- **Show your autograder results and describe each algorithm:**
  - Q1. Depth First Search

```
Question q1
===========
*** PASS: test_cases/q1/pacman_1.test
***      pacman layout:          mediumMaze
***      solution length: 130
***      nodes expanded:         146

### Question q1: 5/5 ###


Finished at 17:55:44

Provisional grades
==================
Question q1: 5/5
------------------
Total: 5/5
```

大致上如 slide 上所述，創建一個 Stack (LIFO) 作為 frontier 儲存節點的 state 以及路徑，需要是 Stack (LIFO) 的原因是在探索節點的 successor 時，使用 LIFO 才能夠優先繼續在某個節點往深處探索，當探索完以後，再回到初始分岔的節點繼續往別條路探索，這是 dfs 的特性；以及一個 set()名為 explored_set 來儲存已訪問過的節點。接著一個 while loop，當 frontier 不為空時，開始處理 dfs，把 frontier 中的 state 以及 actions（即路徑）pop 出來，接著檢查若此 state 為 Goal，則回傳路徑，並結束搜索；若此 state 不為 Goal 也不在 explored_set 中，則把此 state 加進 explored_set，並去看他的 successors，若 successor 也不在 explored_set 中，則把新的路徑更新，並且把此 successor 的 state 以及路徑 push 進 frontier。這整套流程即為 dfs 的邏輯概念。

- Q2. Breadth First Search

```
Question q2
===========
*** PASS: test_cases/q2/pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 5/5 ###


Finished at 18:06:49

Provisional grades
==================
Question q2: 5/5
------------------
Total: 5/5
```

和 dfs code 邏輯差不多，只是儲存節點的資料結構要使用 Queue (FIFO)，這樣才能在廣度探索時，依序探索同一層的節點，其餘概念都與 dfs 一樣。

- Q3. Uniform Cost Search

```
Question q3
===========
*** PASS: test_cases/q3/ucs_4_testSearch.test
***     pacman layout:          testSearch
***     solution length: 7
***     nodes expanded:         14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']

### Question q3: 10/10 ###


Finished at 18:20:15

Provisional grades
==================
Question q3: 10/10
------------------
Total: 10/10
```

和前面兩個 search 的 code 邏輯相差也不多，只是儲存節點的資料結構需要用 Priority Queue，因為需要在每次探索時，以 cost 最低者優先。故其餘處理都一樣，只是需要每次探索時，更新到此節點的 cost。

- Q4. A* Search (null heuristic)

```
Question q4
===========
*** PASS: test_cases/q4/astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']

### Question q4: 15/15 ###


Finished at 18:25:38

Provisional grades
==================
Question q4: 15/15
------------------
Total: 15/15
```

A* search 與其他 search 方法最不一樣的地方就是，需要看到達每個節點的 path cost 以及 heuristic cost，即$f(n) = g(n) + h(n)$。這邊採取的作法是：多建立一個 dictionary 來紀錄各節點的 cost。其他邏輯和 UCS 一樣，只需要多一個條件：在找 successor 時，若 successor 尚未在 explored_set，以及新找到的路徑的 cost 比原本找到的路徑的 cost 還小時，要進行更新。這兩個重點即為 A* search 的精髓。

- Q5. Breadth First Search (Finding all the Corners)

```
Question q2
===========
*** PASS: test_cases/q2/pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 5/5 ###


Question q5
===========
*** PASS: test_cases/q5/corner_tiny_corner.test
***     pacman layout:          tinyCorner
***     solution length:                28

### Question q5: 5/5 ###


Finished at 19:40:39

Provisional grades
==================
Question q2: 5/5
Question q5: 5/5
------------------
Total: 10/10
```

這裏實作的是 CornersProblem 的 getSuccessors 部分，大概念有兩個，第一個就是檢查有沒有撞到牆；第二個就是檢查是否到達角落，若有的話在角落清單的 list 中註記為已走過這個角落。

- Q6. A* Search (Corners Problem: Heuristic)

```
Question q4
===========
*** PASS: test_cases/q4/astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']

### Question q4: 15/15 ###


Question q6
===========
*** PASS: heuristic value less than true cost at start state
path: ['North', 'East', 'East', 'East', 'East', 'North', 'Nor
t', 'South', 'South', 'South', 'South', 'South', 'South', 'We
, 'East', 'East', 'East', 'East', 'North', 'North', 'East', '
South', 'South', 'South', 'South', 'North', 'North', 'North',
path length: 106
*** PASS: Heuristic resulted in expansion of 774 nodes

### Question q6: 9/9 ###


Finished at 19:52:02

Provisional grades
==================
Question q4: 15/15
Question q6: 9/9
------------------
Total: 24/24
```

這裡應該是整份作業中最難的一個 part 了，原本想說最簡單的 heuristic 就是，找當下離自己最近的點，就這樣一直做一下去，結果發現只拿到 6/9 分。那感覺就只能用 DSA 中學到的 MST 來做了，於是就開始實作 MST。MST （最小生成樹）的概念就是將目前所在位置，以及剩餘的所有節點連在一起，建立一個無向圖 (Undirected Graph)，而兩節點的邊代表兩節點間的最短路徑，再對這無向圖計算最小生成樹，而計算出來的 MST 總權重代表了在訪問所有特定點的過程中的最小代價。再將此 MST 的總權重作為估計代價，當作 heuristic function 中的一部分。剩下皆與 A* search (null heuristic) 一樣。

- **Describe the difference between Uniform Cost Search and A* contours**

  UCS 為每次搜尋時皆以當下的最小成本來搜尋，如此一來，最後可以確保找到最佳路徑。但是在較為複雜的地圖中，此種搜尋方法也有可能導致非常低效（例如：一直找遠離終點的方向）；

  而 A* search 則是結合了 UCS 以及 Greedy Search 的概念，計算的不是當下最小的成本，而是在加入一個 heuristic function（例如：離終點的成本），計算當下的最適成本的搜尋方法。

  用以下表格來簡單比較：

	成本評估	效率	最佳解
UCS	僅考慮到達當前節點的 path cost	可能會搜尋很多無關位置，因為他搜尋的方法與終點無關	保證找到最佳解
A* Search	結合了 path cost & heuristic cost	透過 heuristic function 導引，會往導向終點的方向搜索	保證找到最佳解

- **Describe the idea of Admissibility Heuristic**

  可接受性啟發式不高估實際成本，也就是說他估計的成本最多等於最小實際成本。舉例來說：Manhattan Distance, Euclidean Distance，都是可接受性啟發式的例子。他們能保證在找到最優解的同時，提升演算法的效率。