手把手教你 N皇后 回溯解法（Solving The N Queens Puzzle Using Backtracking）

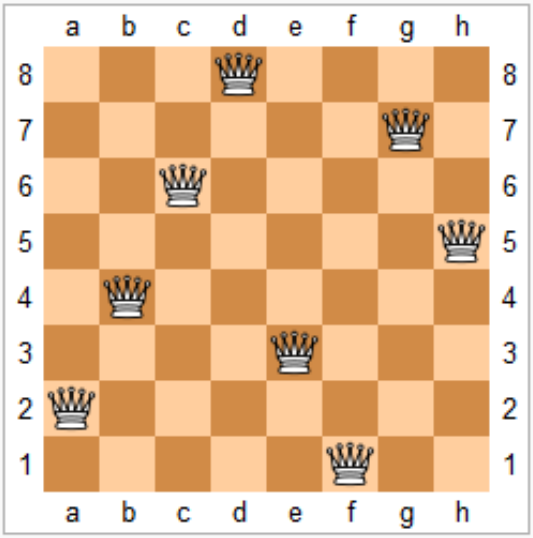
写在前面

本人**没有国际象棋经验**，仅仅是看了几篇攻略写的本题，正好是**每日打卡**，就把这个问题拿出来讲一讲。

本笔记实现完全是 C++，当然如果你是其他平台，**算法的思路是一种常量**，只要抓住方法，一样可以实现。

问题描述

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.



One solution to the eight queens puzzle

Given an integer n, return the number of distinct solutions to the n-queens puzzle.

Example:

```
Input: 4
Output: 2
Explanation: There are two distinct solutions to the 4-queens puzzle as shown
below.
[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
```

```
   "..Q."],

  ["..Q.",  // Solution 2
   "Q...",
   "...Q",
   ".Q.."]
]
```

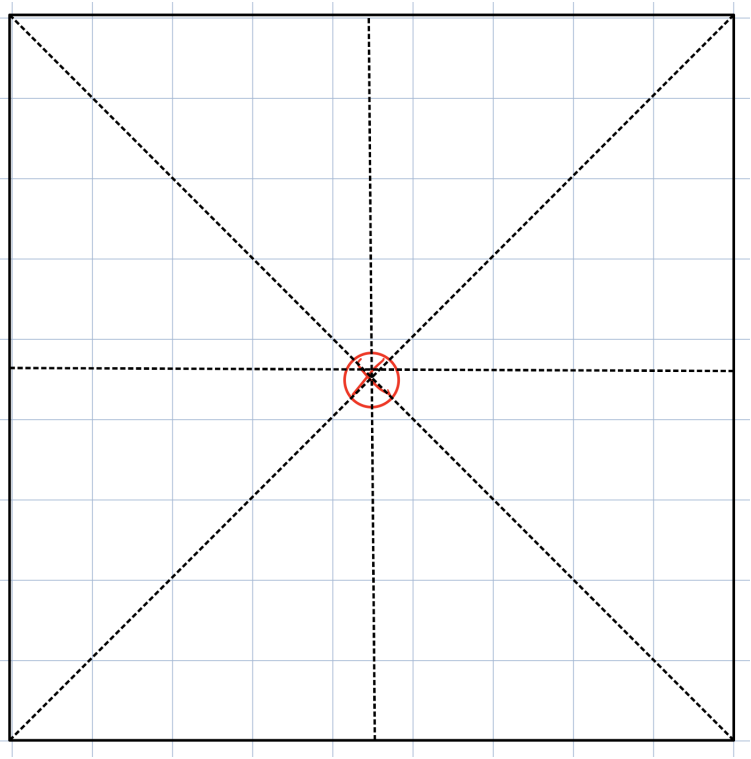
来源：力扣（LeetCode）

解题思路

基本规则

要了解八皇后问题的解法，必须先了解一些国际象棋的规则，不用太多，只要了解**皇后**的活动范围即可。

皇后作为一个棋子，其活动空间**相当自由**：



> 虚线表示皇后的活动空间

Try not to understand it. Try to feel it.

