

N 皇后问题

一、实验目的

- 1、理解 N 皇后问题的基本概念与数学模型。
- 2、掌握递归与回溯算法的设计思想及实现方法。
- 3、学会使用 Python + PyQt5 + Matplotlib 进行算法可视化。
- 4、通过程序运行与图形化展示，加深对搜索算法效率与结果规律的理解。

二、实验原理

1、问题描述

N 皇后问题是经典的组合优化问题：
在 $N \times N$ 的国际象棋棋盘上，放置 N 个皇后，使得任意两个皇后都不在同一行、同一列或同一条对角线上。目标是求出所有可能的放置方式（即所有解）。

2、算法思想——回溯法

回溯算法是一种深度优先搜索（DFS）思想的实现：

- 每次在一行中选择一个合法列放置皇后；
- 若当前放置不冲突，则继续递归搜索下一行；
- 若出现冲突（不可放置），则回溯到上一步，尝试其他列；
- 当所有行都放置完毕时，记录一个完整解。

3、冲突检测逻辑

为了判断某位置是否能放置皇后，定义三个标记数组：

标记数组	含义	长度
<code>label_y[c]</code>	第 c 列是否已有皇后	n
<code>label_up[r+c]</code>	主对角线（左上到右下）是否冲突	$2n-1$
<code>label_do[r-c+n-1]</code>	副对角线（右上到左下）是否冲突	$2n-1$

若三个标志均为 0，则位置 (r, c) 安全。

4、可视化原理

程序利用 Matplotlib 动态绘制棋盘，每个格子颜色交替显示，皇后以 “♛” 符号标记。
通过 PyQt 界面按钮可查看所有解法，增强算法直观性。

三、实验环境

- 1、Python 版本：3.12
- 2、图形界面库：PyQt5
- 3、可视化库：Matplotlib
- 4、在 PyCharm Community Edition 下编译
(运行时 py 文件需要和 background.png 文件在统一文件夹下运行以加载背景图)

四、程序代码

```
import sys
import matplotlib
matplotlib.use("Qt5Agg")
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QLabel, QLineEdit, QPushButton, QMessageBox, QGraphicsDropShadowEffect
)
from PyQt5.QtGui import QPixmap, QColor, QFont
from PyQt5.QtCore import Qt

# ===== 全局变量 =====
success_count = []
current_index = 0

# ===== n 皇后算法 =====
def solve_n_queens(n):
    success_count = []
    label_y = [0] * n
    label_up = [0] * (2 * n - 1)
    label_do = [0] * (2 * n - 1)
    board = [-1] * n

    def search(r):
        if r == n:
            success_count.append(board[:])
            return
        for c in range(n):
            if not label_y[c] and not label_up[r + c] and not label_do[r - c + n
- 1]:
                board[r] = c
                label_y[c] = label_up[r + c] = label_do[r - c + n - 1] = 1
                search(r + 1)
                label_y[c] = label_up[r + c] = label_do[r - c + n - 1] = 0

    search(0)
    return success_count

# ===== 绘制棋盘 =====
def draw_board(n, queens):
    fig, ax = plt.subplots(figsize=(6, 6), dpi=120)
    ax.set_xlim(0, n)
    ax.set_ylim(0, n)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect('equal')
```

```

    for i in range(n):
        for j in range(n):
            color = 'white' if (i + j) % 2 == 0 else 'lightgrey'
            rect = plt.Rectangle((j, n - i - 1), 1, 1, facecolor=color,
                                edgecolor='black', lw=1.5)
            ax.add_patch(rect)

    for r, c in enumerate(queens):
        ax.text(c + 0.5, n - r - 0.5, "♔", ha='center', va='center', fontsize=28,
               color='black')

    fig.patch.set_alpha(0)
    ax.patch.set_alpha(0)
    plt.close(fig)
    return fig

# ===== 主窗口 =====
class NQueenApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("N 皇后可视化")
        self.setGeometry(200, 80, 1100, 750)

        # 背景图
        self.bg_label = QLabel(self)
        pix = QPixmap("background.png")

        self.bg_label.setPixmap(pix)
        self.bg_label.setScaledContents(True)
        self.bg_label.lower()

        # 中心容器
        self.central = QWidget(self)
        self.setCentralWidget(self.central)

        # 主布局
        self.main_layout = QHBoxLayout(self.central)
        self.main_layout.setContentsMargins(50, 50, 50, 50)

        # ----- 棋盘区域 -----
        self.canvas_container = QVBoxLayout()
        self.main_layout.addLayout(self.canvas_container, 2)
        self.canvas = None

        # ----- 控件面板 -----
        self.control_panel = QWidget(self)
        self.control_panel.setFixedWidth(250)
        self.control_panel.setStyleSheet("""

