資料結構 HW5

系級：電機系大三

姓名：林凡皓

學號：B103012002

一、 Explain build_graph( ) and search_from_graph_dfs( )

- build_graph( ) :

  build_graph 函數主要由多個 for loop 組成，最外層的兩個 for loop 會針對迷宮矩陣中每一個位置做迭代。每一次迭代中，會先去檢查該位置是否為通路，如果是通路，就會記錄當前位置的 ID，並判斷該位置是否已經儲存過，沒有儲存過就將該位置儲存。接著去判斷該位置的鄰居是否為通路，這邊也是利用 for loop 去迭代上下左右四個方向。每一次迭代中都先判斷該鄰居是否為通路，如果是的話就記錄位置並判斷該位置使否有被儲存過，如果有就將當前位置與鄰居位置之間加上一條邊，這裡需要呼叫兩次 add_edge 因為要加入雙向的邊。

- search_from_graph_dfs( ) :

  search_from_graph_dfs( )主要透過 Stack 來實現。

  首先會先抓取 start vertex 並判斷該 vertex 是否存在。將 start vertex 的顏色設設定成灰色，代表說該 vertex 正在被探索，並初始化探索時間。透過 while loop 來探索迷宮，當 stack 內還有東西時，while loop 就會持續執行。While loop 一開始先去查看 stack 頂部的資料，將 all_neighbors_visited 參數設定成 True，並判斷是否要更改該 vertex 的顏色，接著針對上下左右進行搜索。搜索過程會先去判斷鄰居是否被搜索過，如果沒有的話，將該鄰居加入 stack 中，並將鄰居的 previous 設定成目前的 vertex，all_neighbors_visited 參數也設定成 False，代表說還有鄰居未被探索。當所有鄰居皆被探索後，將目前 vertex 的顏色改成黑色，並將其從 stack 中移除。接著判斷該 vertex 是否為出口，如果是出口的話，將(row, column)加入到 path 中並將 path 回傳。

二、 Design of my program

- search_from_graph_bfs( ):

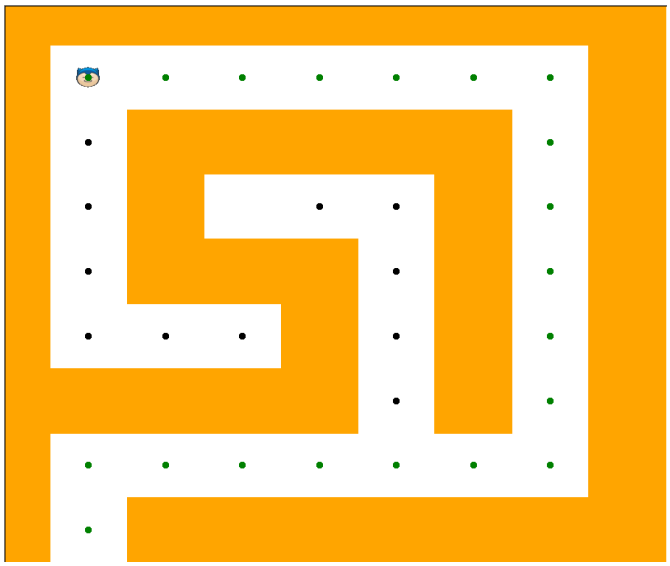
  1. 實現方法 :

     整體實現方式與 dfs 類似，主要差別在於 bfs 採用 queue 做實現。首先會先抓取 start vertex 並判斷該 vertex 是否存在。將 start vertex 的顏色設設定成灰色，代表說該 vertex 正在被探索，並初始化探索時間。透過 while loop 來探索迷宮，當 queue 內還有東西時，while loop 就會持續執行。While loop 一開始將 queue 中最前面的 vertex 移出來，接著針對上下左右進行搜索，判斷每一個

鄰居的顏色是否為白色，如果是白色代表說該位置還沒被搜索，則將該位置加入 queue 中，並將該鄰居的 previous 設定成當前在探索的 vertex。接著去判斷當前 vertex 是否為出口，如果是的話就將當前位置加入 path 中，並沿著 pervious 一路將搜索出來的路徑加入 path 中，最後再回傳 path。

