

《数据结构》上机实验报告

第 3 次上机

学号： 202011140104

姓名： 李馨

学院： 物理学系

专业： 物理学

教师： 郑新

日期： 2022.10.14

1. 实验要求

1.上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2.独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3.上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

1. 实验过程
   1. 问题描述：八皇后问题

设在初始状态下在国际象棋棋盘上没有任何棋子(皇后)。然后顺序在第1行，第2行，…。第8行上布放棋子。在每一行中有8个可选择位置，但在任一时刻，棋盘的合法布局都必须满足3个限制条件，即任何两个棋子不得放在棋盘上的同一行、或者同一列、或者同一斜线上。

**设计要求**：

1、试用递归的方法编写算法，求解并输出此问题的所有合法布局。

**2、**试用非递归的方法编写算法，求解并输出此问题的所有合法布局。

* 1. 问题分析与关键代码（Queen.cpp）
     1. **递归算法（**void putQueen(int i)**）**

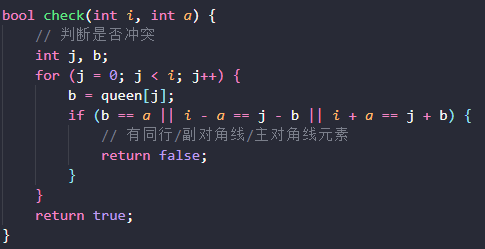
使用回溯法，从第1行开始先在一个位置放下棋子，然后在第2行放下棋子，放完后需要判断**是否满足限制条件**。如果不满足，需要进行**回溯**，在第2行重新**换一个位置**放下棋子，如果所有位置都不行，回溯至第1行，更换第1行棋子的位置。如果满足，对下一行（第3行）进行同样的操作，如果第3行所有位置都不行，再回溯第2行。之后是第4行、第5行……直到8行全部满足要求，输出此解。

一般地，对于第i行，从第1列开始放下棋子，判断是否满足限制条件，如不满足，放至下一列（右移），如果所有位置都不满足，则回溯至第i-1行，让第i-1行的棋子放至原本位置的下一列（右移）。当8行全都满足时，也再度回溯至第7行，将其棋子右移，寻找其他解。

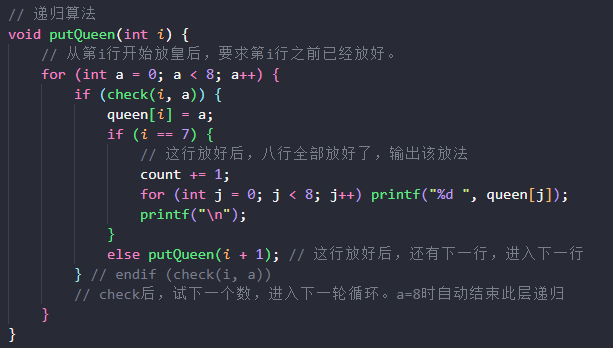
* + - 1. 全局变量：



* + - * 1. 布局数组int queen[8]，queen[i]=a表示第i行第a列放下棋子。（为方便叙述，行、列从0开始计数）
        2. 解的个数int count。
      1. 为了方便反复判断在第i行第a列放下棋子时是否与前面已放的棋子（第j行第b列）冲突（同行/同列/同斜线），需要写一个check函数，如下所示：



* + - 1. 这里的算法中回溯通过循环和该层递归的同时结束来实现：当putQueen(i)的for循环结束时（即第i行所有位置都试过了），这一层递归函数也结束了，回到了外层函数，即putQueen(i-1)，相当于回溯。递归函数代码：



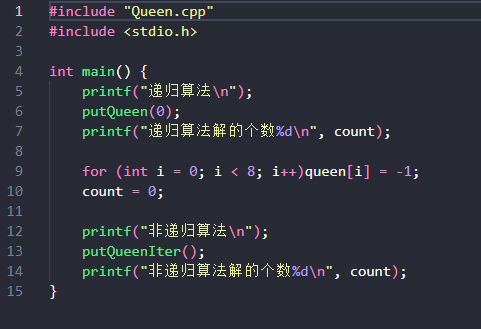
* + 1. **非递归算法（**void putQueenIter()**）**

