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

1. 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。

1. 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。