目录

目录
1.项目背景
2.需求分析
3.设计思路
- 棋盘
- 递归求解

4.核心代码说明
- 函数说明
  - 初始化棋盘
  - 放置判断
  - 放置棋子
  - 打印棋盘
  - 递归函数
- 函数接口说明

5.使用方法及函数功能演示
- 输入数据
- 输出数据

1.项目背景

八皇后问题是一个古老而著名的问题，是回溯算法的经典问题。该问题是十九世纪著名的数学家高斯在1850年提出的：在8*8的国际象棋棋盘上，安放8个皇后，要求没有一个皇后能够“吃掉”任何其它一个皇后，即任意两个皇后不能处于同一行，同一列或者同一条对角线上，求解有多少种摆法。

高斯认为有76种方案。1854年在柏林的象棋杂志上不同的作者发表了40种不同的解，后来有人用图论的方法得到结论，有92中摆法。

本程序拓展了N皇后问题，即皇后个数由用户输入。

2.需求分析

根据用户输入的N值，计算输出N皇后问题的所有可能情况

3.设计思路

棋盘

由于每一行最多只能够放置一个棋子，所以常用一维数组存储棋盘：

`board[i]=j`表示在第i行第j列放置皇后

棋盘初始化值为-1，表示没有该行没有棋子

递归求解

通过递归方法求解，以第0行为参数调用递归函数，递归函数`placeQueen`思路如下：

- 当前行是否为第N行，若是，说明0~N-1行已经执行完毕，棋盘布局已经完成，输出布局，方法计数+1，退出递归
- 从第0列开始至第N-1列，检测当前列是否满足放置条件（同行、同列、对角线是否有棋子）
  - 不满足：继续检测下一列
  - 满足：以下一行为行号参数递归调用`placeQueen`函数
- 如果该行所有列检测完后都不满足放置条件，则清空棋盘，说明该种放置方法无解

最后输出共计有多少种方法

4.核心代码说明

函数说明

初始化棋盘

```
//Initial the chessboard
void initial(int* board,int N) {
  for (int i = 0; i < N; i++) {
    board[i] = -1;
  }
}
```

将棋盘各行存储的列号初始化为-1

放置判断

```
//Judge whether the point can be placed
bool valid(int *board,int N,int row, int col) {
  for (int i = 0; i < N; i++) {
    //As we use single dimension array,we needn't consider conflict of row,we just judge the conflict of column of diagonal
    if (board[i] == col || (abs(i - row) == abs(board[i] - col) && board[i]!= -1))
      return false;
  }
  return true;
}
```

判断当前位置能否满足放置条件，因为用行号为下标存储列号，所以不需要考虑行号冲突，通过遍历棋盘数组检测是否在列与对角线上满足条件（没有棋子），返回一个bool值，`true`表示可以防止，`false`表示不可以放置

放置棋子

```
//Place queen on the position
void place(int *board,int row, int col) {
    board[row] = col;
}
```

用行号为下标存储列号

打印棋盘

```
//Print the chessboard
void printBoard(int *board,int N) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      if (j == board[i])
        cout << 'X';
      else
        cout << '0';
    }
    cout << endl;
  }
  cout << endl;
}
```

读取`board[]`数组，将其转化为可视化的棋盘布局，X表示该位置有棋子，0表示没有

递归函数

```
//Use recursion to find solutions
void placeQueen(int *board, int N, int row) {
  //If the last row have found,print the solution
  if (row == N) {
    printBoard(board,N);
    counter++;
  }
  else {
    for (int i = 0; i < N; i++) {
      if (valid(board,N, row, i)) {
        place(board,row, i);
        placeQueen(board,N,row + 1);
      }
    }
    //If this row cannot find answer,clean the board
    board[row] = -1;
  }
}
```

