**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： n皇后问题**

**学院： 计算机与软件学院 专业： 软件工程**

**报告人： 徐霞 学号： 2015150290 班级： 3**

**同组人： 无**

**指导教师： 杨烜**

**实验时间： 2017/10/14-2017/10/27**

**实验报告提交时间： 2017/10/27**

**教务处制**

1. **实验目的**

（1） 掌握回溯法设计思想。

（2） 掌握n皇后问题的回溯法解法。

1. **实验内容**

1.  编写测试代码测试八皇后问题。

测试数组（M，N），其中M代表皇后所在的行，N代表皇后所在的列。

例如，

第一组测试数据（1，4）、（2，7）、（3，3）、（4、8）、（5，2）、（6，5）、（7，1）、（8，6）

第二组测试数据（1，5）、（2，2）、（3，4）、（4，7）、（5，3）、（6，8）、（7，6）、（8，1）

第三组测试数据（1，4）、（2，2）、（3，7）、（4，3）、（5，6）、（6，8）、（7，5）、（8，1）

       判断测试数据（可能得到的解）是否满足八皇后问题要求。

2．对于n皇后问题，整个程序中应该包括主函数模块，摆放皇后的函数模块，以及判断皇后的位置是否摆放正确的判断模块。对于模块间的关系，在运行主函数的过程中会调用摆放皇后的函数模块，在摆放皇后的函数模块中，又会调用判断皇后位置是否摆放正确的判断模块。

3. 如果能实现可视化输出计算结果（包括回溯过程），如图1，可加分。

1. **实验要求**

1. 对n≥4的问题依次求解，请提供你机器上能求解的问题最大规模。

2. 利用蛮力法验证回溯法的正确性。

3. 对不同的n，统计剪枝前后搜索的节点个数，剪枝前节点数就是完备树搜索的节点个数，剪枝后就是根据八皇后问题的特点，提前结束搜索之后的遍历节点个数。并画图比较。

4. 在blackboard提交电子版实验报告。源代码和PPT作为实验报告附件上传。

5. 在实验完成之后，将进行一次PPT介绍。

6. 在实验报告中要求详细说明蛮力法和回溯法的实现思想，给出完备树节点个数计算公式，具体说明如何进行剪枝。

1. **实验过程**

**问题描述：**n皇后问题是一个以国际象棋为背景的问题：如何能够在n×n的国际象棋棋盘上放置n个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？为了达到此目的，任两个皇后都不能处于同一条横行、纵行或斜线上。

**回溯法思想：**

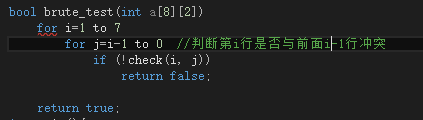
1. 在第i行，第j列放置皇后（i≤n，j≤n），判断此位置是否与前面i-1行的皇后不会发生冲突。
2. 如果发生冲突，则继续在第i行，在第j列之后找寻合适的位置。
3. 如果能够找到，则继续向下一行放置皇后。
4. 如果不能找到，则回溯到第i-1行，重新放置第i-1行皇后的位置，回到步骤1。
5. 当在最后一行能够找到皇后的合适位置，则此时找到一个解。
6. 直到已经回溯到第1行，且第一行的所有位置均已放置过皇后，算法结束。

**数据测试：**

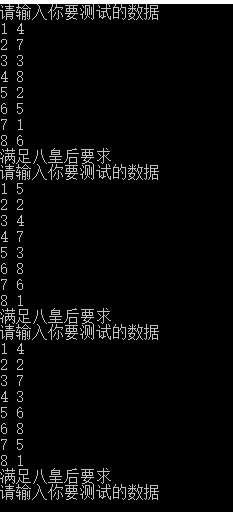
**测试8皇后问题：**

伪代码分析：

1. 皇后位置用二位数组存储
2. 从第二行开始，每次判断该行的皇后位置是否与前面行的皇后位置冲突（是否同列，是否处于对角线上）
3. 如果冲突，则结束判断，返回false
4. 如果循环结束还未检测到冲突，则返回true。