八皇后问题本身的历史也是相当悠久，它在1848年就被提出，并在1850年由 Franz Nauck 给出第一组解法。

直到1972年，Edsger Dijkstra 给出了**基于DFS回溯的程序结构化解法**，这便是今天我们讨论的主题。

我并不认为我是一个数学很好的程序员，实际上，真正意义上数学好的程序员并不多。程序员应该**Think In Computers**，

而不是**Totally Understand The PUZZLE**。也就是说，繁复的计算工作基本让计算机完成，而程序员要做的是，对问题有个**感性认知**。

算法描述

先回溯到问题本身

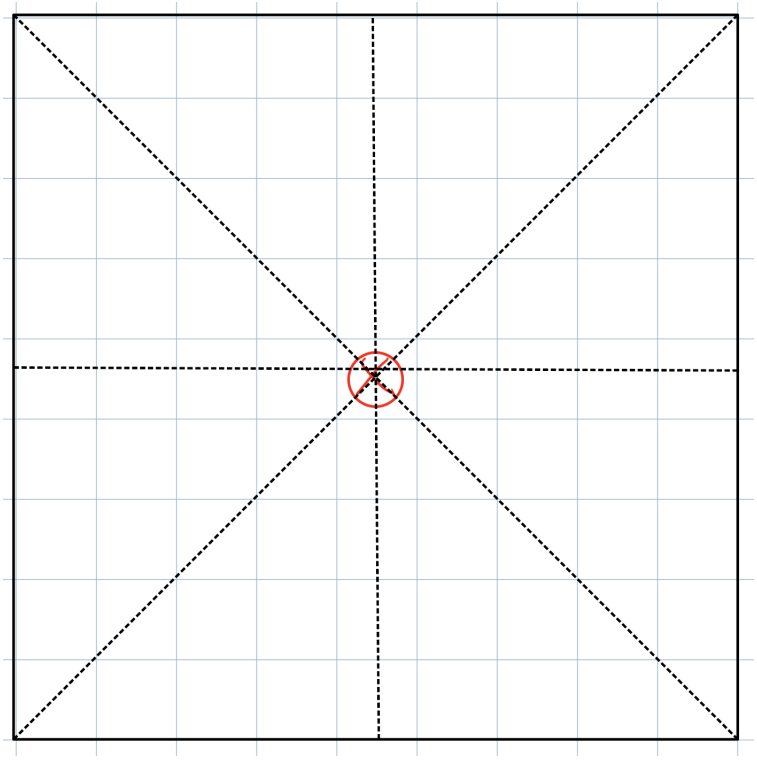
回溯算法的本质，即用**深度优先的方法**（DFS）遍历一颗状态树。

而此时，你可能会一脸懵逼，**什么是DFS？什么又是状态树？状态树又是如何遍历的？**

Be patient，我们先回到**问题本身**。

八皇后问题，是给我们八个皇后棋子，找到一个**摆法**，使得每个皇后都**无法相互攻击**到。

说白了，就是每个皇后的 **左上角、右下角、上下左右** 都 **没有其他皇后** 。



