

算法思想：

首先分析一下 $2N$ 皇后问题中的 N ：

1. 当 N 为偶数时：其实只要求得一组可行解即可，另外一组可行解可以由当前解沿 $N \times N$ 矩阵的中轴线作对称变换得到。因为 N 为偶数，所以不会存在黑白皇后位置冲突的情况。

例如： $N=4$ 时，求得一组可行解为：3 1 4 2，按着这个思路，变换得到另一组可行解为：2 4 1 3。可以判断，这样的两组解合并起来是符合题目要求的。

2. 当 N 为奇数时：沿中轴线对称的方式不再可行，因为肯定有一个皇后会落在中轴线上。此时，可以考虑通过中心点作点对称的情况。

这里要注意，如果求得的可行解存在关于中心点对称的皇后摆放(或有皇后位于中心点)，那么此解不合要求，需要重新再求；直到没有两个皇后的位置是关于中心点对称，另一组解可以通过对当前解关于中心点作点对称变换得到。

例如， $N=5$ 时，求得一组可行解为：2 4 1 3 5，按着这个思路，变换得到另一组可行解为：1 3 5 2 4。可以判断，这样的两组解合并起来是符合题目要求的。

从上可以看出，其实 $2N$ 皇后问题在大多数情况下并不需要二次求解 N 皇后，只需求得一组可行解即可。

那么，接下来的事情就好办多了。只需要利用目标算法求得 N 皇后问题一个符合要求的解即可。

1. **爬山算法**属于局部搜索算法的一份子，因此是一种解决最优化问题的启发式算法。

在实际运用中，爬山算法不会前瞻与当前状态不直接相邻的状态，只会选择比当前状态价值更好的相邻状态，所以简单来说，爬山算法就是向价值增长方向持续移动的循环过程。

初始状态下，每行每列都是一个皇后，之后的求解就是对存在冲突的列，寻找其他的冲突列来进行交换。因为寻找是随机的，所以每次程序的运行结果可能也存在一定的偶然误差。

求解过程中，当得到了局部极大值时，如果不是全局最优解（即冲突数为 0），则随机生成初始状态，重新求解，直到得到全局最优解。

2. **CSP 算法**的实现与爬山算法有不少的相同部分。比如初始值的生成，同样，也都是采用交换列的方式来实现消解冲突的。

求解时，遍历每一个皇后，选择当前行的皇后与其他皇后冲突最少的列（相同则随机选），交换当前行与目标行皇后。若遍历完后没求得解，则再遍历，直到求得解。

算法的空间复杂度:

在实现 N 皇后问题求解的时候,我采用了大小为 N 的一维数组保存当前的皇后摆放状态,数组下标为当前皇后所在的行,数组元素的值即为当前皇后摆放的列。这样的做法将一个二维问题降到一维来存储,有效地节省了空间开销。

由上面的算法思想分析可知:同一时刻,只有一组当前状态和一组相邻状态占用了空间。故算法的时间复杂度为 $O(n)$ 。

算法的时间复杂度:

由于在随机重启爬山算法中,采用随机数来选定初始状态,初始状态中的冲突数为 $O(N^2)$ 。程序的具体运行时间具有一定的随机性,因为只有正确的尝试才会使得冲突数减少。假定交换尝试的总次数为 M , M 的值具有较强的随机性,平均情况下的时间复杂度应该为 $O(MN^2)$ 。

CSP: 初始状态中的冲突数为 $O(N^2)$,此后每次要交换时更新冲突数时为 $4N=O(N)$ (求交换前两行冲突数及交换后两行冲突数)。遍历皇后 $O(N)$,遍历总次数为 M , M 的值具有较强的随机性,故时间复杂度为 $O(N^2+MN^2)=O(MN^2)$ 。

实验结果说明:

我特意写了一份回溯法求解 N 皇后问题的程序,作为对比参照。

下面是程序运行计时的结果对比:

当 $N=10$ 时:

```
The run time is: 0.007s.  
10皇后共计724个解
```

图 1: 回溯法求解 10 皇后问题

```
Running...
Found a solution. Tries: 1. Steps: 76
The run time is: 0.035s.
请按任意键继续. . .
```

input.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

10|

图 2：爬山算法求解 10 皇后问题

分析：N=10 的取值比较小，两种方法几乎都是瞬间完成了求解。

把 N 再取得大一些：

```
The run time is: 318.096s.
16皇后共计14772512个解
```

图 3：回溯法求解 16 皇后问题

```
Running...
Found a solution. Tries: 15. Steps: 155
The run time is: 0.813s.
请按任意键继续. . .
```

input.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

20|

图 4：爬山算法求解 20 皇后问题

分析：此时就可以明显地看出差异了。N=16 时，回溯法求解就已经花费了 318 秒；而 N=20 时，爬山算法的耗时也还在 1 秒以内。从这里就可以看出 AI 算法对程序运行效率的提升是相当惊人的。

