高斯皇后问题

杨克昌　刘志辉

摘　要：在利用枚举求解著名的高斯 8 皇后问题基础上，分别应用回溯与递归设计求解一般 n 皇后问题，从而拓广了 8 皇后问题。

关键词：高斯；8 皇后问题；回溯；递归

1　高斯 8 皇后问题

国际象棋的棋盘是一个 8 行×8 列共 64 格的方格棋盘，本讲座将涉及一般 n 行×n 列以至 n 行×m 列的广义棋盘。

在国际象棋中，皇后可以吃掉同行、同列或同一与棋盘边框成 45 度角的斜线上的任何棋子，是攻击力最强的。

数学大师高斯 (Gauss) 于 1850 年由此引申出著名的 8 皇后问题：在国际象棋的 8×8 方格的棋盘上如何放置 8 个皇后，使得这 8 个皇后不能相互攻击，即没有任意两个皇后处在同一横排，同一纵列，或同一与棋盘边框成 45 度角的斜线上。

实际上，8 皇后问题是高斯借助国际象棋高度抽象出来的一个形象有趣的组合数学问题。图 1 就是高斯 8 皇后问题的一个解。

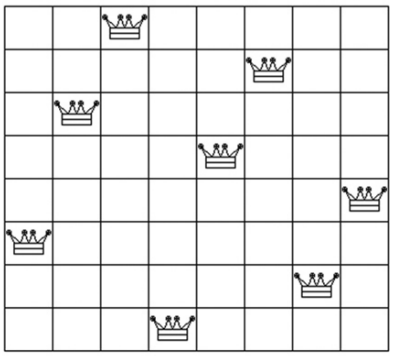

图 1　高斯 8 皇后问题的一个解

图 1 中的 8 个皇后既不同行，也不同列，也没有同处一对角线上，即任意两个皇后都不相互攻击。

2　枚举求解 8 皇后问题

首先从简单开始，试应用“蛮力”策略枚举求解 8 皇后问题。

2.1　8 皇后问题解的表示

试用一个 8 位数表示高斯 8 皇后问题的一个解：8 位数的第 k 个数字为 j，表示方格棋盘上的第 k 行的第 j 格放置一个皇后。由此可知，图 1 所示的解可表示为 36258174。

2.2　枚举算法设计

(1)　设置枚举循环

设置枚举 8 位数的循环变量 a，应用数字 1~8 不重复且从高位开始尽可能使相邻两位数字不相连的策略，确定循环变量 a 的取值范围为区间 [13526847，86473152]。而数字 1~8 的任意一个排列的数字和为 9 的倍数，即数字 1~8 的任意一个排列均为 9 的倍数，因而枚举循环步长可确定为 9。

(2)　任两个皇后不允许处在同一横排，同一纵列

要求 8 位数中数字 1~8 各出现一次，不能重复。

设置 f 数组，分离 a 的 8 个数字，用 f (x) 统计 a 中数字 x 的个数。若 f (1) ~f (8) 均等于 1，即数字 1~8 在 a 中各出现 1 次。否则返回。

(3)　任两个皇后不允许处在同一斜线上

设置 g 数组，若 a 的第 k 个数字为 x，则 g (k) =x。要求解的 8 位数的第 j 个数字与第 k 个数字的绝对值不等于 j–k (设置 j>k)。若出现