仍是回溯法的思路，通过for循环对第i行进行依次处理，如果该行所放位置可行，前进（i=i+1）；如果该行所有位置都不可行，回溯（i=i-1）。

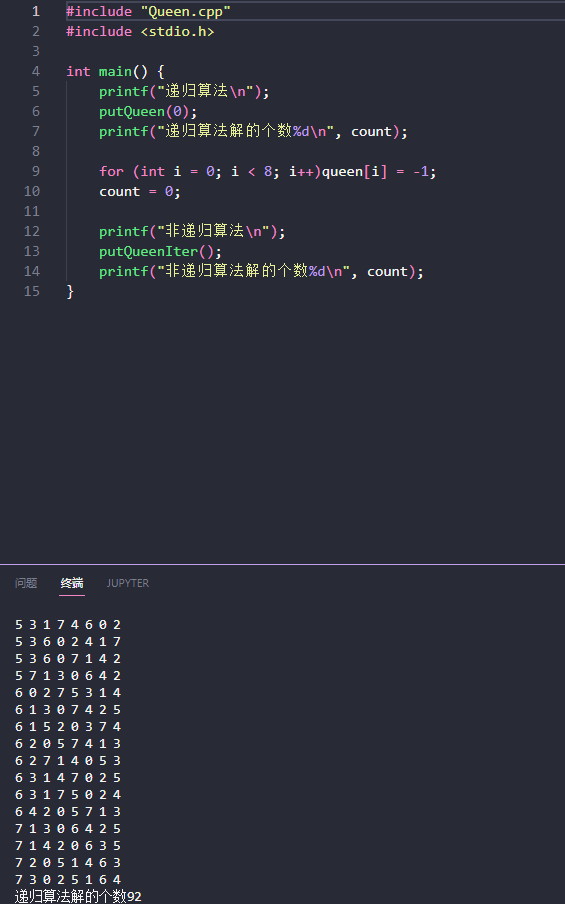
* + - 1. 这里的回溯可以一直回溯到第一行，当第一行所有位置也都试过后，算法结束，这一结束条件的判断放至循环开始处。
      2. 算法细节见代码注释：



