八皇后问题求解

林子清
2212279
智能科学与技术
nkuailzq@gmail.com

1 问题概述

八皇后问题，是一个古老而著名的问题：如何能够在 8Œ8 的国际象棋棋盘上放置八个皇后，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。最早是由国际西洋棋棋手马克斯 · 贝瑟尔于 1848 年提出，之后陆续有数学家对其进行研究，其中包括高斯和康托。八皇后问题可以推广为更一般的 N 皇后摆放问题：这时棋盘的大小变为 NŒN，而皇后个数也变成 N。

2 实验目的

1. 通过求解皇后问题，熟悉深度优先搜索法技术
2. 理解递归回溯算法思想，进而推广到 N 皇后问题
3. 对实验进行图形化界面设计，实现按步或按解的展示

3 实验内容

1. 实现八皇后问题的解法统计
2. 将八皇后问题推广到 N 皇后
3. 图形化界面设计

4 实验步骤

代码通过`Board` 封装了 N 皇后问题的求解器和棋盘的绘制，并使用`tkinter` 构建 UI。代码已经发布在`https://github.com/nkulzq/AILab.git`

4.1 N 皇后求解器

求解器中利用深度优先遍历为每一行中符合`check ()` 要求的位置分配皇后，并且在解出现后回溯。值得注意的是，为了实现连续运行和分步运行的公用，我们利用`yield` 形成步骤列表，并在求解器中遍历列表。这样实现同时还做到了可以单步求解和连续求解的混合使用。

```
def check(self, col, row):
    for i in range(row):
        if self.res[i] == col\
            or self.res[i] + i == row + col or\
            self.res[i] - i == col - row:
            return False
    return True

def solve(self, row):
    self.clear_board()
    self.draw_solution()
    yield
    if row == self.size:
        self.num_solutions += 1
        return
    for col in range(self.size):
        if self.check(col, row):
            self.res[row] = col
            yield from self.solve(row + 1)
            self.res[row] = -1
            self.clear_board()
            self.draw_solution()
            yield
```

Listing 1: 深度优先搜索

```
def solve_step(self):
try:
    next(self.solver)
except StopIteration:
    self.finish = True
```

Listing 2: 分步求解器

```
def solve_continuous(self):
while True:
    try:
        next(self.solver)
        self.canvas.update()
    except StopIteration:
        self.finish = True
        break
```

Listing 3: 连续求解器

4.2 棋盘

棋盘类将绘制封装起来，在求解器中可以直接调用其中的绘制与重绘函数。

```
class Board:
def __init__(self, canvas, size=4):
    self.canvas = canvas
    self.size = size
    self.square_size = 400 // size
    canvas_width = self.square_size * size
    canvas_height = self.square_size * size
    canvas.config(width=canvas_width, height=canvas_height)
    self.piece_radius = self.square_size // 4
    self.res = [-1] * self.size
    self.num_solutions = 0
    self.draw_board()
    self.solver = self.solve(0)
    self.finish = False

def reset(self):
    self.square_size = 400 // self.size
    canvas_width = self.square_size * self.size
    canvas_height = self.square_size * self.size
    self.canvas.config(width=canvas_width, height=canvas_height)
    self.piece_radius = self.square_size // 4
    self.res = [-1] * self.size
    self.num_solutions = 0
    self.draw_board()
    self.solver = self.solve(0)
    self.finish = False
```

Listing 4: 棋盘类

```
def clear_board(self):
    self.canvas.delete('pieces')

def draw_board(self):
square_size = 400 // self.size
for row in range(self.size):
    for col in range(self.size):
        x1 = col * square_size
        y1 = row * square_size
        x2 = x1 + square_size
        y2 = y1 + square_size
        color = 'black' if (row + col) % 2 == 0 else 'white'
        self.canvas.create_rectangle\
        (x1, y1, x2, y2, fill=color, outline='')

def draw_solution(self):
    for row, col in enumerate(self.res):
        if col != -1:
            x_center = col * self.square_size + self.square_size // 2
            y_center = row * self.square_size + self.square_size // 2
            self.canvas.create_oval(
                x_center - self.piece_radius,\
                y_center - self.piece_radius,
                x_center + self.piece_radius,\
                y_center + self.piece_radius,
                fill='red', tags='pieces'
            )
```

Listing 5: 绘制

4.3 UI

在 UI 中我们设计了单步运行，连续运行，皇后数量输入，显示解的数量这几个功能按键。其中显示解的数量要求只有在求解器完成求解后才会变更为可访问状态。同时为了界面的美观，我们重写了库中的messagebox 类，并且新增了按钮状态绑定函数。

```
def get_size():
    board.size = int(size_entry.get())
    board.reset()


def show_num_solutions():
    message = MessageBox(\
    root, "number of solutions",\
    "{} queen problem has {} solutions"\
    .format(board.size, board.num_solutions))
    root.wait_window(message)


def solve_step():
    board.solve_step()
    toggle_button_state(button_solutions, board.num_solutions)


def solve_continuous():
    board.solve_continuous()
    toggle_button_state(button_solutions, board.num_solutions)
```

