北京師範大學
BEIJING NORMAL UNIVERSITY

《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

一、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。

二、实验过程

a) 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。

b) 问题分析与关键代码（Queen.cpp）

i. 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：

```
int queen[8] = {-1,-1,-1,-1,-1,-1,-1,-1};
int count = 0;
```

a) 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）

b) 解的个数 int count。

2. 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：

```
bool check(int i, int a) {
    // 判断是否冲突
    int j, b;
    for (j = 0; j < i; j++) {
        b = queen[j];
        if (b == a || i - a == j - b || i + a == j + b) {
            // 有同行/副对角线/主对角线元素
            return false;
        }
    }
    return true;
}
```

3. 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：

```
// 递归算法
void putQueen(int i) {
    // 从第i行开始放皇后，要求第i行之前已经放好。
    for (int a = 0; a < 8; a++) {
        if (check(i, a)) {
            queen[i] = a;
            if (i == 7) {
                // 这行放好后，八行全部放好了，输出该放法
                count += 1;
                for (int j = 0; j < 8; j++) printf("%d ", queen[j]);
                printf("\n");
            }
            else putQueen(i + 1); // 这行放好后，还有下一行，进入下一行
        } // endif (check(i, a))
        // check后，试下一个数，进入下一轮循环。a=8时自动结束此层递归
    }
}
```

ii. 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：

```
// 非递归算法
void putQueenIter() {
    int a = 0;
    for (int i = 0; i < 8; i++) {
        if (i == -1) break;     // 判断是否都试完了，退出循环

        for (; a < 8; a++) {
            // 对给定的第i行进行试数
            if (check(i, a)) {
                queen[i] = a;
                // 如果是最后一行：
                if (i == 7) {
                    // 是可行解，计数并打印
                    count++;
                    for (int j = 0; j < 8; j++) printf("%d ", queen[j]);
                    printf("\n");
                    continue;   // 不进入下一行，继续试数，直到a=8
                }
                // 不是最后一行，第i行满足了，进入下一行：
                a = 0; // 进入下一行前调整a，即要重新从0试数
                break; // 进入下一行
            }
        } // endfor (; a < 8; a++)

        // 该行所有数都试过了（有可能已经成功并输出了，也有可能失败，这无关紧要）
        if (a == 8) {
            // 回溯：再让前一行试下一个数。没有前一行可回溯了则结束算法
            a = queen[--i];
            a++;
            queen[i] = -1;
            i--;        // 与for循环中i++抵消
        }
    } // endfor (i = 0; i < 8; i++)
}
```

c) 运行结果（main.cpp）

i. 测试代码：

```
1  #include "Queen.cpp"
2  #include <stdio.h>
3
4  int main() {
5      printf("递归算法\n");
6      putQueen(0);
7      printf("递归算法解的个数%d\n", count);
8
9      for (int i = 0; i < 8; i++)queen[i] = -1;
10     count = 0;
11
12     printf("非递归算法\n");
13     putQueenIter();
14     printf("非递归算法解的个数%d\n", count);
15 }
```

ii. 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）：

```cpp
#include "Queen.cpp"
#include <stdio.h>

int main() {
    printf("递归算法\n");
    putQueen(0);
    printf("递归算法解的个数%d\n", count);

    for (int i = 0; i < 8; i++)queen[i] = -1;
    count = 0;

    printf("非递归算法\n");
    putQueenIter();
    printf("非递归算法解的个数%d\n", count);
}
```

