AI-HW2-40647001S 郭奕成

1. **機器軟硬體規格與作業系統、開發軟體版本、如何執行。連絡電話。**
   硬體：
   CPU：AMD R5-2600X
   RAM：DDR4 8Gx2
   軟體：
   OS：Windows 10 home
   Language：Python 3.8 64bits
   如何執行：
   使用Python 3.8或相近的版本執行下面兩個程式。
   1. idastar.py
   2. random-restart-hill-climbing.py

   聯絡電話：0975-389-568

2. **輸入檔五份，盤面尺寸要有大有小、難度不一。說明如何製作這些測試用的輸入檔(可考慮使用亂數產生)。**
   data-generator.py
   依使用者設定盤面的大小N、要有幾個聖誕老人，產生亂數座標先將這些老人設定在盤面上，在用亂數決定鹿車要放在老人的哪個方向，如果跑出來的老人跟鹿車數量不合則重跑一次，再來計算每一列每一行的鹿車數量，最後將這些資料拿去做goal test，如果符合則產生成功，輸出那個沒有包含鹿車的盤面，如果不符合則再重新產生一次。

3. **原始碼中要加註解，說明如何執行這兩支程式。**
   idastar.py：
   可以自行設定inputFileName, outputFileName這兩個變數，在呼叫idaStar(, maxBound)可設定最大的bound，如果都超過會印出訊息並結束函式。

   random-restart-hill-climbing.py
   可以自行設定inputFileName, outputFileName這兩個變數，在呼叫randomRestartHillClimbing(, times)可設定最多restart幾次，如果超過會印出訊息並結束函式。程式碼的最上面也可以設定亂數種子。

4. **第一支程式。使用什麼方法、資料結構、技術(操練要項)。測試結果表現如何、耗用的時間跟空間為何。可以解到多大盤面的題目。用例子輔助說明。**
   a. 方法：IDA* Search
   b. 資料結構：NxN list
   c. 盤面怎麼表示：NxN char list
   d. 棋盤全部資訊要存在每個節點嗎：每個節點都有存全部的空格、老人、鹿車，每一列每一行的鹿車數量則不存在節點裡。
   e. 節點要存那些局部資訊：局部資訊只有像是節點目前的cost會存在遞迴函式。
   f. 走步要如何產生：每個還未展開過的老人都向上下左右四個方面展開放上鹿車，並標記老人表示已經展開過。
   g. 需要判別重複：不用，因為展開的時候會找未標記過的，所以一定不會跟目前路徑上的節點有重複的。

h. 最後答案如何取出來：最後會是list儲存著每一步驟，這個list的最後一步就是答案，再將標記過的老人變回'c'字元、加上盤面大小、加上每一列每一行的鹿車數量來符合輸出格式。
i. 會不會跑不停：不會，bound > total number of clauses之前就會找到答案，如果沒有則會輸出無解，或是在超過使用者設定的max bound之後也會停止。
j. 記憶體會不會爆掉：如果盤面大小、老人數量很多則有可能，老人越多遞迴深度越深，C個老人，每次展開4C, 4^2(C-1), 4^3(C-2), ... , 4^c1，(C^2+C)C!，乘以盤面的大小N^2，所以空間複雜度為C^2C!N^2。
k. 所得結果會是最佳解嗎：從一開始隨機配置鹿車，操作到答案的步驟會是最佳解。從另一個角度來看，每一步放一台鹿車，成本一定會是total number of clauses，所以沒有什麼最佳解的問題。或從符合遊戲條件為觀點，IDA* Search如果沒找到解，就會輸出無解，也沒有什麼最佳解的問題。
l. 使用哪一種heuristic結果比較好：第一種違反的規定是鹿車在橫行、直行、斜行上相鄰，可以計算出有幾對鹿車違反。第二種違反的規定是橫行或直行的鹿車數量要跟方框外的數字相同，可以計算出數量的差距，因為變更鹿車的位置會使原本的那一行列的數量減一，新的位置的那一行列的數量加一，差距會是2，所以算出來的數量差距要除以2。按上面的方式可以算出鹿車衝突的組數、列的數量差距，行的數量差距，這三個數字，在根據admissible heristic的規定，取這三個數字中最大的當作最終結果，這樣便可以找到最佳解(從隨機配置的鹿車開始調整的最少步驟)。如果不是上述的方式的話，另一個是把三個數字都加起來，heuristic的值會變很大，bound增加的速度會增快，有可能因此比較快進入這個盤面就是需要的cost，而比較快找到答案，但也可能因此超過盤面需要的cost，而多走訪了很多其他的路徑，造成計算變慢。
m. 碰到dead end可提早backtracking：碰到dead end的時候因為沒地方可以展開所以回傳空的list，list裡面沒東西就不會再往下執行，會提早backtracking。
n. 如何估算時間及空間的消耗量：老人越多遞迴深度越深，C個老人，每次展開4C, 4^2(C-1), 4^3(C-2), ... , 4^c1，(C^2+C)C!，乘以盤面的大小N^2，所以空間複雜度為C^2C!N^2。時間複雜度也是大概相同。

