Discrete Mathematics Assignment 1 Report

Hung Lun-Tsan B11505050

October 18, 2024

1. Introduction

在一個 n×n 的網格中，船艦可以從起始位置 (0, 0) 移動到目標位置 (n-1, n-1)。船艦可以向右，或向下移動，這樣路徑都是直接向目標前進，不會有過頭回頭的狀況。
爲了讓結果可再現，我們把 random seed 設爲 11505050。
程式碼在 github 上 (link)。

名詞定義

可行路徑-可以從起始位置走到目標位置的 path 即爲可行路徑
最佳路徑-我們定義 path with least turn 爲最佳路徑
障礙物密度-我們先算出網格在該障礙物密度下會有多少格障礙物 (取 floor)，然後隨機選取格子。

2. Visualization

我們用 html 檔案來視覺化網格，在此檔案中，我們運用 javaScript 來進行演算法的部分。
使用者可以自由選擇 n 的值以及障礙物密度，接下來他就會隨機生成出 n by n 的網格以及障礙物 (紅色)。
接著用 **DP 的方法開始尋找所有可行路徑**，並用 **BFS 尋找最佳路徑**如圖1。
如果有找到，就將其中一個最佳路徑畫成黃色，如果沒有的話，就會說 No path found 如圖2。

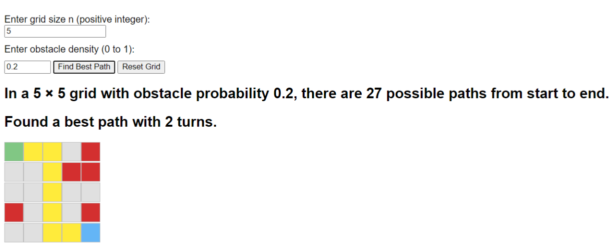

Figure 1: Best path is shown in the image

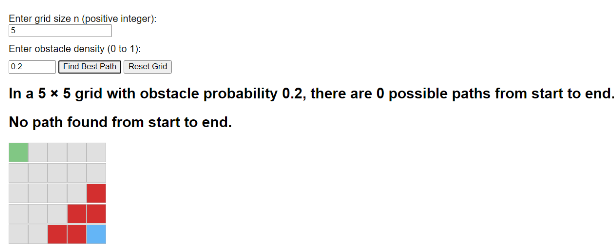

Figure 2: No possible path found

3. Possible Paths

在這部分，我們探討了在不同障礙物密度下，隨著網格大小增加，所有**可行路徑的變化趨勢**。圖3 顯示了不同障礙物密度對可行路徑數量的影響。
我們將每個 case 模擬 10000 次，然後取平均來當可行路徑，此圖 **y 軸是用 log scale** 來顯示，讓我們比較好看資料的趨勢。
可以看到障礙物密度爲 0 或是很小的時候，**可行路徑跟網格數量呈現指數增加的關係**，但是當密度太高的時候，就會趨於平緩，在密度 0.5 時，可行路徑跟網格大小幾乎沒關係，當密度到 0.6 時，就幾乎找不到可行路徑了。

