人工智慧作業二

姓名：許哲安
學號：40871107H

1.

MacBook Pro (15-inch, 2019)
Processor: 2.3 GHz 8-Core Intel Core i9
Memory: 16 GB 2400 MHz DDR4
Graphics: Radeon Pro 560X 4 GB
Intel UHD Graphics 630 1536 MB
OS: Mac os Big Sur

開發語言：python
會選擇這樣的規格沒有什麼特別原因，就剛好開學時師大旁的電腦店有優惠，所以就購買了
電話：0905235578

2. 輸入檔製作方式：

有解盤面的製作方式主要有兩種，第一種是先由 goal state，人工一個一個換 blank 的位置，使得此輸入必定有解，另外一種是從 n puzzle game app 輸入 app 的 n puzzle 測試，而無解盤面是上網搜尋無解盤面的 state，再去測試

3. 如何執行程式：

兩支程式只能解單個方塊(1*1)所構成的 N*P 的 puzzle，並且只有 1 個 blank 的問題，無法解 1*2、2*1 或 2*2 所構成的 N*P 的 puzzle
由於我是利用 Replit 跑，所以最直接的方式就是將程式碼在 replit 的頁面執行

4. IDS

方法：IDS 主要是利用限制深度的 DFS 去做搜索，先去看初始盤面是否有解，如果有解就跑 IDS，limit 值用 For loop through，IDS 的執行是先檢查 current node 是不是 goal state，如果不是就把 current node 加入 open set，然後再 expand current node，先檢查是否去過，沒去過再把 expand node 加入 frontier，持續做到找到 goal state 或達到 limit 值

資料結構：利用 linked list ，link parent node
操練要項：

1. 盤面表示：利用 2D list 表示盤面

2. 走步產生：先尋找 blank 的位置，在測試是否能將 blank 往左、右、上或下移動，如果可以，就創造新的節點

3. 節點資訊：
   state: 盤面
   parent: 此節點的 parent
   move: parent node 要如何移動，才能產生此節點(Ex: parent node 8 往左移，那就會存 8L)
   depth:此節點的深度

4. 是否需判別重複：是，為了節省時間與空間

5. 是否會跑不停：理論上不會，因為有設置 limit 值，假如跑到 limit 值還找不到解，他就會停下來

6. 記憶體是否會爆：如果盤面過大或是 steps 過多會造成記憶體爆炸

7. 結果是否為最佳解：是， loop through limit 值就是在看這次我們 step 最多做 limit 次，所以會是最佳解

8. 使用哪種 Heuristic：IDS 無 Heuristic

9. 如何估算時間與空間消耗量：時間上是利用 python 模組 timeit，空間消耗量是運用 python 模組 os, psutil，如果是人工估算的話，估計主要需要考慮盤面大小與盤面的難度等

是否會找到最佳解：不一定，由於我 limit 值最多 loop 到 49，也就是說需要超過 49 個 steps，就無法找到解，另外過程中也有可能因為記憶體爆掉，而導致程式 signal killed

5. IDA*

方法：與 IDS 的差別不大，主要是將 IDS 的 limit 值，改成由 Heuristic 去估測，找到大於 current f-limit 時就停下，並且也會在過程中更新 next f 的值，next f 的值是這輪大於 f-limit 的所有值中的 min

資料結構：利用 linked list ，link parent node

操練要項：

1. 盤面表示：利用 2D list 表示盤面

2. 走步產生：先尋找 blank 的位置，在測試是否能將 blank 往左、右、上或下移動，如果可以，就創造新的節點

3. 節點資訊：
   state: 盤面
   parent: 此節點的 parent

move: parent node 要如何移動，才能產生此節點(Ex: parent node 8 往左移，那就會存 8L)
depth:此節點的深度
h: heuristic 值
f: cost + heuristic

4. 是否需判別重複：不需要，重複的節點 cost 會依據重複的 path 之值變得較大，在做 idA*時，就會停在那個節點

5. 是否會跑不停：如果盤面太大、step 太多，會跑不停

6. 記憶體是否會爆：理論上不會，因為 idA*是重複產生節點，唯一可能造成記憶體爆炸的情況應該是在 expand node 時，一直無法超過 f-limit，所以一直 expand，假如過程中 node 太多，可能會造成記憶體爆炸

