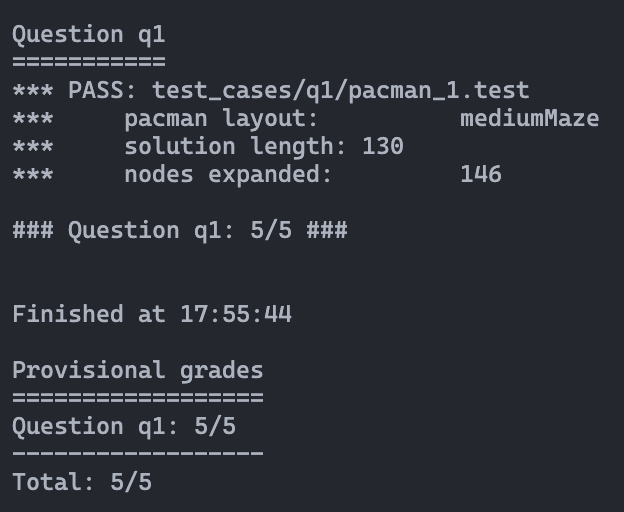
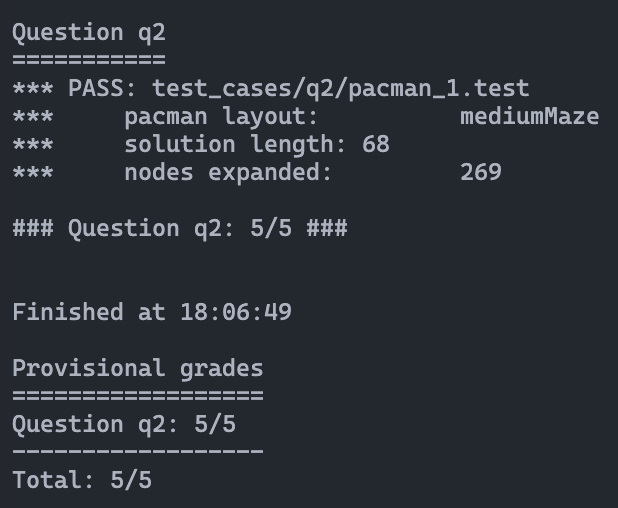
Hw1\_r10723057

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

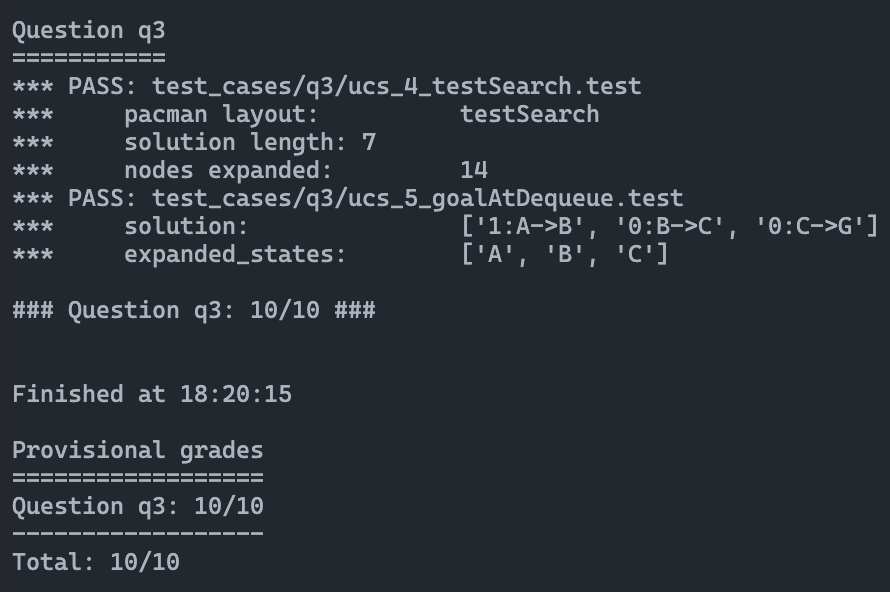


大致上如slide上所述，創建一個Stack (LIFO) 作為frontier儲存節點的state以及路徑，需要是Stack (LIFO) 的原因是在探索節點的successor時，使用LIFO才能夠優先繼續在某個節點往深處探索，當探索完以後，再回到初始分岔的節點繼續往別條路探索，這是dfs的特性；以及一個set()名為explored\_set來儲存已訪問過的節點。接著一個while loop，當frontier不為空時，開始處理dfs，把frontier中的state以及actions（即路徑）pop出來，接著檢查若此state為Goal，則回傳路徑，並結束搜索；若此state不為Goal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

