重庆邮电大学

课程大作业

学年学期：2019 -2020　学年(2)学期

课程名称：算法分析与设计C

学生学院：计算机科学与技术学院

专业班级：数据科学与大数据专业 班级04082002

学生学号：2020211416

学生姓名：吴易巧

联系电话：15922769772

重庆邮电大学教务处印制

报告正文

学生姓名	吴易巧			学生学号	2020211416
题目	八皇后问题				
组长	贡献量	成员 2	贡献量	成员 3	贡献量
羊欣瑶	33.3%	张思宇	33.3%	吴易巧	33.3%

八皇后问题

一、题目简述

在 n×n 格的国际象棋上摆放 n 个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。

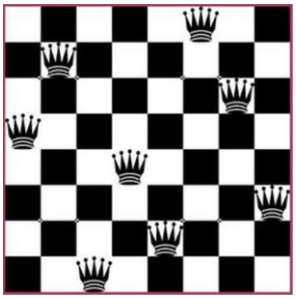


图 1-1 八皇后问题图

二、算法思想

2.1 暴力枚举法

首先对问题进行分析，其本质就是在一个 8x8 的二维空间中，使特定两个数据间需要满足一定的位置条件。因此，最简单也最暴力的办法，就是遍历所有可能的结果，使 64 个数据之间相互计算，也就是要进行C_{64}^{8}的排列组合，检查每一种的可行性。

在此之上进行优化，我们可以根据已知的行规则来进行八层循环，主要代码如下，这显然也是非常消耗算力的。

```
for(solu[1] = 1; solu[1] <= 8; solu[1]++)
    for(solu[2] = 1; solu[2] <= 8; solu[2]++)
        for(solu[3] = 1; solu[3] <= 8; solu[3]++)
            for(solu[4] = 1; solu[4] <= 8; solu[4]++)
                for(solu[5] = 1; solu[5] <= 8; solu[5]++)
                    for(solu[6] = 1; solu[6] <= 8; solu[6]++)
                        for(solu[7] = 1; solu[7] <= 8; solu[7]++)
```

```
8.                          for(solu[8] = 1; solu[8] <= 8; solu[8]++)
9.                              if(不发生矛盾) print;
```

2.2 回溯法

在上述算法的基础上，我们想尽量回避一些在前面的计算中已经不可能成立的情况，于是我们引入了经典的回溯思想进行剪枝。回溯法的核心思想是：

①如果发现错误立即改正，比如上面说到的，我们这里不可能让两个皇后出现在同一列的事情发生，如果有，则重新选择位置。

②如果某一个皇后在一行没有任何位置可以放下去（无论如何都会跟前面的皇后冲突），那么我们有理由认为是前面的皇后放的位置不合理，因此回退到上一个皇后重新放置，这既是回溯。下图是一个四皇后的搜索示意图。