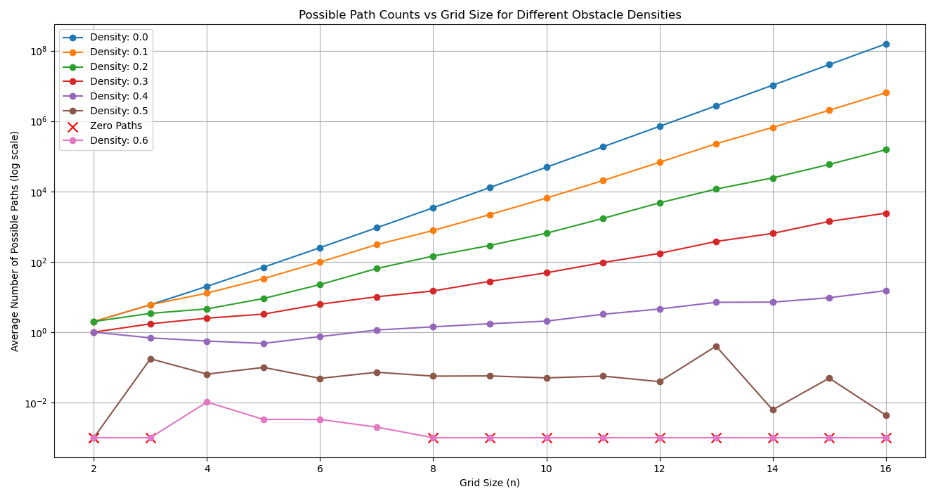

Figure 3: Possible Path Counts vs Grid Size for Different Obstacle Densities

4. Best Path Trend

在這部分，我們分析了**最佳路徑數量**隨著網格大小增加的變化趨勢。圖4 顯示了在不同網格大小下，最佳路徑的數量變化，此處先**假設沒有障礙物**。
可以看到，最佳路徑的數量跟 n 的關係應該是要看**組合公式**的結果，如果沒有障礙物的情況下，最佳路徑一定是走邊邊的兩條，但是次佳路徑就會跟 n 平方相關，也是因爲組合公式有 n 平方的關係。

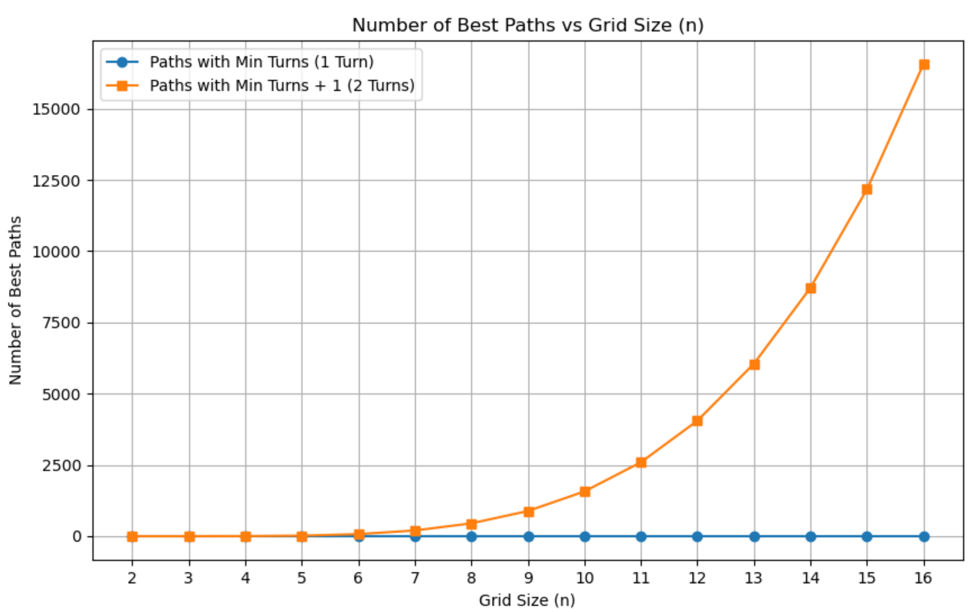

Figure 4: Number of Best Paths vs Grid Size (n)

5. Best Path Count With Different Obstacle Densities

在這部分，我們模擬 10000 次來找到最佳路徑的個數，來得到有障礙物的狀態下**最佳路徑數量**隨著網格大小增加的變化趨勢。
圖5中可以看到，密度爲 0 時，就跟上個部份一樣，都是兩個最佳路徑，當密度變高的時候，同時存在的最佳路徑就變多了，當密度爲 **0.2 時同時存在的最佳路徑最多**。
當超過之後會因爲很多時候**完全無可行路徑**，讓平均的最佳路徑降下來，當密度到 0.4 時，隨著網格加大，平均的最佳路徑會低於 1，代表大多數時候都是完全找不到最佳路徑。

