第5章 递归

5.1 简单递归算法设计

5.1.1 LeetCode509——斐波那契数★


视频讲解

【题目解读】

斐波那契数通常用 $F(n)$表示，形成的序列称为斐波那契数列。该数列由 0 和 1 开始，后面的每项数字都是前面两项数字的和。也就是：

$F(0)=0, F(1)=1$

$F(N)=F(N-1)+F(N-2)$，其中 $N>1$

设计一个算法在给定 $N(0\leqslant N\leqslant 30)$时计算 $F(N)$。在选择“C 语言”时要求设计如下函数：

```
int fib(int N) { }
```

例如 $N=2$ 时，$F(2)=F(1)+F(0)=1+0=1$，结果为 1；$N=3$ 时，$F(3)=F(2)+F(1)=1+1=2$，结果为 2。

解法 1

【解题思路】

根据斐波那契数的定义直接采用递归算法求解。

【设计代码】

```
int fib(int N)
{   if (N==0 || N==1)
        return N;
    return fib(N-1)+fib(N-2);
}
```

【提交结果】

执行结果：通过。执行用时 12ms，内存消耗 5.6MB(编程语言为 C 语言)。

解法 2

【解题思路】

在上述递归函数中包含重复的计算，例如 $F(3)=F(2)+F(1)$，$F(4)=F(3)+F(2)$，$F(5)=F(4)+F(3)$，在计算 $F(5)$时需要重复计算 $F(4)$和 $F(3)$，为此用一个 F 数组存放计算的结果，即 $F[i]$存放 fib$[i]$，初始时 F 数组的所有元素为 -1，一旦 $F[i]$已经求出(此时 $F[i]\neq -1$)，直接返回 $F[i]$即可。

【设计代码】

```
int F[35];                  //全局数组
int fib1(int N)             //递归函数
{   if (N==0 || N==1)
    {   F[N]=N;
```

```
        return F[N];
    }
    if (F[N]!=-1) return F[N];
    F[N]=fib1(N-1)+fib1(N-2);
    return F[N];
}
int fib(int N)              //求解算法
{   for (int i=0;i<35;i++)
        F[i]=-1;
    return fib1(N);
}
```

【提交结果】

执行结果：通过。执行用时 0ms，内存消耗 5.4MB(编程语言为 C 语言)。

解法 3

【解题思路】

由于 fib(n)仅与前面两项 fib($n-2$)和 fib($n-1$)相关，采用迭代方法将递归转换为非递归。

【设计代码】

```
int fib(int N)
{   if (N==0 || N==1)
        return N;
    int a=0,b=1,c;
    for (int i=2;i<=N;i++)
    {   c=a+b;
        a=b;
        b=c;
    }
    return c;
}
```

【提交结果】

执行结果：通过。执行用时 0ms，内存消耗 5.5MB(编程语言为 C 语言)。

视频讲解

5.1.2 LeetCode50——Pow(x,n)★★

【题目解读】

设计一个算法求 x^n，其中 x 是[−100,100]的实数，n 是整数(可能是负整数)。在选择“C 语言”时要求设计如下函数：

```
double myPow(double x,int n) { }
```

例如，$x=2.0$，$n=2$，结果为 $2^2=4.000\ 00$；$x=2.0$，$n=-2$，$2^{-2}=1/2^2=1/4=0.25$，结果为 0.250 00。

【解题思路】

采用递归方法求解。设 $f(x,n)$用于计算 $x^n(n>0)$，则有以下递归模型：

$f(x,n)=x$　　当 $n=1$ 时

$f(x,n)=x*f(x,n/2)*f(x,n/2)$　当 n 为大于 1 的奇数时

$f(x,n)=f(x,n/2)*f(x,n/2)$　当 n 为大于 1 的偶数时

当 $n<0$ 时，结果为 $1/f(x,-n)$。需要注意的是，当 $n=-2\ 147\ 483\ 648$（即 -2^{31}）时，$-n$ 会发生溢出，为此将 n 改为用 long 类型的变量 m 存放。