7. 結果是否為最佳解：是，在 loop through 時，是根據 f-limit，而 f-limit 是 g+h，判斷是否越來越接近 goal-state，每次只要超過 f-limit，下一 round 都是抓此輪超過 f-limit 的 min，當作這一輪的 f-limit，所以不會有非最佳解的情況

8. 使用哪種 Heuristic：使用 manhattan distance

9. 如何估算時間與空間消耗量：時間上是利用 python 模組 timeit，空間消耗量是運用 python 模組 os, psutil，如果是人工估算的話，估計主要需要考慮盤面大小與盤面的難度等

是否會找到最佳解：如果跑得出來，一定是最佳解，但如果盤面太大、太複雜，有可能跑不停

6.比較 IDS 與 IDA*測試盤面表現（時間與空間）：

IDA* vs IDS

	Input	IDA* Time (sec)	IDA* Memory (MB)	IDS Time (sec)	IDS Memory (MB)
Input 1	3 3 0 1 2 5 7 3 4 8 6	0.012235369000336	33.11328125	4.56896328100037	36.0625
Input 2	3 3 1 3 5 4 2 0 7 8 6	0.002134930000465	33.1484375	0.063280066000061	33.31640625
Input 3	3 3 2 4 3 5 6 8 1 7 0	0.068374290999599	32.91796875	22.6739121770001	44.1875
Input 4	3 3 1 8 2 0 4 3 7 6 5	0.010601877999761	32.94921875	2.41312610600016	37.11328125
Input 5	3 2 5 1 2 3 0 4	0.055928902000232	33.0859375	3.24704969999948	36.73046875
Input 6	5 5 2 3 8 4 5 1 7 9 10 15 6 17 12 18 14 11 0 13 20 24 21 16 22 23 19	0.172625858000174	33.4296875	Signal killed	Signal killed
Input 7	5 5 16 10 1 4 3 15 8 23 12 2 6 13 5 9 18 7 20 19 11 17 21 22 14 24 0	Run over 18 min	Run over 18 min	Didn't test it	Didn't test it
Input 8	3 3 2 7 4 6 3 1 5 8 0	28.1127099119994	33.08984375	Signal killed	Signal killed

在時間與記憶體方面，IDA*都優過於 IDS，甚至在 input 8 IDA*需要跑 28 秒的情況下，IDS 直接跑不出來，另外一個比較奇特的點是在 input 2 與 input 8 於 IDA* 的比較，很明顯 Input8 比較複雜，需要 28 秒才解出來，可是時間運用很長，記憶體運用卻比 Input 2 運用的還少，理論上 Input8 要比較多，這部分可能是記憶體計算錯誤。

而假如是比較複雜的盤面，例如 Input7，我測試 IDA*時，跑了超過 18 分鐘還沒跑出來，所以就沒有測試 IDS 了，另外 Input 8，IDA*跑了 28 秒，可是 IDS 最後 signal killed，顯示出的確 IDA*與 IDS 相比，有著顯著的差別。

7. 困難：
    1. 選擇 c 或 python
    2. python 語言較不熟悉
    3. 只先完成正常的 N*P puzzle，作業要求的 puzzle 由於時間與做法，沒有嘗試

8.參考文獻：

(1) 8-puzzle/8_Puzzle.ipynb at master · ZeyadZanaty/8-puzzle · GitHub

參考 class 的設置、get_path、get neighbors、dfs

(2) **Restricting Moves When Solving the NxM–puzzle (mcgill.ca)**

參考如何判斷是否有解

Theorem 1. An NxP-puzzle with an odd number of columns can be solved only if NI is an even number. An NxP-puzzle with an even number of columns and an even number of rows can be solved only if (NI + Row of the blank) is an even number. An NxP-puzzle with an even number of columns and an odd number of rows can be solved only if (NI + Row of the blank) is an odd number.