Assignment 1

Ming Min, Yinan Sun, Tianyu Cheng

Part 1. Heavy N-queens problem

1.1 Instruction

In this part, the programming language we use is Python 2.7. The program can take input of N value for the N-queens problem, and 1 for A*, 2 for greedy hill climbing.

First I'm gonna prove that the heuristic function is not admissible in this case.

Proof: Using counter example, let suppose an easy case in 4-Queens problem. Suppose we are in such a state, as show in table 1. The heuristic function value is 10 + 3 = 13. In order to reach our goal, we need to move Queen in the left most column one step down. Thus the real cost will be 10 + 1*1=11, which is small than the heuristic function value, thus it's not admissible.

	Q		
Q			Q
		Q	

table 1

	Q		
			Q
Q			
		Q	

goal

For hill climbing part, I use 50 restarts. Once it reaches the goal, the program stops and return the result. For A* search, it is done with iterative deepening in order to save memory space.

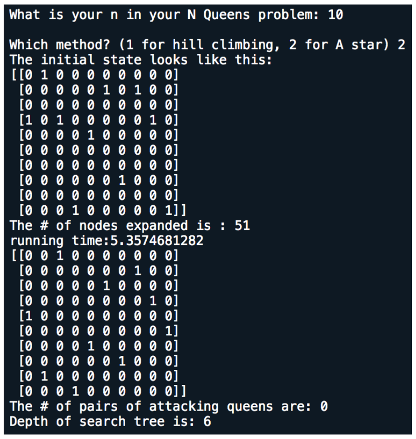
The program also output a sequence of moves. It may need some explanation. The moves is a sequence of number pairs [i, j]. Then meaning for [i, j] is at this step, you move the queen on j+1 column to i+1 row from previous state. For example, [[0, 7], [1, 4], [6, 0], [7, 3], [6, 6], [8, 0], [0, 2], [9, 7], [0, 2]], [0,7] means you need to move the queen on 8th column to 1st row, and still in 8th column. Then [1,4] means you need to move the queen on 5th column to 2nd row, and still in 5th column, etc.

1.2 Experiments

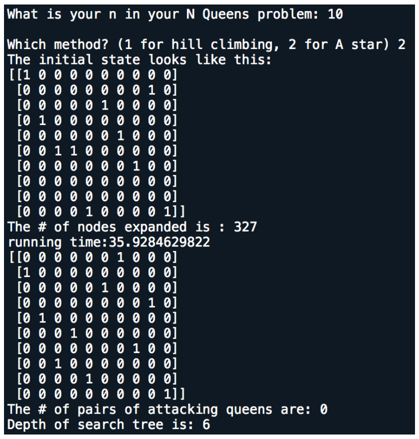
1.2.1 How large of a puzzle can be solved in 10 seconds?

The time for each method is different for the same puzzle size with different start state. Some start states are very easily to reach target, some are extremely worse. So in this part, I'll just give a approximate solution.

When using A* method, solving a 9-Queens puzzle is almost guaranteed in 10 seconds. But when N=10, sometime it can be solved, but sometime not.
Example:

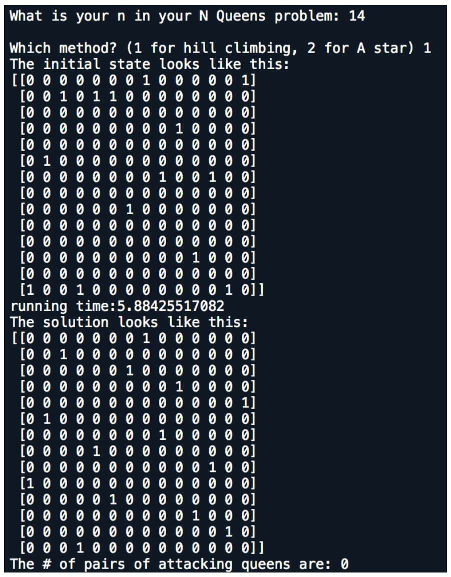


```
What is your n in your N Queens problem: 10

Which method? (1 for hill climbing, 2 for A star) 2
The initial state looks like this:
[[0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 1]]
The # of nodes expanded is : 51
running time:5.3574681282
[[0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]]
The # of pairs of attacking queens are: 0
Depth of search tree is: 6
```

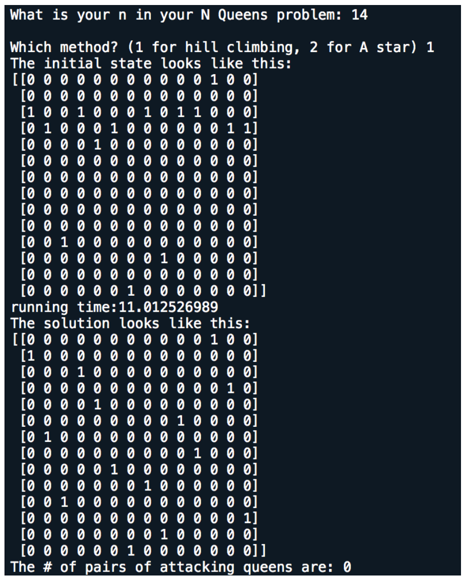


```
What is your n in your N Queens problem: 10

Which method? (1 for hill climbing, 2 for A star) 2
The initial state looks like this:
[[1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [0 0 1 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 1]]
The # of nodes expanded is : 327
running time:35.9284629822
[[0 0 0 0 0 0 1 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]]
The # of pairs of attacking queens are: 0
Depth of search tree is: 6
```

