人工智慧導論 Program assignment#1

學號：411086006　　學生：張愷和

所有程式皆使用 C++，並用 wsl 上 linux 環境中的 gcc 編譯檔案

一、使用 Iterative deeping search 解決 n-puzzle 問題

我先介紹一下 IDS.cpp 檔案中所寫的 function。

1. solvable，透過判斷輸入的 start 和 goal 的 inversion 次數，如果皆為奇數和偶數，那盤面有解，如果為一奇數一偶數，盤面無解。
2. IDS，for 迴圈進行計算限制深度，對每次 DFS 沒找到就加一次深度下去搜索。
3. DFS，輸入 start 和 goal，用 Depth first search 來尋找解的深度以及 saved_state。

那我就先從 main()開始解釋八，我在盤面的輸入以及輸出是以字串表示，先判斷是否有解，如果有解就開始搜尋，沒有解就輸出”Not found”，有解就跳到 IDS function 開始搜索，跳入 IDS function 時，就會開始對 DFS 限制深度，只要一達到深度 DFS 就停止搜索，並回傳有沒有找到，沒有找到繼續增加深度，有找到就跳回 main()，進入 DFS function 時，先宣告一些參數，moves，用來判斷 0 的位置在下一步可以往哪個二為陣列(0~8)的方向走，s，這裡使用 pair 宣告<起始盤面, 深度=0>，用來儲存 parent node 所展開的 child，由於是使用 stack 儲存，根據 FILO 的效果，可以一直往深的地方下去探索，record，用來儲存已經走過的盤面，需要先把 start 的盤面儲存進去，宣告完後，直接進入 while 迴圈，這裡因為有 main()中判斷有無解，應該是不會到跳出 while 迴圈的狀況，總之如果 s.empty(全部 state 都搜尋完了)就跳出迴圈，return -1 表示找不到，進入迴圈，先儲存現在 s 的 size 作為迴圈次數，接著進入 for 迴圈把 parent node 從 s pop 出來，接著判斷參數，如果是 goal state 的話，就輸出深度(pair.second)以及 record 所儲存的參數，接著如果 parent node 的 state 不為 goal state，就尋找 parent state ‘0’所在的位置，也就是空格的位置，在這邊要先判斷是否深度已經達到了所限制的深度，如果達到了限制深度，就不繼續對 node 產生 child，反而往回推一個 node，重

複迴圈，如果沒有達到限制深度，接著使用 for (int k : moves[dir])去移動 0 可以移動到那些位置，假設 0 的位置在 4，那會產生 0 移向{1, 3, 5, 7}，這些陣列的位置，也就是多生成了 4 個 child，在每次生成 child，就去判斷這項 child 有無紀錄過，如果沒有記錄過，就把 child state 和 parent 的深度加 1，push 進 pq 進行排列優先順序判斷，然後順便把沒有記錄過的盤面 insert 進 record，紀錄盤面，接著就重複迴圈如果 stack 先空，就回 IDS 增加限制深度，如果找到，就印出深度以及 saved state 然後 return 1。我在寫完之後發現說，我的程式在做 DFS 的時候，會因為紀錄 saved state 的關係，導致後面 parent 先出現了，可是之前的 parent 還沒搜索到，有可能這個 parent 的 child 就是 goal，但是 parent 到了 limit 就要回去搜索，這導致之前沒搜索過的 parent 就不會被 expand，導致沒有辦法找到最佳解，所以我改成了用 pair 順便儲存同一層重複的 parent，這可以解決上述最佳解的問題，不過產生了新的問題是，儲存的 saved state 會有非常多重複的，所以我最後還是改回原來的，順帶一提，我寫這演算法跑 8-puzzle 沒有什麼太大的問題，不過在跑 15-puzzle 就跑不太出來了。

