
c:4
c:5
```

2. 以下递归算法用于对数组 $a[i..j]$中的元素进行归并排序：

```
void mergesort(int a[],int i,int j)
{   int m;
    if (i!= j)
    {   m = (i + j)/2;
        mergesort(a,i,m);
        mergesort(a,m + 1,j);
        merge(a,i,j,m);
    }
}
```

求执行 mergesort(a,0,$n-1$)的时间复杂度。其中，merge(a,i,j,m)用于两个有序子序列 $a[i..m]$和 $a[m+1..j]$的合并，是非递归函数，它的时间复杂度为 O(合并的元素个数)。

解：设 mergesort(a,0,$n-1$)的执行时间为 $T(n)$，分析得到以下递归关系。

$$T(n)=\begin{cases}O(1) & n=1\\ 2T(n/2)+O(n) & n>1\end{cases}$$

其中，$O(n)$为 merge()所需的时间，设为 cn(c 为常量)。因此：

$$T(n)=2T\left(\frac{n}{2}\right)+cn=2\left[2T\left(\frac{n}{2^2}\right)+\frac{cn}{2}\right]+cn=2^2T\left(\frac{n}{2^2}\right)+2cn=2^3T\left(\frac{n}{2^3}\right)+3cn$$

$$\vdots$$

$$=2^kT\left(\frac{n}{2^k}\right)+kcn=2^kO(1)+kcn$$

由于$\frac{n}{2^k}$趋近于 1，$k=\log_2 n$，所以 $T(n)=2^{\log_2 n}O(1)+cn\log_2 n=n+cn\log_2 n=O(n\log_2 n)$。

3. 已知 $A[0..n-1]$为实数数组，设计一个递归算法求这 n 个元素的平均值。

解：设 avg(A,i)返回 $A[0..i]$中共 $i+1$ 个元素的平均值，则递归模型如下。

$$\text{avg}(A,i)=\begin{cases}A[0] & \text{当 } i=0 \text{ 时}\\ (\text{avg}(A,i-1)\times i+A[i])/(i+1) & \text{其他情况}\end{cases}$$

初始调用为 avg(A,$n-1$)。对应的递归算法如下：

```
double avg(double A[],int i)
{   if (i==0)
        return(A[0]);
    else
        return((avg(A,i-1)*i+A[i])/(i+1));
}
```

求 $A[n]$中 n 个元素平均值的调用方式为 avg(A,$n-1$)。

4. 设计一个算法求正整数 n 的位数。

解：设 $f(n)$为整数 n 的位数，其递归模型如下。

$$f(n)=\begin{cases}1 & \text{当 } n<10 \text{ 时}\\ f(n/10)+1 & \text{其他情况}\end{cases}$$

对应的递归算法如下：

```
int fun(int n)
{   if (n<10)
        return 1;
    else
        return fun(n/10)+1;
}
```

5. 上楼可以一步上一阶，也可以一步上两阶，设计一个递归算法，计算共有多少种不同的走法。

解：设 $f(n)$表示 n 阶楼梯的不同的走法数，显然 $f(1)=1$，$f(2)=2$(两阶有一步一步走和两步走两种走法)。$f(n-1)$表示 $n-1$ 阶楼梯的不同的走法数，$f(n-2)$表示 $n-2$ 阶楼梯的不同的走法数，对于 n 阶楼梯，第 1 步上一阶有 $f(n-1)$种走法，第 1 步上两阶有 $f(n-2)$种走法，则 $f(n)=f(n-1)+f(n-2)$。对应的递归算法如下：

```
int fun(int n)
{   if (n==1 || n==2)
        return n;
    else
        return fun(n-1)+fun(n-2);
}
```

6. 设计一个递归算法，利用顺序串的基本运算求串 s 的逆串。

解：经分析，求逆串的递归模型如下。

$$f(s)=\begin{cases}s & \text{若 } s=\varnothing\\ \text{Concat}(f(\text{SubStr}(s,2,\text{StrLength}(s)-1)),\text{SubStr}(s,1,1)) & \text{其他情况}\end{cases}$$

递归思路是：对于 $s=$"$s_1s_2\cdots s_n$"的串，假设"$s_2s_3\cdots s_n$"已求出其逆串，即 f(SubStr(s,2,StrLength(s)－1))，再将 s_1(为 SubStr(s,1,1))单个字符构成的串连接到最后即得到 s 的逆串。对应的递归算法如下：