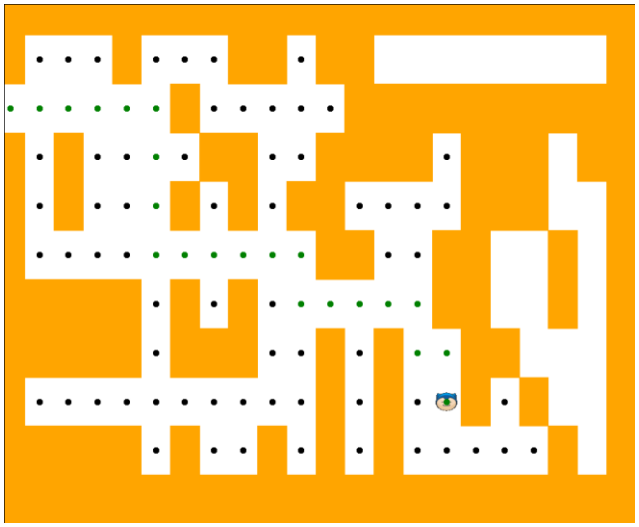
2. 執行結果：

- Maze 1：



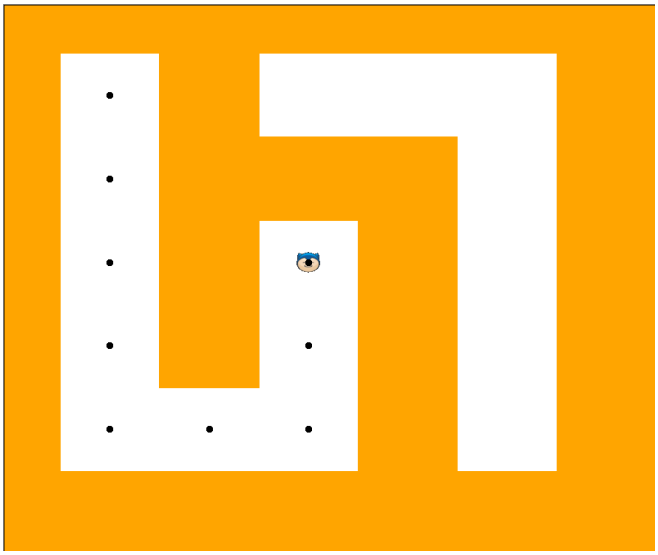
```
Selected algorithm (bfs) path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6),
(1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4),
(7, 3), (7, 2), (7, 1), (8, 1)]
Path to exit found by bfs. Total path length: 19
Correct path from file: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7),
 (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3),
 (7, 2), (7, 1), (8, 1)]
The search path matches the correct path!
```

- Maze 2：



```
Selected algorithm (bfs) path: [(8, 15), (7, 15), (7, 14), (6, 14), (6, 13), (6,
 12), (6, 11), (6, 10), (5, 10), (5, 9), (5, 8), (5, 7), (5, 6), (5, 5), (4, 5),
 (3, 5), (2, 5), (2, 4), (2, 3), (2, 2), (2, 1), (2, 0)]
Path to exit found by bfs. Total path length: 21
```

➢ Maze 3 :



```
Selected algorithm (bfs) path: []
No path found by bfs.
```

- search_from_graph_dijkstra( ):

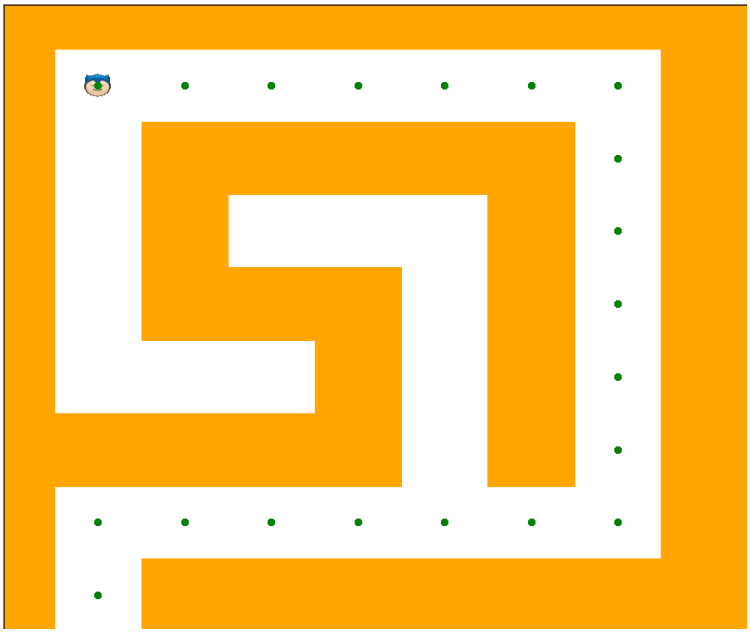
1. 實現方法 :