再來分成四種版本來做分析：

1. 原版
2. 每次展開的時候只針對一個老人，successors最多只有4個，bound一定為老人的數量
3. 原版+把heristic計算方式改為每一行列的雪橇數量跟衝突的雪橇組數三個變數相加
4. 第二版+把heristic計算方式改為每一行列的雪橇數量跟衝突的雪橇組數三個變數相加

N(盤面尺寸)	6	8	10	10	12	12	20
C(老人數量)	9	12	12	16	12	20	12
space ratio	0.5	0.625	0.76	0.68	0.8333	0.7222	0.94
1.run time(s)	1.628	57.933	N/A	N/A	N/A	N/A	N/A
1.memory(MB)	4.4	4.6	N/A	N/A	N/A	N/A	N/A

2.run time(s)	1.243	4.571	4.56	14.309	2.892	N/A	N/A
2.memory(MB)	4.3	4.3	4.4	4.4	4.5	N/A	N/A
3.runtime(s)	0.149	N/A	N/A	N/A	N/A	N/A	N/A
4.runtime(s)	0.277	0.433	0.124	0.19	0.55	10.594	0.154

第一版隨著老人跟盤面尺寸增加，執行時間暴增。

第二版比第一版快很多，老人跟盤面尺寸增加對於執行時間的影響下降很多，每次只針對一個老人展開，而不是全部都展開，這樣會讓解題速度加快。

第三版確實會因為bound增加過快而使計算量暴增。

第四版，要把maxBound設定大一點，不然會因為bound增加太快，很快就超出設定而輸出無解。但解題速度是四個版本中最快的，每次只針對一個老人展開、heristic的值抓大一點，會讓解題速度加快。

我覺得這道題目其實不需要設定admissiable heuristic來追求最佳解(最佳路徑)，最後只要算得出來解答就好，把heuristic設大一點反而可以增加解題速度，所以在這邊大的heuristic是比較好的。space ratio越高的執行時間會較短，但盤面尺寸跟老人數量也都會增加執行時間。相同老人數量之下，盤面尺寸越大則space ratio增加，執行時間下降。相同盤面尺寸下，老人數量增加則space ratio下降，執行時間增加。記憶體用量都不大，隨盤面尺寸跟老人數量增加而增加，但影響也不會太大。盤面20以內的表現大概是這樣，要再上去的話因為產生輸入的程式也要跑很久，隨機算出來的盤面很可能不符合遊戲規定而無法產生，所以比較仔細的觀察的只分析上面這些數據。盤面尺寸100，老人數量12，使用第四版執行的時間為7.187s，可見盤面尺寸大很多、老人數量固定的時候，執行時間還不會暴增太多。當盤面尺寸100，老人數量20的時候，第二版的執行時間有個8.79s就跑出來，有個跑很久都還沒跑出來，會這樣是因為bound有可能會因為heuristic設的太高而暴增，徒增很多計算量而導致解題時間暴增。