$$|g(j)-g(k)|=j-k$$

表明 g (j) 与 g (k) 同处在与棋盘边框成 45°角的斜线上，则返回。

(4)　输出解

在枚举范围内同时通过以上 (2) (3) 两道筛选的 8 位数即为一个解，打印输出 (每行打印 5 个解)，同时用变量 sum 统计解的个数。

2.3　枚举求解 8 皇后问题程序

```
// 枚举求解高斯 8 皇后问题
#include <iostream.h>
#include <cmath>
void main()
{
 int sum,x,f[9],g[9];
 long y;      bool succeed;
 sum=0;
```

```
    cout<<" 高斯 8 皇后问题的解为 :"<<endl;
    for  (long  a  =13526847;a  <=86473152;a  =a  +9)

// 步长为 9 枚举 8 位数
      {
        succeed=true;
// 初始化循环变量满足解的要求
        for(int k=1;k<=8;k++) f[k]=0;
        for(y=a,k=1;k<=8;k++)
        {
            x=y%10;y=y/10;
  f[x]++;g[k]=x;    // 分离 a 各个数字并用 f,g 数组统计
        }
        for(k=1;k<=8;k++)
            if(f[k]>1 || f[k]==0)
  {succeed=false;break;}  // 出现数字 1--8 不为 1 次,返回
        if(! succeed) continue;
        for(k=1;k<=7;k++)
            for(int j=k+1;j<=8;j++)
    if(abs(g[j]-g[k])==j-k)  // 同处在 45 度角的斜线上,返回
                        {succeed=false;k=7;break;}
        if(! succeed) continue;
        cout<<a<<" ";          // 输出 8 皇后问题的解
        if(++sum%5==0) cout<<endl;
      }
     cout<<endl<<" 高斯 8 皇后问题共有以上"<<sum<<"个
解。"<<endl;
    }
```

2.4　程序运行与说明

高斯 8 皇后问题的解为：

15863724　16837425　17468253　17582463　24683175

25713864　25741863　26174835　26831475　27368514

…

74258136　74286135　75316824　82417536　82531746

83162574　84136275

高斯 8 皇后问题共有以上 92 个解。

枚举设计简单而有效，从程序的实际运行看，搜索时间还是可以接受的。

如果枚举范围简单地定为 [12345678,87654321] ,循环步长定为 1，也是可行的，只是增加了循环次数。尽管以上枚举设计注意在枚举范围与循环步长的确立上进行了适当优化处理，枚举循环量依然较大，且对每一个循环变量 a 均需进行 8 个数字的分离统计，并实施两道筛选，操作量大，导致求解效率比较低。

之所以首先选用枚举设计，说明应用简单的“蛮力”策略也可以求解这些经典名题，同时为下面效率较高的回溯与递归设计提供参照。

3　回溯求解 n 皇后问题

3.1　问题提出

把国际象棋棋盘从 8 行×8 列拓广到一般 n 行×n 列，要求在 n×n 棋盘上放置 n 个皇后使它们互不攻击，即成为 n 皇后问题。它是 8 皇后问题的直接推广。

根据键盘输入的正整数 n （n>3），试求出 n 皇后问题的所有解。

3.2　回溯设计要点

设置数组 q_pos (n)，数组元素 q_pos (cur_row) 表示当前行 cur_row 的皇后位于第 q_pos (cur_row) 列。

求 n 皇后问题的一个解，即寻求 q_pos 数组的一组取值，该组取值中每一元素的值互不相同 (即没有任两个皇后在同一行或同一列)，且第 cur_row 个元素与第 k 个元素相差不为 |cur_row-k|，(即任两个皇后不在同一 45°角的斜线上)。

问题的解空间是由整数 1~n 组成的 n 项数组，其约束条件是没有相同整数且每两个整数之差不等于其所在位置之差。

在设置的永真循环中，q_pos (cur_row) 从 1——n 范围内取一个值。

为了检验 q_pos (cur_row) 是否满足上述要求，设置标志变量 succeed，succeed 赋初值 true。q_pos (cur_row) 逐个与其前面的元素 q_pos (k) 比较:

若 q_pos (cur_row) =q_pos (k) or |q_pos (cur_row) -q_pos (k) |=cur_row-k ,即出现相列或同处一对角线，标注 succeed=false;

若出现 succeed 为假，则表明 q_pos (cur_row) 不满足要求，q_pos (cur_row) 增 1 后再试，依此类推。

若 cur_row=n 且 succeed 为真，则满足要求，输出这组解，并用 sum 统计解的个数。若 n>10，输出解的各数值间用空格或“,”隔开。

若 cur_row<n 且 succeed 为真，表明 q_pos (cur_row) 满足要求但数组 q_pos 的取值还不到 n 个数，则 cur_row 增 1 后，q_pos (cur_row) 从 1 开始赋值继续。