* 1. 运行结果（main.cpp）
     1. 测试代码：



* + 1. 运行结果（第i个数为a表示第i行第a列放下棋子，行、列皆从0开始计数）：



递归算法

0 4 7 5 2 6 1 3

0 5 7 2 6 3 1 4

0 6 3 5 7 1 4 2

0 6 4 7 1 3 5 2

1 3 5 7 2 0 6 4

1 4 6 0 2 7 5 3

1 4 6 3 0 7 5 2

1 5 0 6 3 7 2 4

1 5 7 2 0 3 6 4

1 6 2 5 7 4 0 3

1 6 4 7 0 3 5 2

1 7 5 0 2 4 6 3

2 0 6 4 7 1 3 5

2 4 1 7 0 6 3 5

2 4 1 7 5 3 6 0

2 4 6 0 3 1 7 5

2 4 7 3 0 6 1 5

2 5 1 4 7 0 6 3

2 5 1 6 0 3 7 4

2 5 1 6 4 0 7 3

2 5 3 0 7 4 6 1

2 5 3 1 7 4 6 0

2 5 7 0 3 6 4 1

2 5 7 0 4 6 1 3

2 5 7 1 3 0 6 4

2 6 1 7 4 0 3 5

2 6 1 7 5 3 0 4

2 7 3 6 0 5 1 4

3 0 4 7 1 6 2 5

3 0 4 7 5 2 6 1

3 1 4 7 5 0 2 6

3 1 6 2 5 7 0 4

3 1 6 2 5 7 4 0

3 1 6 4 0 7 5 2

3 1 7 4 6 0 2 5

3 1 7 5 0 2 4 6

3 5 0 4 1 7 2 6

3 5 7 1 6 0 2 4

3 5 7 2 0 6 4 1

3 6 0 7 4 1 5 2

3 6 2 7 1 4 0 5

3 6 4 1 5 0 2 7

3 6 4 2 0 5 7 1

3 7 0 2 5 1 6 4

3 7 0 4 6 1 5 2

3 7 4 2 0 6 1 5

4 0 3 5 7 1 6 2

4 0 7 3 1 6 2 5

4 0 7 5 2 6 1 3

4 1 3 5 7 2 0 6

4 1 3 6 2 7 5 0

4 1 5 0 6 3 7 2

4 1 7 0 3 6 2 5

4 2 0 5 7 1 3 6

4 2 0 6 1 7 5 3

4 2 7 3 6 0 5 1

4 6 0 2 7 5 3 1

4 6 0 3 1 7 5 2

4 6 1 3 7 0 2 5

4 6 1 5 2 0 3 7

4 6 1 5 2 0 7 3

4 6 3 0 2 7 5 1

4 7 3 0 2 5 1 6

4 7 3 0 6 1 5 2

5 0 4 1 7 2 6 3

5 1 6 0 2 4 7 3

5 1 6 0 3 7 4 2

5 2 0 6 4 7 1 3

5 2 0 7 3 1 6 4

5 2 0 7 4 1 3 6

5 2 4 6 0 3 1 7

5 2 4 7 0 3 1 6

5 2 6 1 3 7 0 4

5 2 6 1 7 4 0 3

5 2 6 3 0 7 1 4

5 3 0 4 7 1 6 2

5 3 1 7 4 6 0 2

5 3 6 0 2 4 1 7

5 3 6 0 7 1 4 2

5 7 1 3 0 6 4 2

6 0 2 7 5 3 1 4

6 1 3 0 7 4 2 5

6 1 5 2 0 3 7 4

6 2 0 5 7 4 1 3

6 2 7 1 4 0 5 3

6 3 1 4 7 0 2 5

6 3 1 7 5 0 2 4

6 4 2 0 5 7 1 3

7 1 3 0 6 4 2 5

7 1 4 2 0 6 3 5

7 2 0 5 1 4 6 3

7 3 0 2 5 1 6 4

递归算法解的个数92

非递归算法

0 4 7 5 2 6 1 3

0 5 7 2 6 3 1 4

0 6 3 5 7 1 4 2

0 6 4 7 1 3 5 2

1 3 5 7 2 0 6 4

1 4 6 0 2 7 5 3

1 4 6 3 0 7 5 2

1 5 0 6 3 7 2 4

1 5 7 2 0 3 6 4

1 6 2 5 7 4 0 3

1 6 4 7 0 3 5 2

1 7 5 0 2 4 6 3

2 0 6 4 7 1 3 5

2 4 1 7 0 6 3 5

2 4 1 7 5 3 6 0

2 4 6 0 3 1 7 5

2 4 7 3 0 6 1 5

2 5 1 4 7 0 6 3

2 5 1 6 0 3 7 4

2 5 1 6 4 0 7 3

2 5 3 0 7 4 6 1

2 5 3 1 7 4 6 0

2 5 7 0 3 6 4 1

2 5 7 0 4 6 1 3

2 5 7 1 3 0 6 4

2 6 1 7 4 0 3 5

2 6 1 7 5 3 0 4

2 7 3 6 0 5 1 4

3 0 4 7 1 6 2 5

3 0 4 7 5 2 6 1

3 1 4 7 5 0 2 6

3 1 6 2 5 7 0 4

3 1 6 2 5 7 4 0

3 1 6 4 0 7 5 2

3 1 7 4 6 0 2 5

3 1 7 5 0 2 4 6

3 5 0 4 1 7 2 6

3 5 7 1 6 0 2 4

3 5 7 2 0 6 4 1

3 6 0 7 4 1 5 2

3 6 2 7 1 4 0 5

3 6 4 1 5 0 2 7

3 6 4 2 0 5 7 1

3 7 0 2 5 1 6 4

3 7 0 4 6 1 5 2

3 7 4 2 0 6 1 5

4 0 3 5 7 1 6 2

4 0 7 3 1 6 2 5

4 0 7 5 2 6 1 3

4 1 3 5 7 2 0 6

4 1 3 6 2 7 5 0

4 1 5 0 6 3 7 2

4 1 7 0 3 6 2 5

4 2 0 5 7 1 3 6

4 2 0 6 1 7 5 3

4 2 7 3 6 0 5 1

4 6 0 2 7 5 3 1

4 6 0 3 1 7 5 2

4 6 1 3 7 0 2 5

4 6 1 5 2 0 3 7

4 6 1 5 2 0 7 3

4 6 3 0 2 7 5 1

4 7 3 0 2 5 1 6

4 7 3 0 6 1 5 2

5 0 4 1 7 2 6 3

5 1 6 0 2 4 7 3

5 1 6 0 3 7 4 2

5 2 0 6 4 7 1 3

5 2 0 7 3 1 6 4

5 2 0 7 4 1 3 6

5 2 4 6 0 3 1 7

5 2 4 7 0 3 1 6

5 2 6 1 3 7 0 4

5 2 6 1 7 4 0 3

5 2 6 3 0 7 1 4

5 3 0 4 7 1 6 2

5 3 1 7 4 6 0 2

5 3 6 0 2 4 1 7

5 3 6 0 7 1 4 2

5 7 1 3 0 6 4 2

6 0 2 7 5 3 1 4

6 1 3 0 7 4 2 5

6 1 5 2 0 3 7 4

6 2 0 5 7 4 1 3

6 2 7 1 4 0 5 3

6 3 1 4 7 0 2 5

6 3 1 7 5 0 2 4

6 4 2 0 5 7 1 3

7 1 3 0 6 4 2 5

7 1 4 2 0 6 3 5

7 2 0 5 1 4 6 3

7 3 0 2 5 1 6 4

非递归算法解的个数92

三、总结（实验中遇到的问题、取得的经验、感想等）

①不考虑是否递归，最先想到的是穷举法，而后搜集资料后才尝试回溯法，事实上回溯法也是一种穷举，只不过会根据限制条件不断排除一些可能，减少了不必要的穷举。对于“回溯”的实现，有了具体的体悟，一层递归结束后回到上一层递归便是一种回溯。

②可以从输出结果看到，这两种算法的搜索路径是一样的，只是编写方式不同。递归算法代码量显然更少、更简洁。不过我觉得都比较难写出来，理解难度上感觉差异也不大。