CS 540: Introduction to Artificial Intelligence
Homework Assignment # 4

Assigned: 2/26
Due: 3/5 before class

Hand in your homework:

If a homework has programming questions, please hand in the Java program. If a homework has written questions, please hand in a PDF file. Regardless, please zip all your files into hwX.zip where X is the homework number. Go to UW Canvas, choose your CS540 course, choose Assignment, click on Homework X: this is where you submit your zip file.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:30 a.m., and it is handed in between Wednesday 9:30 a.m. and Thursday 9:30 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Eight Queens Puzzle [60 points]

This is a programming question. The solution to the programming problem should be coded in Java, and you are required to use only built-in libraries to complete this homework. Please submit a single zip file named hw4.zip, which should contain a source code file named EightQueen.java with no package statements, and make sure your program is runnable from command line on a department Linux machine. We provide a skeleton EightQueen.java code that you can optionally use, or you can write your own.

In this question you will implement heuristic search algorithms – hill climbing varieties and simulated annealing – for the eight queens puzzle.

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard in such a way that no two queens attack each other. In other words, a solution requires that no two queens share the same row, column, or diagonal. For more background on the eight queens puzzle see `https://en.wikipedia.org/wiki/Eight_queens_puzzle`.

Write a program **EightQueen.java** with the following command line format, where the command line input has variable length[1]:

```
$java EightQueen FLAG [seed] board
```

where $FLAG$ is an integer that specifies the output of the program (see below). *seed* is a seed of a random number generator (only required in part c and e; discarded otherwise). *board* is a string of length 8 that specifies the board as an array where the index is the column and the value is the row. Each position in the string takes values in integer 0-7. To illustrate, the board below is represented as 13572064. Note that the board is a solution to the eight queens puzzle. That is, no two queens share the same row, column, or diagonal.

```
---------------------------------
|   |   |   |   |   | Q |   |   |
---------------------------------
| Q |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |   | Q |   |   |   |
---------------------------------
|   | Q |   |   |   |   |   |   |
---------------------------------
|   |   |   |   |   |   |   | Q |
---------------------------------
|   |   | Q |   |   |   |   |   |
---------------------------------
|   |   |   |   |   |   | Q |   |
---------------------------------
|   |   |   | Q |   |   |   |   |
---------------------------------
```

(Part a, 5 points) You will first implement a heuristic function of an eight queens board. In particular, the function returns the heuristic cost of a given board that is equal to the number of queen pairs sharing the same

[1] We provide code skeleton that already handles variable length input.

row, column, or diagonal. The board above, which is a solution, has a heuristic cost of 0, because there are 0 pairs of queens attacking each other.

When FLAG=100, print the heuristic cost of a given board.

```
$java EightQueen 100 01234567
28

$java EightQueen 100 40057263
1

$java EightQueen 100 13572064
0
```

(Part b, 10 points) To solve this eight queens puzzle, we need to take steps to reduce the heuristic cost to zero. To evaluate which moves are best, we will calculate the heuristic costs of the board after one move. For simplicity, we will only move queens up or down vertically along individual columns.

When FLAG=200, perform calculation of all the heuristic costs of the board after one move, find moves (i.e. successors) with the lowest heuristic cost (the steepest moves), and print corresponding resulting boards after those moves. You will print the heuristic cost in the last line. If no move exists, you should simply produce no output at all.

Important: We ask you to implement the following order among successors. Whenever you append moves into a list, append them according to moves from small index to large index and from small value to large value. That is, the list is sorted by new queen position's index and value in ascending order, with index having higher precedence than value. Recall that we are to represent the board as an array where the index is the column and the value is the row. This order will be used throughout the program, so that the output is well-defined. For example:

```
$java EightQueen 200 13572063
13572064
0

$java EightQueen 200 13572064 (note: produce no output)


$java EightQueen 200 01234567
11234567
31234567
51234567
71234567
00234567
02234567
04234567
06234567
01134567
```

```
01334567
01534567
01734567
01204567
01224567
01244567
01264567
01231567
01233567
01235567
01237567
01234067
01234267
01234467
01234667
01234517
01234537
01234557
01234577
01234560
01234562
01234564
01234566
22
```

(Part c, 15 points) Perform hill climbing (HC) algorithm by randomly choosing a steepest move from part b, print the corresponding board and its heuristic cost (space-separated) with a current iteration number (starting from 0 to XX) prefixed, then repeat the process for the number of iterations equal to XX. XX can be 00 to 99. For example, if FLAG=300, the cutoff is 0. In HC, you will do a goal test, but will NOT expand it. If FLAG=301, the cutoff is one. In HC, you will expand the initial state (i.e. put its moves into a stack in the order we specified in Part b). You will randomly select a move out, perform goal-check (and terminate the program if goal-check succeeds). But you will not expand any of these moves. If a solution is reached, the algorithm should stop and print the word "Solved".

