Report of Homework 1

Description of how I implemented the four algorithms:

Twists I use to make my algorithms run faster or reduce themeory requirements:

When expanding successors for each node, I will not take the opposite action of my last move, because that will move back to the last configuration. The implementation is:

```
not problem.getInverseAction(direction) == actions[-1]
```

1. A*

Use a priority queue to store the (nodes, actions) in open list, and the priority is the f value of each node.

While the open list is not empty, pop the node with the lowest f value and decide if it is a goal. If it is the goal, return actions, if not, insert the succssors of this node into open list with priority and continue the loop of poping the node with highest priority.

2. IDA*

Use recursion to search. Update bound for each recursion. First get the heuristic value of start node as the bound. From start node, run the search function.

In the search function, first get the f value of the node, and decide if the f value is larger than bound. If it is, then return the f value of this node. If not, then expand the node and get the successors of this node. For each successor, still run the search function.

So for each recursion, the algorithm expands all the nodes that have a f value lower than the bound, and get the minimum f value that is larger than the bound as the bound for next iteration.

3. AWA*

Assign a f'value as the sum of g value and a weight times heuristic for each node and use this value as the priority of each node. Because heuristic is not consistent, through the iteration, for a configuration, it is possible to find a path costing less from start node to it than before. So, the algorithm moves the configuration to open and re-expand it.

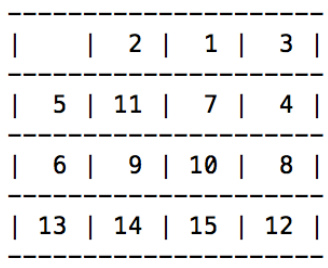
The algorithm uses the f value as the bound of the optimal solution cost to prune the search space and coverge to optimal solution. Search will still continue after finding a solution in order to find the optimal solution.

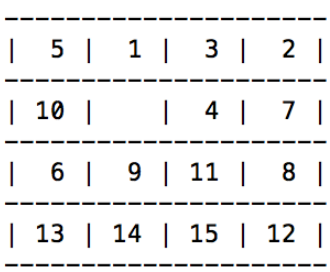
4. ARA*

The algorithm is anytime search based on weighted A*. It first uses a relatively larger weight and through interation, reduces the weight up till 1. The algorithm reuses the computation from last iteration so that saves time and computation. Within limit time, it will find a suboptimal solution very quickly and finally find the optimal solution.

My approach to running experiments and collecting the data:

For the four algorithms, I use five configurations.
Two provided on the blackboard, one provided on Piazza, and the other two are randomly generated by moving 100 steps from the goal state, which are given as:





How to run:

Run **sixteenPuzzle.py** and choose the No. for configuration, heuristic function and search algorithm.

How to collect data:

1. Time

The timer will start counting as soon as you enter the number of search algorithm and the timer stops when it finds the optimal solution. The except is ARA*, which counts the time when the first, second, …, optimal solution has been found.

2. Space

For the four algorithms, whenever they put one node into open list, I will plus 1 to the number of visited nodes, and whenever they put one node into closed list, I will plus 1 to the number of expanded nodes.
For the max number of open plus closed list, I will judge at each iteration that if the length of open list plus the length of closed list is greater than the current max number of open plus closed list, if yes, then update the max number, if not, then do nothing.
For AWA* and ARA* algorithm, whenever an already expanded node has been visited once more and the new cost of this node is smaller than the old cost, then plus 1 to the number of moved nodes. For ARA*, if the node has not been expanded before but has been visited, then if the new cost is greater than the old cost, it will be moved to INCONS list and plus 1 to the number of INCONS.

Experimental runs:

I use ‘blank’ as the moving tile and return the moves of this ‘blank’ tile.
All four algorithms can obtain an optimal goal. The comparison of the four algorithms’ running time and memory requirements is listed in the table.
For AWA*, I assign w = 1.3 and for ARA*, I assign w = 1.6 at first and decrease w by 0.2 for each iteration.
Here, I denote the heuristic No. as 1 and 2, which represents for No. of misplaced tiles and total No. of manhanttan distance, respectively.