【设计代码】

```
double Pow(double x, long n)          //用递归算法求 x^n(n>0)
{   if (n==1)
        return x;
    double p=Pow(x,n/2);
    if (n%2==1)
        return x*p*p;                 //n 为奇数
    else
        return p*p;                   //n 为偶数
}
double myPow(double x, int n)
{   long m=n;
    if (m==0) return 1;
    if (m>0) return Pow(x,m);
    else return 1/Pow(x,-m);
}
```

【提交结果】

执行结果：通过。执行用时 4ms，内存消耗 5.3MB（编程语言为 C 语言）。

5.1.3 LeetCode206——翻转链表★


视频讲解

【题目解读】

给定一个不带头结点的单链表 head，设计一个算法将其翻转。在选择“C 语言”时要求设计如下函数：

```
struct ListNode* reverseList(struct ListNode* head) { }
```

例如，head=[1,2,3,4]，翻转后的结果是[4,3,2,1]。

【解题思路】

对于不带头结点的单链表 head（含两个或者两个以上的结点），设 f(head)的功能是翻转单链表 head 并且返回翻转单链表的首结点 nh，其过程如图 5.1 所示。

小问题 f(head->next)用于翻转单链表 head->next 并且返回翻转单链表的首结点 nh，执行后 head 结点（即 a_1 结点）的 next 指向 a_2 结点，为了将 a_1 结点链接到 a_2 结点之后，即将 a_1 结点变成尾结点，执行 head->next->next=head，再将 a_1 结点（由 head 指向它）的 next 域置为空，最后返回 nh。

【设计代码】

```
struct ListNode* reverseList(struct ListNode* head)
{   if (head==NULL || head->next==NULL)
        return head;
    struct ListNode* nh=reverseList(head->next);
    head->next->next=head;
```

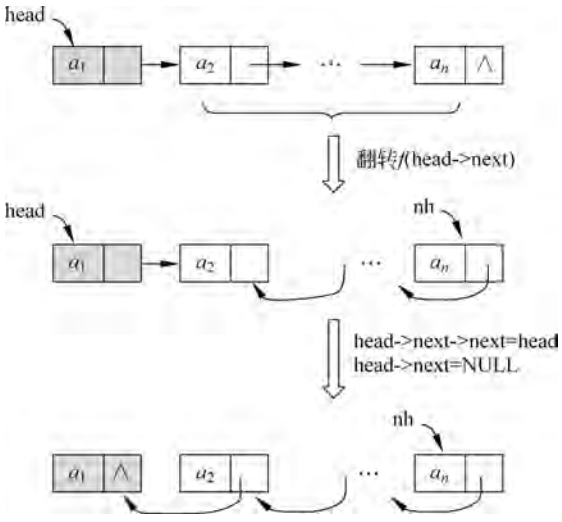

图 5.1　递归翻转单链表 head 的过程

```
    head->next=NULL;
    return nh;
}
```

【提交结果】

执行结果：通过。执行用时 4ms，内存消耗 6.4MB(编程语言为 C 语言)。


视频讲解

5.1.4　LeetCode234——回文链表★

【题目解读】

给定一个不带头结点的单链表 head，设计一个算法判断是否为回文链表，如果是，返回 true，否则返回 false。在选择“C 语言”时要求设计如下函数：

```
bool isPalindrome(struct ListNode* head) { }
```

例如，head=[1,2]，结果为 false；head=[1,2,2,1]，结果为 true。

【解题思路】

先讨论采用递归方法反向输出一个单链表的所有结点值，即单链表为$[a_1,a_2,\cdots,a_n]$时输出结果为$[a_n,a_{n-1},\cdots,a_1]$。

设 f(head)的功能是反向输出一个单链表 head 的所有结点值，为“大问题”，则 f(head->next)的功能是反向输出单链表 head->next 的所有结点值，为“小问题”。对应的递归模型如下：

f(head)≡不做任何事情　　当 head=NULL 时

f(head)≡f(head->next)；输出结点 head 的值　　其他情况

对应的递归算法如下：

```
void reverse(struct ListNode* head)          //反向输出 head 的结点值
{   if (head!=NULL)
    {   reverse(head->next);
        printf("%d ",head->val);             //输出 head 结点值
    }
}
```

