第 5 章

回　溯　法

5.1　回溯法的算法框架

回溯法有通用的解题法之称，是程序设计中常用的一种算法设计技术。它不是按照某种公式或确定的法则来求问题的解，而是通过试探和纠正错误的策略，找到问题的解。这种方法一般是从一个原始状态出发，通过若干步试探，最后达到目标状态并终止。

从理论上来说，回溯法就是在一棵搜索树中从根结点出发，找到一条达到满足某条件的子结点的路径。在搜索过程中，对于每一个中间结点，它的位置以及向下搜索过程是相似的，因此完全可以用递归来处理。典型的例子有著名的 8 皇后问题等。

5.1.1　明确定义问题的解空间

通常将问题的解空间组织成树或图的形式，用回溯法求解时常遇到的两类典型的解空间树为子集树和排列树。

例 5-1　n＝3 时的 0-1 背包问题的解空间树(子集树)。

0-1 背包的解空间可以用完全二叉树表示，如图 5-1 所示，其中有 8 个叶结点，对应的解分别为{0,0,0}、{0,0,1}、{0,1,0}、{0,1,1}、{1,0,0}、{1,0,1}、{1,1,0}、{1,1,1}。

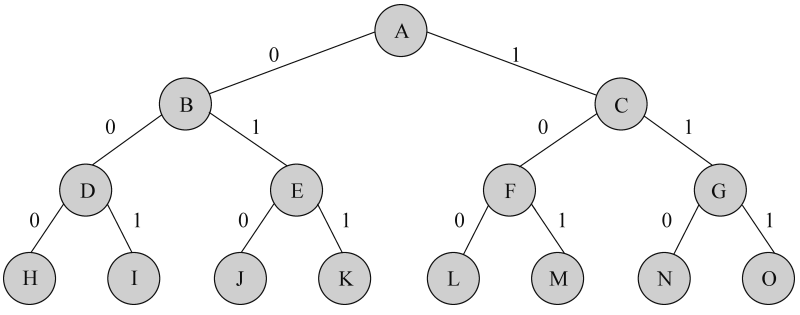


图 5-1　0-1 背包问题的解空间树

设 w＝[16,15,15]，p＝[45,25,25]，c＝30，以深度优先的方式系统地搜索问题的解：从根结点出发，A→ B→D→ H→……

子集树通常有 2^n 个叶结点，其结点总个数为 $2^{n+1}-1$。遍历子集树的任何算法均需 $\Omega(2^n)$。

例 5-2　旅行商问题的解空间树(排列树)。

给定一个有 n 个顶点的带权图 G，如图 5-2 所示，要在 G 中找出一条有最小费用(权)的周游路线。

图 5-3 树中结点的编号按深度优先搜索的顺序给出，树中有 6 个叶结点，表示周游路线有 6 条。

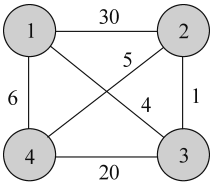


图 5-2 4 个顶点的带权图 G

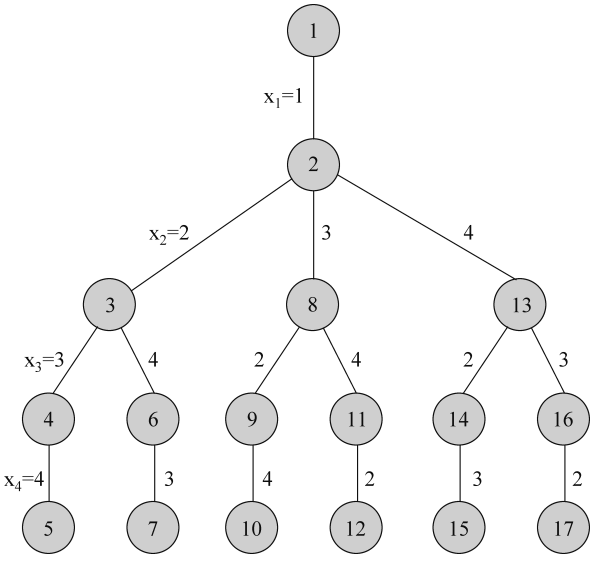


图 5-3 旅行商问题的解空间树

排列树通常有 n!个叶结点，遍历子集树的任何算法均需 Ω(n!)。

5.1.2 运用回溯法解题的步骤

运用回溯法解题的步骤如下：

(1) 针对所给问题，定义问题的空间解；

(2) 确定易于搜索的解空间结构；

(3) 以深度优先的方式搜索解空间，并在搜索过程中用剪枝函数(约束函数 constrain(t)或限界函数 bound(t))避免无效搜索。

5.1.3 回溯法的算法框架

1. 递归回溯的算法框架

递归回溯的算法框架如下：

```
void backtrack(int t)
{
    if (t>n)
        output(x);                    /* 搜索到叶结点，输出 */
    else
```

```
        for (i=f(n,t); i<=g(n,t); i++)
        {
            x[t]=h(i);
            if (constrain(t) && bound(t))
                backtrack(t+1);
        }
}
```

注意：调用一次 backtrack(1)可完成整个回溯搜索过程。其中，