问题 终端 JUPYTER

```
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
```

递归算法

0 4 7 5 2 6 1 3	2 5 1 4 7 0 6 3	3 1 7 5 0 2 4 6	4 2 0 5 7 1 3 6
0 5 7 2 6 3 1 4	2 5 1 6 0 3 7 4	3 5 0 4 1 7 2 6	4 2 0 6 1 7 5 3
0 6 3 5 7 1 4 2	2 5 1 6 4 0 7 3	3 5 7 1 6 0 2 4	4 2 7 3 6 0 5 1
0 6 4 7 1 3 5 2	2 5 3 0 7 4 6 1	3 5 7 2 0 6 4 1	4 6 0 2 7 5 3 1
0 6 4 7 1 3 5 2	2 5 3 1 7 4 6 0	3 6 0 7 4 1 5 2	4 6 0 3 1 7 5 2
1 3 5 7 2 0 6 4	2 5 7 0 3 6 4 1	3 6 2 7 1 4 0 5	4 6 1 3 7 0 2 5
1 4 6 0 2 7 5 3	2 5 7 0 4 6 1 3	3 6 4 1 5 0 2 7	4 6 1 5 2 0 3 7
1 4 6 3 0 7 5 2	2 5 7 1 3 0 6 4	3 6 4 2 0 5 7 1	4 6 1 5 3 0 2 7
1 5 0 6 3 7 2 4	2 6 1 7 4 0 3 5	3 7 0 2 5 1 6 4	4 6 1 7 5 3 0 2
1 5 7 2 0 3 6 4	2 6 1 7 5 3 0 4	3 7 0 4 6 1 5 2	4 6 2 0 5 7 1 3
1 6 2 5 7 4 0 3	2 7 3 6 0 5 1 4	3 7 4 2 0 6 1 5	4 6 2 7 1 4 0 5
1 6 4 7 0 3 5 2	3 0 4 7 1 6 2 5	4 0 3 5 7 1 6 2	5 0 4 1 7 2 6 3
1 7 5 0 2 4 6 3	3 0 4 7 5 2 6 1	4 0 7 3 1 6 2 5	5 1 6 0 2 4 7 3
2 0 6 4 7 1 3 5	3 1 4 6 0 2 7 5	4 0 7 5 2 6 1 3	5 1 6 0 3 7 4 2
2 4 1 7 0 6 3 5	3 1 6 2 5 7 0 4	4 1 3 5 7 2 0 6	5 2 0 6 4 7 1 3
2 4 1 7 5 3 6 0	3 1 6 2 5 7 4 0	4 1 3 6 2 7 5 0	5 2 0 7 3 1 6 4
2 4 6 0 3 1 7 5	3 1 6 4 0 7 5 2	4 1 5 0 6 3 7 2	5 2 0 7 4 1 3 6
2 4 7 3 0 6 1 5	3 1 6 4 2 0 7 5	4 1 7 0 3 6 2 5	5 2 4 6 0 3 1 7
2 5 1 4 7 0 6 3	3 1 7 4 6 0 2 5	4 2 0 5 7 1 3 6	5 2 4 6 0 7 3 1
2 5 1 6 0 3 7 4	3 5 0 4 1 7 2 6	4 2 0 6 1 7 5 3	5 2 4 7 0 3 1 6
2 5 1 6 4 0 7 3	3 5 7 1 6 0 2 4	4 2 7 3 6 0 5 1	5 2 4 7 3 1 6 0
2 5 3 0 7 4 6 1	3 5 7 2 0 6 4 1	4 6 0 2 7 5 3 1	5 2 6 0 3 1 7 4
2 5 3 1 7 4 6 0	3 6 0 7 4 1 5 2	4 6 0 3 1 7 5 2	5 2 6 1 3 7 0 4
2 5 7 0 3 6 4 1	3 6 2 7 1 4 0 5	4 6 1 3 7 0 2 5	5 2 6 1 7 4 0 3
2 5 7 0 4 6 1 3	3 6 4 1 5 0 2 7	4 6 1 5 2 0 3 7	5 2 6 3 0 7 1 4
2 5 7 1 3 0 6 4	3 6 4 2 0 5 7 1	4 6 1 5 3 0 2 7	5 2 6 3 1 7 4 0
2 6 1 7 4 0 3 5	3 7 0 2 5 1 6 4	4 6 1 7 5 3 0 2	5 2 7 0 3 1 6 4
2 6 1 7 5 3 0 4	3 7 0 4 6 1 5 2	4 6 2 0 5 7 1 3	5 2 7 0 4 6 1 3
2 7 3 6 0 5 1 4	3 7 4 2 0 6 1 5	4 6 2 7 1 4 0 5	5 3 0 4 1 7 2 6
3 0 4 7 1 6 2 5	4 0 3 5 7 1 6 2	5 0 4 1 7 2 6 3	5 3 1 6 0 2 4 7
3 0 4 7 5 2 6 1	4 0 7 3 1 6 2 5	5 1 6 0 2 4 7 3	5 3 1 6 0 7 4 2
3 1 4 6 0 2 7 5	4 0 7 5 2 6 1 3	5 1 6 0 3 7 4 2	5 3 1 7 4 6 0 2
3 1 6 2 5 7 0 4	4 1 3 5 7 2 0 6	5 2 0 6 4 7 1 3	5 3 6 0 2 4 1 7
3 1 6 2 5 7 4 0	4 1 3 6 2 7 5 0	5 2 0 7 3 1 6 4	5 3 6 0 7 1 4 2
3 1 6 4 0 7 5 2	4 1 5 0 6 3 7 2	5 2 0 7 4 1 3 6	5 7 1 3 0 6 4 2
3 1 7 4 6 0 2 5	4 1 7 0 3 6 2 5	5 2 4 6 0 3 1 7	6 0 2 7 5 3 1 4
3 5 0 4 1 7 2 6	4 2 0 5 7 1 3 6	5 2 4 6 0 7 3 1	6 1 3 0 7 4 2 5
3 5 7 1 6 0 2 4	4 2 0 6 1 7 5 3	5 2 4 7 0 3 1 6	6 1 5 2 0 3 7 4