Listing 6: 按钮函数

```
import tkinter as tk
from tkinter import ttk
from n_queen_prob import Board
from utils import MessageBox, toggle_button_state
import sv_ttk

root = tk.Tk()
root.title('n-queen problem')
root.geometry("750x500+630+80")
canvas = tk.Canvas(root)
board = Board(canvas)
button_step = ttk.Button(root, text="step", command=solve_step)
button_continuous = ttk.Button(root, text="continuous", command=solve_continuous)
size_entry = ttk.Entry(root)
button_submit = ttk.Button(root, text="submit", command=get_size)
button_solutions =\
ttk.Button(root, text="solutions",\
command=show_num_solutions, state=tk.NORMAL if board.finish else tk.DISABLED)
canvas.place(x=50, y=50)
button_step.place(x=550, y=25, width=100, height=30)
button_continuous.place(x=550, y=125, width=100, height=30)
size_entry.place(x=550, y=225, width=100, height=30)
button_submit.place(x=550, y=325, width=100, height=30)
button_solutions.place(x=550, y=425, width=100, height=30)
sv_ttk.set_theme("dark")
root.mainloop()
```

Listing 7: 主界面

```
class MessageBox(tk.Toplevel):
    def __init__(self, master, title, message):
        super().__init__(master)
        self.title(title)
        self.geometry("300x150+630+80")
        self.transient(master)
        self.grab_set()
        self.message_label = ttk.Label(self,\
        text=message, wraplength=250, justify=tk.LEFT)
        self.message_label.pack(pady=20, padx=20)
        self.ok_button = ttk.Button(self, text="OK", command=self.destroy)
        self.ok_button.pack(pady=10)


    def toggle_button_state(button, state):
        if state:
            button.config(state=tk.NORMAL)
        else:
            button.config(state=tk.DISABLED)
```

Listing 8: 重写库函数

5 实验结果

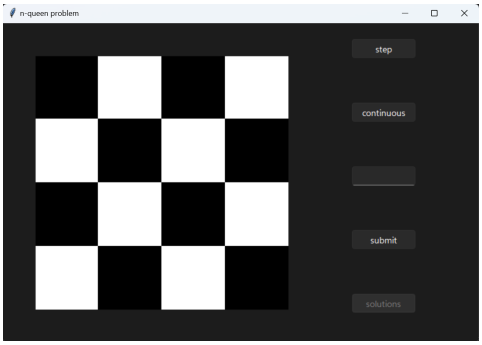

图 1: 初始界面

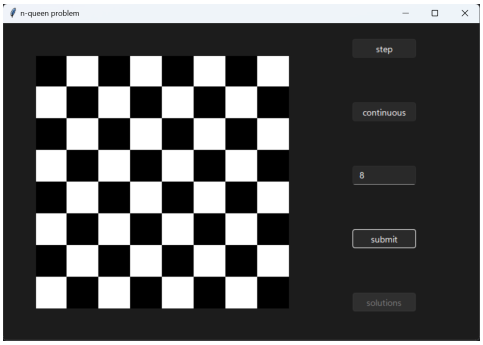

图 2: 重置皇后数

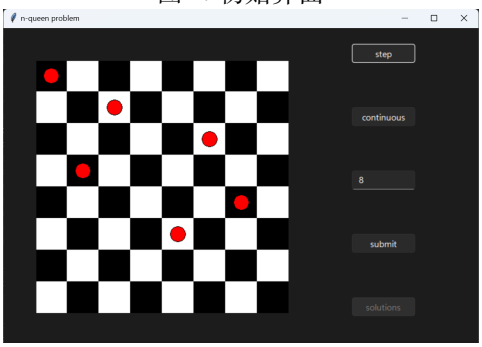

图 3: 运行过程

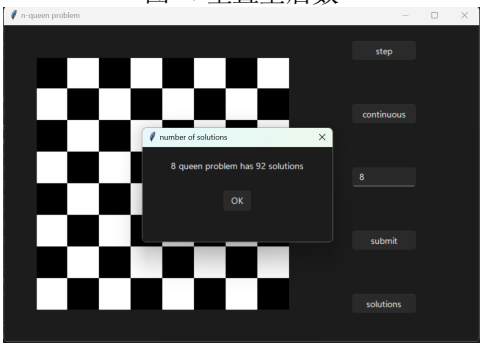

图 4: 报告解的数量

6 分析总结

n 皇后问题的解的个数是一个 **NP-Complete** 问题，若使用递归求解则其时间复杂度为 $\mathcal{O}(n!)$，这样的问题在 n 增大时会变得难以求解。但这个问题的简化问题：找出一个可行解已经被找到了构造方法，时间复杂度变为 $\mathcal{O}(1)$，而我们还可以使用一些启发式算法，例如以冲突个数为代价函数进行局部微调搜索的方法获取更多可行解。

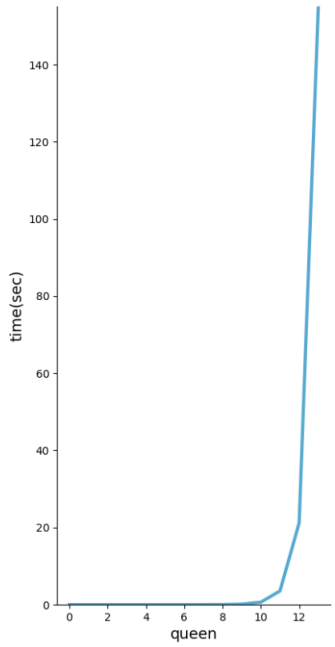

图 5: 消耗时间

References

[1]Hoffman, E. J., et al. "Constructions for the Solution of the m Queens Problem." Mathematics Magazine, vol. 42, no. 2, 1969, pp. 66–72. JSTOR, https://doi.org/10.2307/2689192.