```
#include "sqstring.cpp"                    //顺序串的基本运算算法
SqString invert(SqString s)
{   SqString s1,s2;
    if (StrLength(s)>0)
    {   s1 = invert(SubStr(s,2,StrLength(s) - 1));
        s2 = Concat(s1,SubStr(s,1,1));
    }
    else
        StrCopy(s2,s);
    return s2;
}
```

7. 设有一个不带表头结点的单链表 L，设计一个递归算法 count(L)求以 L 为首结点指针的单链表的结点个数。

解：对应的递归算法如下。

```
int count(LinkNode *L)
{   if (L==NULL)
        return 0;
    else
        return count(L->next) + 1;
}
```

8. 设有一个不带表头结点的单链表 L，设计两个递归算法，traverse(L)正向输出单链表 L 中的所有结点值，traverseR(L)反向输出单链表 L 中的所有结点值。

解：对应的递归算法如下。

```
void traverse(LinkNode *L)
{   if (L==NULL) return;
    printf("%d",L->data);
    traverse(L->next);
}
void traverseR(LinkNode *L)
{   if (L==NULL) return;
    traverseR(L->next);
    printf("%d",L->data);
}
```

9. 设有一个不带表头结点的单链表 L，设计两个递归算法，del(L,x)删除单链表 L 中第一个值为 x 的结点，delall(L,x)删除单链表 L 中所有值为 x 的结点。

解：对应的递归算法如下。

```
void del(LinkNode *&L,ElemType x)
{   LinkNode *t;
    if (L==NULL) return;
    if (L->data==x)
    {   t=L;L=L->next;free(t);
        return;
    }
    del(L->next,x);
}
void delall(LinkNode *&L,ElemType x)
{   LinkNode *t;
    if (L==NULL) return;
    if (L->data==x)
    {   t=L;L=L->next;
        free(t);
    }
    delall(L->next,x);
}
```

10. 设有一个不带表头结点的单链表 L，设计两个递归算法，maxnode(L)返回单链表 L 中的最大结点值，minnode(L)返回单链表 L 中的最小结点值。

解：对应的递归算法如下。

```
ElemType maxnode(LinkNode *L)
{   ElemType max;
    if (L->next==NULL)
        return L->data;
    max=maxnode(L->next);
    if (max>L->data) return max;
    else return L->data;
}
ElemType minnode(LinkNode *L)
{   ElemType min;
    if (L->next==NULL)
        return L->data;
    min=minnode(L->next);
    if (min>L->data) return L->data;
    else return min;
}
```

11. 设计一个模式匹配算法，其中模式串 t 中含一个或多个通配符'*'，每个'*'可以和任意子串匹配。对于目标串 s，求其中匹配模式串 t 的一个子串的位置('*'不能出现在 t 的开头)。

解：采用 BF 模式匹配的思路，当 $s[i]$和 $t[j]$比较时，若 $t[j]$为'*'，j++跳过 t 的当前'*'，取出 s 中对应'*'的字符及其之后的所有字符构成的字符串，即 SubStr(s,i+1,s.length−i)，其中 i+1 是 s 中对应'*'字符的字符的逻辑序号。再取出 t 中'*'字符后面的所有字符构成的字符串，即 SubStr(t,j+1,t.length−j)，递归对它们进行匹配，若返回值

大于－1，表示匹配成功，返回 start。当 i 或者 j 超界后结束循环，再判断如果是 j 超界，返回 start，否则返回－1。对应的递归算法如下：

