3. 比特匠心&模拟退火

1. 引言

一场充满“高科技”气息的热身晚会结束了。大家这才想起营会还没有名字呢。经过一番激烈的讨论，最后由自诩为“计算机界写诗最好”的徐老师一锤定音，就叫“比特匠心”吧，我们不就是传说中善用 0 和 1 思考的锦鲤（~~码农~~）吗？

于是“比特匠心”们开始思考经典的八皇后问题。这个古老的问题大家都不陌生，就是在 8*8 的棋盘上放置八个皇后（Queen），使得棋盘上任意两个皇后都不会互相攻击。根据国际象棋规则，每个皇后会攻击到与自己在同一行（列），同一左斜（右斜）对角线上的其他皇后。下图是八皇后问题的一个解，可以用向量表示为：`[f, d, g, a, h, b, e, c]`，分别表示每行的皇后位置（列号）。

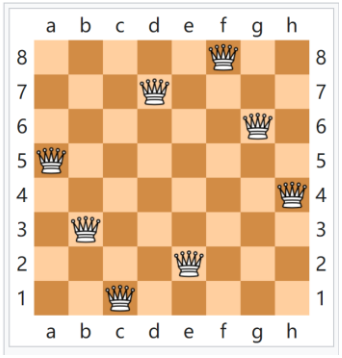

图 1. 八皇后问题的一个解

八皇后问题可以扩展为 n-queens 问题，在 n*n 的棋盘上放置 n 个皇后，n 称为问题的阶。下图是 1~27 阶问题的所有可能解，**你能从中发现什么现象？**

n	fundamental	all
1	1	1
2	0	0
3	0	0
4	1	2
5	2	10
6	1	4
7	6	40
8	12	92
9	46	352
10	92	724
11	341	2,680
12	1,787	14,200
13	9,233	73,712
14	45,752	365,596
15	285,053	2,279,184
16	1,846,955	14,772,512
17	11,977,939	95,815,104
18	83,263,591	666,090,624
19	621,012,754	4,968,057,848
20	4,878,666,808	39,029,188,884
21	39,333,324,973	314,666,222,712
22	336,376,244,042	2,691,008,701,644
23	3,029,242,658,210	24,233,937,684,440
24	28,439,272,956,934	227,514,171,973,736
25	275,986,683,743,434	2,207,893,435,808,352
26	2,789,712,466,510,289	22,317,699,616,364,044
27	29,363,495,934,315,694	234,907,967,154,122,528

图 2. N 皇后问题的所有可能解

求 n-queens 问题的所有可能的解的个数，其最为传统的解法就是回溯法（back-tracking）。该算法的主要思路是每行放置一个皇后，不断试探，不行重来。最重要的步骤就是如何判断最新放入棋盘的皇后是否与已有的皇后冲突。

许多算法教材只给出了比较经典的方法。但是不要忘了，我们是“比特匠心”哦！所以，必须求问：如果使用位运算，是否可以找到一种更为优雅的解法呢？让我们先通过下面的程序，回顾一下位运算吧。

```cpp
#include <iostream>
using namespace std;

main() {
   unsigned int a = 60;   // 60 = 0011 1100
   unsigned int b = 13;   // 13 = 0000 1101
   int c = 0;

   c = a & b;             // 12 = 0000 1100
   cout << "Line 1 - Value of c is : " << c << endl ;

   c = a | b;             // 61 = 0011 1101
   cout << "Line 2 - Value of c is: " << c << endl ;

   c = a ^ b;             // 49 = 0011 0001
   cout << "Line 3 - Value of c is: " << c << endl ;

   c = ~a;                // -61 = 1100 0011
   cout << "Line 4 - Value of c is: " << c << endl ;

   c = a << 2;            // 240 = 1111 0000
   cout << "Line 5 - Value of c is: " << c << endl ;

   c = a >> 2;            // 15 = 0000 1111
   cout << "Line 6 - Value of c is: " << c << endl ;

   return 0;
}
```

图 3. 位运算 C++程序

上面程序只给出了位运算的基本用法。下面程序片段中的两行位运算的代码，是位运算的比较高级的用法。**你能看出他们分别实现了什么功能吗？**

```cpp
int x, y, r;
r = y ^ ((x ^ y) & -(x < y));
for (r = 0; x != 0; ++r)     x &= x - 1;
```

图 4. 奇妙的位运算。

上述代码节选自 MIT 公开课的第 3 课 Bit Hacks，斯坦福大学有一个更详细的教程。

2. 八皇后问题与位运算

我们接下来看看如何将位运算运用到解决八皇后问题中去。本部分内容来自网络，有所修改。在此对原作者表示衷心感谢:)