- t 表示递归深度，即当前扩展结点的深度；
- n 表示解空间树的高度；
- f(n,t)表示当前扩展结点处未搜索过的子树的起始编号；
- g(n,t)表示当前扩展结点处未搜索过的子树的终止编号；
- h(i)表示在当前扩展结点处 x[t]的第 i 个可选值；
- constrain(t)表示在当前扩展结点处的约束函数；
- bound(t)表示在当前扩展结点处的限界函数；
- backtrack(t+1)表示对其相应子树作进一步搜索。

2. 迭代回溯的算法框架

如果采用树的非递归深度优先遍历，可将回溯法表示为一个非递归的迭代过程如下：

```
void iterativebacktrack(int t)
{
    int t=1;
    while(t>0)
    {
        if (f(n,t)<=g(n,t))
        for (i=f(n,t);i<=g(n,t);i++)
        {
            x[t]=h(i);
            if (constrain(t) && bound(t))
            {
                if (solution(t))
                    output(x);      /*搜索到叶结点,输出*/
                else t++;
            }
        }
        else
            t--;
    }
}
```

注意：算法的 while 循环结束后可完成整个回溯搜索过程。其中，

- t 表示递归深度，即当前扩展结点的深度；
- n 表示解空间树的高度；
- f(n,t)表示当前扩展结点处未搜索过的子树的起始编号；
- g(n,t)表示当前扩展结点处未搜索过的子树的终止编号；
- h(i)表示在当前扩展结点处 x[t]的第 i 个可选值；
- constrain(t)表示在当前扩展结点处的约束函数；
- bound(t)表示在当前扩展结点处的限界函数；
- solution(t)表示在当前扩展结点处是否已得到问题的一个可行解。

5.2 n 皇后问题

5.2.1 问题描述

n 皇后问题要求一个 n×n 格的棋盘上放置 n 个皇后，使得她们不能相吃。按照国际象棋的规则，一个皇后可以吃掉与她处于同一行、同一列或同一对角线上的任何棋子，因此每一行只能摆放一个皇后。n 皇后问题等价于要求在一个 n×n 格的棋盘上放置 n 个皇后，使得任何两个皇后不能放在同一行、同一列或同一对角线上。

5.2.2 算法设计

对于 n 皇后问题，皇后的位置用一个一维数组 x[1..n]来存放，其中 x[i]表示第 i 行皇后放在第 x[i]列。下面主要是判断皇后是否安全的问题。

(1) 用一维数组 x[1..n]来表示已经解决了不在同一行的问题；

(2) 对于列，当 j!=k 时，要求 x[j]!=x[k]；

(3) 对于左上右下的对角线的每个元素有相同的“行－列”值；对于左下右上的对角线有相同的“行＋列”值。若两个皇后分别占有(j,x[j])和(k,x[k])两个位置，则两个皇后在同一对角线上的条件是：j－x[j]＝k－x[k]或 j＋x[j]＝k＋x[k]，即 j－k＝x[j]－x[k]或 j－k＝x[k]－x[j]。因此，同一对角线上的条件可归纳为|j－k|＝|x[k]－x[j]|。