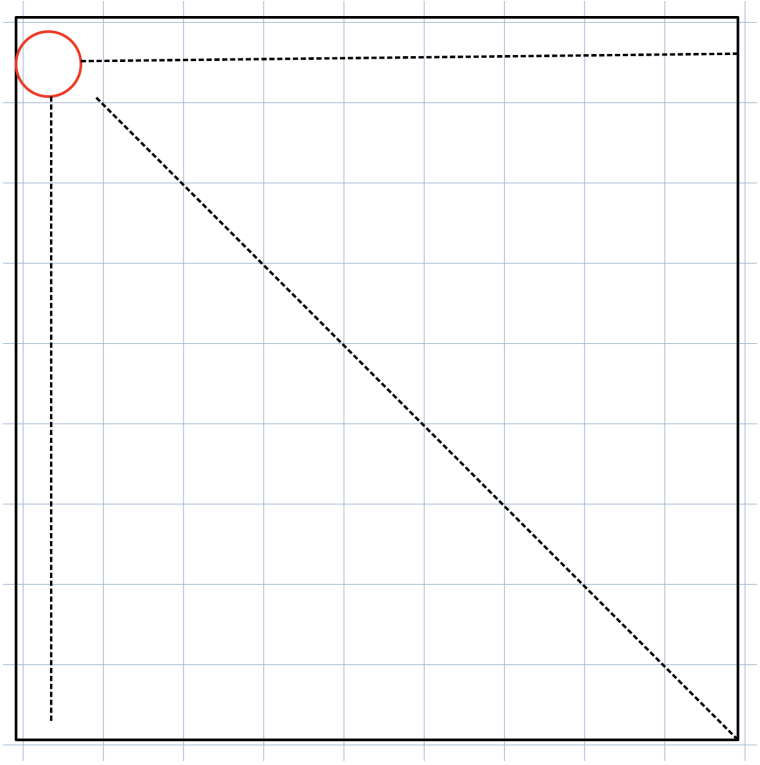
（上图虚线所及的格子都不能有其他皇后）

既然不能在同一行同一列，我们应该**一行一行**地放皇后。

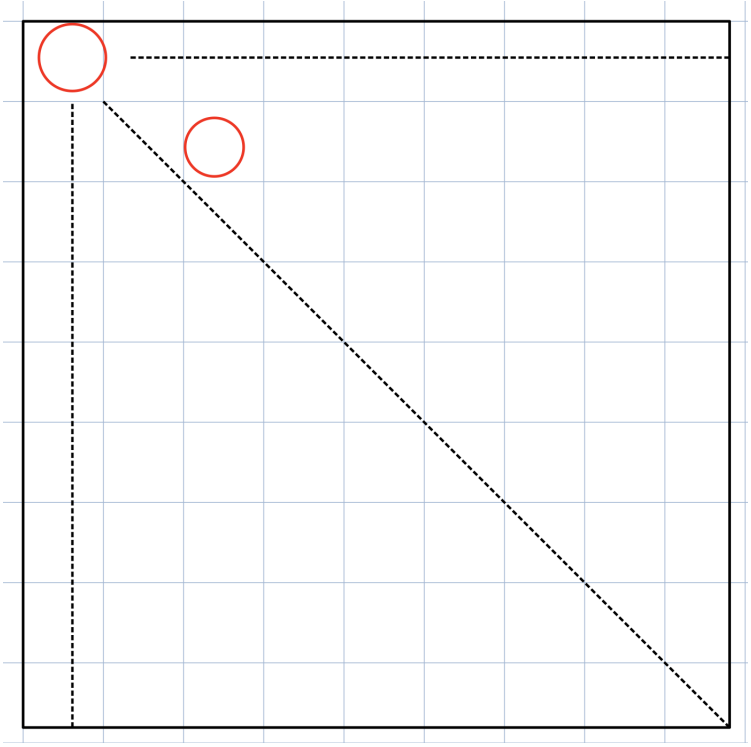
一步一步来

假设我们把第一个皇后放在如图所示的位置，

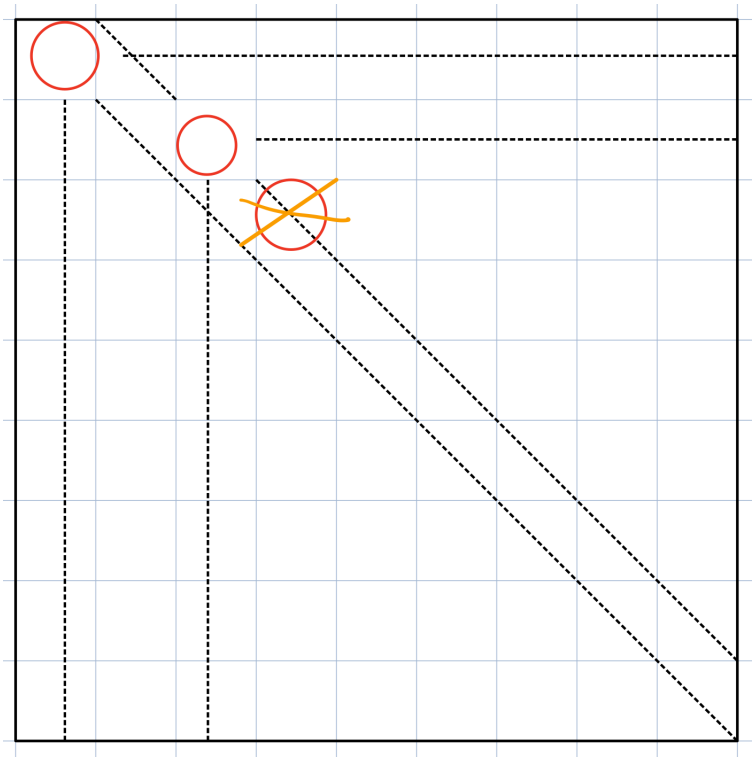
第二行 未被虚线覆盖到的位置 就是**下一个皇后**可以摆放的位置



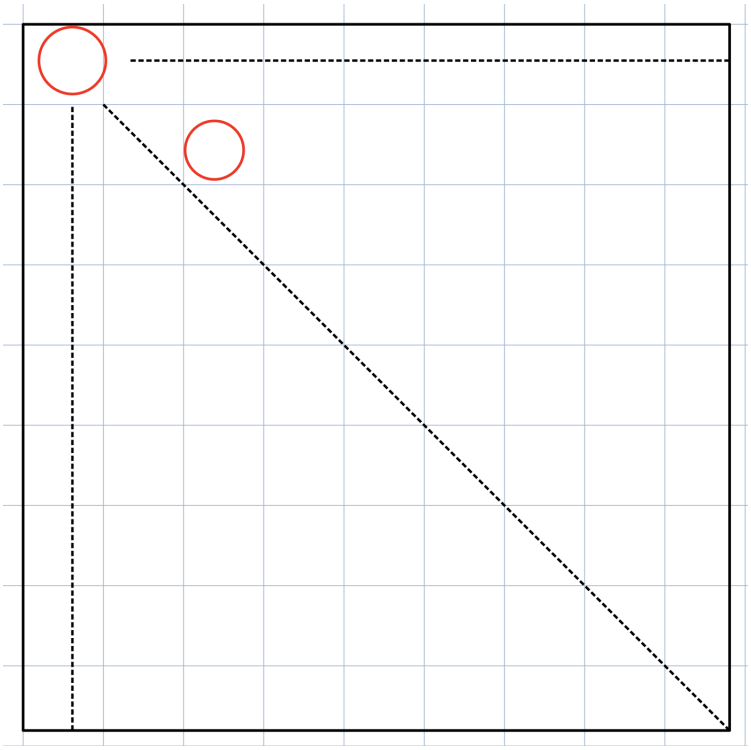