数据测试：



可见，测试数据都是符合n皇后的摆放要求的。

**递归-蛮力法 VS 递归-回溯法：**

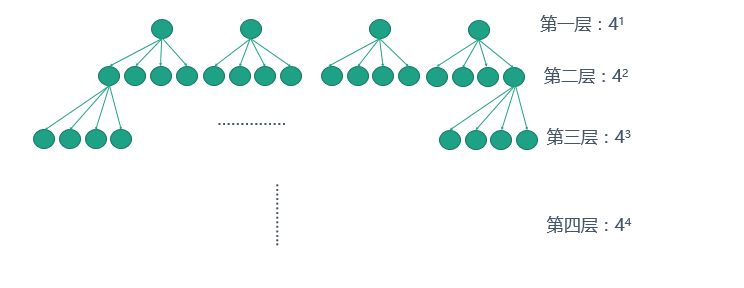
递归-蛮力法：每放置一个皇后，就继续向下一行放置，不需判断他的合理性，当每一行都放置好了皇后之后，再判断是否合理。

递归-回溯法：每放置一个皇后，就判断该位置是否合理，合理则继续向下一行放置，不合理则在该行继续放置，找不到时回溯到上一行。此时，不合理的位置的下面的放置都不会再进行，达到了剪枝的效果。

遍历的结点数的计算：

递归-蛮力法：

以4后问题为例

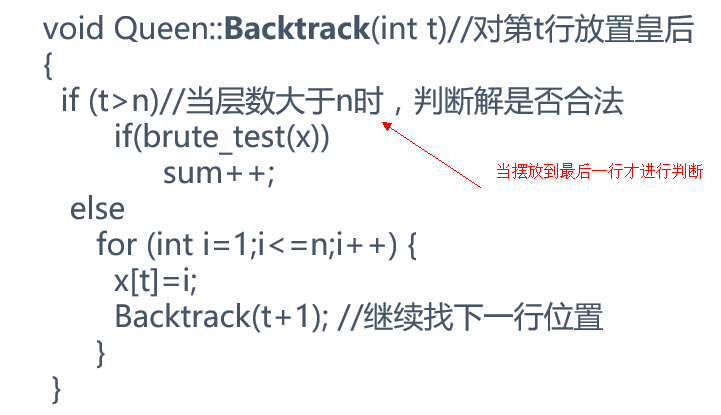


第i层有ni个 结点，所以总共结点数knot=n1+n2+n3+…+nn=n(nn-1)/(n-1);