判断是否可放置一个新皇后的算法如下：

```
int place(int k)
{
    int i;
    for (i=1;i<k;i++)
        if (x[i]==x[k] || abs(x[i]-x[k])==abs(i-k))
            return(0);
    return(1);
}
```

5.2.3 算法实现

下面给出 n 皇后问题的递归回溯算法的实现代码，有兴趣的读者可以自行调试完成 n 皇后问题的迭代回溯算法的实现。

```
void backtrack(int t)
{
    int i,xx;
    if (t>n)
       for (i=1;i<=n;i++)
           printf("%d",x[i]);
    else
       for (i=0;i<n;i++)
        {
           x[t]=i;
           if (t==1 && x[t]>=n/2 )
               break;
           if (place(t)==1)
               backtrack(t+1);
        }
}
```

5.2.4 8 皇后问题的不等效解分析与实现

1. 8 皇后问题的不等效解

会下国际象棋的人都很清楚：皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何在 8×8 的棋盘上放置八个皇后，使它们不能相吃，这就是古老而著名的 8 皇后问题。显然，每行只能摆放一个皇后，问题转换为求每行上皇后所在的列。

该问题是 19 世纪著名的数学家高斯于 1850 年提出的，当时高斯认为有 76 种方案。1854 年在柏林的象棋杂志上不同的作者发表了 40 种不同的解，后来有人用图论的方法解出 92 种结果，如图 5-4 所示，现已证明了结果的正确性。在数学游戏的书常讨论到 8 皇后问题的对称解。在计算机文献方面，Peter Naur 在 1972 年讲解了对称的问题，Rodney W.Topor 在 1982 年用群论讨论过并有 PASCAL 程序。N.Wirth 在其名著《算法＋数据结构＝程序》中编程求得 8 皇后问题的 92 个解之后说："实际上只有 12 个真正不同的解；我们的程序不能识别对称的解。"随后列出这 12 个真正不同的解（即不等效解）。下面以图示的方式总结规律并设计一个有效的求解算法，并用 C 程序加以识别，N.Wirth 那句"我们的程序不能识别对称的解"已成过去式。

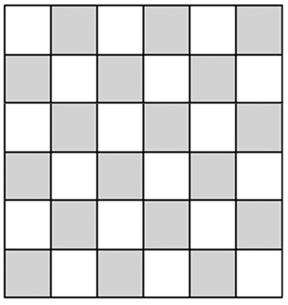
2. 8 皇后问题的不等效解分析

对于 n 皇后问题，可以发现一些解是另一些解的简单反射或旋转的结果。比如，n＝4 时的两个解(1,3,0,2)和(2,0,3,1)在反射意义下是等效的。n＝6 时，其 4 个解等效情

```
QUEEN PROBLEM:  n=8,   total results are:

04752613 25317460 46152037 71306425 31625740 06471352 73025164 52460317
05726314 24175360 36415027 71420635 41362750 06357142 72051463 53602417
13572064 36271405 31750246 27360514 46027531 50417263 64205713 41506372
14602753 52617403 42057136 47306152 35720641 30471625 63175024 25160374
14630752 52613704 52074136 37046152 25703641 40731625 63147025 25164073
15063724 53174602 35041726 57130642 42736051 20647135 62714053 24603175
15720364 26175304 31475026 37420615 46302751 40357162 62057413 51602473
16257403 41357206 47302516 17502463 30475261 60275314 61520374 36420571
16470352 31625704 52470316 37025164 25307461 40752613 61307425 46152073
24170635 35716024 53607142 42061753
24730615 25713064 26174035 31746025 51603742 46031752 53047162 52064713
25147063 46137025 41703625 25704613 36074152 52073164 52630714 31640752

total result of 8 queen:92

its real different result:12
04752613 05726314 13572064 14602753 14630752 15063724 15720364 16257403
16470352 24170635 24730615 25147063
```