3 5 7 2 0 6 4 1	4 2 7 3 6 0 5 1	5 2 4 7 3 1 6 0	6 2 0 5 7 4 1 3
3 6 0 7 4 1 5 2	4 6 0 2 7 5 3 1	5 2 6 0 3 1 7 4	6 2 7 1 4 0 5 3
3 6 2 7 1 4 0 5	4 6 0 3 1 7 5 2	5 2 6 1 3 7 0 4	6 3 1 4 7 0 2 5
3 6 4 1 5 0 2 7	4 6 1 3 7 0 2 5	5 2 6 1 7 4 0 3	6 3 1 7 5 0 2 4
3 6 4 2 0 5 7 1	4 6 1 5 2 0 3 7	5 2 6 3 0 7 1 4	6 4 2 0 5 7 1 3
3 7 0 2 5 1 6 4	4 6 1 5 3 0 2 7	5 2 6 3 1 7 4 0	7 1 3 0 6 4 2 5
3 7 0 4 6 1 5 2	4 6 1 7 5 3 0 2	5 2 7 0 3 1 6 4	7 1 4 2 0 6 3 5
3 7 4 2 0 6 1 5	4 6 2 0 5 7 1 3	5 2 7 0 4 6 1 3	7 2 0 5 1 4 6 3
4 0 3 5 7 1 6 2	4 6 2 7 1 4 0 5	5 3 0 4 1 7 2 6	7 3 0 2 5 1 6 4
4 0 7 3 1 6 2 5	5 0 4 1 7 2 6 3	5 3 1 6 0 2 4 7	
4 0 7 5 2 6 1 3	5 1 6 0 2 4 7 3	5 3 1 6 0 7 4 2	
4 1 3 5 7 2 0 6	5 1 6 0 3 7 4 2	5 3 1 7 4 6 0 2	
4 1 3 6 2 7 5 0	5 2 0 6 4 7 1 3	5 3 6 0 2 4 1 7	
4 1 5 0 6 3 7 2	5 2 0 7 3 1 6 4	5 3 6 0 7 1 4 2	
4 1 7 0 3 6 2 5	5 2 0 7 4 1 3 6	5 7 1 3 0 6 4 2	
4 2 0 5 7 1 3 6	5 2 4 6 0 3 1 7	6 0 2 7 5 3 1 4	
4 2 0 6 1 7 5 3	5 2 4 6 0 7 3 1	6 1 3 0 7 4 2 5	
4 2 7 3 6 0 5 1	5 2 4 7 0 3 1 6	6 1 5 2 0 3 7 4	
4 6 0 2 7 5 3 1	5 2 4 7 3 1 6 0	6 2 0 5 7 4 1 3	
4 6 0 3 1 7 5 2	5 2 6 0 3 1 7 4	6 2 7 1 4 0 5 3	
4 6 1 3 7 0 2 5	5 2 6 1 3 7 0 4	6 3 1 4 7 0 2 5	
4 6 1 5 2 0 3 7	5 2 6 1 7 4 0 3	6 3 1 7 5 0 2 4	
4 6 1 5 3 0 2 7	5 2 6 3 0 7 1 4	6 4 2 0 5 7 1 3	
4 6 1 7 5 3 0 2	5 2 6 3 1 7 4 0	7 1 3 0 6 4 2 5	
4 6 2 0 5 7 1 3	5 2 7 0 3 1 6 4	7 1 4 2 0 6 3 5	
4 6 2 7 1 4 0 5	5 2 7 0 4 6 1 3	7 2 0 5 1 4 6 3	
5 0 4 1 7 2 6 3	5 3 0 4 1 7 2 6	7 3 0 2 5 1 6 4	
5 1 6 0 2 4 7 3	5 3 1 6 0 2 4 7		
5 1 6 0 3 7 4 2	5 3 1 6 0 7 4 2		
5 2 0 6 4 7 1 3	5 3 1 7 4 6 0 2		
5 2 0 7 3 1 6 4	5 3 6 0 2 4 1 7		
5 2 0 7 4 1 3 6	5 3 6 0 7 1 4 2		
5 2 4 6 0 3 1 7	5 7 1 3 0 6 4 2		
5 2 4 6 0 7 3 1	6 0 2 7 5 3 1 4		
5 2 4 7 0 3 1 6	6 1 3 0 7 4 2 5		
5 2 4 7 3 1 6 0	6 1 5 2 0 3 7 4		
5 2 6 0 3 1 7 4	6 2 0 5 7 4 1 3		
5 2 6 1 3 7 0 4	6 2 7 1 4 0 5 3		
5 2 6 1 7 4 0 3	6 3 1 4 7 0 2 5		
5 2 6 3 0 7 1 4	6 3 1 7 5 0 2 4		
5 2 6 3 1 7 4 0	6 4 2 0 5 7 1 3		
5 2 7 0 3 1 6 4	7 1 3 0 6 4 2 5		
5 2 7 0 4 6 1 3	7 1 4 2 0 6 3 5		
5 3 0 4 1 7 2 6	7 2 0 5 1 4 6 3		
5 3 1 6 0 2 4 7	7 3 0 2 5 1 6 4		
5 3 1 6 0 7 4 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			

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 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 6 2 7 1 4 0 5
3 6 4 1 5 0 2 7
3 6 4 2 0 5 7 1
3 6 4 2 0 7 5 1
3 7 2 0 5 1 6 4
4 0 2 7 5 3 1 6
4 0 3 5 7 1 6 2
4 0 7 3 1 6 2 5
4 0 7 3 1 6 5 2
4 1 3 5 7 0 2 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 6 1
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。