```
#include "sqstring.cpp"                    //顺序串的基本运算算法
int findpat(SqString s,SqString t)
{   int i=0,j=0,k,start;
    while (i<s.length && j<t.length)
    {   if (t.data[j]=='*')
        {   j++;                           //跳过*
            k=findpat(SubStr(s,i+1,s.length-i),SubStr(t,j+1,t.length-j));
            if (k>-1)                      //找到了
                return start;
        }
        else if (s.data[i]==t.data[j])
        {   i++;
            j++;
        }
        else
        {   i=i-j+1;
            start=i;
            j=0;
        }
    }
    if (j>=t.length)
        return start;
    else
        return -1;
}
```

5.3 补充练习题及参考答案

5.3.1 单项选择题

习题答案

1. 一个正确的递归算法通常包含_____。

   A. 递归出口　　B. 递归体

   C. 递归出口和递归体　　D. 以上都不包含

2. 递归函数 $f(1)=1, f(n)=f(n-1)+n(n>1)$ 的递归出口是_____。

   A. $f(1)=1$　　B. $f(1)=0$　　C. $f(0)=0$　　D. $f(n)=n$

3. 递归函数 $f(1)=1, f(n)=f(n-1)+n(n>1)$ 的递归体是_____。

   A. $f(1)=1$　　B. $f(0)=0$

   C. $f(n)=f(n-1)+n$　　D. $f(n)=n$

4. 计算 $f(n)=\frac{1}{1\times 2}+\frac{1}{2\times 3}+\cdots+\frac{1}{n(n+1)}$，采用的递归模型为_____。

A. $f(n)=\frac{1}{1\times2}+\frac{1}{2\times3}+\cdots+\frac{1}{n(n+1)}$

B. $f(n)=\frac{1}{1\times2}+\frac{1}{2\times3}+\cdots+\frac{1}{(n-1)n}+f(n-1)$

C. $f(1)=\frac{1}{2}, f(n)=f(n-1)+\frac{1}{n(n+1)}$

D. $f(n)=f(n-1)+\frac{1}{n(n+1)}$

5. 在将递归算法转换成对应的非递归算法时，通常需要使用_____保存中间结果。

A. 队列　　B. 栈　　C. 链表　　D. 树

6. 函数 $f(x,y)$定义如下：

$$f(x,y)=\begin{cases}f(x-1,y)+f(x,y-1) & \text{当 } x>0 \text{ 且 } y>0 \text{ 时}\\ x+y & \text{否则}\end{cases}$$

则 $f(2,1)$的值是_____。

A. 1　　B. 2　　C. 3　　D. 4

7. 一个递归问题可以用递归算法求解，也可以用非递归算法求解，但单从执行时间来看，通常递归算法比非递归算法_____。

A. 快　　B. 慢　　C. 相同　　D. 无法比较

8. 以下关于递归的叙述中错误的是_____。

A. 一般而言，使用递归解决问题比使用循环解决问题需要定义更多的变量

B. 递归算法的执行效率相对较低

C. 递归算法的执行需要用到系统栈

D. 以上都是错误的

9. 在实现递归调用时需要利用系统栈保存参数值。在处理传值参数时，需要为对应形参分配空间，以存放实参的_____。

A. 空间　　B. 副本　　C. 代码地址　　D. 地址

10. 在实现递归调用时需要利用系统栈保存参数值。在处理引用型参数时，需要保存实参的_____。

A. 空间　　B. 副本　　C. 值　　D. 地址

5.3.2 填空题


习题答案

1. 将 $f(n)=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$转化成递归函数，其递归出口是__①__，递归体是__②__。

2. 有以下递归过程：

```
void print(int w)
{   int i;
    if (w!= 1)
    {   print(w - 1);
```

```
        for (i = 0;i <= w;i++)
            printf(" % 3d",w);
        printf("\n");
    }
}
```

调用语句 print(4)的结果是_____。

3. 有以下递归过程：

```
void reverse(int m)
{   printf(" % d",n % 10);
    if (n/10 != 0)
        reverse(n/10);
}
```

调用语句 reverse(582)的结果是_____。

4. 递推式 $f(1)=0, f(n)=f(n/2)+1$ 的解是_____。

5. 设 $a[0..n-1]$是一个含 n 个整数的数组，求该数组中所有元素之和的递归定义是_____。

6. 有以下递归算法：

```
void fun(int n)
{   if (n > 0)
    {   printf(" % d ",n);
        fun(n - 1);
        fun(n - 1);
    }
}
```

执行 fun(3)的输出是_____。

7. 有以下递归算法：

```
void fun(int n)
{   if (n > 0)
    {   fun(n - 1);
        fun(n - 1);
        printf(" % d ",n);
    }
}
```

执行 fun(3)的输出是_____。

5.3.3 判断题

习题答案

1. 判断以下叙述的正确性。

(1) 任何递归算法都有递归出口。

(2) 递归算法的执行效率比功能相同的非递归算法的执行效率高。

(3) 任何能够正确执行的递归算法都能转换成功能等价的非递归算法。

(4) 任何递归算法都是尾递归。

(5) 通常递归的算法简单、易懂、容易编写,而且执行的效率也高。

(6) 尾递归算法可以通过循环转换成非递归算法。

2. 判断以下叙述的正确性。

(1) 有以下递归算法:

```
int fun(int n)
{   if (n==1 || n==0)
        return n;
    else
        return n+fun(n/2);
}
```

其中递归体是 $n==1$ 或 $n==0$ 时返回 n。

(2) 有以下递归函数:

```
int fun(int n)
{   if (n==1 || n==0)
        return n;
    else
        return n+fun(n-2);
}
```

执行 fun(6)的返回结果是 10。

(3) 以下算法中没有循环语句,其时间复杂度为 $O(1)$:

```
int fun(int n)
{   if (n==1)  return 1;
    else return n*fun(n-1);
}
```

(4) 以下算法中没有循环语句,其空间复杂度为 $O(1)$:

```
int fun(int n)
{   if (n==1)  return 1;
    else return n*fun(n-1);
}
```

5.3.4 简答题

习题答案

1. 简述递归算法的优缺点。

2. 推导出求 x 的 n 次幂的递归模型。

3. 推导出求 x 的 n 次幂的递归模型,要求最多使用 $O(\log_2 n)$次递归调用。

4. 采用递归方法,将含有 n 个字符的 C/C++字符串 s 中最后一个为 x 的字符改为 y,

给出其递归模型。

5. 某递归算法的求解时间复杂度的递推式如下：

$$T(n)=\begin{cases}1 & \text{当 } n=0 \text{ 时}\\ T(n-1)+n+3 & \text{当 } n>0 \text{ 时}\end{cases}$$

求该算法的时间复杂度。

6. 设 $a[0..n-1]$是含有 n 个元素的整数数组，写出求 n 个整数之积的递归定义。

7. 以下算法是计算两个正整数 u 和 v 的最大公因数的递归函数，给出其递归模型。

```
int gcd(int u,int v)
{   int r;
    if ((r = u % v) == 0)
        return(v);
    else
        return(gcd(u,r));
}
```

8. 计算以下算法中的语句频度(不计 return 语句的返回)。

```
double rsum(double a[], int n)
{   if (n<=0)
        return a[0];
    else
        return rsum(a,n-1)+a[n-1];
}
```

9. 分析以下算法的时间复杂度(其中问题规模 $n=j-i+1$)。

```
void fun(ElemType A[],int i,int j,ElemType &max,ElemType &min)
{   int mid;
    ElemType gmax,gmin,hmax,hmin;
    if (i == j)
    {   max = min = A[i];
        return;
    }
    if (i == j-1)
    {   if (A[i]<A[j])
        {   max = A[j];min = A[i];}
        else
        {   max = A[i];min = A[j];}
        return;
    }
    mid = (i+j)/2;
    fun(A,i,mid,gmax,gmin);
    fun(A,mid+1,j,hmax,hmin);
    max = (gmax>hmax?gmax:hmax);
    min = (gmin<hmin?gmin:hmin);
}
```

10. 设有算法如下：

```
int Find(ElemType a[],int s,int t,ElemType x)
{   int m=(s+t)/2;
    if (s<=t)
    {   if (a[m]==x)
            return m;
        else if (x<a[m])
            return Find(a,s,m-1,x);
        else
            return Find(a,m+1,t,x);
    }
    return -1;
}
```

分析执行 Find(a,0,$n-1$,x)的时间复杂度。

5.3.5 算法设计题

1.【递归算法设计】有一个不带头结点的单链表，其结点类型为 LinkNode。设计一个递归算法，删除并释放以 h 为首指针的单链表中的所有结点。

解：设 h 为不带头结点的单链表(含 n 个结点)，则 $h->$next 也是一个不带头结点的单链表(含 $n-1$ 个结点)，两者除了相差一个结点以外，其他都是相似的。设 release(h)的功能是删除并释放单链表 h 中的所有结点。其递归模型如下：

$$\text{release}(h)\equiv\begin{cases}\text{不做任何事情} & \text{当 } h \text{ 为空表时}\\ \text{release}(h->\text{next})\text{；释放 } h \text{ 结点} & \text{其他情况}\end{cases}$$

对应的递归算法如下：

```
void release(LinkNode *h)
{   if (h!=NULL)
    {   release(h->next);
        free(h);                //释放 h 结点
    }
}
```

2.【递归算法设计】设计一个递归算法，利用串的基本运算 SubStr()判断字符 x 是否在串 s 中。

解：设串 $s=$"$a_1a_2\cdots a_n$"，Find(s,x)的值表示 x 是否为串 s 的元素，若是则返回真，否则返回假。本题的递归模型如下：

$$\text{Find}(s,x)=\begin{cases}0 & \text{如果 } s \text{ 为空串}\\ 1 & \text{如果 } a_1=x\\ \text{Find}(\text{"}a_2\cdots a_n\text{"},x) & \text{其他情况}\end{cases}$$

对应的递归算法如下：

```
#include "SqString.cpp"                 //包含顺序串的定义和基本运算函数
bool Find(SqString s,char x)
```

```
{   SqString s1;
    if (s.length == 0)
        return false;
    else if (s.data[0] == x)                    //a1 = x
        return true;
    else
    {   s1 = SubStr(s,2,s.length - 1);          //s1 = "a2 … an"
        return(Find(s1,x));
    }
}
```

3.【递归算法设计】一个人赶着鸭子去每个村庄卖,每经过一个村子卖出所赶鸭子的一半又一只,这样他经过了 7 个村子后还剩两只鸭子。设计一个算法求他出发时共赶了多少只鸭子?

解:设 fun(i)表示经过 i 个村子后还剩下的鸭子数,依题意有以下递归模型。

$$\text{fun}(i)=\begin{cases}2 & \text{当 } i=7 \text{ 时}\\ 2\times\text{fun}(i+1)+1 & \text{当 } i<7 \text{ 时}\end{cases}$$

对应的递归算法如下:

```
int fun(int i)
{   if (i == 7)
        return 2;
    else
        return 2 * fun(i + 1) + 1;
}
```

调用 fun(0)求得出发时共赶鸭子数为 383 只。

4.【递归算法设计】求解猴子吃桃问题。海滩上有一堆桃子,5 只猴子来分。第一只猴子把这堆桃子分为 5 份,多了一个,这只猴子把多的一个扔入海中,拿走了一份。第 2 只猴子把剩下的桃子又平均分成 5 份,又多了一个,它同样把多的一个扔入海中,拿走了一份,第 3、第 4、第 5 只猴子都是这样做的,问海滩上原来最少有多少个桃子?

解:设 fun(i)表示第 i 个猴子分桃子前的桃子总数。显然,第 5 只猴子分桃子后的桃子总数为 m(相当于第 6 个猴子分桃子前的桃子总数,可以是任何大于或等于 0 的整数)。$f(n+1)$ 应该是$(f(n)-1)/5$ 的 4 倍,即 $f(n+1)=4[(f(n)-1)/5]$,求出 $f(n)=5f(n+1)/4+1$,而 $f(n)$一定为整数,所以 m 应该取保证所有 $5f(n+1)/4$ 整除的最小整数。依题意有以下递归模型:

$\text{fun}(6)=m$　　第 5 只猴子分桃子后的桃子总数为 m

$\text{fun}(n)=(\text{fun}(n+1)+1)\times 5$　　当 $n>1$ 时

对应的递归算法如下:

```
bool isn(int x,int y)       //x 整除 y 时返回 true
{   if (x % y == 0)
        return true;
    else
```

```
        return false;
}
int fun(int n,int m)
{   if (n==6)
        return m;
    else
    {   if (isn(5*fun(n+1,m),4))
            return (5*fun(n+1,m)/4+1);
        else                              //当m不合适时返回-1
            return -1;
    }
}
int pnumber()
{   int k;
    int m=0;                              //m从0开始试探
    while(true)
    {   k=fun(1,m);
        if (k!=-1)
            break;
        m++;
    }
    return k;
}
```

pnumber()的计算结果是3121,所以海滩上原来最少有3121个桃子。

5.【递归算法设计】有以下递归计算公式:

$$C(n,0)=1 \qquad n\geqslant 0$$

$$C(n,n)=1 \qquad n\geqslant 0$$

$$C(n,m)=C(n-1,m)+C(n-1,m-1) \qquad n>m,n\geqslant 0,m\geqslant 0$$

设计一个递归算法和一个非递归算法求$C(n,m)$。

解:对应的递归算法如下。

```
int fun(int n,int m)
{   if (n>=0 && m==0 || n>=0 && m==n)
        return 1;
    else
    {   if (n>m && n>=0 && m>=0)
            return(fun(n-1,m)+fun(n-1,m-1));
        else
        {   printf("n,m值不正确\n");
            return(-1);
        }
    }
}
```

用一个数组a存放$C(m,n)$的值,对应的非递归算法如下:

```
int fun1(int n,int m)
{   int a[M][N] = {0},i,j;
    for (i = 0;i <= n;i++)
    {   a[i][0] = 1;
        a[i][i] = 1;
    }
    for (j = 1;j <= m;j++)
        for (i = j + 1;i <= n;i++)
            a[i][j] = a[i - 1][j] + a[i - 1][j - 1];
    return a[n][m];
}
```

6.【递归算法设计】编写一个递归算法，读入一个字符串（以“.”作为结束），要求打印出它们的倒序字符串。

解：首先获取用户的按键，如果不是'.'字符，则递归调用该过程，否则显示该字符。对应的算法如下：

```
void reverse()
{   char ch;
    scanf("%c",&ch);
    if (ch!= '.')
    {   reverse();
        printf("%c",ch);
    }
}
```

7.【递归算法设计】设计一个程序求解全排列问题：输入 n 个不同的字符，给出它们所有的 n 个字符的全排列。

解：将 n 个不同的字符存放在字符串 str[0..$n-1$]中，设 f(str,k,n)表示输出 str[k..$n-1$](共 $n-k$ 个字符)所有字符全排列，而 f(str,$k+1$,n)表示输出 str[$k+1$..$n-1$](共 $n-k-1$ 个字符)所有字符全排列，前者是大问题，后者为小问题。递归模型 f(str,k,n)如下：

$$f(\text{str},k,n)\equiv\begin{cases}\text{输出产生的解} & \text{若 } k=n-1\\ \text{对于 } k\sim n-1 \text{ 的 } i,\text{str}[i]\text{与 str}[k]\text{交换位置；} & \text{其他情况}\\ \quad f(\text{str},k+1,n)\text{；} & \\ \quad \text{将 str}[k]\text{与 str}[i]\text{交换位置(恢复环境)；} & \end{cases}$$

对应的算法如下：

