# 《程序设计进阶与实践》实验报告

| 姓名 | 王昱 | 学号 | PB21030814 | 日期 | 2022.4.10 |
|---|---|---|---|---|---|
| 实验名称 | N 皇后问题 | | | | |

实验环境：
CPU：Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz  2.21 GHz
内存：16.0 GB (15.8 GB 可用)
操作系统：Windows 11 家庭中文版
软件平台：Visual Studio Code 1.65.2

## 一、 问题分析与求解思路

采用最小冲突法来解决 N 皇后问题，具体如下：

① 初始化 N 个皇后的位置，并允许位置有冲突。

② 考虑某行（或者某列）的一个皇后，计算它的冲突数，然后与该行（或该列）的其他位置比较看哪个冲突数更小，并将皇后移动到冲突数最小的位置。

③ 不断地执行②，直到没有冲突为止。

## 二、核心代码说明

```
void Initialize(void)
{
    for (int i = 0; i < N; i++)
    {
        QueenPosition[i] = i;
    }
    Randomize(QueenPosition, 0, N);
    for (int i = 0; i < N; i++)
    {
        Row[i] = 1;
        Col[i] = 1;
    }
    for (int i = 0; i <2* N-1; i++)
    {
        MainDiagonal[i] = 0;
        CounterDiagonal[i] = 0;
    }
    for (int i = 0; i < N; i++)
    {
        MainDiagonal[GetMainDiagonalPosition(i, QueenPosition[i])]++;
        CounterDiagonal[GetCounterDiagonalPosition(i, QueenPosition[i])]++;
    }
}
```

**① 初始化函数：**

先解释各个数组的含义：**QueenPosition[row]=col 代表第 row 行 col 列有皇后；MainDiagonal[i]代表编号为 i**（以 row-col+N-1 为编号）**的主对角线**（左上到右下的对角线）**上的皇后数；CounterDiagonal[i]代表编号为 i**（以 row+col 为编号）**的副对角线**（右上到左下的对角线）**上的皇后数；Row[i]代表第 i 行上的皇后数；Col[i]代表第 i 列上的皇后数。**

再解释调用的函数：**Randomize 函数**的作用是将 N 个皇后打乱，也就是随机排成

一种形式(可冲突)；**GetMainDiagonalPosition 和 GetCounterDiagonalPosition 函数**是为了得到主副对角线编号。

函数功能实现过程：使用循环赋值让第 i 行第 i 列有皇后，然后调用 Randomize 函数随机排列，由于每行和每列均有且仅有一个皇后故将每行和每列的皇后数赋值为 1，主对角、副对角线根据皇后位置进行计算，如果在一个对角线上出现皇后则该编号的对角线对应值加一。

```
bool Adjust(int Row)
{
    int CurrentCol = QueenPosition[Row];
    int PerfectCol = CurrentCol;
    int MinConflict = Col[PerfectCol] +
     MainDiagonal[GetMainDiagonalPosition(Row, PerfectCol)] - 1 +
      CounterDiagonal[GetCounterDiagonalPosition(Row, PerfectCol)] - 1;
    for (int i = 0; i < N; i++)
    {
        if (i == CurrentCol)
        {
            continue;
        }
        int Conflict = Col[i] +
         MainDiagonal[GetMainDiagonalPosition(Row, i)] +
         CounterDiagonal[GetCounterDiagonalPosition(Row, i)];
        if (Conflict < MinConflict)
        {
            MinConflict = Conflict;
            PerfectCol = i;
        }
    }
    if (PerfectCol != CurrentCol)
    {
        Col[CurrentCol]--;
        MainDiagonal[GetMainDiagonalPosition(Row, CurrentCol)]--;
        CounterDiagonal[GetCounterDiagonalPosition(Row, CurrentCol)]--;
        Col[PerfectCol]++;
        MainDiagonal[GetMainDiagonalPosition(Row, PerfectCol)]++;
        CounterDiagonal[GetCounterDiagonalPosition(Row, PerfectCol)]++;
        QueenPosition[Row] = PerfectCol;
        if (Col[CurrentCol] == 1 &&
         Col[PerfectCol] == 1 &&
         MainDiagonal[GetMainDiagonalPosition(Row, PerfectCol)] == 1 &&
         CounterDiagonal[GetCounterDiagonalPosition(Row, PerfectCol)] == 1)
        {
            return A_Solution();
        }
    }
    return false;
}
```

② **调整行函数：**利用最小冲突算法调整行

**函数参数是待调整的行**。将当前列记为 CurrentCol；最优的列记为 PerfectCol 并把当前列的值赋给它；

**最小的冲突记为 MinConflict**，它的初值为 Row 行 Perfect 列的主对角线上皇后数-1+ Row 行 Perfect 列的副对角线上皇后数-1+Perfect 列上皇后数（对角线皇后数都要减一是因为当前 Perfect 列上有皇后，避免重复算同一个皇后）；

**将该行其他列的冲突记为 Conflict**，它的值为当前列上的皇后数+该行、列主对角线上的皇后数+该行、列副对角线上的皇后数（不需要减去 1 是因为一行只有一个皇后）。

**如果 Conflict 小就把当前列赋给 Perfect；如果 Perfect 和 Current 列不一致，则需要将皇后移到 Perfect 列上，并改变相应的数组元素。**

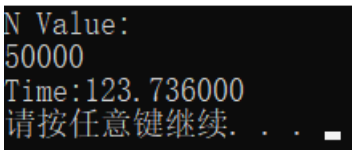
如果交换后 Row 行 Perfect 列上的皇后不与其他皇后冲突并且 Current 列上有一个皇后，那么就调用 A_Solution 函数（就是简单通过 for 循环和 if 语句来判断是否符合解条件的函数）对整个棋盘进行判定；否则返回 false。