我们继续摆放第二个皇后，放在合理的位置



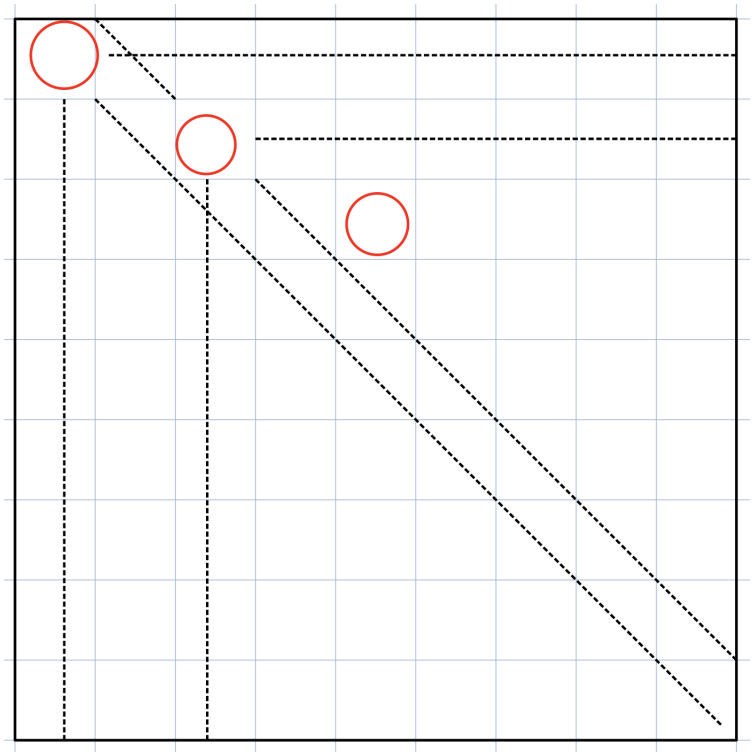
假如，我们在摆放第三个皇后时，很不幸，她被摆到了**错误的位置**



那么我们就像 杀手皇后的败者食尘 一样**回退**到第二个皇后已经摆好的时间

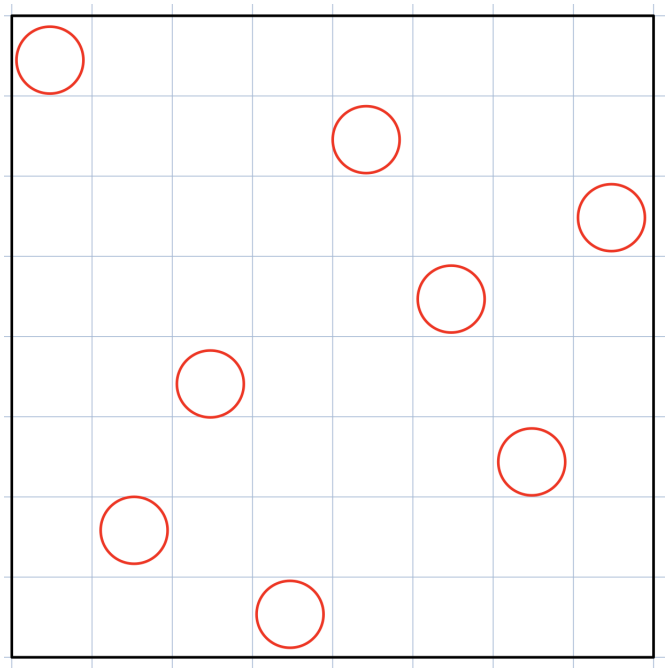


既然看到了 **未来** 不如意，我们就重新思考，把第三个皇后**往后放一列**，逆转未来！



这样就合理了。

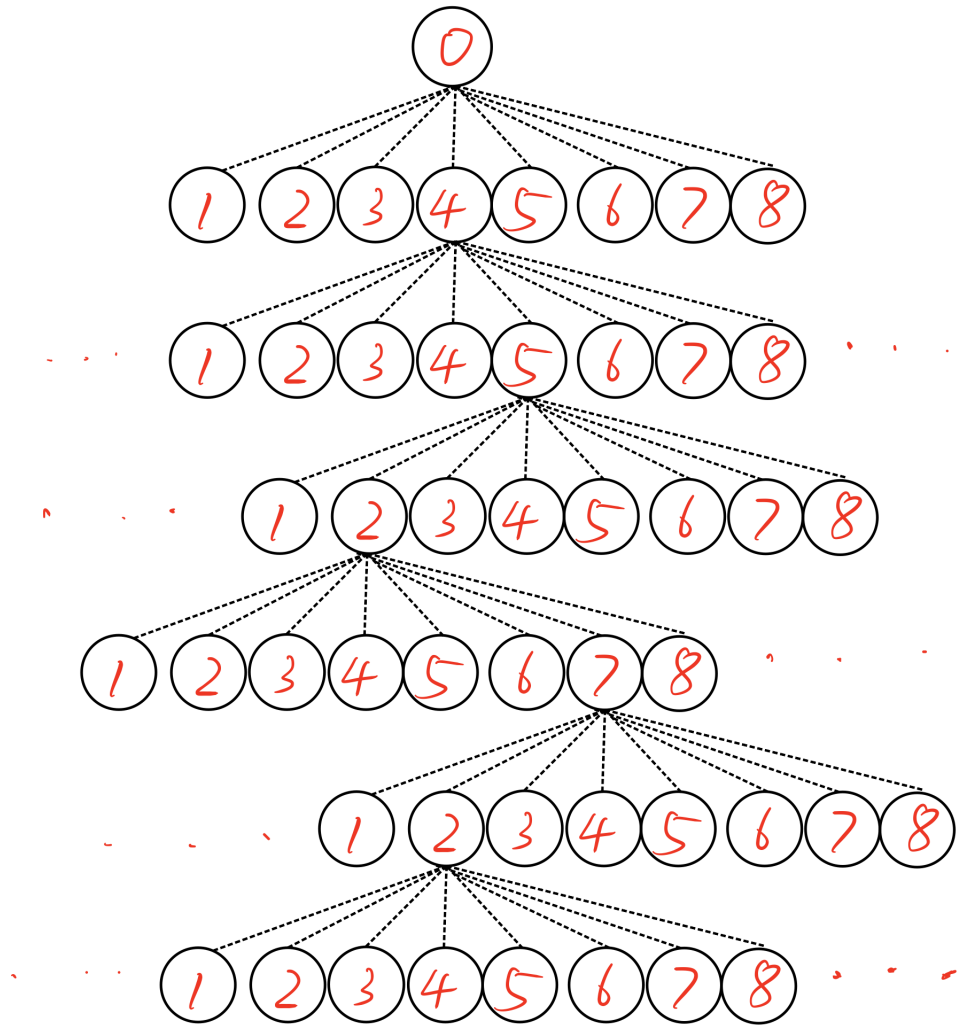