```

```

        background-color: rgba(0, 0, 0, 160);
        border-radius: 15px;
    """)
    shadow = QGraphicsDropShadowEffect()
    shadow.setBlurRadius(25)
    shadow.setOffset(0, 0)
    shadow.setColor(QColor(0, 0, 0, 150))
    self.control_panel.setGraphicsEffect(shadow)

    control_layout = QVBoxLayout(self.control_panel)
    control_layout.setContentsMargins(20, 20, 20, 20)
    control_layout.setSpacing(15)

    title = QLabel("N 皇后问题")
    title.setFont(QFont("Microsoft YaHei", 16, QFont.Bold))
    title.setStyleSheet("color: white;")
    control_layout.addWidget(title)

    label = QLabel("输入 n: ")
    label.setFont(QFont("Microsoft YaHei", 14, QFont.Bold))
    label.setStyleSheet("color: white;")
    control_layout.addWidget(label)

    self.input = QLineEdit()
    self.input.setFixedWidth(80)
    self.input.setStyleSheet("""
        QLineEdit {
            background-color: rgba(255,255,255,200);
            border: none;
            border-radius: 5px;
            padding: 3px;
            font-size: 14px;
        }
    """)
    control_layout.addWidget(self.input)

    self.run_btn = QPushButton("执行")
    self.next_btn = QPushButton("下一个解法")
    for btn in [self.run_btn, self.next_btn]:
        btn.setStyleSheet("""
            QPushButton {
                background-color: rgba(255,255,255,180);
                border-radius: 8px;
                padding: 6px;
                font-size: 14px;
            }
            QPushButton:hover {
                background-color: rgba(255,255,255,230);
            }
        """)

```

```

        """)
        btn.setCursor(Qt.PointingHandCursor)

self.run_btn.clicked.connect(self.run)
self.next_btn.clicked.connect(self.show_next)
control_layout.addWidget(self.run_btn)
control_layout.addWidget(self.next_btn)

self.info_label = QLabel(" ")
self.info_label.setStyleSheet("font-size:13px; color:white;")
control_layout.addWidget(self.info_label)
control_layout.addStretch(1)

# 把控制面板固定在左下角
self.main_layout.addWidget(self.control_panel, alignment=Qt.AlignLeft |
Qt.AlignBottom)

def resizeEvent(self, event):
    super().resizeEvent(event)
    self.bg_label.resize(self.size())

# ----- 功能 -----
def run(self):
    global success_count, current_index
    text = self.input.text().strip()
    if not text.isdigit() or int(text) <= 0:
        QMessageBox.warning(self, "错误", "请输入正整数 n。")
        return
    n = int(text)

    QApplication.processEvents()

    success_count = solve_n_queens(n)
    current_index = 0
    if not success_count:
        self.info_label.setText("无解")
        self._clear_canvas()
    else:
        self.info_label.setText(f"共 {len(success_count)} 种解。当前第 1 种
")
        self.show_solution(n, success_count[0])

def show_next(self):
    global current_index
    if not success_count:
        QMessageBox.information(self, "提示", "请先执行计算。")
        return
    n = len(success_count[0])
    current_index = (current_index + 1) % len(success_count)