When FLAG=3XX, command line argument *seed* is required. If $seed = -1$, you actually do not set the seed (this allows you to generate different random moves). Otherwise, you should set the seed to *seed*. To set the seed in Java, use the following code:

```
Random rng = new Random();
if (seed != -1) rng.setSeed(seed);
```

To randomly choose a steepest move, you should generate a new random integer $r \in [0, |A| - 1]$, where A is a set of all steepest moves and $|A|$ denotes the cardinality of A. The random integer is used as an index to get the element at a specified position in a list. This should be done with:

```
// successors is an ArrayList instance containing all steepest moves
int r = rng.nextInt(successors.size());
State move = successors().get(r);
```

Important: We ask you to implement the order among successors like in part b, so that when the same seed is set, the output is well-defined.

```
$java EightQueen 305 1 12375643
0:12375643 14
1:02375643 10
2:02175643 7
3:04175643 5
4:04175603 3
5:24175603 2

$java EightQueen 315 1 12375643
0:12375643 14
1:02375643 10
2:02175643 7
3:04175643 5
4:04175603 3
5:24175603 2
6:24175607 2
7:24175307 1
8:24175304 1
9:26175304 0
Solved

$java EightQueen 315 1 13572063
0:13572063 2
1:13572064 0
Solved

$java EightQueen 315 1 13572064
0:13572064 0
Solved

$java EightQueen 300 1 11111111
0:11111111 28

$java EightQueen 300 1 13572064
0:13572064 0
Solved
```

(Part d, 15 points) We will look at an additional way to solve eight queens puzzle. So far, we had to go through every move and check every possible square. Now, we will perform first-choice hill climbing that, instead of trying to find the best move at each step, we will simply try to find a better move. In particular, we will choose the first move with a heuristic cost that is lower than a current heuristic cost. Again, we iterate through elements in the list from first to last in order specified in part b and c.

When FLAG=4XX, perform a first-choice hill climbing algorithm step, print the board and its heuristic cost (space-separated), then repeat the process for the number of iterations equal to XX. XX can be 00 to 99. If a solution is reached, the algorithm should stop and print the word "Solved" in the last line. If the algorithm gets stuck on local optima, the algorithm should also terminate and print the word "Local optimum" in the last line. A current board is at local optima if all heuristic costs of the board after one move are less than or equal to the current heuristic cost.

```
$java EightQueen 405 12375643
0:12375643 14
1:02375643 10
2:00375643 8
3:20375643 7
4:20175643 5
5:20135643 4

$java EightQueen 415 12375643
0:12375643 14
1:02375643 10
2:00375643 8
3:20375643 7
4:20175643 5
5:20135643 4
6:27135643 3
7:27135043 2
8:27135040 1
Local optimum

$java EightQueen 415 13572063
0:13572063 2
1:13572061 1
2:13572064 0
Solved

$java EightQueen 415 13572064
0:13572064 0
Solved

$java EightQueen 400 11111111
0:11111111 28
```

```
$java EightQueen 400 13572064
0:13572064 0
Solved
```

(Part e, 15 points) Perform simulated annealing algorithm[2], print the board and its heuristic cost (space-separated), then repeat the process for the number of iterations equal to XX. XX can be 00 to 99. If a solution is reached, the algorithm should stop and print the word "Solved" in the last line. For this part, we set the anneal rate and initial temperature to be 0.95 and 100, respectively.

When FLAG=5XX, command line argument *seed* is required and setting *seed* can be done like in part c. Recall that if $seed = -1$, you actually do not set the seed (this allows you to generate different random moves). Otherwise, you should set the seed to *seed*.

Important: We ask you to use only one instance of *Random* with the same sequence of method calls for each iteration, so that it generates and returns identical sequences of numbers, and the output is well-defined. Formally, define the sequence of method calls as $(i_1, i_2, d_1, i_3, i_4, d_2, i_5, i_6, d_3, ...)$ where k is positive integers, d_k is the k$^{\text{th}}$ nextDouble() call for the k$^{\text{th}}$ iteration, and i_k is the k$^{\text{th}}$ nextInt() call. The new queen position picked at random at k$^{\text{th}}$ iteration is specified by i_{2k-1} as *index* and i_{2k} as *value*. This should be done with:

```
Random rng = new Random();
if (seed != -1) rng.setSeed(seed);

...

for each iteration:
    int index = rng.nextInt(7);
    int value = rng.nextInt(7);
    double prob = rng.nextDouble();

    ...
```

where *index* and *value* are used to specify a new queen position picked at random. *prob* is used to simulate the probability of acceptance.

Print results similar to part c. For example,

```
$java EightQueen 505 1 12375643
0:12375643 14
1:12374643 11
2:12374663 10
3:12364663 11
4:12364563 9
5:02364563 10
```

[2]Its pseudo-code is on slide 14 from "Advanced search part 2: simulated annealing".

```
$java EightQueen 515 1 12375643
0:12375643 14
1:12374643 11
2:12374663 10
3:12364663 11
4:12364563 9
5:02364563 10
6:02664563 11
7:02662563 8
8:02662263 7
9:02362263 7
10:02362263 7
11:02362263 7
12:02363263 8
13:02353263 7
14:02313263 7
15:02363263 8

$java EightQueen 515 1 13572063
0:13572063 2
1:13574063 3
2:13574063 3
3:13564063 4
4:13564563 8
5:03564563 11
6:03664563 12
7:03662563 8
8:03662263 6
9:03362263 6
10:03362263 6
11:03362263 6
12:03363263 9
13:03353263 9
14:03313263 9
15:03363263 9

$java EightQueen 515 1 13572064
0:13572064 0
Solved

$java EightQueen 500 1 11111111
0:11111111 28

$java EightQueen 500 1 13572064
0:13572064 0
```

```
Solved
```