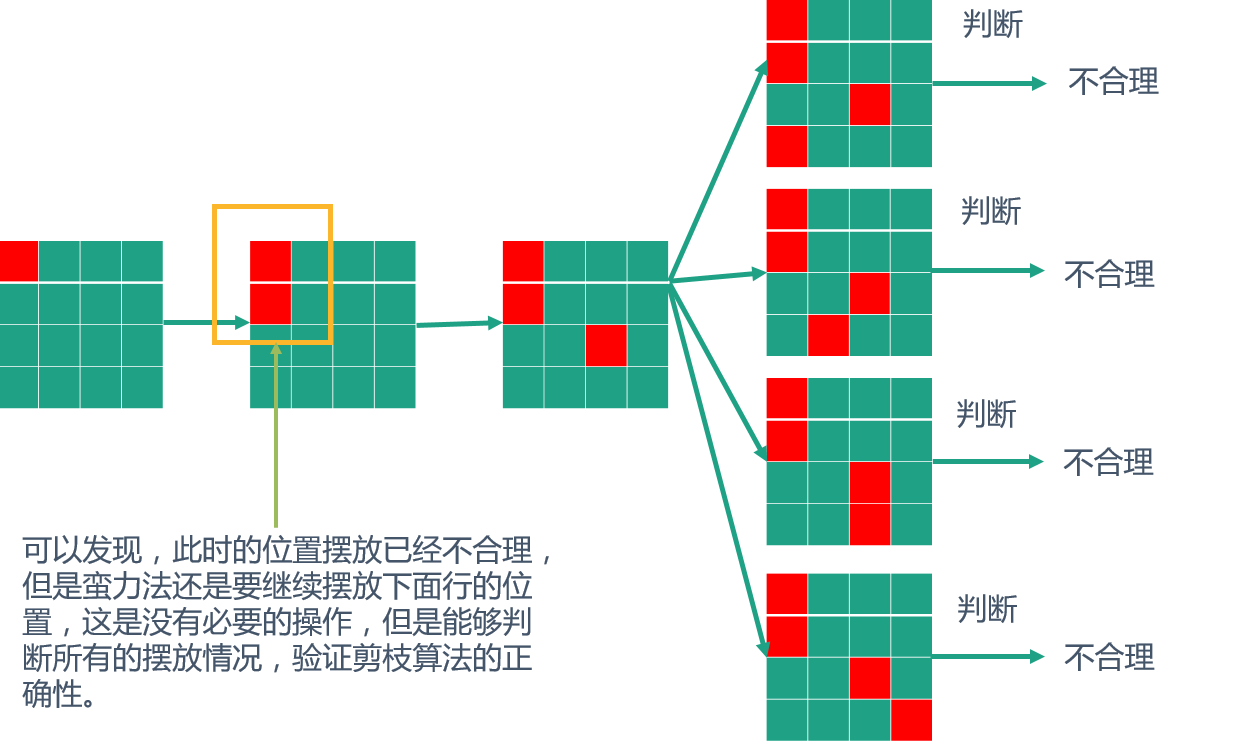
递归-回溯法：无法使用公式来计算，因为无法提前知道剪枝的位置，只能在代码中计算：每当放置一个皇后时，结点数就加一，不论这种放置方法是否合理，因为此时这个结点已经生成，并且要去判断它是否合理，尽管不合理还是要计算进去，剪枝只是不合理结点之后的结点不计算进去。