首先判断行号，若行号为N，说明布局已经完成，输出当前布局，方法计数器+1；

若布局未完成，从该行0~N-1列，依次执行一下操作：

- 检测当前列是否满足放置条件（同行、同列、对角线是否有棋子）
  - 不满足：继续检测下一列
  - 满足：以下一行为行号参数递归调用`placeQueen`函数
- 如果该行所有列检测完后都不满足放置条件，则清空棋盘，说明该种放置方法无解

最后输出共计有多少种方法

函数接口说明

返回值类型	成员函数名	参数	属性	功能
void	initial	(int *board,int N)	非成员函数	初始化棋盘
bool	valid	(int *board,int N,int row, int col)	非成员函数	判断能否放置棋子
void	place	(int *board,int row, int col)	非成员函数	在棋盘的row行col列放置棋子
void	printBoard	(int *board,int N)	非成员函数	打印棋盘
void	placeQueen	(int *board,int N,int row)	非成员函数	递归函数

5.使用方法及函数功能演示

输入数据

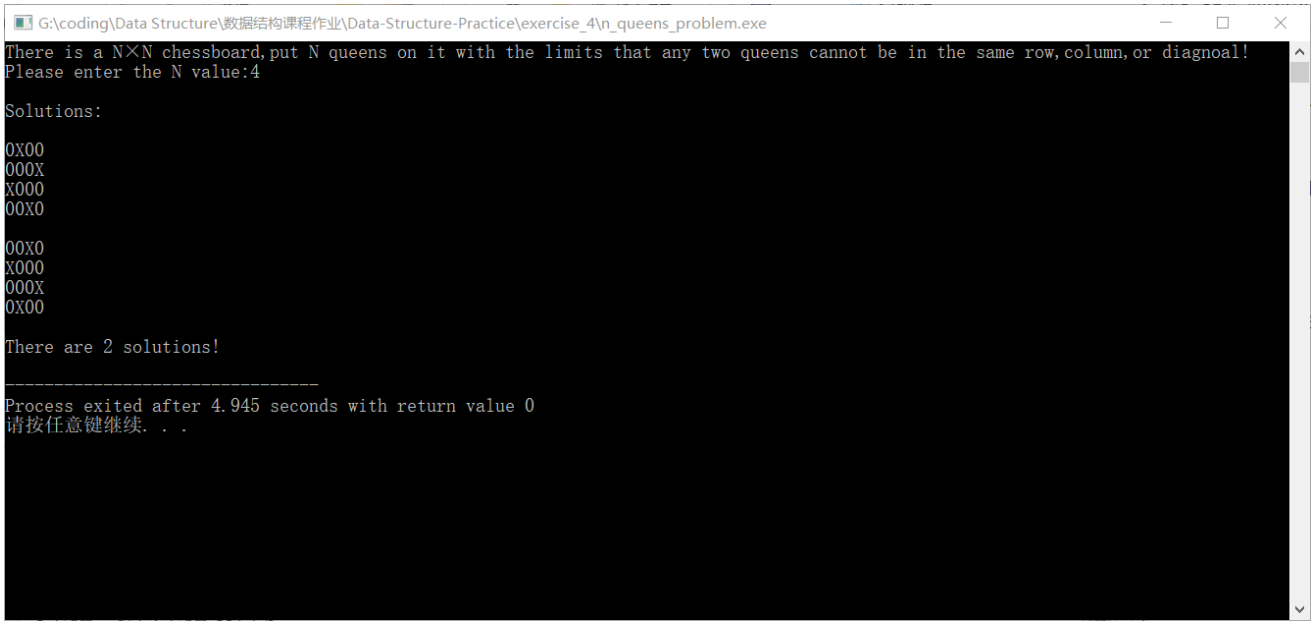
双击运行`n_queens_problem.exe`，启动程序，根据提示输入N值

```
There is a N×N chessboard,put N queens on it with the limits that any two queens cannot be in the same row,column,or diagnoal!
Please enter the N value:4
```

输出数据

输入完N值后，程序会自动计算出所有符合条件的情况，输出分为两个部分：

- 每种情况的具体排布
- 所有情况总数



程序结束，需再次运行则重启