離散數學 作業一 程式碼說明

隨班附讀 b06106035 吳欣樺 [Github Repo]

（一）繪圖所需套件

利用 numpy & matplotlib 來繪製圖表，如果只要查看程式碼運行結果，可註解掉 import 這兩個套件的程式碼，以及呼叫 `draw_plot` 此函式的部分。

（二）執行程式碼

輸入 `python3 main.py` 後，在 terminal 中可以輸入 `n` 以及 `obstacle`(障礙物密度)的值，以查看不同網格數量的可行路徑變化。

（三）程式碼架構

1. 基本變數創建
    - 障礙物生成：透過 `gen_obstacle` 此函式隨機選取第 1 至 (n ^ 2 - 2) 個網格放置障礙物，回傳包含障礙物位置編號的陣列。
    - 網格生成：透過 `create_board` 函式回傳一個 n * n 的陣列。傳入障礙物位置，將其所在位置的值設為 -1。如果想查看生成的網格以及障礙物的位置，可使用 `print_board` 印出網格。
2. 情境一實作：向下或向右走所需的成本相同時的可行/最佳路徑數
    - 動態規劃： `numb_of_path_dp` 會接收上面生成的 n * n 陣列 (i.e. `board`)，其中 `dp` 為用來記憶子問題解的 n*n 陣列。 `dp[i][j]` 紀錄走到 `board[i][j]` 的可行路徑數量，其值等於兩個方向的路徑數總和。當 `board[i][j] = -1` 時，可行路徑數量為 0。
    - 遞迴法： `num_of_path_recursion` 亦接收 `board` 陣列，依序嘗試每種走法。遞迴結束條件有以下幾種：
        - 走到終點，代表可行路徑存在，回傳 1
        - 走到障礙物，回傳 0
        - 走到無效位置，回傳 0

3. 情境二實作：轉彎時成本 **+ 1** 的最佳路徑數
    - 動態規劃：`num_of_shortest_path_dp` 接收 `board` 陣列，其中 `dp[i][j][k]=(min_cost, number_of_path)`。
        - 變數說明：
            - `dp` 為一個 3 維陣列，每個陣列中的元素為一個 Tuple，第一個值 `min_cost` 代表走到 `board[i][j]` 所需的最小成本，第二個值 `number_of_path` 代表花費最小成本到達 `board[i][j]` 的路徑總數（i.e. 最佳路徑數）。若 `board[i][j]` 為障礙物，則到達成本為無限大，可行路徑數為 0。
            - `k` 代表路徑方向（0=從左邊來 / 1=從上面來）
        - 運作機制：
            - 想到達每個位置 `board[i][j]` 有兩種可能路徑：從左邊（`board[i][j - 1]`）或是上面（`board[i - 1][j]`）。為了記錄是否轉彎，因此分開計算兩種方向的成本以及路徑數（i.e. `dp[i][j][0]` 和 `dp[i][j][1]`）。當考慮某一方向的成本時，函式會去參照到達前一個位置所需的成本，比較轉彎/不轉彎的成本，選出較低的路徑。比如想計算從左邊到達 `board[i][j]` 的最小成本與最佳路徑數，也就是計算 `dp[i][j][0]` 時，會經歷以下幾個步驟：
                1. 考慮到達 `board[i][j - 1]` 的兩條路徑各自所需成本：`dp[i][j - 1][0]` 以及 `dp[i][j - 1][1]` tuple 中的第一個值
                2. 計算最佳路徑成本：最小成本為 `min(dp[i][j - 1][0][0], dp[i][j - 1][1][0] + 1)` ，其中因為 `dp[i][j - 1][1]` 代表從上方到達 `board[i][j - 1]` ，因此轉了一次彎，所需成本需 + 1。
                3. 計算最佳路徑總數：取最佳路徑 tuple 中的第二個值。如果兩個路徑成本相同，則路徑數等於兩個方向的最佳路徑總數相加。
    - 遞迴法：`num_of_shortest_path_recursion`
        - 變數說明：`min_cost` 代表最佳路徑所需的成本，`path_count` 代表最佳路徑總數
        - 運作機制：與上述動態規劃的做法相似，都是透過 0 和 1 代表路徑方向，每次遞迴不同方向的路徑時，都會傳入下一步的座標、以及更新後

的路徑成本。到達終點時，如果當前路徑所需的成本小於 `min_cost` ，則更新 `min_cost` 為當前路徑的成本，並且將 `path_count` 重新設定為 1。如果到達終點時，當前路徑所需的成本等於 `min_cost` ，則將 `path_count` + 1。

4. 求期望值

    - 由於障礙物皆隨機產生，因此透過大量的隨機試驗重複計算不同障礙物分佈下的可行/最佳路徑總數，再求平均值後可以得到較為收斂的值。`get_expected_value` 函式接收不同的 `n` 值、障礙物密度、以及計算路徑數的函式，並進行 `t` 次重複計算，回傳平均值。