A1. Search algorithms tests design

To find the pros and cons of different search algorithms and strategy when applying on the 8-puzzle, we designed a sequence of test cases to acquire related information and performance data. The normal goal would be complex enough to test, so the spiral goal is not considered. The test is divided into three levels including simple, moderate, and difficult, which correspond to 4 steps, 18 steps, and 31 steps (the most complex situation) achieving the goal. There are 10 combinations of search methods and options, and 6 kinds of results in each level. The following sample code indicates corresponding 3 initial layouts and the part of test code implementing to the moderate situation.

```
# simple                  # moderate               # difficult
LAYOUT_1 = [ [1,2,3],     LAYOUT_2 = [ [1,4,2],     LAYOUT_3 = [ [8,6,7],
             [5,0,6],                  [6,3,5],                  [2,5,4],
             [4,7,8] ]                 [0,7,8] ]                 [3,0,1] ]
EP1 = EightPuzzle( LAYOUT_1, NORMAL_GOAL )
EP2 = EightPuzzle( LAYOUT_2, NORMAL_GOAL )
EP3 = EightPuzzle( LAYOUT_3, NORMAL_GOAL )

T2_1=search(EP2, 'BF/FIFO', 40000, loop_check=True)
T2_2=search(EP2, 'BF/FIFO', 1400000)
T2_3=search(EP2, 'DF/LIFO', 80000, loop_check=True)
T2_4=search(EP2, 'DF/LIFO', 120000, loop_check=True, randomise=True)
T2_5=search(EP2, 'DF/LIFO', 100000)
T2_6=search(EP2, 'DF/LIFO', 100000, randomise=True)
T2_7=search(EP2, 'BF/FIFO', 300, heuristic=bb_misplaced_tiles,
loop_check=True)
T2_8=search(EP2, 'BF/FIFO', 200, heuristic=bb_manhattan, loop_check=True)
# cost = len(path)
T2_9=search(EP2, 'BF/FIFO', 3000, cost=cost, heuristic=bb_misplaced_tiles,
loop_check=True)
T2_10=search(EP2, 'BF/FIFO', 500, cost=cost, heuristic=bb_manhattan,
loop_check=True)

TEST_RESULTS ={'BFS_LT': T2_1,       # BFS loop true
               'BFS_LF': T2_2,       # BFS loop false
               'DFS_LT_RF': T2_3,    # DFS loop true  & random false
               'DFS_LT_RT': T2_4,    # DFS loop true  & random true
               'DFS_LF_RF': T2_5,    # DFS loop false & random flase
               'DFS_LF_RT': T2_6,    # DFS loop true  & random true
               'h1': T2_7,           # misplaced_tiles      loop true & fix
               'h2': T2_8,           # manhattan            loop true & fix
               'h1A*': T2_9,         # misplaced_tiles  A*  loop true & fix
               'h2A*': T2_10}        # manhattan        A*  loop true & fix
```

A2. Result tables

Simple test

Test	#max	Result	#gen	#inQ	Time(s)	cost
BFS_LT:	1000	Y	67	19	0.0	4
BFS_LF:	1000	Y	207	134	0.01	4
DFS_LT_RF:	1000	Y	184	56	0.01	58
DFS_LT_RT:	100000	!	171672	38274	94.09	!
DFS_LF_RF:	60000	!	60001	35999	71.11	!
DFS_LF_RT:	100000	!	100001	64659	125.99	!
h1:	100	Y	13	5	0.01	4
h2:	100	Y	13	5	0.0	4
h1A*:	100	Y	13	5	0.0	4
h2A*:	100	Y	13	5	0.0	4

LT: loop check true LF: loop check false RT: randomize true RF: randomize false

h1: misplaced tile h2: manhattan

Moderate test

Test	#max	Result	#gen	#inQ	Time(s)	cost
BFS_LT:	40000	Y	70191	10814	56.65	18
BFS_LF:	1400000	!	1400001	908366	126.7	!
DFS_LT_RF:	80000	Y	122275	32712	80.31	37836
DFS_LT_RT:	120000	!	213597	42531	192.76	!
DFS_LF_RF:	100000	!	100001	59999	176.15	!
DFS_LF_RT:	100000	!	100001	64695	199.71	!
h1:	300	Y	396	98	0.58	34
h2:	200	Y	253	61	0.02	34
h1A*:	3000	Y	4045	936	0.18	18
h2A*:	500	Y	819	181	0.03	18