若 q_pos (n) =n，则回溯到前一个数组元素 q_pos (n-1) 增 1 赋值 (此时，q_pos (n) 又从 1 开始) 再试。

若 q_pos (n-1) =n，同样回溯到前一个数组元素 q_pos (n-2) 增 1 赋值再试。

一般地，若 q_pos (cur_row) =n (cur_row>1)，则回溯到前一个数组元素 q_pos(cur_row-1) 增 1 赋值再试。

直到 q_pos (1) =n 时，已无法回溯，意味着已完成回溯试探，求解结束。

3.3　回溯求解 n 皇后问题程序

```
// 回溯求解 n 皇后问题
```

```
#include <iostream.h>
#include <cmath>
void main()
{
 int cur_row,n,q_pos[20];
 long sum=0; bool succeed;
 cout<<"请输入整数 n(n>3):";
 cin>>n;
 cout<<n<<"皇后问题的解:"<<endl;
 cur_row=1;q_pos[1]=1;
 while (1)
   {
      succeed=true;
      for(int k=cur_row-1;k>=1;k--)
            if(q_pos[cur_row]==q_pos[k] ||
abs(q_pos[cur_row]-q_pos[k])==cur_row-k)
                  {succeed=false;break;}
// 同列或同处一对角线上时返回
      if(succeed && cur_row==n)
// 已有 n 个数字且满足条件时输出解
      {
            for(int j=1;j<=n;j++)
cout<<" "<<q_pos[j];  // 输出解的数值间用空格隔开
            cout<<"   ";
            if(++sum%5==0) cout<<endl;
      }
      if(cur_row<n && succeed)
            {cur_row++;q_pos[cur_row]=1;continue;}
      while (q_pos [cur_row] ==n && cur_row >1)
cur_row--; // 往前回溯
      if(q_pos[cur_row]==n && cur_row==1) break;
      else q_pos[cur_row]=q_pos[cur_row]+1;
 }
   cout<<endl<<"共"<<sum<<"个解."<<endl;
}
```

3.4 程序运行示例与说明

请输入整数 n (n>3) :6

6 皇后问题的解:

2 4 6 1 3 5　3 6 2 5 1 4　4 1 5 2 6 3　5 3 1 6 4 2

共 4 个解.

运行程序若输入 n=8，即输出高斯 8 皇后问题的所有 92 个解。

4 递归求解 n 皇后问题

注意到递归也具有回溯功能，应用递归自然也可以求解 n 皇后问题。

4.1 递归设计要点

递归函数 put (cur_row) 的设计针对 n 皇后问题数字解的 n 个整数中的第 cur_row 个整数 q_pos (cur_row) 展开。

设 q_pos (cur_row) 取值为 i (1,2,…,n)，q_pos (cur_row) 逐一与已取值的 q_pos (k) (k=1,2,,cur_row-1) 比较：

若满足 q_pos (cur_row) =q_pos (k) or |q_pos (cur_row) - q_pos (k) |=cur_row-k，即同行或同列或同对角线，显然不符合题意要求，记 succeed=false, 即所取 q_pos (cur_row) 不妥，表示该行该列已放不下皇后，于是 q_pos (cur_row) 继续下一个 i 取值。

否则，符合题意要求，保持 succeed 为真，即所取 q_pos (cur_row) 妥当。此时检测所完成的行数：

若 cur_row=n 成立，完成了 n 行，按格式输出一个数字解，并用 sum 统计解的个数；

若 cur_row =n 不成立，未完成 n 行，继续调用 put (cur_row+1)，探讨下一行取值。

若 q_pos (cur_row) 取值到 n 仍不妥，则返回（回溯）到调用 put (cur_row) 的 put (cur_row-1) 环境下，继续 q_pos (cur_row-1) 的下一个取值。

最后若 q_pos (1) 取值到 n 仍不妥，则返回到调用 put (1) 的主程序，输出解的个数 sum 或“无解”,程序结束。

4.2 递归程序

```
// 递归求解 n 皇后问题主程序
#include <iostream.h>
#include <cmath>
int n,q_pos[30];
long sum=0;
// n 皇后问题递归函数
void put(int cur_row)
{
 if(cur_row<=n)
 {
  bool succeed;
      for(int i=1;i<=n;i++)
// 探索当前行 cur_row 第 i 格放置皇后
      {
            q_pos[cur_row]=i; succeed=true;
            for(int k=1;k<=cur_row-1;k++)
                  if(q_pos[cur_row]==q_pos[k] ||
   abs(q_pos[cur_row]-q_pos[k])==cur_row-k )
      {succeed=false;break;}
// 若第 i 格放不下,则 succeed 为假
   if(succeed)    // 若当前行第 i 格可放,则检测是否满 n 行
            {
if(cur_row==n)       // 若已放满到 n 行时,则打印出一个解
                  {
                        for (int j=1;j<n;j++)
                              cout<<q_pos[j]<<",";
                        cout<<q_pos[n]<<"  ";
                        if(++sum%5==0) cout<<endl;
                  }
```

```
                    else  put(cur_row+1);
// 若没放满 n 行,则继续放下一行
                }
            }
        }
    }
    // 主调函数
    void main()
    {
     cout<<"求解 n 皇后问题,请确定 n(n>3): ";
     cin>>n;
       put(1);              // 从第 1 行开始放置皇后
     cout<<endl<<n<<"皇后问题共"<<sum<<"个解。"<<endl;

    }
```