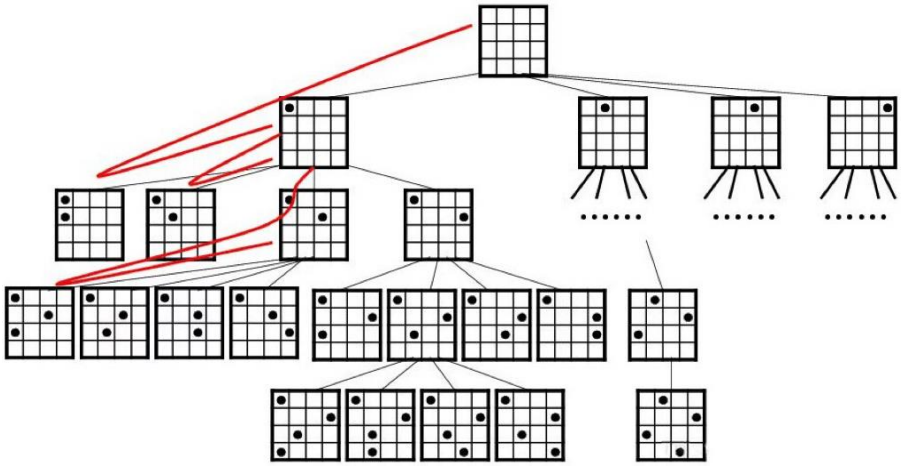


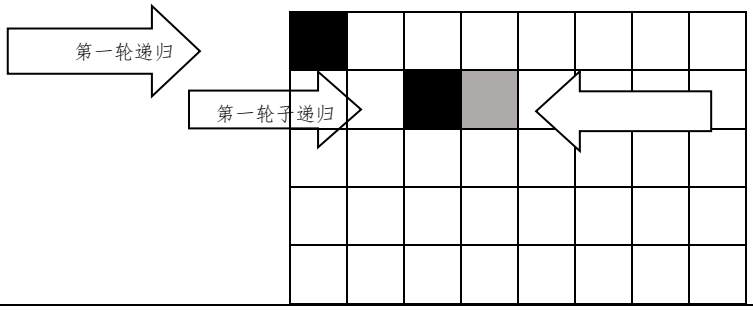
图 2-1 回溯思想流程图

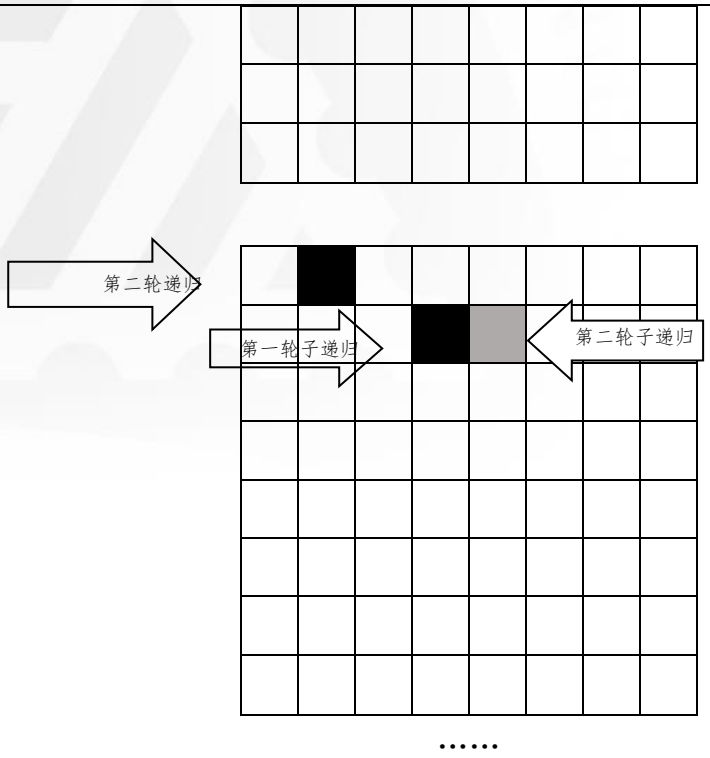
我们通过两个方法实现了该思想，即递归和迭代。

2.3 递归实现

我们发现对每一个皇后摆放时都需遵循同一规则，因此可以采用递归的主要思想来对每一个皇后摆放位置进行是否合理的判断。

递归主要分为对每个皇后的摆放和对某个皇后在各列上摆放的判断：





这里将棋盘抽象为一个一维数组，数组值索引代表行，数组值代表列。两数值差的绝对值等于两数值索引差的绝对值可以表示在同一条斜线上。

2.4 迭代实现

可以利用迭代来实现回溯，即将回溯的皇后位置又当作新的值再次进行赋值，但其本质仍是回溯，与递归实现完成的事情相同。

该实现方法与递归的不同在于：①迭代判断冲突的方式为每次选定两个数据后进行一次计算，迭代则是在每选定一个数据后标记冲突的位置，这会牺牲空间而减少计算开销；②在递归中，我们使用一个一维数组来存储每行皇后的列信息，实际上数组的顺序隐藏着皇后的行信息；在迭代中，我们使用四个一维数组来分别存储行、列、两个对角线的冲突信息；③另外，在迭代中，我们使用了栈来记录已经放置到棋盘上的皇后的状态。