从中看出，执行小问题 reverse(head->next)时 head 依次输出的结点是 a_n、a_{n-1}、…、a_2，再输出 a_1 即得到 head 的反向输出结果。为了判断 head 是否为回文链表，再设置一个全局变量 first 正向遍历 head，若 first 与 head 结点值不相同，说明不是回文链表。

【设计代码】

```
struct ListNode * first;                          //全局变量
bool ispal (struct ListNode * last)               //递归算法
{   if(last!=NULL)                                //递推到尾结点为止
    {   if(!ispal(last->next))
            return false;
        if(last->val!=first->val)
            return false;
        first=first->next;
    }
    return true;
}
bool isPalindrome(struct ListNode * head)         //求解算法
{   if (head==NULL || head->next==NULL)
        return true;
    first=head;
    return ispal(head);
}
```

【提交结果】

执行结果：通过。执行用时 12ms，内存消耗 11.2MB(编程语言为 C 语言)。

5.1.5 LeetCode24——两两交换链表中的结点★★

视频讲解

【题目解读】

给定一个不带头结点的单链表 head，设计一个算法两两交换相邻的结点，交换方式是结点地址交换而不是结点值交换。在选择“C 语言”时要求设计如下函数：

```
struct ListNode * swapPairs(struct ListNode * head) { }
```

例如，单链表 head=[1,2,3,4]，交换后 head 变为[2,1,4,3]；单链表 head=[1,2,3,4,5]，交换后 head 变为[2,1,4,3,5]。

【解题思路】

采用递归方法求解。设不带头结点的单链表 head 存放的元素为(a_0,a_1,a_2,…)，f(head)是大问题，用于两两交换链表 head 中的结点。

(1) 若单链表 head 为空或者只有一个结点(head=NULL 或者 head->next==NULL)，交换后的结果单链表没有变化，返回 head。

(2) 否则，让 last 和 p 分别指向 a_1 和 a_2 结点，如图 5.2 所示，显然 $f(p)$为小问题，用于两两交换链表 p 中的结点。f(head)的执行过程是，先交换 last 和 head 结点(让 head 指向 a_1 结点，last 指向 a_0 结点)，再置 last->next=$f(p)$，最后返回 head。

【设计代码】

```
struct ListNode * swapPairs(struct ListNode * head)
```

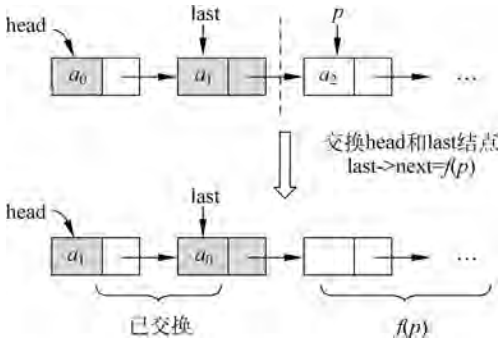

图 5.2　有两个或者两个以上结点时 f（head）的执行过程

```
{    if (head==NULL || head->next==NULL)
         return head;                          //为空或者只有一个结点的情况
     struct ListNode* last=head->next;         //last 指向 a1
     struct ListNode* p=last->next;            //p 指向 a2
     last->next=head;                          //交换 head 和 last 结点
     head=last;
     last=head->next;
     last->next=swapPairs(p);
     return head;
}
```

【提交结果】

执行结果：通过。执行用时 0ms，内存消耗 5.9MB（编程语言为 C 语言）。

5.2　复杂递归算法设计


视频讲解

5.2.1　LeetCode59——螺旋矩阵 Ⅱ ★★

【题目解读】

给定正整数 n，设计一个算法创建一个螺旋矩阵，其中元素值为 $1\sim n^2$，按螺旋方式排列。在选择“C++语言”时要求设计如下函数：

```
class Solution {
public:
    vector<vector<int>> generateMatrix(int n) { }
};
```