```
void print(char str[],int n)          //输出一个排列
{   for (int i = 0;i < n;i++)
        printf("%c ",str[i]);
    printf("\n");
}
void perm(char str[],int k,int n)
{   int i;
```

```
    if (k == n - 1)
        print(str,n);
    else
    {   for (i = k;i < n;i++)
        {   swap(str[k],str[i]);//交换 str[k]与 str[i]
            perm(str,k + 1,n);
            swap(str[k],str[i]);//交换 str[k]与 str[i]
        }
    }
}
```

设计以下主函数：

```
int main()
{   int n = 3;
    char a[4] = "123";
    printf("123 的全排列如下:\n");
    perm(a,0,n);
    return 1;
}
```

程序的执行结果如下：

```
123 的全排列如下:
1 2 3
1 3 2
2 1 3
2 3 1
3 2 1
3 1 2
```

8.【递归算法设计】设计一个程序求解组合问题：从自然数 1～n 中任取 r 个数的所有组合。

解：设 comb(a,m,k)为从 1～m 中任取 k 个数的所有组合(每个组合放在数组 a 中，由于组合与元素的顺序无关，不妨设 $a[0]<a[1]<a[2]<\cdots$)。当组合的最后一个数字(只能取 k～m 中的一个数)选定后，其后的数字是从余下的 $m-1$ 个数中取 $k-1$ 个数的组合。求解组合问题的递归模型如下：

$$\text{comb}(a,m,k)\equiv\begin{cases}\text{输出产生的解} & \text{若 } k=1\\ \text{对于 } k\sim m \text{ 的 } i,a[k-1]=i; & \text{其他情况}\\ \quad \text{comb}(a,i-1,k-1); & \end{cases}$$

对应的程序如下：

```
#include <stdio.h>
#define MaxSize 10
int n,r;                                    //全局变量
void print(int a[])                         //输出一个组合
```

```
{   int j;
    for (j = r - 1;j >= 0;j -- )
        printf(" %d ",a[j]);
    printf("\n");
}
void comb(int a[ ],int m,int k)
{   int i;
    for (i = m;i >= k;i -- )
    {   a[k - 1] = i;
        if (k > 1)
            comb(a,i - 1,k - 1);
        else
            print(a);
    }
}
int main( )
{   int a[MaxSize];
    printf("输入 n,r(r <= n):");
    scanf(" %d %d",&n,&r);
    printf("1..%d 中 %d 个的组合结果如下:\n",n,r);
    comb(a,n,r);
    printf("\n");
    return 1;
}
```

程序的一次执行结果如下：

```
输入 n,r(r <= n):5 3 ↙
1..5 中 3 个的组合结果如下:
5 4 3
5 4 2
5 4 1
5 3 2
5 3 1
5 2 1
4 3 2
4 3 1
4 2 1
3 2 1
```

9.【递归算法设计】棋子移动问题：有 $2n$ 个棋子($n \geqslant 4$)排成一行，开始位置为白色全部在左边，黑色全部在右边。移动棋子的规则是每次必须同时移动相邻的两个棋子，颜色不限，可以左移也可以右移到空位上去，但不能调换两个棋子的左右位置。每次移动必须跳过若干个棋子(不能平移)，要求最后能够移成黑白相间的一行棋子。

解：$n=4$ 的求解过程如下。

```
          1 2 3 4 5 6 7 8 9 10
初始状态: ○○○○●●●●——
  第 1 步: ○○○——●●●○●      // mvtosp(4): 即位置 4 开头的两个棋子与"--"交换
  第 2 步: ○○○●○●●——●      // mvtosp(8): 即位置 8 开头的两个棋子与"--"交换
  第 3 步: ○——●○●●○○●      // mvtosp(2): 即位置 2 开头的两个棋子与"--"交换
  第 4 步: ○●○●○●——○●      // mvtosp(7): 即位置 7 开头的两个棋子与"--"交换
  第 5 步: ——○●○●○●○●      // mvtosp(1): 即位置 1 开头的两个棋子与"--"交换
```

用数组 $c[1..2n+2]$存放棋子，用 init()函数对其所有元素初始化。设 mv(n)表示求解 $2n$ 个棋子移动的问题，则求解棋子移动问题的递归模型如下：

$$mv(n)\equiv\begin{cases}\text{直接求解} & n=4\\ \text{将 } c[n]、c[n+1]\text{棋子对移动到 } c[2n+1]、c[2n+2]\text{处，即 mv}(n); & \text{其他情况}\\ \quad\text{将 } c[2n-1]、c[2n]\text{棋子对移动到 } c[n]、c[n+1]\text{处，即 mv}(2n-1);\\ \quad\text{mv}(n-1);\end{cases}$$

对应的程序如下：