再取一些更大的 N 值来看看爬山算法程序的运行情况：

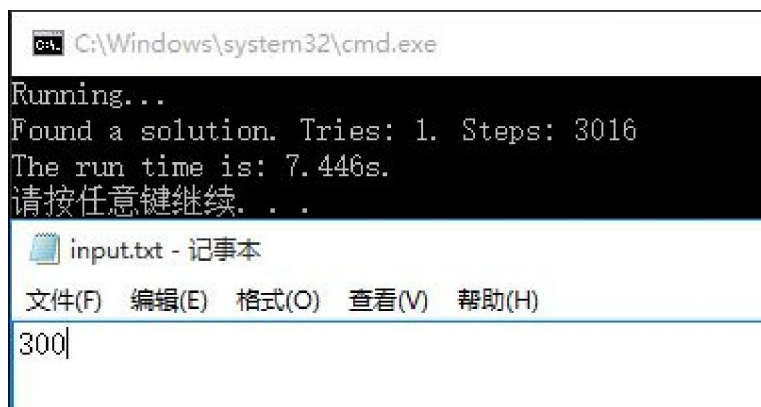
```
Running...
Found a solution. Tries: 1. Steps: 3890
The run time is: 1.367s.
请按任意键继续. . .
```

input.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

100|

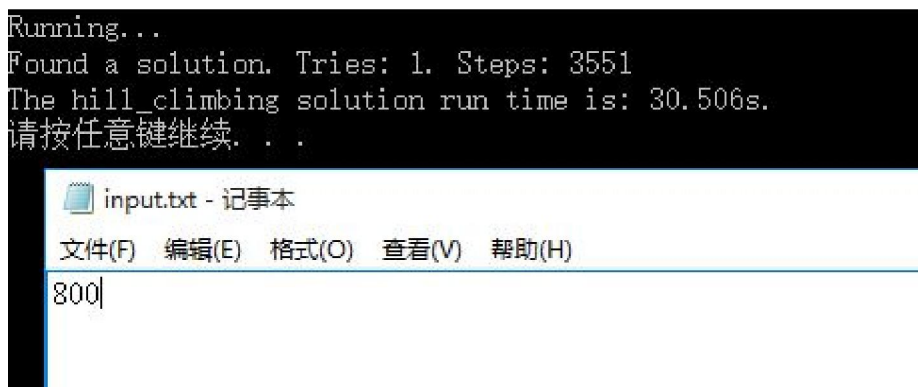
图 5：爬山算法求解 100 皇后问题



```
CA: C:\Windows\system32\cmd.exe
Running...
Found a solution. Tries: 1. Steps: 3016
The run time is: 7.446s.
请按任意键继续. . .

input.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
300|
```

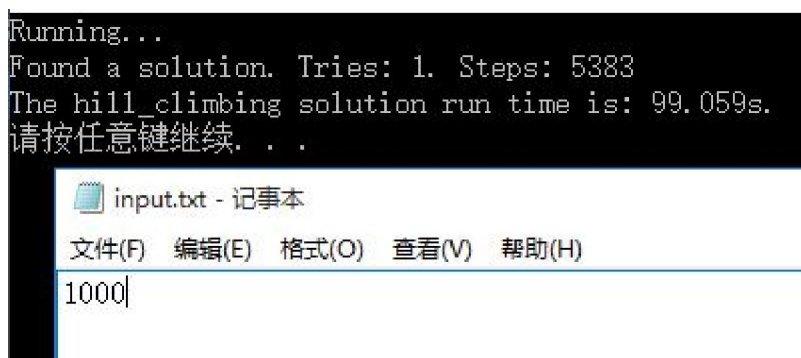
图 6：爬山算法求解 300 皇后问题



```
Running...
Found a solution. Tries: 1. Steps: 3551
The hill_climbing solution run time is: 30.506s.
请按任意键继续. . .

input.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
800|
```

图 7：爬山算法求解 800 皇后问题



```
Running...
Found a solution. Tries: 1. Steps: 5383
The hill_climbing solution run time is: 99.059s.
请按任意键继续. . .

input.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1000|
```

图 8：爬山算法求解 1000 皇后问题

最后，再来看一下 CSP 算法的求解效果：

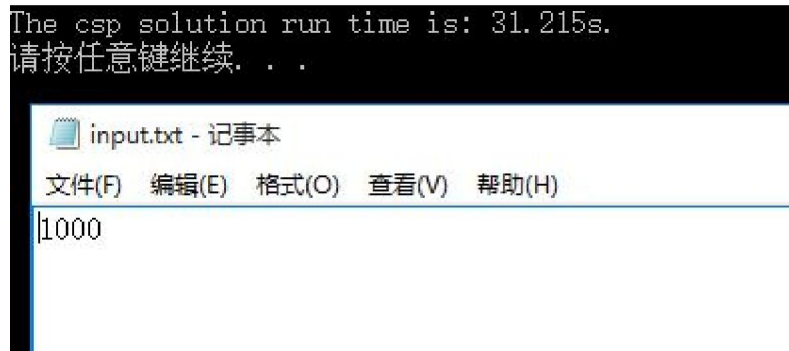


图 9: CSP 算法求解 1000 皇后问题

分析：从这里就可以很明显地看出 CSP 的算法运行效率要高于随机爬山算法。

简单分析一下原因，两种算法的时间复杂度都为 $O(MN^2)$ ，但是：

1. CSP 的 M 值要远远小于爬山算法的 M
2. 在实际的编写时，CSP 的交换尝试时间耗费为 $4N=O(N)$ ，而爬山算法因为我维护了一个冲突列的容器，所以每次交换尝试的时间耗费都为 $O(N^2)$ 。