【解题思路】

采用递归方法求解。用二维数组 $a[n][n]$存放 n 阶螺旋矩阵，初始化所有元素为 0，vector 向量的定义和初始化如下：

```
vector<vector<int>> ans(n,vector<int>(n,0));
```

设 $f(x,y,\text{start},n)$用于创建左上角为(x,y)、起始元素值为 start 的 n 阶螺旋矩阵，如图 5.3 所示，该矩阵共 n 行 n 列，它是大问题；$f(x+1,y+1,\text{start},n-2)$用于创建左上角

为$(x+1,y+1)$、起始元素值为 start 的 $n-2$ 阶螺旋矩阵，该矩阵共 $n-2$ 行 $n-2$ 列，它是小问题。

例如，如果 4 阶螺旋矩阵为大问题，那么相应地 2 阶螺旋矩阵就是一个小问题，如图 5.4 所示。

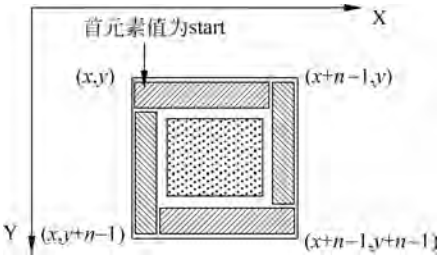

图 5.3 n 阶螺旋矩阵

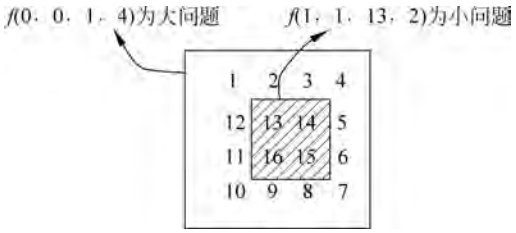

图 5.4 $n=4$ 时的大问题和小问题

对应的递归模型(大问题的 start 从 1 开始)如下：

$f(x,y,start,n)$≡不做任何事情　　当 $n\leqslant 0$ 时

$f(x,y,start,n)$≡产生只有一个元素的螺旋矩阵　　当 $n=1$ 时

$f(x,y,start,n)$≡产生(x,y)的那一圈　　当 $n>1$ 时

$f(x+1,y+1,start,n-2)$

【设计代码】

```
class Solution {
public:
    vector < vector < int >> generateMatrix(int n)
    {   vector < vector < int >> ans(n, vector < int >(n,0));
        Spiral(ans,0,0,1,n);                                    //求 n 阶螺旋矩阵 ans
        return ans;
    }
    void Spiral(vector < vector < int >> &a,int x,int y,int start,int n)    //递归创建螺旋矩阵
    {   if (n<=0) return;                                       //递归结束条件
        if (n==1)                                               //矩阵大小为 1 时
        {   a[x][y]=start;
            return;
        }
        for (int j=x;j<x+n-1;j++)                               //上一行
        {   a[y][j]=start;
            start++;
        }
        for (int i=y;i<y+n-1;i++)                               //右一列
        {   a[i][x+n-1]=start;
            start++;
        }
        for (int j=x+n-1;j>x;j--)                               //下一行
        {   a[y+n-1][j]=start;
            start+=1;
        }
        for (int i=y+n-1;i>y;i--)                               //左一列
```

```
            {    a[i][x]=start;
                 start++;
            }
            Spiral(a,x+1,y+1,start,n-2);                    //递归调用
        }
};
```

【提交结果】

执行结果：通过。执行用时 0ms，内存消耗 6.3MB(编程语言为 C++语言)。

视频讲解

5.2.2 LeetCode51——n 皇后

【题目解读】

n 皇后问题就是在 $n\times n$ 的棋盘上放置 n 个皇后，并且使所有皇后彼此之间不能相互攻击，即所有皇后不同行、不同列和不同左右两条对角线。给定一个整数 $n(1\leqslant n\leqslant 9)$，返回所有不同的 n 皇后问题的解决方案。

每一种解法包含一个不同的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.' 分别代表了皇后和空位。在选择“C++语言”时要求设计如下函数：

```
class Solution {
public:
    vector<vector<string>> solveNQueens(int n) { }
};
```