二、使用 Uniform cost search 解決 n-puzzle 問題

我先介紹一下 UCS.cpp 檔案中所寫的 function。

1. comp，藉由判斷每個 node 的深度(這裡深度儲存在 pair.second 的位置)，用來排列 priority_queue 大小，作為優先搜索順序。
2. solvable，透過判斷輸入的 start 和 goal 的 inversion 次數，如果皆為奇數和偶數，那盤面有解，如果為一奇數一偶數，盤面無解。
3. UCS，輸入 start 和 goal，用 Uniform cost search 來尋找解的深度以及 saved_state。

那我就先從 main()開始解釋八，我在盤面的輸入以及輸出是以字串表示，先判斷是否有解，如果有解就開始搜尋，沒有解就輸出”Not found”，有解就跳到 UCS function 開始搜索，跳入 UCS function 時，先宣告一些參數，moves，用來判斷 0 的位置在下一步可以往哪個二為陣列(0~8)的方向走，pq，這裡使用 pair 宣告<起始盤面, 深度=0>，用來儲存 parent node 所展開的 child，並排列下一次的優先搜索，record，用來儲存已經走過的盤面，需要先把 start 的盤面儲存進去，宣告完後，直接進入 while 迴圈，這裡因為有

main()中判斷有無解，應該是不會到跳出 while 迴圈的狀況，總之如果 pq.empty(全部 state 都搜尋完了)就跳出迴圈，return -1 表示找不到，進入迴圈，先儲存現在 pq 的 size 作為迴圈次數，接著進入 for 迴圈把 parent node 從 pq pop 出來，接著判斷參數，如果是 goal state 的話，就輸出深度(pair.second)以及 record 所儲存的參數，接著如果 parent node 的 state 不為 goal state，就尋找 parent state '0'所在的位置，也就是空格的位置，接著使用 for (int k : moves[dir])去移動 0 可以移動到那些位置，假設 0 的位置在 4，那會產生 0 移向{1, 3, 5, 7}，這些陣列的位置，也就是多生成了 4 個 child，在每次生成 child，就去判斷這項 child 有無紀錄過，如果沒有記錄過，就把 child state 和 parent 的深度加 1，push 進 pq 進行排列優先順序判斷，然後順便把沒有記錄過的盤面 insert 進 record，紀錄盤面，接著就重複迴圈直到找到 goal state。在寫這份程式的時候，其實有先寫了 breath first search，然後在理解 Uniform cost search 的時候才發現深度，其實就代表了 0 移動的步數，所以其實這兩項 search 跑出來的 depth 和 saved state 根本沒有差。

三、使用 Greedy best first search 解決 n-puzzle 問題

我先介紹一下 GBFS.cpp 檔案中所寫的 function。

1. Struct value(v)，定義一個物件，用來儲存盤面，heuristic 數值，以及深度，heuristic 和深度都先初始為 0。
2. comp，藉由判斷每個 node 的 heuristic 數值(這裡 heuristic 數值儲存在物件.h_value 的位置)，用來排列 priority_queue 大小，作為優先搜索順序。
3. solvable，透過判斷輸入的 start 和 goal 的 inversion 次數，如果皆為奇數和偶數，那盤面有解，如果為一奇數一偶數，盤面無解。
4. heuristic_function，輸入 cur.state 以及 goal state，先用兩個 for 迴圈儲存 goal state 的每個數字的 row 和 col 值，接著用曼哈頓距離計算 cur.state 各個數字，與 goal state 各個數字相差的距離相加起來，就是這個 state 的 heuristic 值。
5. GBFS，輸入 start 和 goal，用 Greedy best first search 來尋找解的深度以及 saved_state。

