CS 421 Algorithms (Summer 2021)

Programming Assignment #1, Due on 7/06/2021, Tuesday (11PM)

Introduction:

This assignment will ask you to write JAVA programs to solve the Knight's Tour problem using the `exhaustive search with backtracking` technique. The technique solves problems by searching all possible next steps. If the search cannot go further, it requires going back to the previous step. There are many well known algorithms using this technique such as the DFS search on a graph and the 8-queen problem.

Description:

The Knight Tour problem is described as follows. A chess board has n rows and n columns, where $n > 2$. From its current position, a knight's next position will be either two rows and one column or one row and two columns from the current position.

Given n and a starting position, the problem tries to find the sequence of moves for the knight to travel every position on the chess board exactly once. For example, one possible solution for the problem with $n = 8$ and starting position (0, 0) is

	0	1	2	3	4	5	6	7
0	1	38	55	34	3	36	19	22
1	54	47	2	37	20	23	4	17
2	39	56	33	46	35	18	21	10
3	48	53	40	57	24	11	16	5
4	59	32	45	52	41	26	9	12
5	44	49	58	25	62	15	6	27
6	31	60	51	42	29	8	13	64
7	50	43	30	61	14	63	28	7

Table 1: The integer i in each entry indicates the i-th move of the knight

I would suggest to have three java files for this problem, which are `KnightTour.java` (driver program), `KnightBoard.java` and `Position.java`.

Search Strategies:

This assignment will ask you to implement three different ways to search for a solution.

- Basic Search: Try next eligible move from a position A in a clockwise sequence as shown in Table 2.

  To improve the search time of the knight tour problem, we can apply a heuristic while performing the search. There are two potential search heuristics.

			8^{th}		1^{st}		
		7^{th}				2^{nd}	
				A			
		6^{th}				3^{rd}	
			5^{th}		4^{th}		

Table 2: The sequence of next eligible moves in a basic search

- Heuristic I: When there are multiple eligible next moves, try the one that is closer to the boarder of the chess board first. If there is a tie, pick the one based on the clockwise sequence. To measure how close a position A is to the boarder, a "distance" of the position A to the boarder can be calculated as (the smaller vertical distance from A to the horizontal boarders) + (the smaller horizontal distance from A to the vertical boarders). For example, in Table 2, the "distance" of the position A to the boarder is $3 + 3 = 6$.
- Heuristic II (Warnsdorff's heuristic): When there are multiple eligible next moves, try the one that has the fewest onward moves first. If there is a tie, pick the one based on the clockwise sequence.

What you need to do:

1. Implement three options to search the chess board as below:

   `java KnightTour <0/1/2 (no/heuristicI/heuristicII search)> <n> <x> <y>`

   where n is the size of the chess board, (x, y) is the starting position at x row and y column. The program should print the total number of moves tried to reach a solution or to reach a conclusion of no solution. You can find some sample results in

   `/home/JHyeh/cs421/labs/lab1/files/sample_output`

2. Run your program three times, one for each search option, with $n = 7, x = 1, y = 1$ and report the number of moves in each search in a README file. In addition, if there is a solution, please print the solution (as Table 1) for each search in the README file. Note that if a given problem instance has no solution, then the heuristic search cannot improve the search efficiency.

submission:

Submit your program from `onyx` by copying all of your java files to an empty directory (with no subdirectories) and typing the following FROM WITHIN this directory:

```
submit jhyeh cs421 p1
```