例如 $n=4$ 时，4 皇后问题有两个解，如图 5.5 所示，返回的结果是[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]。

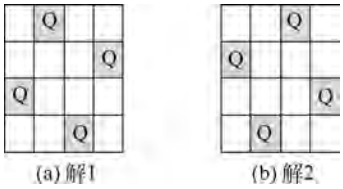

图 5.5 4 皇后问题的两个解

解法 1

【解题思路】

采用递归方法求解。用整数数组 $q[N]$存放 n 皇后问题的求解结果，因为每行只能放一个皇后，$q[i](1\leqslant i\leqslant n)$的值表示第 i 个皇后所在的列号，即该皇后放在$(i,q[i])$的位置上。对于图 5.5(a)的解 1，$q[1..4]=\{2,4,1,3\}$(为了简便，不使用 $q[0]$元素)。

对于(i,j)位置上的皇后，是否与已放好的皇后$(k,q[k])(1\leqslant k\leqslant i-1)$有冲突呢？显然它们不同列，若同列，则有 $q[k]==j$；对角线有两条，如图 5.6 所示，若它们在任一条对角线上，则构成一个等腰直角三角形，即满足条件$|q[k]-j|==|i-k|$。所以只要满足以下条件就存在冲突，否则不存在冲突：

$(q[k]==j)\ ||\ (\text{abs}(q[k]-j)==\text{abs}(i-k))$

设 queen(i,n)是在 $1\sim i-1$ 行上已经放置了 $i-1$ 个皇后，用于在 $i\sim n$ 行放置剩下的

$n-i+1$ 个皇后，则 queen($i+1$,n)表示在 1～i 行上已经放置了 i 个皇后，用于在 $i+1$～n 行放置 $n-i$ 个皇后。显然 queen($i+1$,n)比 queen(i,n)少放置一个皇后，所以 queen(i,n)是大问题，queen($i+1$,n)是小问题，则求解皇后问题所有解的递归模型如下：

queen(i,n)≡n 个皇后放置完毕，输出一个解　　若 $i>n$

queen(i,n)≡在第 i 行找到一个合适的位置(i,j)，放置一个皇后　　其他情况

　　queen($i+1$,n)

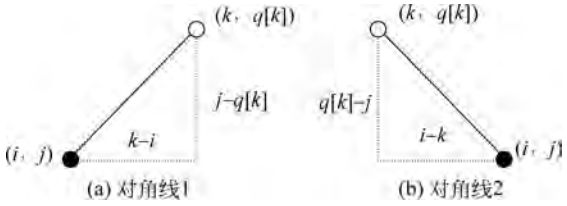

图 5.6　两个皇后构成对角线的情况

【设计代码】

```
class Solution {
    vector<vector<string>> ans;                    //存放所有的解
    int q[12];                                     //存放一个解
public:
    vector<vector<string>> solveNQueens(int n)
    {   queen(1,n);                                //放置 1～n 的皇后
        return ans;
    }
    bool place(int i,int j)                        //测试(i,j)位置能否摆放皇后
    {   if (i==1) return true;                     //第一个皇后总是可以放置
        int k=1;
        while (k<i)                                //k=1～i-1 是已放置了皇后的行
        {   if ((q[k]==j) || (abs(q[k]-j)==abs(i-k)))
                return false;
            k++;
        }
        return true;
    }
    void queen(int i,int n)                        //递归算法:放置 1～i 的皇后
    {   if (i>n)                                   //所有皇后放置结束
        {   vector<string> asolution;              //存放一个解
            for(int j=1;j<=n;j++)
            {   string str(n,'.');                 //存放一个皇后位置的字符串
                str[q[j]-1]='Q';
                asolution.push_back(str);
            }
            ans.push_back(asolution);              //向 ans 中添加一个解
            return;
        }
        for (int j=1;j<=n;j++)                     //在第 i 行上试探每一个列 j
        {   if (place(i,j))                        //在第 i 行上找到一个合适位置(i,j)
            {   q[i]=j;
```

```
                queen(i+1,n);
            }
        }
    }
};
```