伪代码分析：

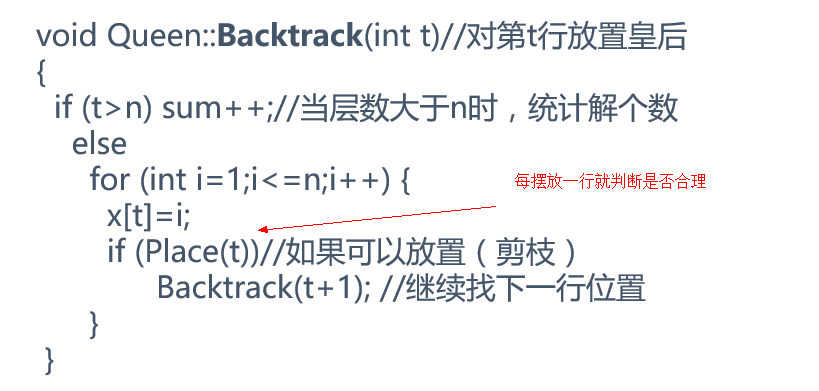
递归-蛮力法：



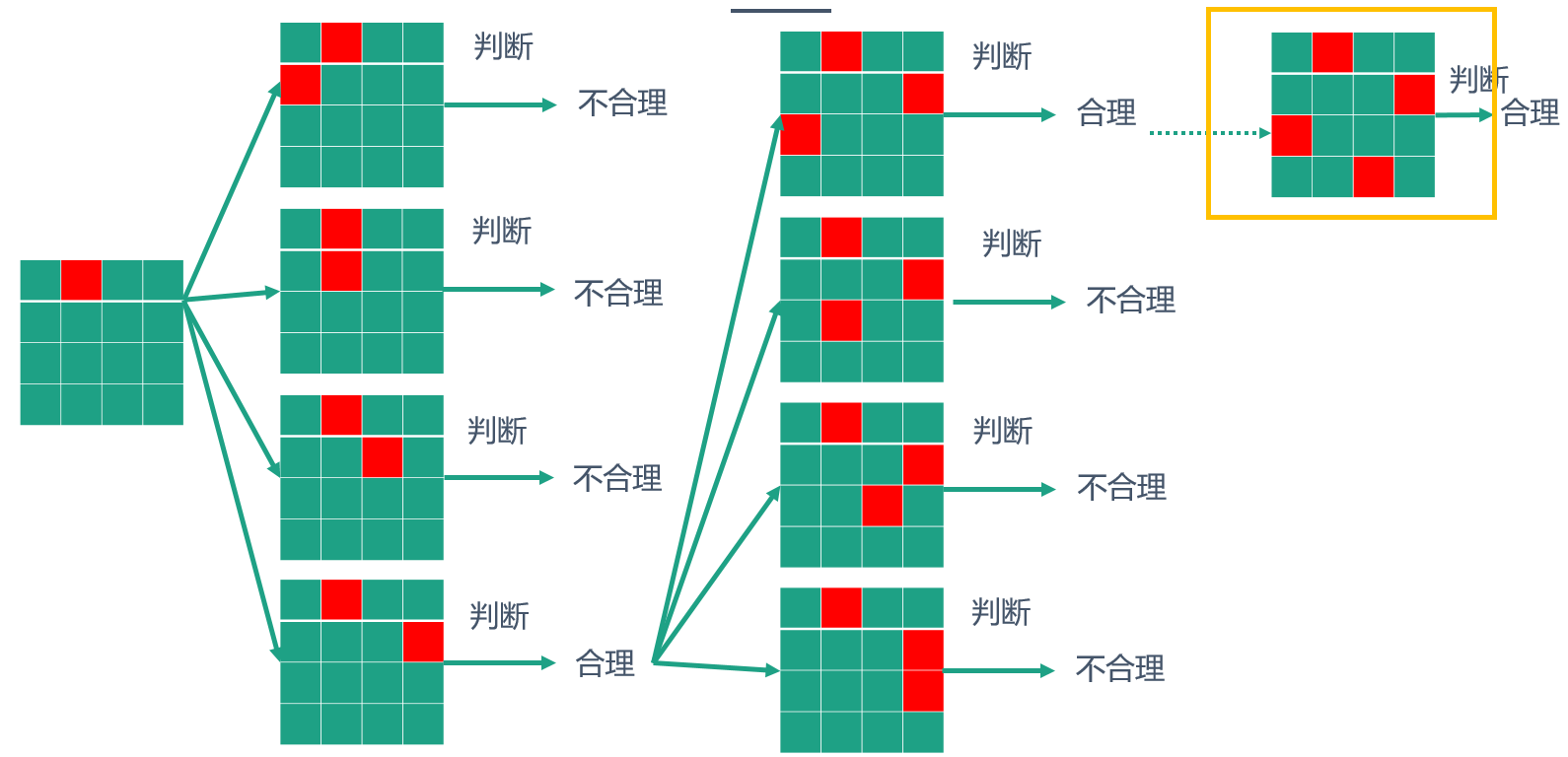
从下图可以看出，蛮力法的缺点：只当所有皇后都摆放好后才进行判断，效率低下



递归-回溯法：



从下图可以看出，回溯法的优点：每摆放一个皇后就对其合理性进行判断，对于不合理的摆放不再摆放下一行，减去不必要的枝节



递归-蛮力法 VS 递归-回溯法 结点数



表1递归蛮力法与递归回溯法遍历的结点数

对于蛮力法套用公式：knot=n(nn-1)/(n-1)