5. **第二支程式。使用什麼方法、資料結構、技術(操練要項)。測試結果表現如何、耗用的時間跟空間為何。可以解到多大盤面的題目。用例子輔助說明。**
    a. 方法：random restart hill climbing
    b. 資料結構：NxN list
    c. 盤面怎麼表示：NxN char list
    d. 棋盤全部資訊要存在每個節點嗎：每個節點都有存全部的空格、老人、鹿車，每一列每一行的鹿車數量則不存在節點裡。
    e. 節點要存那些局部資訊：局部資訊只有像是節點目前的cost會存在遞迴函式。
    f. 走步要如何產生：每個還未展開過的老人都向上下左右四個方面展開放上鹿車，並標記老人表示已經展開過。
    g. 需要判別重複：不用，因為展開的時候會找未標記過的，所以一定不會跟目前路徑上的節點有重複的。

h. 最後答案如何取出來：因為走過去後就把前面的路徑忘記，所以最後只會有一個節點，goal test成功的就是答案，再將標記過的老人變回'c'字元、加上盤面大小、加上每一列每一行的鹿車數量來符合輸出格式。
i. 會不會跑不停：不會，randomRestartHillClimbing()會在超過使用者設定的次數後結束。
j. 記憶體會不會爆掉：不會，走過去後會把前面的都忘記，一次只存一個節點。另外用比較多記憶體的是在展開的時候會先把所有展開的節點都存起來，再做比較，找到最好的，因為這樣程式比較好寫，雖然記憶體用量會大一點，但最多只是CN^2這個大小，算是非常小。
k. 所得結果會是最佳解嗎：以一步驟一步驟來看的話，並不是最佳解。以每次放一台鹿車的觀點來看，則沒有是不是最佳解的問題。以符合遊戲條件為觀點，沒找到答案則會輸出無解，也沒有什麼最佳解的問題。
l. 使用哪一種heuristic結果比較好：第一種違反的規定是鹿車在橫行、直行、斜行上相鄰，可以計算出有幾對鹿車違反。第二種違反的規定是橫行或直行的鹿車數量要跟方框外的數字相同，可以計算出數量的差距，因為變更鹿車的位置會使原本的那一行列的數量減一，新的位置的那一行列的數量加一，差距會是2，所以算出來的數量差距要除以2。按上面的方式可以算出鹿車衝突的組數、列的數量差距，行的數量差距，這三個數字，在根據admissible heristic的規定，取這三個數字中最大的當作最終結果。
m. 碰到dead end可提早backtracking：如果抵達local maximum的時候，不論往哪走都會使節點的值變低，所以不會過去，會結束hill climbing，再重新產生random initial state跑新的hill climbing。
n. 如何估算時間及空間的消耗量：記憶體用最多的就是展開的時候存的所有successors，C個老人，CN^2這個大小。時間複雜度則因為這個演算法主要是靠運氣，時間複雜度不好估算，但根據程式的執行結果來看大多是比IDA* Search還要快蠻多的。

再來分為個版本來做分析：

1. 原版
2. heuristic改為每一行列的雪橇數量跟衝突的雪橇組數三個數字相加

N(盤面尺寸)	6	8	10	10	12	12	20
C(老人數量)	9	12	12	16	12	20	12
space ratio	0.5	0.625	0.76	0.68	0.8333	0.7222	0.94
1.run time(s)	0.157	0.431	1.143	2.081	1.921	19.073	17.233
1.memory(MB)	4.4	4.4	4.4	4.4	4.5	4.6	4.7
2.run time(s)	0.12	0.232	0.307	0.725	0.164	6.063	0.666

第一版，表現意外的好，比IDA* Search好很多，也隨著老人數量跟盤面尺寸增加，執行時間增加，但幅度明顯小很多。記憶體用量則跟IDA* Search差不多，倒是有點意外，可能跟python程式執行的基本需求有關。

第二版，把heuristic設定成大很多之後大幅增加了解題速度。
因為這個演算法也不必要使用admissiable heuristic，所以選擇比較大的heuristic會增加解題速度，會比較好。雖然這個演算法有點看運氣，但執行時間跟盤面難度的比例感覺是差不多。space ratio越高的執行時間會較短，但盤面尺寸跟老人數量也都會增加執行時間。相同老人數量之下，盤面尺寸越大則space ratio增加，執行時間下降。相同盤面尺寸下，老人數量增加則space ratio下降，執行時間增加。記憶體用量都不大，隨盤面尺寸跟老人數量增加而增加，但影響也不會太大。盤面20以內的表現大概是這樣，要再上去的話因為產生輸入的程式也要跑很久，隨機算出來的盤面很可能不符合遊戲規定而無法產生，所以比較仔細的觀察的只分析上面這些數據。當盤面尺寸100，老人數量12的時候，第二版執行時間為2.641s，可見盤面尺寸大很多、老人數量固定的時候，執行時間還不會暴增太多。當盤面尺寸100，老人數量20的時候，第二版的執行時間為36.433s，測試第二個測資為21.32s，再增加一些老人數量，執行時間就增加蠻多的，所以程式在盤面尺寸不要太大的情況下(100內)的執行時間主要還是取決於老人數量跟space ratio，而且解題速度還蠻一致的，不會忽慢忽快，整體看起來是比第一支程式還要好。