那我就先從 main()開始解釋八，我在盤面的輸入是先宣告一個物件，皆著在輸入物件.state 儲存初始盤面的數值，輸出是以字串表示，先判斷是否

有解，如果有解就開始搜尋，沒有解就輸出”Not found”，有解就跳到 GBFS function 開始搜索，跳入 GBFS function 時先宣告一些參數，moves，用來判斷 0 的位置在下一步可以往哪個二為陣列(0~8)的方向走，pq，這裡使用 v 宣告，用來儲存 parent node 所展開的 child，並排列下一次的優先搜索，record，用來儲存已經走過的盤面，需要先把 start 的盤面儲存進去，宣告完後，直接進入 while 迴圈，這裡因為有 main()中判斷有無解，應該是不會到跳出 while 迴圈的狀況，總之如果 pq.empty(全部 state 都搜尋完了)就跳出迴圈，return -1 表示找不到，進入迴圈，先儲存現在 pq 的 size 作為迴圈次數，接著進入 for 迴圈把 parent node 從 pq pop 出來，接著判斷參數，如果是 goal state 的話，就輸出深度(parent.depth)以及 record 所儲存的參數，接著如果 parent node 的 state 不為 goal state，就尋找 parent state ‘0’所在的位置，也就是空格的位置，接著使用 for (int k : moves[dir])去移動 0 可以移動到那些位置，假設 0 的位置在 4，那會產生 0 移向{1, 3, 5, 7}，這些陣列的位置，也就是多生成了 4 個 child，在每次生成 child，就去計算出每個 child 的 heuristic 數值，計算完就存入 child.h_value，接著判斷這項 child 有無紀錄過，如果沒有記錄過，就把 child 和 parent 的深度加 1，push 進 pq 進行排列優先順序判斷，然後順便把沒有記錄過的盤面 insert 進 record，紀錄盤面，接著就重複迴圈直到找到 goal state，現在在打這份報告時在看到我對每次計算 heuristic 數值，都去建一次 goal state 所在的 row col，可是 goal state 不會變，所以只要在 global 的地方先建好，在讓 function 使用就好了。

四、使用 A star search 解決 n-puzzle 問題

我先介紹一下 A*.cpp 檔案中所寫的 function。

1. Struct value(v)，定義一個物件，用來儲存盤面，heuristic 數值，以及深度，heuristic 和深度都先初始為 0。
2. comp，藉由判斷每個 node 的 heuristic 數值(這裡 heuristic 數值儲存在物件.h_value 的位置)，用來排列 priority_queue 大小，作為優先搜索順序。
3. solvable，透過判斷輸入的 start 和 goal 的 inversion 次數，如果皆為奇數和偶數，那盤面有解，如果為一奇數一偶數，盤面無解。
4. heuristic_function，輸入 cur.state 以及 goal state，先用兩個 for 迴圈儲存 goal state 的每個數字的 row 和 col 值，接著用曼哈頓距離計算

cur.state 各個數字，與 goal state 各個數字相差的距離相加起來，就是這個 state 的 heuristic 值。

5. A_star，輸入 start 和 goal，用 A* search 來尋找解的深度以及 saved_state。

那我就先從 main()開始解釋八，我在盤面的輸入是先宣告一個物件，皆著在輸入物件.state 儲存初始盤面的數值，輸出是以字串表示，先判斷是否有解，如果有解就開始搜尋，沒有解就輸出”Not found”，有解就跳到 A_star function 開始搜索，跳入 A_star function 時先宣告一些參數，moves，用來判斷 0 的位置在下一步可以往哪個二為陣列(0~8)的方向走，pq，這裡使用 v 宣告，用來儲存 parent node 所展開的 child，並排列下一次的優先搜索，record，用來儲存已經走過的盤面，需要先把 start 的盤面儲存進去，宣告完後，直接進入 while 迴圈，這裡因為有 main()中判斷有無解，應該是不會到跳出 while 迴圈的狀況，總之如果 pq.empty(全部 state 都搜尋完了)就跳出迴圈，return -1 表示找不到，進入迴圈，先儲存現在 pq 的 size 作為迴圈次數，接著進入 for 迴圈把 parent node 從 pq pop 出來，接著判斷參數，如果是 goal state 的話，就輸出深度(parent.depth)以及 record 所儲存的參數，接著如果 parent node 的 state 不為 goal state，就尋找 parent state ‘0’所在的位置，也就是空格的位置，接著使用 for (int k : moves[dir])去移動 0 可以移動到那些位置，假設 0 的位置在 4，那會產生 0 移向{1, 3, 5, 7}，這些陣列的位置，也就是多生成了 4 個 child，在每次生成 child，就去計算出每個 child 的 heuristic 數值，計算完就存入 child.h_value，接著判斷這項 child 有無紀錄過，如果沒有記錄過，就把 child 和 parent 的深度加 1，接著再把 child.depth 的部分加入 child.heuristic(f = g + h)，push 進 pq 進行排列優先順序判斷，然後順便把沒有記錄過的盤面 insert 進 record，紀錄盤面，接著就重複迴圈直到找到 goal state。