   Dijkstra 主要透過 priority queue 來實現。

   一開始先去抓取初始位置的 ID，並將該位置的距離設定成 0 然後加入 priority queue 中。接著初始化一個 set 為 visited，紀錄拜訪過的 vertex。透過 while loop 來探索迷宮，當 priority queue 內還有東西時，while loop 就會持續執行。每一次迭代都先將 priority queue 中最優先的 vertex 拿出來並將其加入 visited 中，接著先去判斷該 vertex 是否為出口，如果是則利用 while loop 將該位置的 pervious 加入 path 中，最後回傳 path。如果不是出口，利用 get_neighbors 來取得目前位置的鄰居，並將鄰居的 distance 令成當前 distance 加一(每一個路徑的權重皆為 1)，接著去判斷說新的距離是否小於原先的距離，如果小於的話就更新距離並將鄰居的 previous 設定成當前位置。

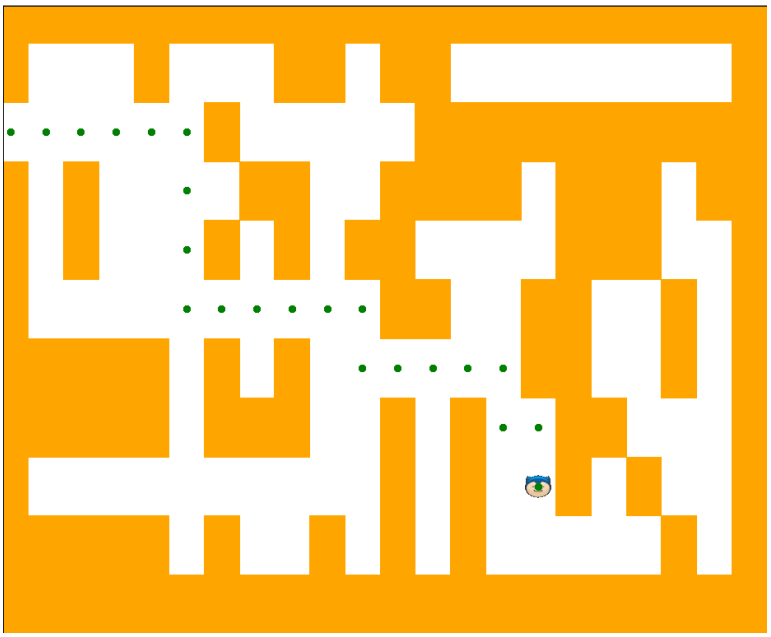
2. 執行結果 :

   ➢ Maze 1:



```
Selected algorithm (dijkstra) path: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1,
 6), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7,
 4), (7, 3), (7, 2), (7, 1), (8, 1)]
Path to exit found by dijkstra. Total path length: 19
Correct path from file: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7),
 (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3),
 (7, 2), (7, 1), (8, 1)]
The search path matches the correct path!
```

- Maze 2 :



```
Selected algorithm (dijkstra) path: [(8, 15), (7, 15), (7, 14), (6, 14), (6, 13)
, (6, 12), (6, 11), (6, 10), (5, 10), (5, 9), (5, 8), (5, 7), (5, 6), (5, 5), (4
, 5), (3, 5), (2, 5), (2, 4), (2, 3), (2, 2), (2, 1), (2, 0)]
Path to exit found by dijkstra. Total path length: 21
```

- Maze 3 :

```
Selected algorithm (dijkstra) path: []
No path found by dijkstra.
```

三、 Analyze the difference between three algorithms

BFS 與 DFS 的差異相當明顯，BFS 會將目前位置的鄰居全部搜索完成後，才去搜索鄰居的鄰居，而 DFS 會從一個鄰居開始一路找到底之後，才對另外一個鄰居做搜索。

DFS 無法保證說找到的路徑為最短路徑，但是他會去將所有路徑都搜索過一次；相對的 BFS 不一定會將所有度竟都搜索過，但是可以保證找到的路徑為最短路徑。

Dijkstra 與 BFS 有些相似，都是可以尋找到最段路徑的演算法，但是 BFS 沒有辦法對有權重的路徑做搜索，而 dijkstra 支援對有權重路徑的圖做搜索。驗算法上的差異在於，BFS 會將為搜索過的鄰居加入 queue 中，而 dijkstra 會在未搜索過的路徑找找到權重最小的路徑做搜索。

四、 What I have learned

在這次作業中，我認為最大的收穫在於找出學習這些演算法的盲點，像是在讀上課講義的時候，我以為我已經很了解 dijkstra 演算法的原理以及運作方式，但是輪到我自己去應用這套演算法時我才發覺到其實有很多細節被我忽略了。

除此之外，雖然說走迷宮這個問題在作業三的時候就已經透過 stack 來實現過一次了，但是這一次使用不同的驗算法來實做相同的問題感覺很不一

樣，而且也能夠更深的體會到不同演算法在同一個問題中的表現差異。