Difficult test

Test	#max	Result	#gen	#inQ	Times	cost
BFS_LT:	180000	!	469739	4087	33.05	!
BFS_LF:	2000000	!	2000001	1287448	249.45	!
DFS_LT_RF:	14000	Y	22530	6169	2.62	7031
DFS_LT_RT:	80000	!	133721	32181	56.27	!
DFS_LF_RF:	50000	!	50001	29999	24.78	!
DFS_LF_RT:	50000	Y	35868	23178	47.69	12689
h1:	1000	Y	923	220	0.11	63
h2:	200	Y	279	65	0.07	47
h1A*:	200000	Y	383506	18328	23.41	31
h2A*:	30000	Y	48694	8520	2.28	31

A3. Result key observations

- **About loop checking:** For the same search algorism, set the loop check option as true would extremely reduce the time of search. It seems like the nodes waiting for checking in inQ is little when loop checking is on. That could make loop checking helpful when using any searching algorism. According to this conclusion, the loop check verification in heuristic and question B was always launched.
- **About randomize:** For DFS, randomization is not very useful. It seems that the sample gives results very unstably (can't be specified in the result diagram) when using randomized DFS. Sometimes, it can give you the answer you want quickly and precisely. But more often than not, it will consume a lot of computing time and eventually fail to give you an answer or give you a very costly answer. It is conjectured that sub-node randomization greatly reduces the chance of DFS encountering the correct path, resulting in long search time and low accuracy.
- **About BFS:** Without using heuristic, loop checked BFS is more likely to find the best path using little time when the situation is not so complicate. However, BFS could hardly find the path when situation is so complex. Furthermore, BFS will generate and check more nodes at an alarming rate, as system complexity increases. It is conjectured that the increase in space complexity increases the number of nodes that BFS needs to check at each layer. This ultimately leads to a large increase in the amount of RAM space required, making it difficult to calculate the results.
- **About Greedy:** When the system is complicate enough, the Best-First algorism would be extremely fast comparing with others. Although it can give solution of problem, the accuracy is poor (high cost) compared with A*. It is convinced that, the more informative, the more accurate for informed algorisms (Hou, 2022). Each choice of the Greedy Best-First is the best in partial choices. However, these local optimal solutions together do not necessarily lead to the overall optimal solution (Hou, 2022).
- **About A*:** A* responds quickly to the environment and has more information than greedy. It uses heuristic information to find the overall optimal path (Hou, 2022). It is obvious in the table of QA2 that when system complexity is low, A* always gives the exact path quickly. However, A* has poor real-time performance, with high computational effort and long computing time per node. So, as the number of nodes increases, the algorithm search efficiency decreases (djlzq, 2019).
- **For heuristic comparison:** the effect of the Manhattan distance far outweighs the number of incorrect digits, because the Manhattan distance takes into account one more factor - distance than just the misplaced digits. And the heuristic function with higher quality has a significant influence on both reducing the algorithm time and the space complexity.

A4. Define heuristic functions

These two functions are defined in EightPuzzle class to call self.goal_layout. Users can conveniently call them as defult misplaced_tiles and manhattan.

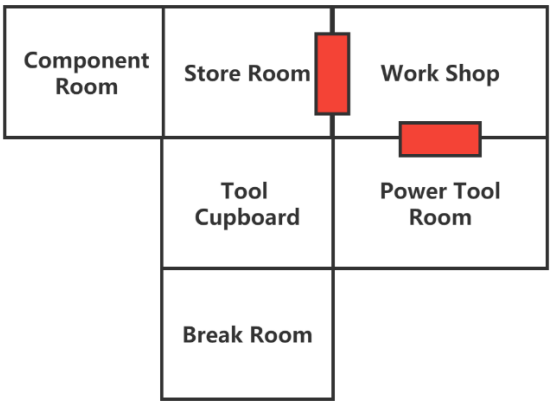
```
class EightPuzzle( SearchProblem ):

    ......

    def my_misplaced_tiles(self, state):
        count = 0
        for i in range(3):
            for j in range(0, 3):
              if state[1][i][j] != self.goal_layout[i][j]:
                count += 1
        return count

    def my_manhattan(self, state):
        count = 0
        for i in range(3):
            for j in range(3):
                val = state[1][i][j]
                x, y = number_position_in_layout(val, self.goal_layout)
                count += abs(x - i)
                count += abs(y - j)
        return count
```

- **my_misplaced_tiles:** Count the number of misplaced tiles in the 8-puzzle as return value of heuristic.
- **my_manhattan:** Return the sum of manhattan distance between each tile and its goal place.

B1. Scenario

Layout

There are six rooms in this house: the component room, the store room, the tool cupboard, the power tool room, the workshop and the break room, all of which are located in the following image, each of which is the same size of a square. All adjoining rooms have doors, the red squares refer to locks and specific keys are required to get through such doors, the black key for the door between the storage room and the workshop, the white key for the door between the power tool room and the workshop.