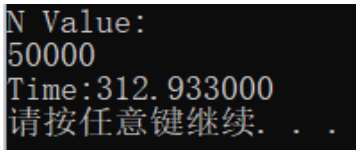
```
int main()
{
    srand((unsigned)time(NULL));
    printf("N Value:\n");
    scanf("%d", &N);
    Initialize();
    // 如果第一次初始化就成功，不需要调整行
    if (A_Solution())
    {
        Print();
        system("pause");
        return 0;
    }
    bool CanExit = false;
    while (!CanExit)
    {
        for (int i = 0; i < N; i++)
        {
            if (Adjust(i))
            {
                CanExit = true;
                break;
            }
        }
    }
    Print();
    system("pause");
    return 0;
}
```

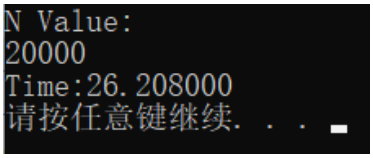
**③ 主函数：**

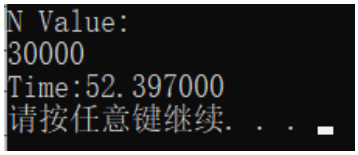
输入 N 规模之后，先初始化数组。A_Solution 函数是判断是否为解的一个函数，如果初始化就找出解则直接输出；如果不能，则进入 while 循环不断调整行中皇后的位置直到不发生冲突。

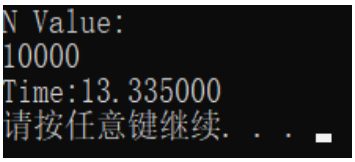
## 三、测试、运行与分析

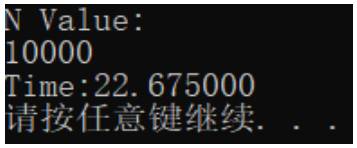


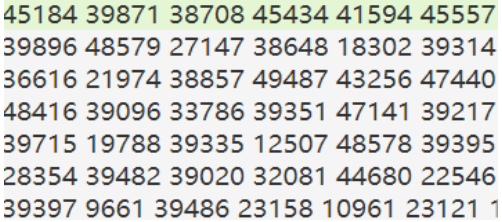



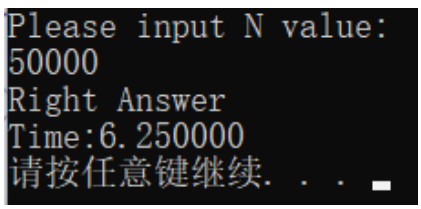



```
Please input N value:
50000
Wrong Answer
Time:0.989000
请按任意键继续. . .
```

**分析：**

由于初始化不同，导致在运行同一个 N 值时所用的时间不同。这里 N 的量级是 $10^4$。输出的格式大概就是如图所示。

验证解时第一个图片验证的是文件中的输出的解（所以是正确的）；第二个是在**修改一个文件中数值之后**验证的。（所以是错误的）

**测试的一些数据：**在 **N=1000** 的时候，调整一次（即调用 Adjust 函数）仅需要几毫秒；在 **N=5000** 时，调整一次大概要 0.1s；在 **N=10000** 的时候，调整一次大概要 0.4s；在 **N=15000** 的时候，调整一次大概要 1s；在 **N=20000** 的时候，调整一次大概要 1.6s；在 **N=25000** 的时候，调整一次大概需要 2.6s；在 **N=30000** 的时候，调整一次大概需要 3.7s；在 **N=35000** 的时候，调整一次大概需要 5.1s；在 **N=40000** 的时候，调整一次大概需要 6.7s；在 **N=45000** 的时候，调整一次大概需要 8.5s；在 **N=50000** 的时候，调整一次大概需要 10.5s。也就是说 N=50000 时调整 30 次可以符合时间要求。N>50000 的时候，由于调整时间的限制，在 5min 之内跑出来的概率比较小。

**四、备注**

查阅文献可知：**当 N 的值足够大时（例如 10000），第一次随机排列总是能通过调整找到一组解，且调整的次数几乎保持不变。**

输出的格式是：按照行递增的顺序依次输出每行中皇后所在列的位置，中间以空格隔开。

例如：四皇后输出为：2 0 1 3

**总结**

N 皇后是比较经典的问题，相关的算法有很多，比如回溯法、爬山法、遗传法、模拟退火法、最小冲突法、构造法等等。本次采用了最小冲突法是因为算法本身理解起来较为容易，且 N 值的规模做起来不算太小。

通过此次实验回顾了文件相关的读写操作、C 语言产生随机数的操作等。

学习了洗牌算法，将 N 皇后的位置随机排成一种形式。

本程序也存在一些不足，比如每次调整时间比较长，导致 N 的大小没有再提升一个量级。在查阅文献的时候也看到一些方法，比如在随机初始化的过程中每行随机产生一个位置，让 N-c 个皇后不冲突其余 c 个皇后可冲突，但不知道怎么具体实现。另外由于只会 C 语言，对语言没有更多的选择，在一定程度上也限制了 N 的规模。

| | |
|---|---|
| 另附（源代码文件名称） | N Queen.c<br>N Queen Test.c |