五、使用 Recursive best first search 解決 n-puzzle 問題

我先介紹一下 A*.cpp 檔案中所寫的 function。

1. Struct value(v)，定義一個物件，用來儲存盤面，heuristic 數值，以及深度，heuristic 和深度都先初始為 0。
2. comp，藉由判斷每個 node 的 heuristic 數值(這裡 heuristic 數值儲存在物件.h_value 的位置)，用來排列 priority_queue 大小，作為優先搜

索順序。

3. solvable，透過判斷輸入的 start 和 goal 的 inversion 次數，如果皆為奇數和偶數，那盤面有解，如果為一奇數一偶數，盤面無解。
4. heuristic_function，輸入 cur.state 以及 goal state，先用兩個 for 迴圈儲存 goal state 的每個數字的 row 和 col 值，接著用曼哈頓距離計算 cur.state 各個數字，與 goal state 各個數字相差的距離相加起來，就是這個 state 的 heuristic 值。
5. RBFS，輸入 start 和 goal，用 Recursive Best first search 來尋找解的深度以及 saved_state。

那我先從 global 宣告的變數解釋，在程式第 59 行和第 60 行，我把 goal 和 record 宣告在這裡，goal 是因為不用再讓 function 多一項引述，而 record 宣告在這裡是因為，RBFS 的 function 有 recursive 的特性，所以他會一直重複呼叫 function，如果我跟前幾項程式宣告在 RBFS 裡面，record 所儲存的 state 有些無法存到，所以我直接宣告在外面，那 main()的部分還是一樣，先判斷是否有解，有解就進行 search，沒解就輸出”Not found”，由於在講義上的 pseudocode 在 function 是回傳兩項數值，所以我就用 pair 宣告 function 來回傳兩項值<有無解，深度>，皆這就來講一下 RBFS function 在做甚麼，一開始也是一樣先宣告一些參數，不過這些參數就跟 A* 宣告的一模一樣，所以我就不再做說明，接著就先判斷是否為 goal state，如果是就回傳 true 和深度，如果不是就對 parent 進行 expand，在 expand 的部分 node.h_value 的計算就跟 A star 一樣，不一樣的點在紀錄是否要數入 pq 來在 search 的部分，為了要跟 parent 的 h_value 比，我除了判斷有沒有經歷過 state 以外，還有去判斷 parent.h_value 有沒有大於 child.h_value，如果有的話就去把 parent.h_value 輸入進 child.h_value，接著 expand 完所有的 child，就進入 recursive search best 的 while 迴圈，在一開始我照著講義上條件改成 true，不過每次跑出來都會跳出，segment fault，不過改成判斷 pq.empty，就正常了，這邊我比較不太了解，總之進入迴圈後，開始判斷 best.h_value 有沒有大於 flimit，如果有就回傳{false, best.h_value}，現在不為最佳解的意思，而接下來宣告 alternative，也就是判斷下一個的 h_value 跟 flimit 相比誰比較小，做 recursive search，直到找到結果。

至於 15-puzzle 的部分，因為我不太了解對於非順序盤面的 puzzle 要如何去判

斷有沒有解，所以我在 solvable 的部分就沒有下去判斷，不過由於是直接更改 8-puzzle move 的部分，所以大部分的原理其實都差不多，沒有太大的更動，至於 goal state 和 start state 的部分我給的狀態會相近一些，因為 15-puzzle 這狀態差太多即使跑下去也要花一些時間，至於說我在 state 的表示上超過 9 的數字，我會用 abcdef 去表示，因為我在輸入及輸出的部分是用 string 來宣告的。