6. **此次作業所碰到的狀況跟困難。**
每個應用的實作都有不小的差距，所以一開始去查N puzzle、N queens的資料的時候，看完了也只有一點點啟發，對這個題目的想法還是很模糊，還是要靠自己仔細思考之後設計程式的細節。從最開始IDA* Search只知道個大概，實際要寫code還是沒有想法，很多東西都不確定，像是盤面最一開始的節點是要只有老人跟空格，還是說要隨機擺鹿車給老人，如果是前者，那這樣就不用iterate了，bound也一定就是總共要有幾個鹿車，這樣就不像是IDA* Search，那如果是後者，但在玩這個遊戲的時候也不會先隨機擺鹿車給老人，這樣算出來的最佳步驟也沒有意義，反正玩遊戲的時候肯定是要按C(老人數量)次去給鹿車，這裡我猶豫了很久，最後採用後者，但又出現另一個問題，既然先隨機配置了鹿車，之後要調整鹿車位置的時候也要知道這台鹿車是屬於誰的，解決辦法就是在盤面上老人的位置標記0~3來代表鹿車放在老人的哪個方向，展開的時候再標記說哪個老人展開了，之後的節點不要再對這個老人展開。程式碼實作的細節也是上網查了很多資料，研究跟思考很久。heuristic function的設計也是不知道做出來的好不好，整個演算法會不會動，硬著頭皮依照遊戲規定先決定可以算出的三個數字，感覺很容易超過自己模擬從隨機鹿車位置到goal state所花的成本，所以最後決定是這三個變數取最大值。我對python這個語言還不是很熟悉，但因為比起其他語言真的是好寫很多，程式碼的份量也少很多，所以還是決定要用python寫，因為很多東西都是不確定的，有時候寫一寫突然想到就跳到別段程式碼做修改，所以在開發的過程都是先把所以東西寫出來，跑起來看有沒有問題，而不是一個模組一個模組做測試，這樣的結果就是第一次跑的時候程式都會有問題，然後就要很痛苦的慢慢debug，debug是必然的結果就是，只是早作晚做的事，只是因為很多東西都不確定，最後再一次de感覺會比較有效率，但缺點就是debug的過程會很迷茫吧，不確定到底問題在哪，不確定要de多久，設一個breakpoint、下一步、下一步、下一步，但是因為程式碼不算少，而且很多迴圈，要抓到問題點要花很多時間，幸好最後有抓出來，有一些小錯誤，數值設錯、誤更改到變數、程式流程想錯等等之類的問題。在寫第二支random restart hill climbing的時候，演算法的細節研究之後，程式碼實作的部分就比較順利一點。兩支程式都寫得差不多了，再來就是產生測資的程式，難度是沒有前面兩支高，但程式碼意外的多，也是花了意外多的時間。然後就是這個產生測資的程式在N跟C大

的時候也要跑很久，所以測試也不太好測。之後再對兩支程式做一些修改，解題速度的差距意外的大。

7. **參考文獻，說明參考了那些部分用於作業中。**

    a. 看他的IDS例子來理解iterative deepening的觀念
    Iterative deepening depth-first search

    b. IDA* Search 參考他解 8-puzzle problem 的觀念跟 heuristic function 的觀念
    演算法筆記- State

    c. IDA* Search 的程式碼有參考這個網站的
    IDA* 迭代加深A star演算法解決15數碼問題——python實現- IT閱讀

    d. 學習hill climbing的觀念
    Introduction to Hill Climbing | Artificial Intelligence

    e. hill climbing的演算法的 pseudocode
    Hill climbing

    f. random restart hill climbing的演算法
    AI第四張講義