直到我们**摆满八行**，就记录一次结果



说到状态树

我们把**每次决定**都认为是一层树的节点

比如选择了第二层四节点（第一层代表还没放皇后），就代表在**第一行第四列**摆放**第一个皇后**，如下图



这里的状态树并不是 Full Tree（如果是Full的话，那么就是代表在棋盘上**放满了皇后**，显然是不合理的）。

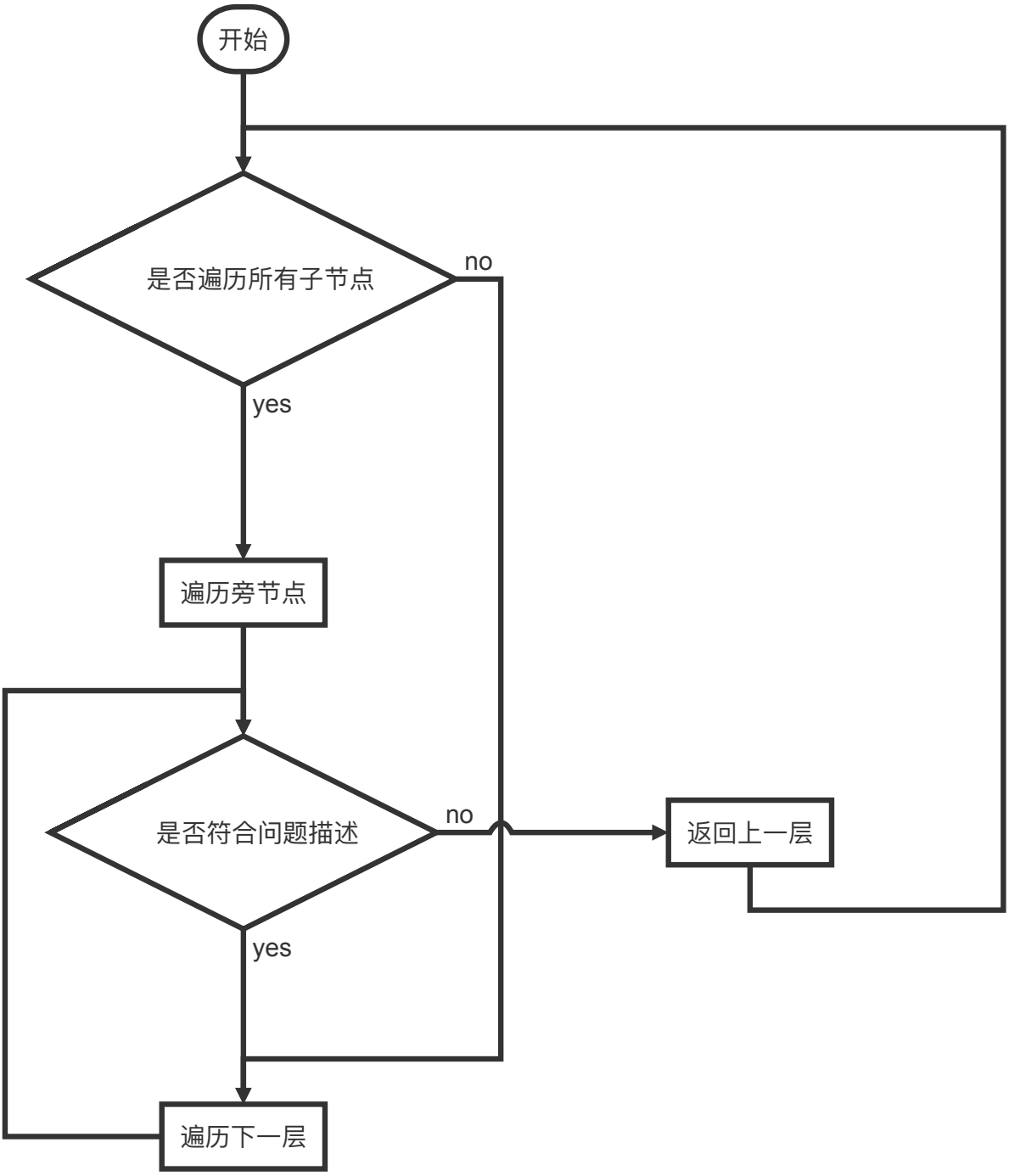
只有符合题目要求的决定会被选择，选择完 **8层**算作一种解，或者说是一种**状态**。