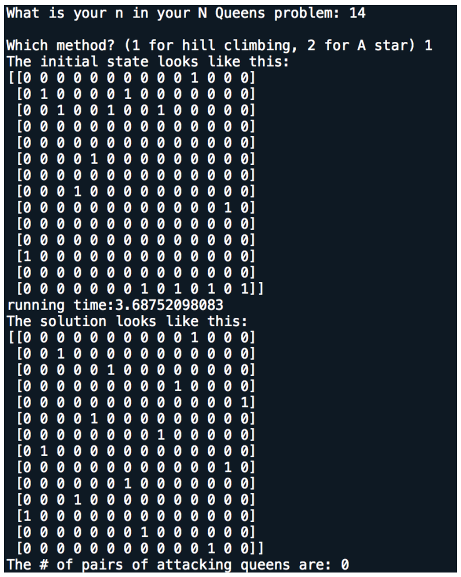
When using greedy hill climbing, it is almost guaranteed to be solved in 10 seconds for most case of 14-Queens puzzle.



```
What is your n in your N Queens problem: 14

Which method? (1 for hill climbing, 2 for A star) 1
The initial state looks like this:
[[0 0 0 0 0 0 0 1 0 0 0 0 0 1]
 [0 0 1 0 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 0 0 0 0 0 0 0 1 0]]
running time:5.88425517082
The solution looks like this:
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0]]
The # of pairs of attacking queens are: 0
```



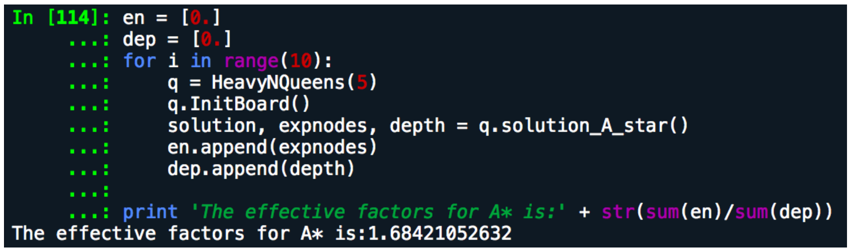
```
What is your n in your N Queens problem: 14

Which method? (1 for hill climbing, 2 for A star) 1
The initial state looks like this:
[[0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 0 0 1 0 1 1 0 0 0]
 [0 1 0 0 0 1 0 0 0 0 0 0 1 1]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]]
running time:11.012526989
The solution looks like this:
[[0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]]
The # of pairs of attacking queens are: 0
```



```
What is your n in your N Queens problem: 14

Which method? (1 for hill climbing, 2 for A star) 1
The initial state looks like this:
[[0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 1 0 0 1 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 1 0 1 0 1]]
running time:3.68752098083
The solution looks like this:
[[0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0]]
The # of pairs of attacking queens are: 0
```

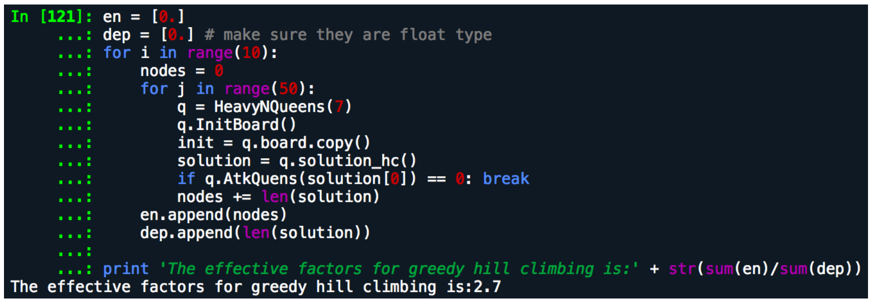
1.2.2 Effective factors

For A* method, I'll use N=5 to determine what's the effective branching factor. The code and result shows below.



```
In [114]: en = [0.]
     ...: dep = [0.]
     ...: for i in range(10):
     ...:     q = HeavyNQueens(5)
     ...:     q.InitBoard()
     ...:     solution, expnodes, depth = q.solution_A_star()
     ...:     en.append(expnodes)
     ...:     dep.append(depth)
     ...:
     ...: print 'The effective factors for A* is:' + str(sum(en)/sum(dep))
The effective factors for A* is:1.68421052632
```

For greedy hill climbing, I'll use N=7 to determine what's the effective branching factor. The code and result shows below.



```
In [121]: en = [0.]
     ...: dep = [0.] # make sure they are float type
     ...: for i in range(10):
     ...:     nodes = 0
     ...:     for j in range(50):
     ...:         q = HeavyNQueens(7)
     ...:         q.InitBoard()
     ...:         init = q.board.copy()
     ...:         solution = q.solution_hc()
     ...:         if q.AtkQuens(solution[0]) == 0: break
     ...:         nodes += len(solution)
     ...:     en.append(nodes)
     ...:     dep.append(len(solution))
     ...:
     ...: print 'The effective factors for greedy hill climbing is:' + str(sum(en)/sum(dep))
The effective factors for greedy hill climbing is:2.7
```

1.2.3 Which approach come up with cheaper solution?

We've done several experiments for this question. The result is shown below in the table,

N	**A* search**	**Greedy hill climbing**
4	13	48
5	31	80
6	949	120
7	1369	210
8	3228	336
9	12160	432

When N is getting larger, the cost A* search is greatly larger than the cost of greedy hill climbing (where we discard the fail climbings).

1.2.4 Which approach takes less time?

Except for small N (=4,5), the running time for greedy hill climbing with restarts is much faster than A*.

N	**A* search**	**Greedy hill climbing**
4	0.0097	0.019

N	A* search	Greedy hill climbing
5	0.036	0.026
6	0.314	0.228
7	1.187	0.198
8	4.569	0.54
9	2.57	0.876