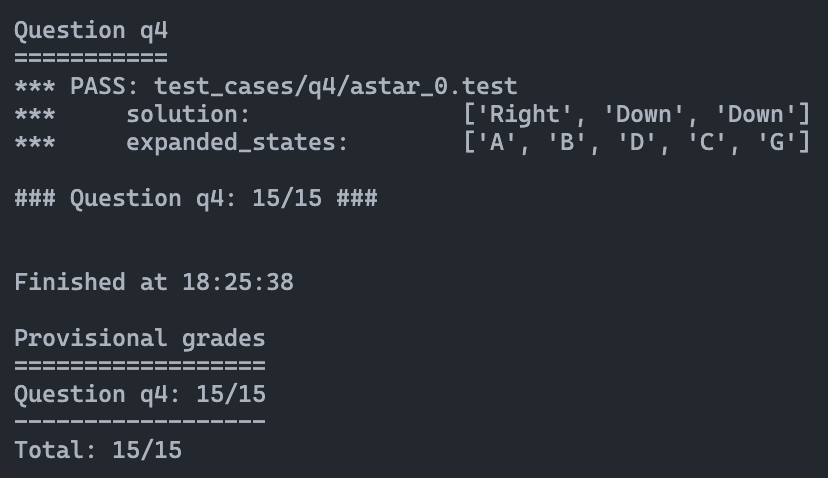


和dfs code邏輯差不多，只是儲存節點的資料結構要使用Queue (FIFO)，這樣才能在廣度探索時，依序探索同一層的節點，其餘概念都與dfs一樣。

* Q3. Uniform Cost Search

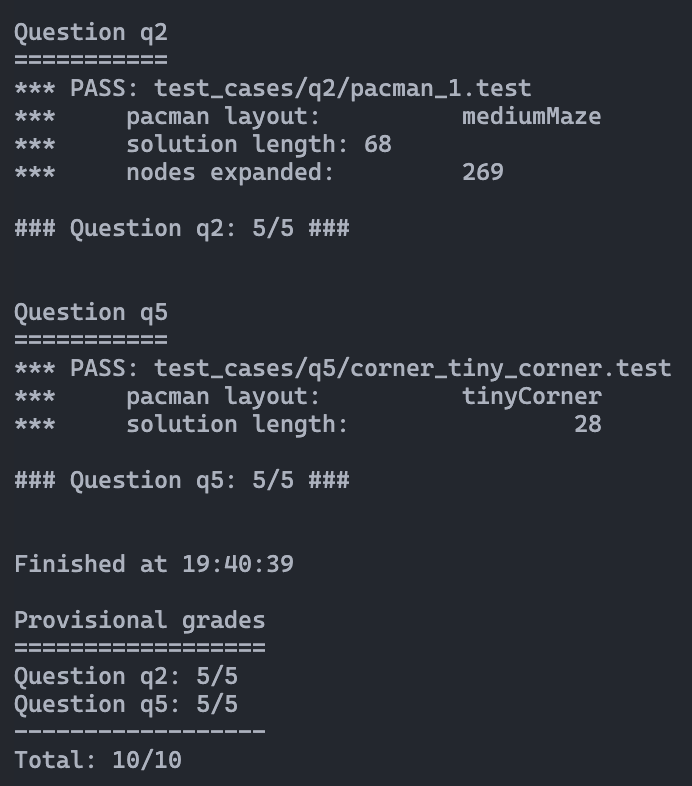
和前面兩個search的code邏輯相差也不多，只是儲存節點的資料結構需要用Priority Queue，因為需要在每次探索時，以cost最低者優先。故其餘處理都一樣，只是需要每次探索時，更新到此節點的cost。

* Q4. A\* Search (null heuristic)



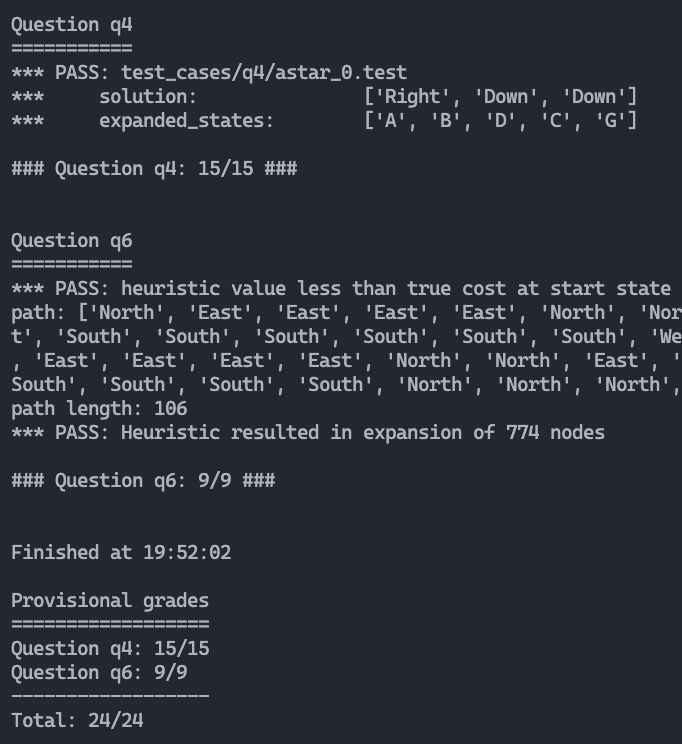
A\* search與其他search方法最不一樣的地方就是，需要看到達每個節點的path cost以及heuristic cost，即。這邊採取的作法是：多建立一個dictionary來紀錄各節點的cost。其他邏輯和UCS一樣，只需要多一個條件：在找successor時，若successor尚未在explored\_set，以及新找到的路徑的cost比原本找到的路徑的cost還小時，要進行更新。這兩個重點即為A\* search的精髓。

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



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)



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。