```

```

        self.info_label.setText(f"共 {len(success_count)} 种解。当前第  
{current_index + 1} 种")
        self.show_solution(n, success_count[current_index])

    def _clear_canvas(self):
        if self.canvas:
            self.canvas.setParent(None)
            self.canvas = None

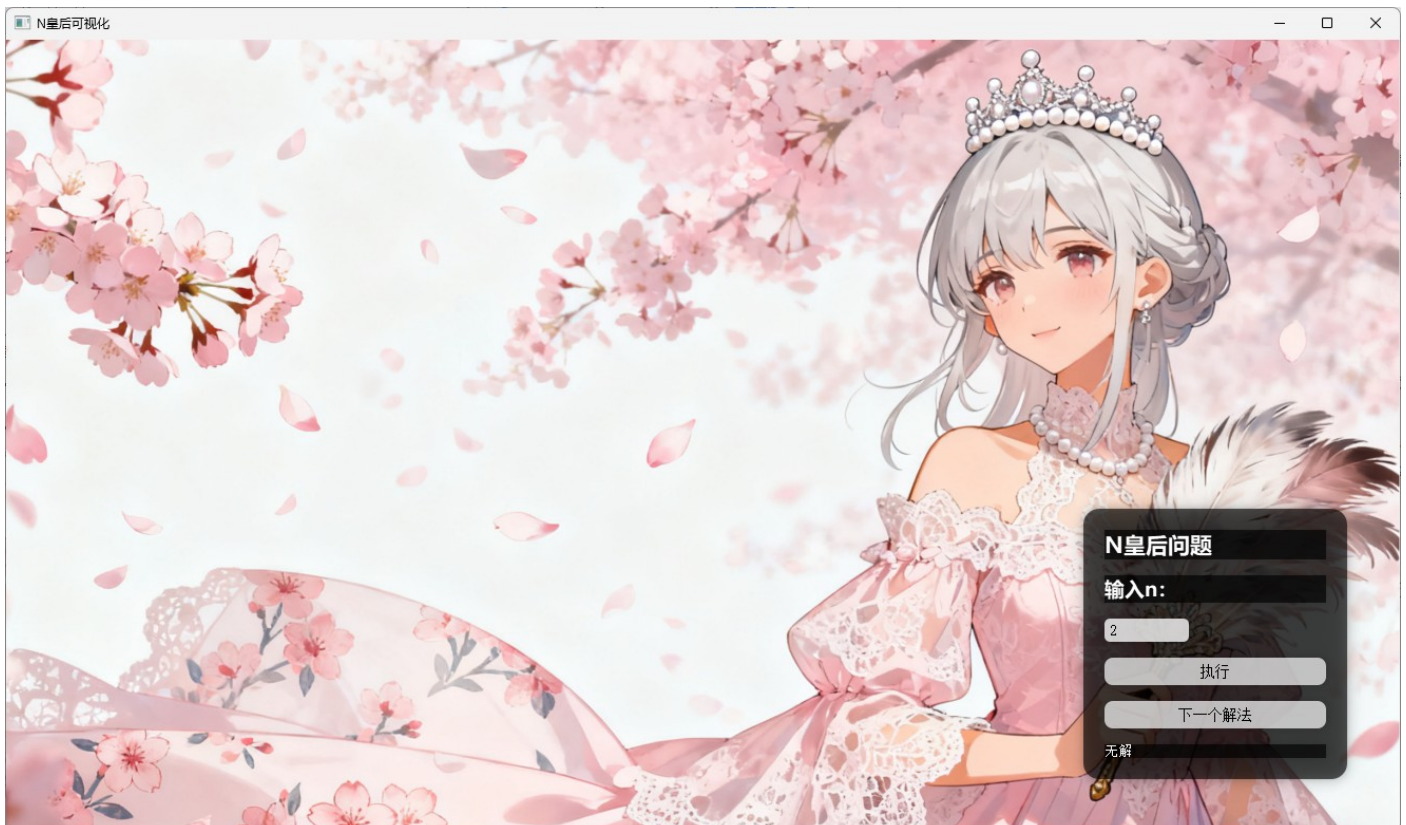
    def show_solution(self, n, queens):
        self._clear_canvas()
        fig = draw_board(n, queens)
        self.canvas = FigureCanvas(fig)
        self.canvas.setStyleSheet("background: transparent;")
        self.canvas_container.addWidget(self.canvas, alignment=Qt.AlignCenter)
        self.canvas.draw()

# ===== 主程序 =====
if __name__ == "__main__":
    app = QApplication(sys.argv)
    w = NQueenApp()
    w.show()
    sys.exit(app.exec_())

```

五、实验结果显示

1、 $N = 2$



2、N = 4

N皇后可视化

	♔		
			♔
♔			
		♔	

N皇后问题

输入n:

4

执行

下一个解法

共 2 种解。当前第 1 种

3、N = 8

N皇后可视化

♔							
				♔			
							♔
					♔		
		♔					
						♔	
	♔						
			♔				

N皇后问题

输入n:

8

执行

下一个解法

共 92 种解。当前第 1 种

六、实验分析总结

1、时间复杂度

回溯算法在最坏情况下需要搜索 $O(N!)$ 种排列，但剪枝优化（列、主对角线、副对角线标记）极大减少了搜索空间。

2、算法特点

优点：实现简单、思路清晰、解的遍历完整；

缺点：在 N 较大时计算量快速增长（指数级）。

3、结果特征

$N=2, 3$ 无解；

$N \geq 4$ 均有解；

随 N 增大，解的数量增长迅速，分布呈对称性。

4、可视化效果分析

图形界面交互良好；Matplotlib 绘制清晰；可通过按钮逐一查看不同解，方便教学展示。