【提交结果】

执行结果：通过。执行用时 4ms，内存消耗 8MB(编程语言为 C++语言)。

解法 2

【解题思路】

视频讲解

采用非递归方法求解。用一个栈 st 存放已经放置好的皇后，其元素为(皇后 i 的行号，皇后 i 的列号)($1 \leqslant i \leqslant n$)。

假设已经放置好了编号为 $1 \sim i-1$ 的 $i-1$ 个皇后，它们的位置均存放在栈 st 中，对于其中某个皇后 e，其位置是(e.x，e.y)。现在要放置皇后 i，考查位置(i，j)是否与前面所有的皇后冲突？按照解法 1 可知，只要满足以下条件就说明位置(i，j)存在冲突，不能在该位置放置皇后 i，否则不存在冲突，可以试探放置皇后 i：

((e.y==j) || (abs(e.y-j)==abs(i-e.x)))

首先将第一个皇后位置(1,0)进栈，栈不空时循环，出栈元素 e，置 i=e.x，j=e.y，处理步骤如下：

(1) 在第 i 行中从列 $j=j+1$ 开始找一个适合的位置，如果找到了与前面放置的全部皇后没有冲突的位置(i，j)，则将第 i 个皇后放置在(i，j)位置，将该皇后位置进栈。

① 如果此时 $i==n$，说明 n 个皇后已放置好，得到一个解，将其转换后添加到 ans 中。

② 否则将($i+1$,0)进栈，表示第 $i+1$ 个皇后从 0 列开始找到一个适合的位置。

(2) 如果第 i 行中从列 j 开始没有找到适合的位置，则回到循环开头，出栈下一个皇后继续查找。

例如，$n=4$ 时求全部解的过程是将[1,0]进栈，如图 5.7(a)所示，栈不空时循环：

(1) 出栈[1,0]，找到第一个皇后的合适位置[1,1]，如图 5.7(b)所示。将[1,1]进栈，同时将[2,0]进栈。

(2) 出栈[2,0]，找到第 2 个皇后的合适位置[2,3]，如图 5.7(c)所示。将[2,3]进栈，同时将[3,0]进栈。

(3) 出栈[3,0]，第 3 个皇后找不到合适位置，出栈第 2 个皇后位置[2,3]，找到第 2 个皇后的合适位置[2,4]，如图 5.7(d)所示。将[2,4]进栈，同时将[3,0]进栈。

(4) 出栈[3,0]，找到第 3 个皇后的合适位置[3,2]，如图 5.7(e)所示。将[3,2]进栈，同时将[4,0]进栈。

(5) 出栈[4,0]，第 4 个皇后找不到合适位置；出栈第 3 个皇后位置[3,2]，第 3 个皇后后面也找不到合适位置；出栈第 2 个皇后位置[2,4]，第 2 个皇后后面也找不到合适位置；出栈第一个皇后位置[1,1]，找到第一个皇后的合适位置[1,2]，如图 5.7(f)所示。将[1,2]进栈，同时将[2,0]进栈。

(6) 出栈[2,0]，找到第 2 个皇后的合适位置[2,4]，如图 5.7(g)所示。将[2,4]进栈，同时将[3,0]进栈。

(7) 出栈[3,0],找到第 3 个皇后的合适位置[3,1],如图 5.7(h)所示。将[3,1]进栈,同时将[4,0]进栈。

(8) 出栈[4,0],找到第 4 个皇后的合适位置[4,3],如图 5.7(i)所示。将[4,3]进栈,此时 $i=4$,产生一个解。

(9) 出栈[4,3],第 4 个皇后后面找不到其他合适位置,出栈第 3 个皇后位置[3,1],第 3 个皇后后面也找不到合适位置;出栈第 2 个皇后位置[2,4],第 2 个皇后后面也找不到合适位置;出栈第一个皇后位置[1,2],找到第一个皇后的合适位置[1,3],如图 5.7(j)所示。