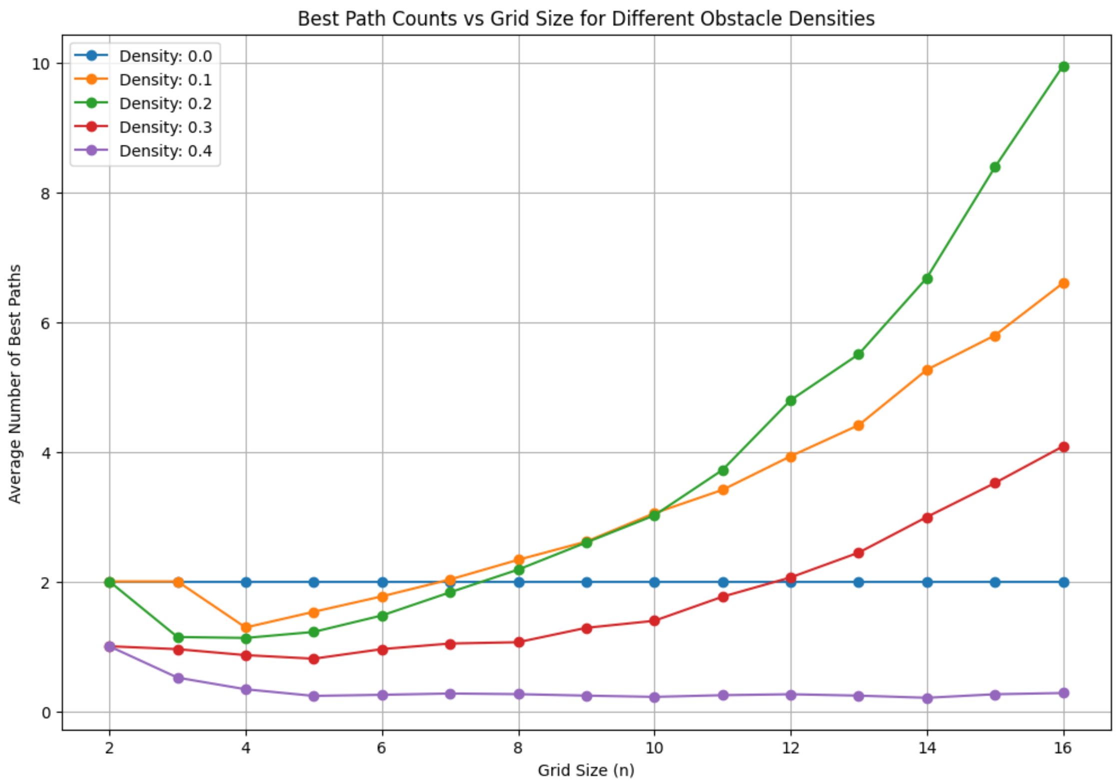

Figure 5: Number of Best Paths vs Grid Size (n) with different obstacles density

6. Best Path Average Turn

這部分顯示了**最佳路徑的平均轉彎次數**，隨著網格大小和障礙物密度的增加而變化的趨勢。圖6 展示了不同障礙物密度下，最佳路徑的平均轉彎次數。
不同於上個部分，在這裡我們要討論的是隨著障礙物密度的增加，最佳路徑要多轉幾次，我們運用 BFS 來找最佳路徑，從圖中可以看到**當障礙物密度越高，同樣的網格大小需要轉越多次**才能走到終點。但是當**密度太大時，可能在 10000 次模擬中都找不到可行路徑**。

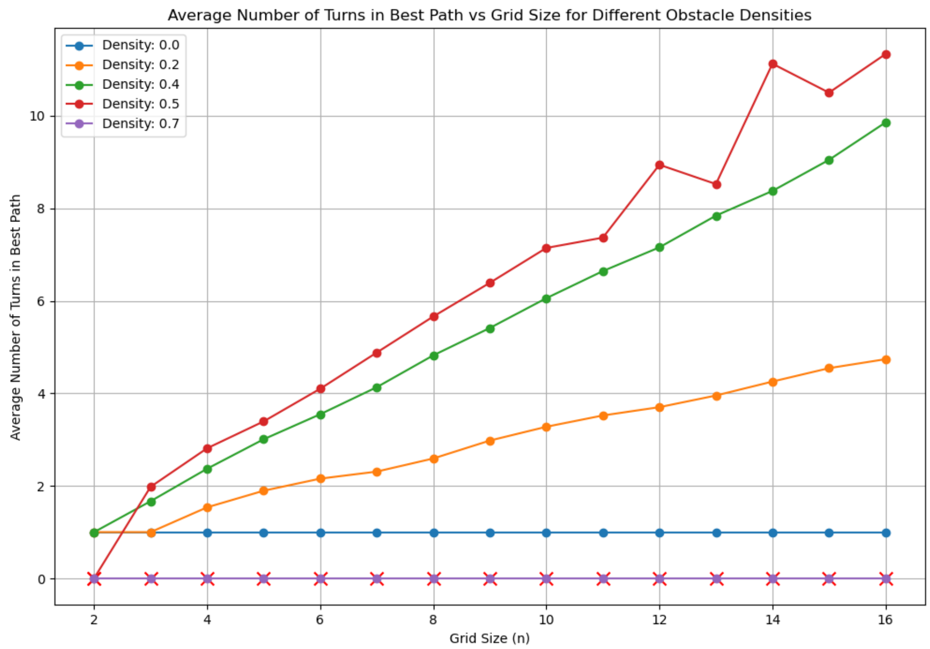

Figure 6: Average Number of Turns in Best Path vs Grid Size for Different Obstacle Densities

