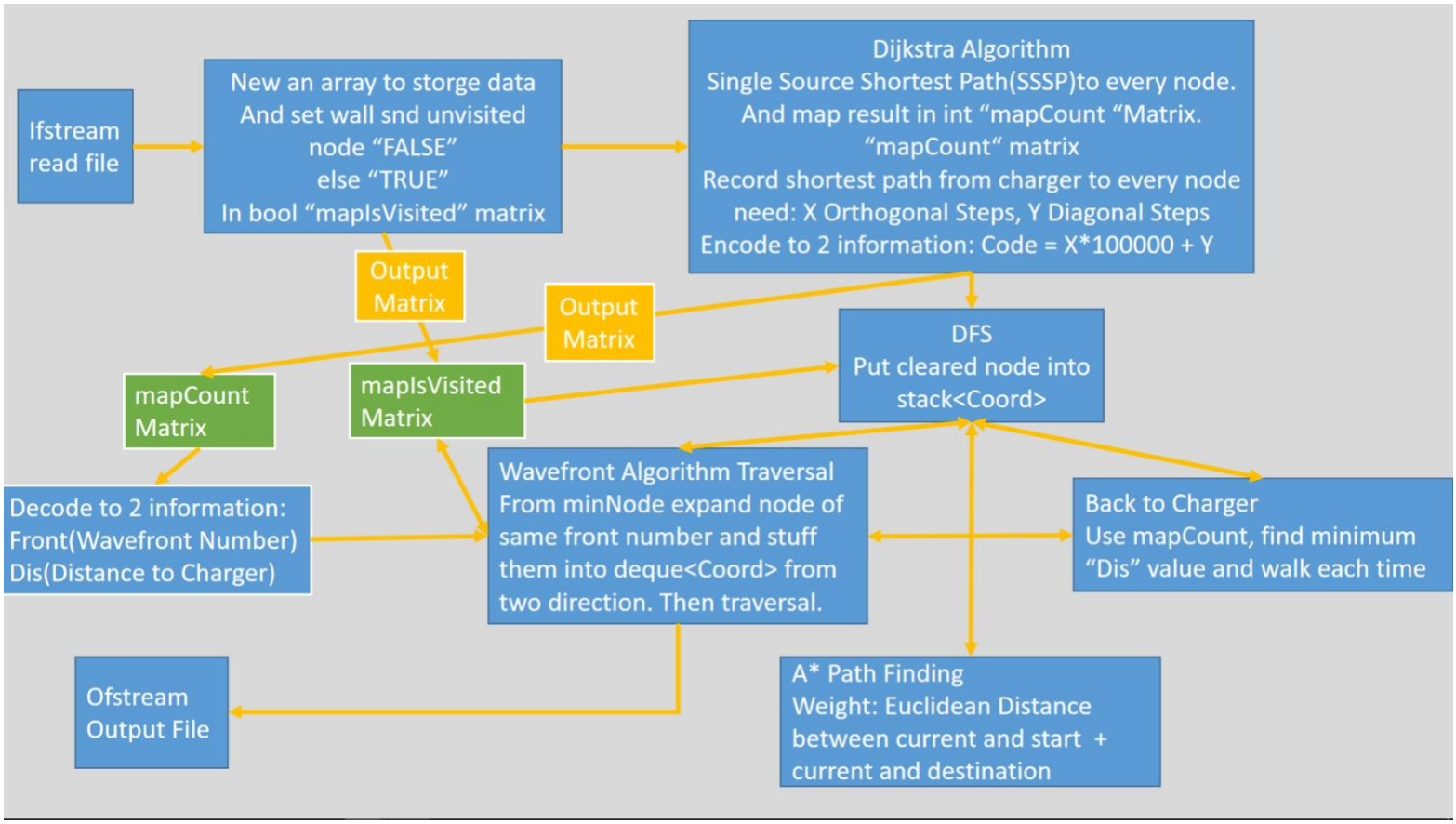
1. Project Description

(1) Program Flow Chart



(2) Detailed Description

參考了三篇論文，一開是先看到 A Survey on Coverage Path Planning for Robotics 這篇概述各種 Coverage 演算法，隨後注意到 Grid-based Coverage using the Wavefront Algorithm 正好與一開始就規劃好的用 Dijksra 做最短回歸路徑作法相似，資料似乎可以共用，而且論文最後評論該作法相對容易實現，就決定使用該方法了。