1. Configuration 1

```
--------------------
|  9 |  5 |  7 |  4 |
--------------------
|  1 |    |  3 |  8 |
--------------------
| 13 | 10 |  2 | 12 |
--------------------
| 14 |  6 | 11 | 15 |
--------------------
```

All the four algorithms found the optimal solution of 22 moves of the ‘blank’ tile: ['left', 'up', 'right', 'down', 'right', 'down', 'left', 'down', 'left', 'up', 'up', 'up', 'right', 'down', 'right', 'up', 'left', 'down', 'down', 'right', 'down', 'right']

	Average running time/s		No. of nodes visited		No. of nodes expanded		No. of nodes moved		Max size of open+closed list	
Heuristic No.	1	2	1	2	1	2	1	2	1	2
A*	400.86	0.56	21864	906	10711	438	0	0	21551	901
IDA*	84.53	0.91	2747333	8437	880667	2745	0	0	2747333	8437
AWA*	30.69	0.29	8932	593	4448	286	0	0	8914	593
ARA*	144.43	0.64	10257	900	5074	424	0	1	9657	900

For ARA*

Heuristic No.	First solution		Second solution	
Heuristic 1	time/s	68.49	time/s	144.43
	w’	1.1	w’	1.0
	moves	22	moves	22
Heuristic 2	time/s	0.64		
	w’	1.0		
	moves	22		

2. Configuration 2

```
--------------------
|  3 |  6 |  9 |  4 |
--------------------
|  5 |  2 |  8 | 11 |
--------------------
| 10 |    | 15 |  7 |
--------------------
| 13 |  1 | 14 | 12 |
--------------------
```

All the four algorithms found the optimal solution of 29 moves of the 'blank' tile: ['down', 'right', 'up', 'right', 'up', 'left', 'left', 'down', 'left', 'up', 'right', 'up', 'left', 'down', 'right', 'right', 'up', 'left', 'down', 'right', 'down', 'left', 'up', 'left', 'down', 'right', 'right', 'right', 'down']

Because for this configuration, heuristic 1 takes too much time (hours), so I only give the results of heuristic 2.

	Running time/s	No. of nodes visited	No. of nodes expanded	No. of nodes moved	Max size of open+closed list
A*	50.69	8163	4044	0	8069
IDA*	180.0	1618040	529715	0	1618040
AWA*	0.94	1394	1208	0	1183
ARA*	4.82	2538	1311	0	2198

For ARA*

Heuristic No.	First solution		Second solution	
Heuristic 2	time/s	0.14	time/s	4.82
	w'	1.260870	w'	1.0
	moves	29	moves	29

3. Configuration 3

```
--------------------
|  5 |  3 |    |  4 |
--------------------
|  7 |  2 |  6 |  8 |
--------------------
|  1 |  9 | 10 | 11 |
--------------------
| 13 | 14 | 15 | 12 |
--------------------
```

All the four algorithms found the optimal solution of 22 moves of the 'blank' tile: ['left', 'down', 'right', 'up', 'left', 'left', 'down', 'down', 'right', 'up', 'up', 'left', 'down', 'right', 'up', 'right', 'down', 'left', 'down', 'right', 'right', 'down']

	Average running time/s		No. of nodes visited		No. of nodes expanded		No. of nodes moved		Max size of open+closed list	
Heuristic No.	1	2	1	2	1	2	1	2	1	2
A*	1163	16.49	36901	4740	17984	2304	0	0	36312	4702
IDA*	641.99	264.77	20593833	2259475	6577953	727529	0	0	20593833	2259475
AWA*	173.29	2.14	19400	2182	9666	1122	0	0	19389	2152
ARA*	388.57	6.10	16647	2339	8023	1159	1	1	15620	1922

For ARA*

Heuristic No.	First solution		Second solution	
Heuristic 1	time/s	191.58	time/s	388.57
	w'	1.1	w'	1.0
	moves	22	moves	22
Heuristic 2	time/s	2.60	time/s	6.10
	w'	1.1	w'	1.0
	moves	22	moves	22