n=4时，knot=340；

n=5时，knot=3905；

n=6时，knot=55986；

……

与测出来的数据一致。

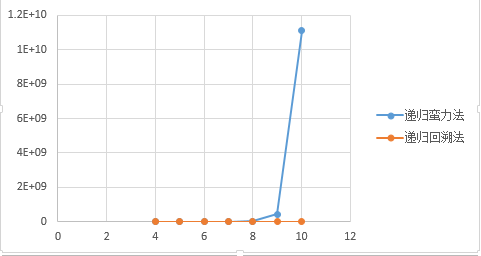


图1 递归蛮力法与递归回溯法遍历的结点数

从二者遍历的结点数就可以看出二者的效率差异，当规模n增大时，二者遍历的结点数差得就越多，回溯法的优势越来越明显，这还可以在时间上体现，如下表，下图：



表2 递归蛮力法与递归回溯法时间

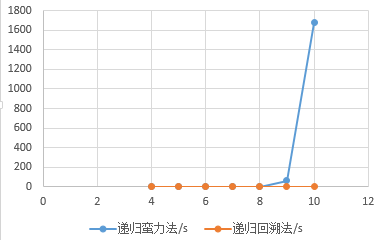


图2 递归蛮力法与递归回溯法时间

**递归-回溯法 VS 非递归-回溯法：**

二者算法思想是一样的，都是每每摆放好一个皇后之后进行判断，合理则进，不合理则退。区别在于前者用递归实现，后者用循环实现。



表3 递归回溯与非递归回溯时间

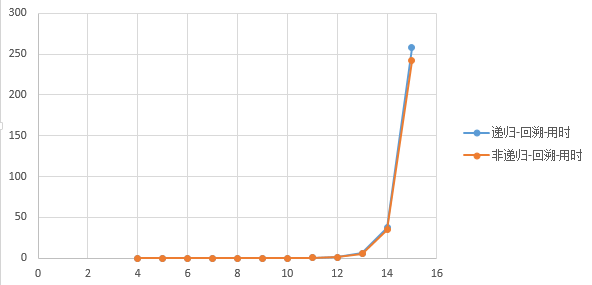


图3 递归回溯与非递归回溯时间

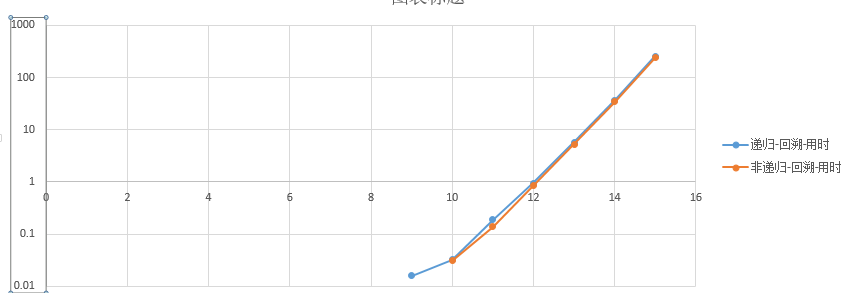


图3-1（纵坐标取对数） 递归回溯与非递归回溯时间

**递归-蛮力法 VS 非递归-蛮力法：**

二者都是穷举所有的可能摆放的n个皇后的位置，区别在于递归-蛮力法使用递归建树来实现，非递归-蛮力法使用循环来实现（类似于钟表的循环）

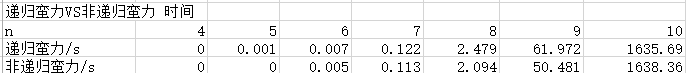


表4 递归蛮力与非递归蛮力时间

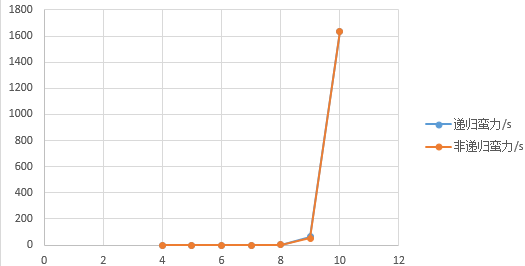


图4 递归蛮力与非递归蛮力时间

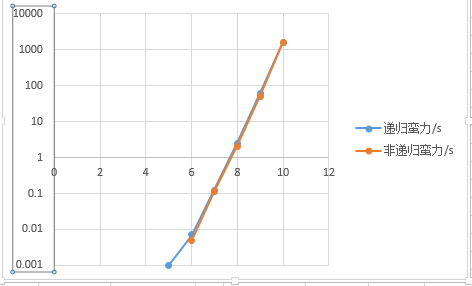


图4-1(纵坐标取对数) 递归蛮力与非递归蛮力时间

**结论：**

从以上的数据测试可以看出，不论是蛮力法还是回溯法，递归与非递归消耗的时间都是差不多的，算法原理是一样的，都是利用的回溯，只是实现的方式不一样，所以在效率上并没有太大差别。回溯法明显比蛮力法效率要高很多，因为回溯法在执行过程中，减去了很多已经确定了不合法的枝节，使得效率大大提高。