```
#include <stdio.h>
#include <string.h>
const int MAX = 100;
char c[MAX][3];
int st = 0,sp,n;                    //全局变量,st 记录移动的步骤,sp 指向为"-"的棋子
void print()                        //输出一个移动步骤
{   printf("  step%-2d:",++st);
    for (int i = 1;i <= 2 * n + 2;i++)
        printf("%s",c[i]);
    printf("\n");
}
void mvtosp(int k)                  //将 c[k]和 c[k+1]两个棋子与"--"交换
{   for (int j = 0;j <= 1;j++)
    {   strcpy(c[sp + j],c[k + j]);
        strcpy(c[k + j],"—");
    }
    sp = k;                         //sp 指向"--"的棋子
    print();                        //输出一个解
}
void mv(int n)                      //求解 2n 个棋子移动的问题
{   if (n == 4)                     //递归出口
    {   mvtosp(4);mvtosp(8);mvtosp(2);
        mvtosp(7);mvtosp(1);
    }
    else                            //求解 n>4 的情况
    {   mvtosp(n);
        mvtosp(2 * n - 1);
        mv(n - 1);
    }
}
```

```
void init(int n)                          //初始化 c 数组
{   int i;
    st = 0;
    sp = 2 * n + 1;                       //sp 指向第 2n + 1 的棋子,即" -- "
    for (i = 1;i <= n;i++)
        strcpy(c[i],"○");
    for (i = n + 1;i <= 2 * n;i++)
        strcpy(c[i],"●");
    strcpy(c[2 * n + 1],"—");
    strcpy(c[2 * n + 2],"—");
}
int main()
{   do
    {   printf("输入 n 值(4 - 20):");
        scanf(" % d",&n);
    } while (n > 20 ‖ n < 4);
    init(n);
    printf("移动过程如下:\n");
    mv(n);
    printf("\n");
    return 1;
}
```

本程序的一次执行结果如下：

