人工智慧 Lab4

0116F137 陳廣能

實驗內容描述：

這份程式碼實作了一個 N-Queens 求解器（包含 8-Queens），使用傳統 DFS 演算法，另外也分析他人 Github 使用的 遺傳演算法，主要步驟：

1. 初始化 Solution class：
    a. 建立一個 `queen_move_directions: List[Tuple[int, int]]` 來維護放下皇后後被封鎖的格子跟皇后位置的相對方向（共有 8 個方向：包含右、右下、下、左下、左、左上、上、右上）
    b. 建立一個 `result: List[List[str]]` 來儲存所有 N-Queens 的結果
    c. 建立一個 `chessBoardSize: int` 用來確定長寬大小，這也可以視為是決定 N-Queens 問題的 N
    d. 建立一個 `output_dir: str` 用來給之後 output 成檔案的儲存位置，由於直接用 Python 去打印結果可能會不太方便。
2. 呼叫 `solveNQueens(n: int) -> List[List[str]]: …`，傳入 n 即可得到結果，透過 `solution.printResult()` 即可立即查看結果。
    a. 呼叫 `solveNQueens(n)` 後再把 `chessBoardSize` 設成 `n`，並初始化 `board: List[str]` 成一個 n * n 內部都是 '.' 的棋盤、`block_count: List[int]` 成一個 n * n 內部都是 0 的棋盤封鎖計數（其中 block_count[i][j] 表示當前狀態下有多少皇后限制了 (i, j) 這個格子，直到這個格子的限制數為 0，該格子才可以放下皇后）。
    b. 呼叫 `backtrack(current_row: int, current_board: List[str], block_count: List[List[int]]) -> None: …` 來透過 DFS 暴力求解所有 N-Queens 問題的解。
        i. 針對 `current_row`，由左往右前進檢查 `block_count[current_row][col]` 哪個格子可以放下皇后，若為 0 則代表可以放下皇后。
        ii. 放下皇后：
            1. `current_board[current_row][col] = 'Q';`
            2. `flip_positions(current_row, col, false, block_count);`

iii. 後在遞迴呼叫 `backtrack(current_row + 1, current_board, block_count)`。
iv. 遞迴回來後要在回溯：
1. `current_board[current_row][col] = '.';`
2. `flip_positions(current_row, col, true, block_count);`

3. 使用 `solution.print_result() -> None: …` 或是 `solution.write_result_to_file(file_name: str) -> None: …` 來輸出結果。

- Note :
  - 我們以 '.' 表示空格，'Q' 表示該格子有皇后。
  - block_count[current_row][col] 的加減運算，因為是向八個方向的所有棋格擴展，所以外包一個 `flip_positions(i, j, is_available, block_count)` 去做考慮或不考慮的翻轉。

```python
import os
from typing import *    # for python3

class Solution:
    def __init__(self):
        self.queen_move_directions: List[Tuple[int, int]] = [
            (1, 0),    # right
            (1, 1),    # right down
            (0, 1),    # down
            (-1, 1),   # left down
            (-1, 0),   # left
            (-1, -1),  # left top
            (0, -1),   # top
            (1, -1),   # right top
        ]
        self.result: List[List[str]] = []
        self.chess_board_size: int = 0
        self.output_dir = './outputs/'

    def is_valid_position(self, i: int, j: int) -> bool:
        return 0 <= i < self.chess_board_size and 0 <= j < self.chess_board_size

    def flip_positions(self, i: int, j: int, is_available: bool, block_count: List[List[int]]) -> None:
        for d_i, d_j in self.queen_move_directions:
            current_row, current_col = i, j
            while self.is_valid_position(current_row, current_col):
                block_count[current_row][current_col] = max(0, block_count[current_row][current_col] + (-1 if is_available else 1))
                current_row += d_i
                current_col += d_j

    def backtrack(self, current_row: int, current_board: List[str], block_count: List[List[int]]) -> None:
        if current_row == self.chess_board_size:
            self.result.append(["".join(row) for row in current_board])
            return

        for col in range(self.chess_board_size):
            if block_count[current_row][col] == 0:
                self.flip_positions(current_row, col, False, block_count)
                current_board[current_row][col] = 'Q'
                self.backtrack(current_row + 1, current_board, block_count)
                current_board[current_row][col] = '.'
                self.flip_positions(current_row, col, True, block_count)

    def solveNQueens(self, n: int) -> List[List[str]]:
        self.chess_board_size = n
        board = [['.'] * n for _ in range(n)]
        block_count = [[0] * n for _ in range(n)]
        self.backtrack(0, board, block_count)
        return self.result  # 改成 return len(self.result) 即可完成 N-Queens II 的問題
```

實驗結果:

- 成功透過上述步驟生成 8 皇后問題的所有(共 92 組解), 並將結果的輸出字串轉換成 FEN 格式, 接著再透過 requests 從 lichess 上直接轉換成圖片。
- **Note**:因為 8 皇后會有 92 組解, 所以透過壓縮檔內的 `screenshots/` 來展示所有來自 lichess 截下來的結果圖片(由於圖片實在太多太大... 請見諒)。

結果討論與實驗心得:

此次實驗花費我相當多的時間, 本來只是單純解 N 或 8 皇后問題是還好, 但還要透過 lichess 呈現結果, 這使得這份作業變得更具有挑戰性, 同時我也從 8 皇后問題上更加熟悉如何透過 backtrack 求解這類 NP 問題, 其中透過嘗試所有解, 每當考慮放下一個皇后時加入路徑限制, 然後回溯時, 再把對於該皇后的限制解除, 如此遞迴求出所有解。最後, 再把他人 github 上面透過遺傳演算法寫的另類版本也抓下來做了註解, 該演算法解決 8 皇后問題, 透過隨機生成、交配、變異與適應度評估反覆優化棋盤配置, 直到找到合法解。解以列表表示, 每輪挑選適應值較高的解進行交配與調整, 最終可有效產生無衝突的皇后排布, 展現演化式搜尋的解題能力。