4.3　程序运行示例与说明

运行程序，输入 n=8，可得 8 皇后问题的 92 个解。

求解 n 皇后问题，请确定 n (n>3) : 5

1,3,5,2,4　1,4,2,5,3　2,4,1,3,5　2,5,3,1,4　3,1,4,2,5

3,5,2,4,1　4,1,3,5,2　4,2,5,3,1　5,2,4,1,3　5,3,1,4,2

5 皇后问题共 10 个解。

比较以上 3 种设计的搜索效率，枚举设计的时间复杂度最高，为 O (10n)；回溯设计与递归设计的时间复杂度都难以用一个关于 n 的确切数量级估算，从实际运行的时间对比，其搜索效率确实高于枚举。

由于以上回溯与递归的搜索流程与操作步骤（含回溯）基本相同，因而这两个算法的搜索效率也基本相等，从实际运行（选择比较大的 n 值）的时间来看也证实了这一点。

参考文献

[1] 谷峰，吴勇，唐俊. 基于遗传算法的 n 皇后问题求解 [J] . 宿州教育学院学报，2002，(04).

[2] 黄建民，罗杰. 八皇后问题的非递归算法设计 [J] .计算机与现代化，2004，(05).

[3] 周康，同小军，许进. 基于闭环 DNA 模型的八皇后问题算法 [J] .计算机工程与应用，2007，(06).

[4] 李鸿. 解决 N—皇后问题的一个遗传算法 [J] . 宿州师专学报，2000，(04).

[5] 杨克昌，刘志辉. 趣味 C 程序设计集锦. 中国水电水利出版社，2010.

（收稿日期：2010–11–21）

（上接第 78 页）

```
   protected void Page_Load(object sender, EventArgs e)
   {
      string sqlStr = "select * from u_tablet where reference-
Type='"
      +Request.QueryString ["referenceType"]+"' and referen-
ceID='"
      + Request.QueryString["referenceID"] + "' ";
       DataTable dtTablet=ExecuteSql(sqlStr);
      HttpContext.Current.Response.Clear();
      HttpContext.Current.Response.Buffer = true;
      HttpContext.Current.Response.Charset = "UTF-8";
        HttpContext.Current.Response.ContentEncoding    =
System.Text.Encoding.UTF8;
       HttpContext.Current.Response.ContentType = "image/
png";
      HttpContext.Current.Response.BinaryWrite ((byte[])dt-
Tablet.Rows[0]["img"]);
      HttpContext.Current.Response.End();
   }

     //SQL 语句执行方法
   private DataTable ExecuteSql(string sqlStr)
   {
        SqlConnection objConn = new SqlConnection();
        objConn.Conne ctionString =  ConfigurationManager.
AppSettings["connStr"];
        objConn.Open();
        DataSet objDs = new DataSet();
        SqlDataAdapter objAdapter = new SqlDataAdapter
(sqlStr, objConn);
        objAdapter.Fill(objDs);
        return objDs.Tables[0];
   }
}
```

4　结语

所开发的程序本身结构比较简单，难度也不大，但文中用到了一些不太常用但又非常有用的技术，例如控件的封装、Flex。这些技术的掌握和灵活应用能使开发更加得心应手，希望能对大家有所帮助。

参考文献

[1] 邱彦林. Flex 第一步——基于 ActionScript3.0 的 Flex 2 应用开发. 清华大学出版社，2007.

[2] 吕晓鹏. 精通 Flex 3.0——基于 ActionScript3.0 实现. 人民邮电出版社，2008.

[3] 张跃延，苏宇，贯伟红. ASP.NET 程序开发范例宝典 (C#) . 第 2 版. 人民邮电出版社，2009.

（收稿日期：2010–11–07）