Room content

- Component room: bolt, gear, spring, white key
- Storage room: bucket, suitcase, toolbox
- Tool room: sledge hammer, anvil, saw, screwdriver
- Power tool room: drill, cutter, polisher
- Workshop: helmet, glasses
- Break room: water, snacks, tissues, black key

Action

The robot's action can be to pick up something, put something down, or go through an unlocked door to another room; if the robot has a key to the door, it can use the key to open the door and lock it immediately after going through it

Goal

The goal is to put some specific items in a specific room

B2. Heuristic

Disclaimer: The length of the path is used when calculating the cost

Heuristic function 1

Number of incorrectly placed items. This heuristic function is admissible because in the optimal case, the position of the incorrectly placed item differs from the target position by only one unlocked door, then the cost is the number of incorrectly placed items * 2 (pick up and put down) and in the rest of the cases, more doors need to be gone through.

Heuristic function 2

Max(m1+n1, m2+n2, …). First calculate the Manhattan distance m1,m2,m3,...,for the robot to the location of each incorrectly placed item, and afterwards calculate the Manhattan distance n1,n2,n3,...,for the location of the incorrectly placed item to the target location, summing the two distances for each item, m1+n1,m2+n2,m3+n3,...,taking the maximum of these terms max, and then max+1(put down). This heuristic function is admissible, because in the optimal case, the robot only needs to put down the goal item it carries, which is one. In other cases, there may be many goal items in different locations, in which robot will pick up the item one by one and put them in the goal location. More complexly, robot may take repetitive path because of the hunting of the key.

There are a few special scenarios to note in implementing this heuristic function. When the item is already at the target location, the two Manhattan distances do not need to be calculated; when the item is being held by the robot, the item cannot be detected in the room, and the heuristic function in this case only needs to return the distance from the robot to the target location.

B3. Investigation

In the first question we already found that it is much more efficient when loop check is true than when it is false, so in this survey of the algorithm we have ignored the case where no loop check is performed.

In the design of the test cases, we also used three problems ranging from simple to complex. In the simplest problem, we had the robot pick up two items at the very beginning position and then move two unlocked rooms. In the medium difficulty, the robot needs to place an item that is in another room into a locked workroom and, in addition to picking up the target item, there is also the task of finding the key. In the most difficult task, there are many forks in the robot's path between the item location and the target location, which greatly increases the complexity of the problem, so the best solution can only be given when using heuristic functions in best first and A* algorithm.

Simple problem

Test	#max	Result	#gen	#inQ	Times	cost
BFS_LT:	50000	Y	22124	9988	7.45	6
DFS_LT_RF:	100000	Y	7710	5362	3.05	877
DFS_LT_RT:	100000	Y	8749	5903	4.0	1413
BEST_1ST:	100	Y	71	43	0.04	6
A*:	2000	Y	1181	500	0.44	6

Moderate test

Test	#max	Result	#gen	#inQ	Times	cost
BFS_LT:	800000	Y	421043	147774	139.68	8
DFS_LT_RF:	200000	Y	47093	33316	35.36	6176
DFS_LT_RT:	5000	Y	3850	2546	1.33	616
BEST_1ST:	5000	Y	1396	395	0.53	10
A*:	50000	Y	5481	2899	7.58	8

Difficult test

Test	#max	Result	#gen	#inQ	Times	cost
BFS_LT:	400000	!	785660	267858	267.55	!
DFS_LT_RF:	100000	Y	37767	26102	19.15	5236
DFS_LT_RT:	100000	Y	3888	2665	1.58	612
BEST_1ST:	20000	Y	42480	10179	16.13	14
A*:	40000	Y	63731	23585	31.83	10

B4. Findings

- Compared to the previous problem, this one is more complex, so whether it is a simple, medium or difficult problem, the number of nodes expanded in the tree and the execution time of different algorithms are greatly increased, and the efficiency of the algorithm is extremely important at this point, with huge differences in performance between different algorithms
- The design of heuristic functions can refer to other problems, for example, for this problem we refer to the 8-puzzle heuristic function, which is the number of misplaced displaced items and the Manhattan distance, respectively, and similar to the performance of two heuristic functions in the 8-puzzle, the second heuristic function is more effective
- When designing heuristic functions, it is important to get as close to the actual cost as is admissible in order to improve the quality of the heuristic function. For example, the first heuristic is only a very rough estimate of the cost, while the second heuristic uses the distance variable for estimation, which is closer to the actual cost
- In problems of high complexity like this scenario, heuristics are particularly important, and in some cases the only way to find the answer to a problem is to rely on the heuristic function

Reference list

djlzq. 2019. *A* algorithm*. [Online]. [Accessed 14 March 2022]. Available from: https://zhuanlan.zhihu.com/p/61633289 .

Hou, J. 2022. Aritificial Intelligence. XJCO2611. 07/08/2022, SWJTU-LEEDS Joint School