如參考資料所列，一開始用 Dijkstra 演算法將整個地圖填入權重值，再使用 DFS 尋訪 Wavefront 進行清掃，途中若遇電池耗盡則使用 Dijkstra 演算法(同時也用來計算 Wavefront Number)的最短距離回歸充電點，再用 stack 回到出發點，若遇到需要跳躍移動到非當前位置鄰接的點，則使用 A*演算法移動到該點。

a) Dijkstra 演算法：

原理上與課堂介紹類似，使用 queue 將探詢點一一塞入，後使用 queue.pop 將 front 元素逐一取出擴展其鄰接 8 點(上下左右、左上、左下、右上、右下)並分別填入權值如下

設A_{mn}的鄰接點為n_{ij}，m、i 為在地圖上位置的 row，n、j 為位置的 column Ortho、diago 分別是變數，分別代表鄰接上下左右四點和對角四點，queue 每為空一次，則各加 1 一次

$$n_{ij} = (diago + 1) \times 100000 + ortho, if\ m - i = \pm 1\ \cap n - j = \pm 1$$

$$n_{ij} = diago \times 100000 + (ortho + 1), if\ \ m - i = \pm 1\ and\ n - j = 0$$

$$or\ \ m - i = 0\ and\ n - j = \pm 1$$

b) Wavefront Hybrid DFS 演算法：

取 Dijkstra 演算的結果 X，X%100000 取 ortho 值，X/100000 取 diago 值，與充電點的最短路徑距離(Distance)為 2*diago + ortho，Wavefront Number(即波前線的編號)diago*4+ortho*3(算法參見 Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot A. Zelinsky1, R.A. Jarvis2, J.C. Byrne2 and S. Yuta3 中的 Modified Wavefront 算法)，每次走訪下一點先與該點的 Distance 比較，若大於電量回歸充電，每次取 mapCount 中鄰接點 Distance 值最小者，則能保證以最短路徑回歸；若尚有充足電力，則尋找鄰接八點 Wavefront Number 最小者 minNode，並用 deque 從 minNode 往兩方擴展鄰接四點中擁有相同 Wavefront Number 者，再從 deque.front、deque.back 挑選一個較接近 minNode 者，依序將 deque 尋訪完畢，並進行下一輪擴展，若周圍無可擴展點，則用 DFS stack.pop 找能尋訪的下一點，並用 A*行走至該處。

c) A*演算法：與網路上的做法類似，基本上就是 Dijkstra 的剪枝優化作法，先宣告一個 class Node，宣告類別參數 public: Coord coord, int value, Node* from，分別為該類別的座標位置(Coord int row, int col)、該點權值、擴展該點的上一點的 Node*，每次均將已尋訪的鄰接未尋訪點塞入 minimum priority_queue，每次取其 top 元素，即為整著 queue 中元素的最小值，也就是所有鄰接未尋訪點的最小權重點。而權重方式則用該點與出發點和該點與目標終點的直線距離相加((Destinatio.row-current.row)* (Destinatio.row-current.row)+(Destinatio.col-current.col)* (Destinatio.col-current.col) + (Start.row-current.row)* (Start.row-current.row)+(Start.col-current.col)* (Start.col-current.col))，如此可以平衡 greedy algorithm 與執行時間。待找到目標點後即可用類別中的 Node* from 找到前一點，輸出行走路徑。

d) Reference:

a) <<A Survey on Coverage Path Planning for Robotics>> BY Enric Galceran and Marc Carreras

b) <<Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot>> BY A. Zelinsky1, R.A. Jarvis2, J.C. Byrne2 and S. Yuta3

c) <<Wave front Method Based Path Planning Algorithm for Mobile Robots>> BY Bhavya Ghai1 and Anupam Shukla2

2. Test case Design

(1) Detailed Description of the Test case:

考量兩種狀況，地圖障礙物隨機凌亂和地圖障礙物排列整齊的狀況。先將地圖一分為二，左為零散，右為整齊，一個優秀的遍歷演算法應該要能適應兩種狀態，並同時給出最佳解，零散地圖是由隨機亂數產生，障礙物機率設為 20%，而在另一邊則是以地圖的大小產生固定的障礙物，將 s 設為正方形地圖的長，在高=2、<=s/6 和寬=s/2+1、<=s/2+s/7 還有高=s/2 - s/3、 <=s-3 和寬=s/2+1+s/6、<=s/2+1+s/3，以及高=s/2+s/3 - s/3、<=s-3 和寬=s/2+1、<=s/2+1+s/10，共產生三個矩形障礙物。