7. Algorithm Comparison

在這部分，我們比較了多種演算法在不同障礙物密度下的執行效率，包括遞歸法、動態規劃、專門解決路徑問題的 Dijkstra 演算法，以及當障礙物密度爲 0 時，可以用組合公式直接算出來。圖7 顯示了各種演算法的執行時間隨著網格大小和障礙物密度的變化。

遞歸法 (Recursive Method)

優點:

- 實現簡單，易於理解和編寫。
- 自然地反映了問題的遞歸結構，適合用於教學和概念驗證。

缺點:

- 計算效率低下，尤其是在網格大小較大時，會出現指數級的時間複雜度。
- 需要大量的遞歸調用，容易導致堆疊溢出（RecursionError）。
- 缺乏記憶化（Memoization）時，會有大量的重複計算。

時間複雜度: $O(2^n)$，其中 n 爲網格的大小。由於每一步都有兩種選擇（向右或向下），遞歸樹的深度爲 $2n$，導致指數級的時間複雜度。
空間複雜度: $O(n)$，主要由遞歸調用的堆疊深度所占用的空間。

動態規劃 (Dynamic Programming Method)

優點:

- 顯著提高計算效率，避免了重複計算。
- 能夠處理較大的網格規模，時間複雜度較低。
- 通常具有線性或多項式級別的時間複雜度，適合實際應用。

缺點:

- 需要額外的空間來存儲中間結果，可能導致較高的空間複雜度。
- 相對於遞歸法，實現較爲複雜。

時間複雜度: $O(n^2)$，其中 n 爲網格的邊長。需要遍歷每個網格點並計算其狀態。
空間複雜度: $O(n^2)$，用於存儲動態規劃表（DP table），其中每個網格點需要存儲兩個方向的最小轉彎次數和路徑數量。

Dijkstra 演算法 (Dijkstra's Algorithm)

優點:

- 能夠有效地找到從起點到終點的最短路徑。
- 通過優先隊列（Heap）的使用，能夠在較大規模的網格中保持良好的性能。
- 易於擴展，能夠處理多種路徑優化問題（如最小轉彎數）。

缺點:

- 實現相對複雜，尤其是在需要同時考慮多個狀態（如方向和轉彎次數）時。
- 如果不加以優化，可能會導致較高的時間和空間消耗。

時間複雜度: $O(E + V \log V)$，其中 V 爲節點數量，E 爲邊的數量。在網格中，$V = n^2$，$E = 2n(n-1)$，因此時間複雜度近似爲 $O(n^2 \log n)$。
空間複雜度: $O(V)$，主要用於存儲圖的結構、最小轉彎次數以及路徑計數。

組合公式法 (Combinatorial Method)

優點:

- 計算速度極快，尤其是在無障礙物的情況下，能夠直接通過數學公式獲得結果。
- 無需遍歷網格，節省大量計算資源。

缺點:

- 僅適用於無障礙物的情況，無法處理有障礙物的場景。
- 依賴於特定的路徑結構（如最小轉彎數），不具備靈活性。

時間複雜度: $O(1)$，直接通過數學公式計算，與網格大小無關。
空間複雜度: $O(1)$，僅需常數空間來存儲計算結果。

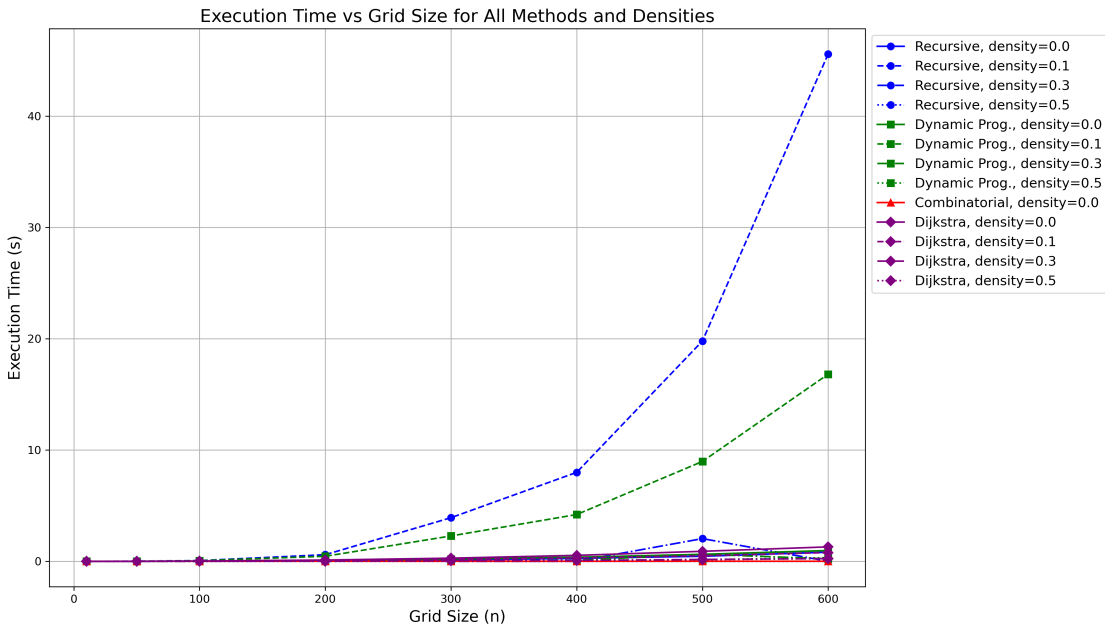

Figure 7: Execution Time For All Methods and Densities

綜合分析

從圖7 可以觀察到，隨著網格大小的增大，各種演算法的執行時間呈現不同的增長趨勢：

- **遞歸法**的執行時間隨著網格大小的增長呈指數級上升，這主要是由於其高時間複雜度和大量的重複計算所致。因此，遞歸法僅適用於小規模的問題。
- **動態規劃**相較於遞歸法，能夠更有效地處理較大的網格，因爲它通過存儲中間結果來避免重複計算。然而，隨著網格大小的增大，其空間複雜度也會顯著增加。
- **Dijkstra 演算法**在處理較大網格時展現出良好的性能，雖然其時間複雜度比動態規劃略高，但仍能夠在合理的時間內完成計算。Dijkstra 演算法的靈活性使其在多種路徑優化問題中具有廣泛應用。
- **組合公式法**由於其計算速度極快，能夠在無障礙物的情況下立即給出結果，是最優的選擇。然而，其應用範圍非常有限，僅適用於特定的問題情境。

總結來説，選擇適當的演算法取決於具體問題的規模和特性。在小規模且無障礙物的情況下，組合公式法和遞歸法可能是簡單且有效的選擇；而在需要處理較大規模或存在障礙物的情況下，動態規劃和 Dijkstra 演算法則更爲適合。

時間複雜度與空間複雜度總結

Table 1: 各演算法的時間複雜度與空間複雜度

演算法	**時間複雜度**	**空間複雜度**
遞歸法	$O(2^n)$	$O(n)$
動態規劃	$O(n^2)$	$O(n^2)$
Dijkstra 演算法	$O(n^2 \log n)$	$O(n^2)$
組合公式法	$O(1)$	$O(1)$

表1 彙總了各演算法的時間複雜度與空間複雜度，便於對比和分析。根據表中的數據，可以看出組合公式法在適用範圍内具有明顯的優勢，而遞歸法在計算效率和空間使用上存在較大劣勢。動態規劃和 Dijkstra 演算法則在性能和應用靈活性之間取得了較好的平衡。

結論

通過對遞歸法、動態規劃、Dijkstra 演算法以及組合公式法的比較分析，我們可以根據具體的應用需求選擇最合適的演算法。在需要處理複雜和大規模的路徑尋找問題時，動態規劃和 Dijkstra 演算法是更爲理想的選擇；而在簡單且無障礙物的情況下，組合公式法則提供了最快速的解決方案。遞歸法雖然在理論上具有一定的教育意義，但在實際應用中因其效率低下而較少使用。