4. Configuration 4

```
--------------------
|    |  2 |  1 |  3 |
--------------------
|  5 | 11 |  7 |  4 |
--------------------
|  6 |  9 | 10 |  8 |
--------------------
| 13 | 14 | 15 | 12 |
--------------------
```

All the four algorithms found the optimal solution of 22 moves of the 'blank' tile: ['right', 'right', 'down', 'left', 'up', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'down', 'left', 'up', 'right', 'right', 'right', 'down', 'down', 'down']

	Average running time/s		No. of nodes visited		No. of nodes expanded		No. of nodes moved		Max size of open+closed list	
Heuristic No.	1	2	1	2	1	2	1	2	1	2
A*	469.3	6.62	23210	3180	11400	1568	0	0	22851	3162
IDA*	216.68	67.72	6683542	601815	2158043	203139	0	0	6683542	601815
AWA*	64.69	1.09	13090	1404	6516	694	0	0	13081	1393
ARA*	184.64	3.18	11628	1542	5712	742	5	3	11138	1286

For ARA*

Heuristic No.	First solution		Second solution	
Heuristic 1	time/s	92.75	time/s	184.64
	w'	1.29	w'	1.0
	moves	22	moves	22
Heuristic 2	time/s	1.18	time/s	3.18
	w'	1.22	w'	1.0
	moves	22	moves	22

5. Configuration 5

```
--------------------
|  5 |  1 |  3 |  2 |
--------------------
| 10 |    |  4 |  7 |
--------------------
|  6 |  9 | 11 |  8 |
--------------------
| 13 | 14 | 15 | 12 |
--------------------
```

All the four algorithms found the optimal solution of 22 moves of the 'blank' tile: ['left', 'down', 'right', 'up', 'right', 'up', 'right', 'down', 'left', 'left', 'left', 'up', 'right', 'right', 'down', 'right', 'up', 'left', 'down', 'right', 'down', 'down']

	Average running time/s		No. of nodes visited		No. of nodes expanded		No. of nodes moved		Max size of open+closed list	
Heuristic No.	1	2	1	2	1	2	1	2	1	2
A*	737.43	5.13	29854	2876	14712	1413	0	0	29363	2846
IDA*	363.42	44.86	11568158	410399	3840966	137650	0	0	11568158	410399
AWA*	29.73	1.56	9168	1784	4909	882	0	2	9087	1771
ARA*	66.42	5.49	8714	2211	4404	1080	0	4	6399	1941

For ARA*

Heuristic No.	First solution		Second solution	
Heuristic 1	time/s	13.31	time/s	66.42
	w'	1.157895	w'	1.0
	moves	22	moves	22
Heuristic 2	time/s	2.45	time/s	5.49
	w'	1.375	w'	1.0
	moves	22	moves	22

Comparison of algorithms' experimental runs:

A* is the fundametal algorithm that the other three use to expand.

Comparing to A*, IDA* is priority-based depth first search. Therefore, IDA*'s memory usage is less than A*. However, because it does not use dynamic search, it will expand the same nodes again and again. As shown in the experimental runs, IDA* visited and expanded far more nodes than A* and costs more time to find the optimal solution.

Comparing to A*, AWA* uses an inconsistent heuristic function which makes the configuration close to a goal more attractive. The algorithm will not stop until all the nodes in the open list have a greater or equal f value than the current solution's f value. AWA* is proved to be able to find the optimal solution. According to my experimental runs, AWA*'s average running time is far less than A* and sometimes performs better than ARA*. It visited and expanded nodes less than other three algorithms. It should also be noted that the choice of w value is important for the performance of AWA*. I tried w to be 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, and I found that w = 1.3 achieves the best average performance.

Comparing to A*, ARA* will always find a sub-optimal solution within a very short time. Because ARA* uses the result from every last iteration, i.e., put nodes in INCONS list into open list, it is memory and time sufficient. By experimental results, ARA* can always find a solution in a short time and an optimal solution in a relative short time comparing to A*, IDA*.

Comparing AWA* and ARA*, AWA* has less memory requirements and spends less time to find the optimal solution.