首先，我们的基本思路与传统一样，每一行只能放并且必须放一个皇后。每次放置一个皇后之后，它就会对后面所有行中，可能放置皇后的位置产生影响。如下图所示，我们已经放置了三个皇后 Q1，Q2，Q3。 接着在下面一行放置 Q4 时，用彩色标注出的区域都是不可行的方案，会与已经放置的三个皇后产生冲突。

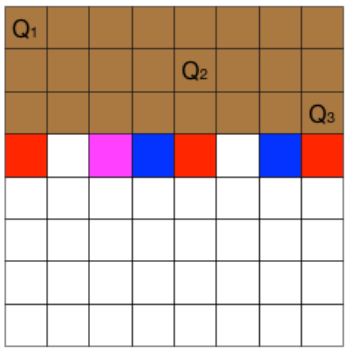

图 5. 表达第四行有无冲突的二进制数（本图来源于网络）

这个冲突其实有三种不同的情况，我们用三个变量 A，B，C 分别表示这三种不同的冲突。这三个变量都是一个八位的二进制数，这八位中，为 1 的表示有冲突，为 0 则表示没有冲突。

(1)与已放置的皇后处于同一列中；

我们用 A 表示这种冲突。例如在图中，第四行会有三个位置因为列攻击而不能再放置皇后（图中大红色方块），因此 A = 1000 1001。

(2)与已放置的皇后处于同一左斜(指向右下方)对角线中；

我们用 B 表示这种冲突。例如在图中，蓝色方块所表示的位置，分别是由于 Q1 和 Q2 的左斜对角线上的攻击而不能够放置新的皇后，因此 B = 0001 0010。

(3)与已放置的皇后处于同一右斜(指向右上方)对角线中。

我们用 C 表示这种冲突。例如图中粉色方块，是由 Q2 的右斜对角线的攻击造成的。因此 C = 0010 0000。

有了 A，B，C 三个变量，我们很容易求出当前行(第 4 行)中，有哪些位置是可以放置皇后的。这个变量我们用 D 来表示，我们可以写出 D = ~(A|B|C)。D 中为 1 的那些位置则是我们可以放置皇后的位置。上图中，D = 0100 0100。此时，我们有两个可能的放置皇后的位置，**那么如何取出每个可能的位置呢?**

这里有一个技巧。我们用 bit 表示可能的一个位置，使用 bit = D & (-D) 即可取出 D 中最右边的 1（LSB，Least Significant Bit）。例如 D = 0100 0100，则-D = 10111100（注意计算机中的数都是补码，-D = ~D + 1）。而 bit = D & (-D) = 0000 0100 取出了最右边的那个 1，代表在第 4 行第 6 列放置一个皇后。那么在此基础上，如何确定第 5 行的 A、B、C 的值呢？

其实也很简单，对于表示列冲突的变量 A 来说，只要使用 A|bit 即可表示下一行的列冲突情况，对于 B，采用(B|bit)<<1，对于 C，采用(C|bit)>>1 即可表示下一行中，对角线冲突的情况。

上述算法的思路来自于 Martin Richards 的论文。请编写 C 语言程序，求出 4~16 阶(通过命令行输入)问题的所有可能解的个数。**（35 分）**

美中不足的是，此算法仅能求出 8 皇后问题的所有解的个数。请对其进行修改，仍采用回溯法，按图 6 的形式输出 4~16 阶(通过命令行输入)问题的一个可行解。#代表空位，Q 代表皇后。**（15 分）**有余力的同学，可以开发类似于图 1 的图形界面(☆▽☆)。

```
+---+---+---+---+
| # | # | Q | # |
+---+---+---+---+
| Q | # | # | # |
+---+---+---+---+
| # | # | # | Q |
+---+---+---+---+
| # | Q | # | # |
+---+---+---+---+
```

图 6. 四皇后问题的一个可行解

3. 求 n-queens 问题可行解的多项式时间算法（20 分）

本算法是由 Rok Sosic 和 Jun Gu 在 1990 年首先提出的，其灵感来源于对图 2 中可行解数量的考察，从图 2 中可以看出：**可行解数量众多**且均匀分布于整个状态空间（棋盘上皇后的所有可能的 n!个排列)中。

4. 使用模拟退火解决 N-queens 问题（30 分）

有关模拟退火的讲解请参考以下资源：

知乎文章：https://zhuanlan.zhihu.com/p/266874840

油管：https://www.youtube.com/watch?v=XNMGq5Jjs5w

更加详细：https://www.youtube.com/watch?v=dg5zUxdAE_E