```
输入 n 值(4 - 20):8 ↙
移动过程如下:
  step1 :○○○○○○○○——●●●●●●●●○●
  step2 :○○○○○○○○●●●●●●●●——○●
  step3 :○○○○○○○——●●●●●●●○●○●
  step4 :○○○○○○●●●●●●●——○●○●
  step5 :○○○○○——●●●●●●○●○●○●
  step6 :○○○○○●●●●●●——○●○●○●
  step7 :○○○○——●●●●●○●○●○●○●
  step8 :○○○○●●●●●——○●○●○●○●
  step9 :○○○——●●●●○●○●○●○●○●
  step10:○○○●○●●——●○●○●○●○●○●
  step11:○——●○●●○○●○●○●○●○●○●
  step12:○●○●○●——○●○●○●○●○●○●
  step13:——○●○●○●○●○●○●○●○●○●
```

10.【递归算法设计】设以字符序列 *abcd* 作为顺序栈 st 的输入，利用 push(进栈)和 pop(出栈)操作，输出所有可能的出栈序列并编程实现整个算法。

解：设 $A=(a_0,a_1,\cdots,a_j)$ 是已出栈的序列，B 是已进栈的序列(如果栈不空)，$C=(c_i,c_{i+1},\cdots,c_{n-1})$ 是尚未进栈的序列(初始时 $i=0$ 表示 C 中有 n 个字符)，描述进栈、出栈的状态由 A、C 两个序列表示即可(由 A、C 序列可以确定 B 序列)。

产生所有出栈序列的过程是如果所有元素都在 A 序列中(栈空且 C 序列中的所有元素处理完)，此时产生一种出栈序列，否则从某个进栈、出栈的状态(如图 5.2 所示的状态称为

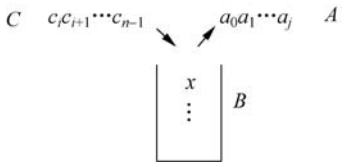

图 5.2 进栈、出栈的状态

初始状态)出发,只有两种操作。

① 从初始状态出发,将 C 中的元素 c_i($i<n$)进栈,此时产生另一种进栈、出栈的状态,从该状态继续进行类似的操作。

② 从初始状态出发,出栈元素 x(栈不空)并将其添加到 A 的末尾,此时产生另一种进栈、出栈的状态,从该状态继续进行类似的操作。

求解过程如图 5.3 所示,其中"从该状态继续进行类似的操作"和产生所有出栈序列的过程是相似的,所以可以采用递归算法。

注意:上述①、②两步都是从初始状态出发的,所以每步执行后都需要将进栈、出栈的状态恢复成初始状态,如第①步进栈了一个元素,其后要将该元素出栈以恢复成原来的初始状态,第②步出栈了一个元素 x,其后要将 x 进栈以恢复成原来的初始状态。

为了方便,用数组 str[0..$n-1$]表示一个进栈序列(其中元素的个数为 n),每个下标对应唯一的元素,所以在进栈、出栈中直接用 0～$n-1$ 的下标来表示,只有在最后输出一个出栈序列时还原为字符序列。用 a 数组表示输出序列,用 j 表示 a 数组中当前元素的位置,st 是一个栈,包含存放元素的 data 数组和栈顶指针 top。

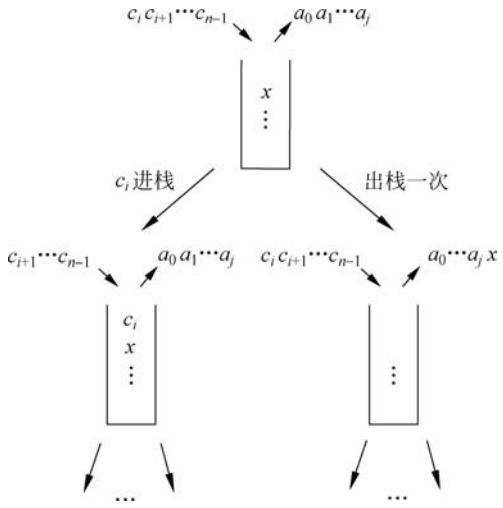

图 5.3 求解过程

前面介绍过,进栈、出栈的状态由 A、C 两个序列表示即可,A 序列中有 j 个元素,用 i 遍历 C 序列(i 从 0 开始),这样进栈、出栈的状态由 i、j 唯一确定。为了简单,在设计栈运算时不考虑溢出情况。完整的程序如下:

```
#include <stdio.h>
#define MaxSize 100
struct stacknode
{    int data[MaxSize];
     int top;
```

```
} st;                                          //全局变量,定义整数顺序栈
int n = 4;                                     //全局变量,定义输入序列中元素的个数
char str[ ] = "abcd";                          //全局变量,指定进栈序列
int sum = 0;                                   //全局变量,累计出栈序列的个数
void initstack()                               //初始化顺序栈
{
    st.top = -1;
}
void push(int x)                               //元素 x 进栈的运算
{   st.top++;
    st.data[st.top] = x;
}
int pop()                                      //退栈运算
{   int temp;
    temp = st.data[st.top];
    st.top--;
    return temp;
}
bool empty()                                   //判断栈空否运算
{   if (st.top == -1)
        return true;
    else
        return false;
}
void process(int i,int a[ ],int j)             //处理 C 序列中 i 位置的元素
{   int x,k;
    if (i >= n && empty())                     //输出一种可能的方案
    {   printf("  ");
        for (k = 0;k < curp;k++)               //输出 a 中的元素序列,构成一种出栈序列
            printf("%c ",str[a[k]]);
        printf("\n");
        sum++;                                 //出栈序列的个数增 1
    }
    if (i < n)                                 //编号为 i 的元素进栈时递归
    {   push(i);                               //i 进栈
        process(i+1,a,j);                      //递归:从该状态继续进行类似的操作
        pop();                                 //出栈以恢复环境
    }
    if (!empty())                              //元素出栈时递归
    {   x = pop();                             //出栈 x
        a[j] = x;                              //将 x 输出到 a 中
        j++;                                   //a 中元素的个数增 1
        process(i,a,j);                        //递归:从该状态继续进行类似的操作
        push(x);                               //进栈以恢复环境
    }
}
int main()
{   int a[MaxSize];
    initstack();
    printf("所有出栈序列:\n");
```

```
    process(0,a,0);                          //i 从 0 开始,j 从 0 开始
    printf("出栈序列的个数:%d\n",sum);
    return 1;
}
```

上述程序的执行结果如下：

```
所有出栈序列:
d c b a
c d b a
c b d a
c b a d
b d c a
b c d a
b c a d
b a d c
b a c d
a d c b
a c d b
a c b d
a b d c
a b c d
出栈序列的个数:14
```