(10) 以此类推,找到第 2 个解,如图 5.7(k)所示,再回退找不到其他解,结束。

从中看出,在求解中既有正向搜索,如第一个皇后→第 2 个皇后→第 3 个皇后等,也有回退,如第 3 个皇后没有合适位置→第 2 个皇后,第 2 个皇后再找到下一个合适位置。因此需要保存搜索的路径,而回退过程总是回退到最近的皇后,即后进先出,所以采用栈保存搜索路径。

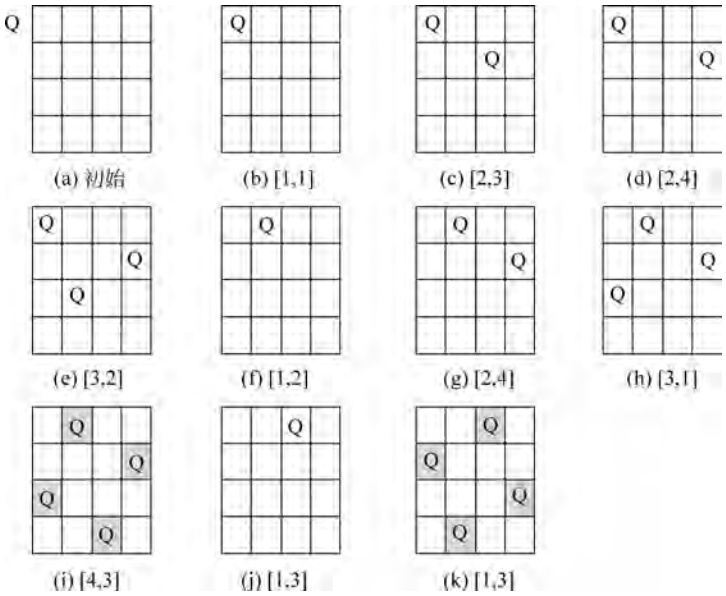

图 5.7 4 皇后求解的过程

【设计代码】

```
struct SNode                                    //栈元素类型
{   int x,y;                                    //存放一个皇后的位置(x,y)
    SNode(int x1,int y1):x(x1),y(y1) { }        //构造函数
};
class Solution {
public:
    vector<vector<string>> solveNQueens(int n)  //非递归算法
    {   int i,j;
        vector<vector<string>> ans;             //存放所有的解
        stack<SNode> st;                        //定义一个栈
        SNode e(1,0);                           //第一个皇后从(1,0)位置开始
        st.push(e);                             //将第一个皇后的(1,0)位置进栈
        while (!st.empty())                     //栈不空时循环
```

```
        {   e=st.top();st.pop();                    //出栈一个皇后
            i=e.x; j=e.y;
            j++;                                    //从下一列开始查找
            while (j<=n)
            {   if(place(i,j,st))                   //(i,j)位置可以放置皇后 i
                {   st.push(SNode(i,j));            //将(i,j)进栈
                    if (i==n)                       //n 个皇后放置好后得到一个解
                        appendans(n,st,ans);        //将这个解转换后添加到 ans 中
                    else                            //n 个皇后没有放完
                        st.push(SNode(i+1,0));      //将下一个皇后(i+1,0)进栈
                    break;
                }
                j++;
            }
        }
        return ans;
    }
    bool place(int i,int j,stack<SNode> st)         //测试(i,j)位置能否摆放皇后
    {   if (i==1) return true;                      //第一个皇后总是可以放置
        while(!st.empty())
        {   SNode e=st.top(); st.pop();
            if ((e.y==j) || (abs(e.y-j)==abs(i-e.x)))
                return false;
        }
        return true;
    }
    void appendans(int n,stack<SNode> st,vector<vector<string>> &ans)   //将 st 中的一个解存
                                                                        //放到 ans 中
    {   vector<string> asolution(n,string(n,'.'));  //存放一个解
        while (!st.empty())
        {   SNode e=st.top(); st.pop();
            asolution[e.x-1][e.y-1]='Q';
        }
        ans.push_back(asolution);                   //向 ans 中添加一个解
    }
};
```

【提交结果】

执行结果：通过。执行用时 68ms，内存消耗 82.6MB(编程语言为 C++语言)。