8. Application and Future

這個部份我們會討論演算法在實際應用的潛力以及未來研究方向。

8.1 應用場景

無人車導航系統：在無人駕駛車輛中，尋找從起點到終點的最佳路徑至關重要。特別是在城市環境中，車輛需要避開各種障礙物，如行人、其他車輛和道路障礙物。通過應用動態規劃和 Dijkstra 演算法，無人車能夠實時計算出最優路徑，確保行駛的安全性和效率。

倉庫自動化運輸：在現代化的倉庫中，自動化機器人需要高效地在倉庫內移動，運送物品。這些機器人需要在滿佈貨物的環境中找到最短且轉彎最少的路徑，以提高運輸效率並減少能源消耗。動態規劃和 Dijkstra 演算法在此類應用中表現出色，能夠應對複雜的環境和多變的障礙物配置。

機器人路徑規劃：在各種機器人應用中，如服務型機器人和工業機器人，路徑規劃是核心任務之一。這些機器人需要在動態環境中靈活移動，避開障礙物並達到目標位置。遞歸法雖然不適用於大規模問題，但動態規劃和 Dijkstra 演算法則能夠高效地處理實時路徑規劃需求。

8.2 未來研究方向

未來的研究可以在以下幾個方面進行擴展和深化，以提升現有演算法的性能和應用範圍：

8.2.1 動態障礙物處理

目前的研究主要針對靜態障礙物，但在實際應用中，障礙物往往是動態的，如行人移動、其他車輛的變動等。因此，未來研究可以著重於開發能夠處理動態障礙物的路徑規劃演算法，這需要考慮時間因素和預測障礙物的移動路徑。

8.2.2 多目標優化

除了最少轉彎之外，還可以引入其他優化目標，如最短距離、最少耗時、最低能耗等。多目標優化能夠滿足不同應用場景下的多樣化需求，提升路徑規劃的靈活性和適應性。

8.2.3 演算法性能提升

針對大規模網格問題，現有的演算法在時間和空間效率上仍有提升空間。未來研究可以探索並行計算、分佈式計算或基於強化學習的方法，以加速路徑規劃過程，滿足實時應用的需求。

8.2.4 結合機器學習

隨著機器學習技術的發展，結合機器學習與傳統路徑規劃演算法，可以進一步提升路徑規劃的智能化和自適應能力。例如，利用深度學習模型預測障礙物的動態行爲，從而提前調整路徑規劃策略。

8.3 技術挑戰

在實際應用中，仍存在一些技術挑戰需要克服：

實時性要求：在動態環境中，路徑規劃需要具備高效的計算能力，以滿足實時反應的需求。

多機器人協調：在多機器人系統中，不僅需要單個機器人的路徑規劃，還需要考慮多機器人之間的協調與避讓，這增加了問題的複雜性。

不確定性處理：在現實環境中，存在許多不確定因素，如障礙物的隨機出現和消失、環境變化等，需要演算法具備良好的魯棒性和適應性。

9. Summary

本報告深入探討了在 n×n 網格中從起點 (0, 0) 到終點 (n-1, n-1) 的所有可行路徑及最佳路徑的尋找問題。我們採用了遞歸法、動態規劃、Dijkstra 演算法以及組合公式法來分析路徑數量和轉彎次數，並比較了這些演算法在不同網格大小和障礙物密度下的執行效率。

通過視覺化和大量模擬實驗，我們發現障礙物密度對可行路徑和最佳路徑的數量有顯著影響。具體而言，在無障礙物或障礙物密度較低的情況下，可行路徑數量呈指數級增長，轉彎次數相對較少。然而，隨著障礙物密度的增加，可行路徑和最佳路徑的數量顯著減少，特別是在密度達到 0.6 以上時，幾乎無法找到可行路徑。

在演算法比較部分，我們發現動態規劃和 Dijkstra 演算法在處理較大規模或存在障礙物的問題時表現出色，能夠在合理的時間內完成計算。而遞歸法由於其高時間複雜度和大量的重複計算，僅適用於小規模問題。組合公式法則在無障礙物的情況下提供了最快速的計算方式，但其應用範圍有限。

總結來說，本次作業不僅加深了我們對路徑規劃問題的理解，還提供了實現和優化各種演算法的實踐經驗。這些技術在現代智能導航、自動化運輸和機器人技術中具有重要的應用價值。未來，我們將繼續探索更高效、更智能的路徑規劃方法，以滿足日益複雜的應用需求。