图 5-4　8 皇后问题的 12 个不等效解(92 个解中最左列)及对应的等效解(同行右侧)

况如图 5-5 所示。



① 135024

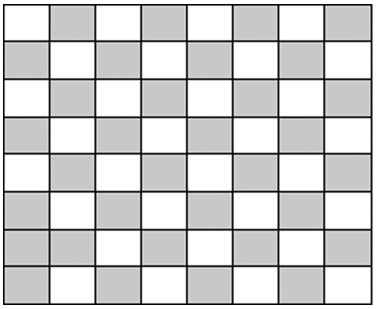
② 251403(左旋90°)

③ 304152 (②的逆)

④ 420531(①的逆)

图 5-5　6 皇后的 1 个不等效解及其 3 个等效解

当 n=8 时,共有 92 个解,其中有 12 个不等效解,如图 5-4 所示。图 5-6 中除第 10 个不等效解演变出 3 个等效解之外,其他 11 个不等效解经棋盘旋转 90°、180°、270°及简单反射后,各可演变出 7 个等效解,如图 5-7 所示。图 5-5、图 5-6 和图 5-7 分别表示了 8 皇后棋子的一个布局,这 3 个图中的多个解只不过是由于看棋盘的角度不同而导致的。



① 24170635

② 35716024(左旋90°)

③ 53607142(①的逆)

④ 42061753(②的逆)

图 5-6　8 皇后第 10 个不等效解 24170635 及其 3 个等效解

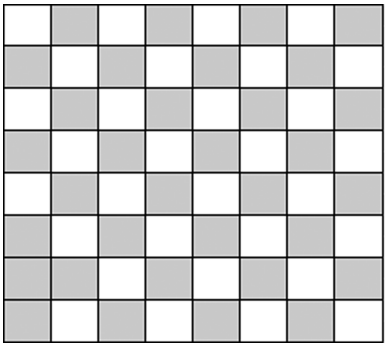


图 5-7　8 皇后第 1 个不等效解 04752613 及其 7 个等效解

对于图 5-5、图 5-6 和图 5-7 不等效解及其所产生的等效解之间的关系，可以发现构成简单反射关系的解正好是一半，只要限制第一个皇后所在的列 x[0]<n/2 即可。而构成旋转关系的解有两种情况，一种只需左旋 90°，另一种需依次左旋 90°、180°、270°。进一步考察属第一种情况的如图 5-5 和图 5-6 所示的不等效解 135024 和 24170635，发现其满足 x[i]+x[n−1−i]=n−1，因此左旋 180°的解与原不等效解一致，左旋 270°的解与左旋 90°的解一致，要给予剔除。另外，当 n=4 时，只有一个不等效解及其构成简单反射关系的一个等效解，不必经过旋转，否则将产生重复的两个解。通过以上分析，可由一个不等效解构造出与其等效的其他所有解。

顺便说明一下，不等效解与其构造出的等效解之间，并非仅仅通过对称(或称简单反射)可找到，所以称它们为等效解更合适，而有的文献中称为对称解。

3. n 皇后问题不等效解的算法实现思路

皇后问题是回溯算法的典型例题，许多算法教材中都给出了解 n 皇后问题的回溯算法，这里进一步讨论的是如何在产生 n 皇后问题不等效解的基础上，通过不等效解构造出与其等效的其他所有解。输出皇后问题不等效解的算法思路如下：

步骤 1. 生成的第一个解必是不等效解，令 count=1，用于标记不等效解的数目，并存储此解到二维数组 b 中，二维数组 b 的第一个下标为 count，第二个下标为 8 个皇后所在的列。另外，用求等效解的函数 dx(见后面程序)生成第一个解的其他 7 个等效解，并用 m 标记已生成 8 个解，同时存储此 8 个解到二维数组 c 中，二维数组 c 的第一个下标为 m，第二个下标为 8 个皇后所在的列。