下图以四皇后为例，使用迭代法放置前两行皇后的冲突标记示意图。

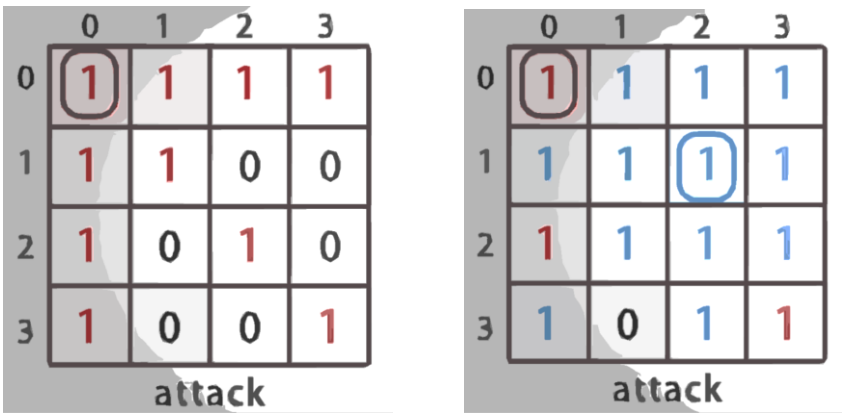


图 2-2 迭代冲突标记示意图

三、实现过程

3.1 递归实现

递归的主要步骤如下：

在第一个皇后摆放在第一列的位置后，递归摆放后续 7 个皇后，若其中第 n 个皇后无法摆放则跳出，回到第 n-1 个皇后将其往右移一步重新摆放，再次对第 n 个皇后进行摆放，依次类推，最后会推演出当第一个皇后摆在第一列时的所有情况；8 个皇后摆放结束后进入第二轮递归，即第一个皇后移到第二列……

主要代码如下：

```
1. void recursion(int n){
2.     if(n==sum){    //8 个皇后摆放结束
3.         result();
4.         return;
5.     }
6.     int i;
7.     for(i=0;i<sum;i++){  //皇后右移摆放
8.         queen[n]=i;
9.         if(isRight(n)==1)
10.             recursion(n+1);    //满足条件，递归进行下一个皇后的摆放
11.     }
12. }
13. int isRight(int n){
14.     int i, flag = 1;
15.     for(i=0;i<n;i++){
16.         if(queen[i]==queen[n]||abs(n-i)==abs(queen[n]-queen[i])){
17.             flag=0;
18.             break;
19.         }
20.     }
21.     return flag;
22. }
```

3.2 迭代实现

迭代的主要步骤如下：

从第一行第一列开始放置皇后，然后在第二行，从左至右，找到第一个可以放置皇后的位置并放置一个皇后。摆放过程中会遇到两种情况：一种情况是，我在某一行找不到放置皇后的合法位置了；另一种情况是，每一行都放置了一个皇后，此时已经构成了一个解，需要寻找下一个解。对于第一种情况，我们需要把当前行的上一行的皇后移除，在代码上

的表现，就是把 q 的位置设置为上一行的皇后位置并从这个位置继续向右，找到第一个可以放置皇后的位置，并放置之；对于第二种情况，前半部分处理与第一种情况相同，后面只需对此种方法计数。

```
do {
  if (solution._size >= N || q.y >= N) //找不到合法位置{
   Queen tmp = PopList(&solution); //将当前皇后弹出回到上一皇后位置
   q.x = tmp.x;
   q.y = tmp.y;
   xarray[q.x] = 0;
   yarray[q.y] = 0;
   sumarray[q.x + q.y] = 0;
   diffarray[q.x - q.y + N ] = 0; //回溯消除标记
    q.y++; //继续向右尝试新的位置
    }
   else{
      while ((q.y < N) && ((xarray[q.x] == 1) || (yarray[q.y] == 1) || (sumarray[q.x + q.y] == 1) || (diffarray[q.x - q.y+N] == 1)){
   //当前位置不合法但不用回溯可以继续尝试的情况
     q.y++;
      }
     if (q.y < N){
     PushList(&solution, q); //将此皇后放入棋盘
     xarray[q.x] = 1;
     yarray[q.y] = 1;
     sumarray[q.x + q.y] = 1;
     diffarray[q.x - q.y+N] = 1; //标记此皇后所在的行、列、对角线
     if (solution._size == N) //已经将皇后放置完成，记录此种方法
      nsolution++;
     q.x++; //完成一行的放置后到下一行寻找恰当的放置位置
     q.y = 0;
    }
   }
 } while ((0 < q.x) || (q.y < N));
 return nsolution;
}
```