那么所有这些**状态**或是解，构成的**集合**被称为状态空间（State Space）。

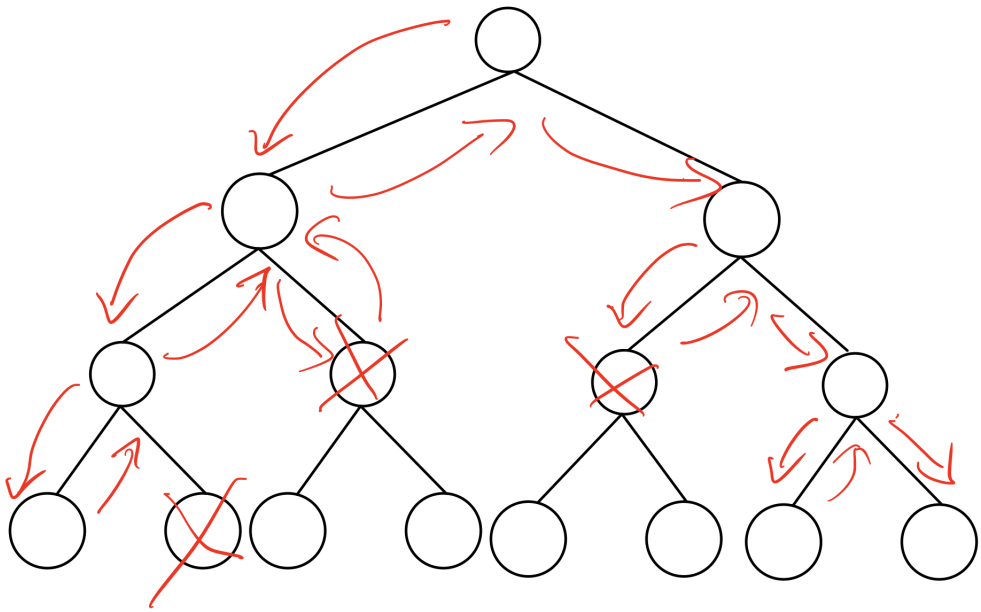
用科幻一些的方法说明，有多少种**解**，就相当于有多少种代表不同选择**平行宇宙**。