步骤 2. 生成第二个解开始，每一个解必须与二维数组 c 中已存的解进行 8 个分量的比较，判别其是新的不等效解还是重复出现的等效解，用 test 函数实现。若是新的不等效解，重复步骤 1，否则，仅标记 sum=sum+1。

下面给出按上述算法思想所实现的求皇后问题不等效解的 C 程序。n=8 的运行结果如图 5-4 所示；当 n=4、6、10 时的运行结果如图 5-8 所示。当 n>8 时，解的数目较大，程序中所用数组要调大，以 n=10 为例，得到的不等效解为 102 个。

```
QUEEN PROBLEM:  n=4,   total results are:
1302  2031
total result of 4 queen:2
its real different result:1
1302

total result of 10 queen:804
its real different result:102
```

```
QUEEN PROBLEM:  n=6,   total results are:
135024  251403  420531  304152
total result of 6 queen:4
its real different result:1
135024
```

图 5-8　皇后问题(n＝4、6、10)的不等效解及对应的等效解

算法实现程序如下：

```
#define N 8
int x[100]={0};
int a[N][N],b[2*N][N],c[100][N],row=0;
static int sum=0,count=0,m=1;
void left90(int x[])                    /*产生左旋 90°的等效解*/
{
     int i;
     for (i=0;i<N;i++)
        a[row][N-1-x[i]]=i;
     row++;
}

void left180(int x[])                   /*产生左旋 180°的等效解*/
{
     int i;
     for (i=0;i<N;i++)
        a[row][N-1-i]=N-1-x[i];
    row++;
}

void left270(int x[])                   /*产生左旋 90°的等效解*/
{
    int i;
    for (i=0;i<N;i++)
        a[row][x[i]]=N-1-i;
    row++;
}

void store(int x[],int count)           /*实现存储不等效解*/
{
    int i;
     for(i=0;i<N;i++)
```

```
        b[count][i]=x[i];
}

void dx(int x[])                          /*用于构造等效解的函数 dx*/
{
    int i,j,flag=0;
    for (i=0;i<N;i++)                     /*临时存储用于构造等效解的原不等效解*/
        a[row][i]=x[i];
    row++;
    for (i=0;i<N/2;i++)                   /*判断左旋次数*/
        if (x[i]+x[N-1-i]!=N-1)
        {
            flag=1;
            break;
        }
    if (N>4&&flag==0)
        left90(x);
    else if (flag)
    {
        left90(x);
        left180(x);
        left270(x);
    }
    for (i=0;i<row;i++)                   /*产生简单反射的等效解*/
        for (j=0;j<N;j++)
            a[row+i][N-1-j]=a[i][j];
    printf("\n");
    for (i=0;i<row*2;i++)                 /*输出该不等效解及其产生的所有等效解*/
    {
        printf(" ");
        for (j=0;j<N;j++)
            printf("%d",c[m][j]=a[i][j]);
        m++;
    }
}

int test(int x[])                         /*测试是否为不等效解*/
{
    int i,j;
    for (i=1;i<=m;i++)
    {
        for (j=0;j<N;j++)
            if (c[i][j]!=x[j])
```

```
                break;
        if (j>=N)
            break;
    }
    if (i>m)
        return(1);
    else
        return(0);
}

int place(int k)                          /*所放位置是否有冲突*/
{
    int i;
    for (i=0;i<k;i++)
        if (x[i]==x[k]||abs(x[i]-x[k])==abs(i-k))
            return(0);
    return(1);
}

void backtrack(int t)                     /*回溯求解*/
{
    int i,xx;
    if (t>N-1)
    {
        sum=sum+1;
        if (sum==1||test(x))
        {
            count++;
            store(x,count);
            row=0;
            dx(x);
        }
        if (sum==48) getch();
    }
    else for (i=0;i<N;i++)
    {
        x[t]=i;
        if (t==0&&x[t]>=N/2 )
            break;
        if (place(t)==1)
            backtrack(t+1);
    }
}
```