四、结果及分析

4.1 结果展示

因为所得到的结果相同，这里只展示一个运行结果。

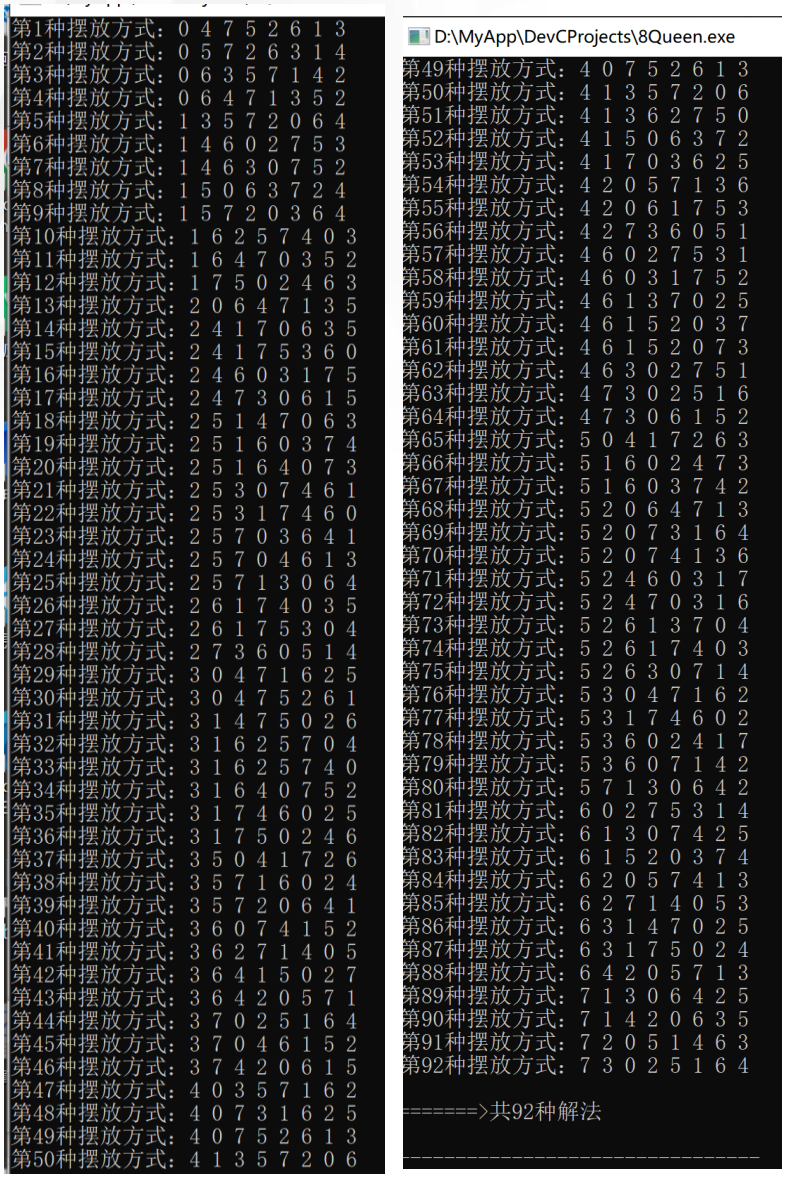


图 4-1 八皇后结果图

4.2 时间复杂度与空间复杂度分析

对于两个算法进行时间复杂度和空间复杂度的分析和比较，结果如下。

表 4-1 算法效率对比

	时间复杂度	空间复杂度	运行时间（n=8）
递归法	O(n!)	O(n)	0.014s
迭代法	O(n!)	O(n)	0.016s

现在我们对这两个算法进行分析，由于它们的本质是相同的，因此它们具有相同的时间复杂度，可以看到递归法和迭代法的运行时间是相差不多的。

回溯法的时间复杂度通常由蒙特卡罗方法计算，蒙特卡罗方法的基本思想是：当求解的问题是某种随机事件出现的概率或数学期望时，通过大量“实验”的方法，以频率估计概率或者得到某些数字特征，将其近似看作问题的解。

我们由蒙特卡罗方法的结论有：排列树的时间复杂度为 O（p(n) * n!），其中，p(n)表示生成一个节点所需的时间复杂度，而在我们使用的迭代和递归中，p(n) = 1，因此时间复杂度为 O（n!）。

由于它们都只需要一维数组的存储空间，因此它们的空间复杂度都为 O（n）。

五、心得体会

①回溯法就像是一颗抽象的树形结构，如果子结点不满足条件，回溯到上一个结点，并“清零”也就是回溯到上一步，重新赋值重新计算；

②在进行需要自身调用自身的情况下，可以采用递归的方法来减少代码量和冗杂的逻辑思考；

③由于采用迭代法而不是传统的递归法解决这个问题，算法需要我们显式地维护一个栈，栈中装载每一行放置的皇后的坐标，通过入栈与出栈，实现回溯。这个栈的结构基于双向链表。