DFS深度优先算法

讲完了**状态树**，我们来讲讲什么是**深度优先搜索**DFS，因为可以把八皇后的决策问题抽象成**状态树**的结构，而后才有**遍历**的说法。按照深度优先遍历的方法来遍历一棵树，可以描述为：



把这个过程具体化大致是这个样子：



代码实现

平台

OS：MacOS Catlina 10.15.6

CPU架构： x86_64

编译器：g++ | Apple clang version 12.0.0 (clang-1200.0.26.2) | Target: x86_64-apple-darwin19.6.0

IDE: Xcode

代码

> 注意：leetcode原题是N皇后问题，其算法的原理与八皇后一致，而下面的实现是面向N皇后实现

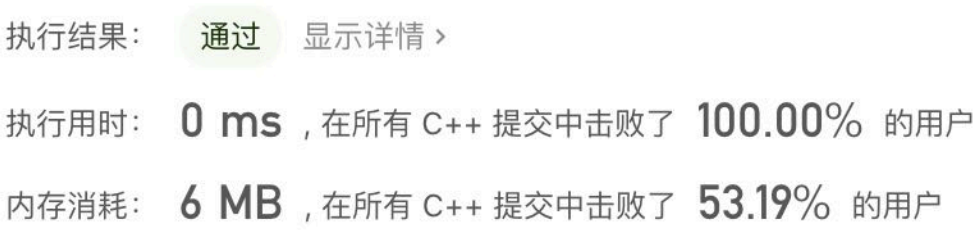
源码

```
class Solution {
public:
        int count=0;
        void  calNqueens(int row,int n,int* result){//call calNqueen(0)
            // 停止条件：到达棋盘最后一行
            // 打印一种解法
            if(row == n){
                this->count++;
```

```
            // printf("----------------\n");
                return;
            }

        // 遍历每一列
        for (int column=0; column< n; column++) {
            //判断是否合理
            if(isOk(row, column,n,result)){
                //记录一种解法
                result[row] = column;
                //遍历下一行
                calNqueens(row+1,n,result);
            }
        }
    }



    // 判断 该行该列 是否符合条件
    // 即 是否存在皇后攻击范围内的棋子
    bool isOk(int row,int column,int n,int* result){
        int leftup = column - 1,rightup = column +1;

        //逐行往上检查
        for(int i = row -1 ;i>=0;--i){

            //第i行的column列（本列）是否有棋子
            if(result[i]== column)return false;

            //考虑左上
            if(leftup>=0){
                if(result[i]==leftup)return false;
            }

            //考虑右上
            if(rightup<=n){
                if(result[i]==rightup)return false;
            }
            --leftup;++rightup;
        }
        return true;

    }
// 在本题OJ中相当于main函数
    int totalNQueens(int n) {
        int* result = new int[n];
        calNqueens(0,n,result);
        return this->count;
    }
```

```
};
```

结果



反思与补充

1. 在解决八皇后的问题的历史上，有计算机结构化程序的解法给出了92种解，而实际上的在这个解法诞生的100年之前，就已经被人用**非计算机**的方式解出了92种解法。
2. 实际上，在八皇后的问题上，92种解法是我们**将棋盘标记了坐标点的结果**，也就是规定了ABCDEFGH、12345678的纵横的具有方向性的坐标的情况下，产生92种解法，我们称之为**全解**（All Solution）。如果我们不考虑坐标，其实92种很多解法都是重复的（棋盘可以对称、翻转），留下的不同解法只有12种，成为**基本解**。（详见维基百科及其reference）
3. 回溯算法的名字的起的很好，让人联想到了时间回溯的情景，将我们生活中的所有抉择看作是一颗树的话，我们如果只做**单一选择**，也就是沿着一条路径遍历，那么如果时间能回溯，也就是回到时间抉择的**父节点**，我们似乎可以重新选择，至少可以选择作出比现在更好的选择。

参考

维基百科

《数据结构与算法之美》 王争

Leetcode #52. N Queen II

致谢

我的父母

程振波老师

以及我爱的人们

祝大家变得更强