**Recursive Back Tracking – Peg Solitaire**

Use recursive backtracking to determine if there exists a series of legal moves from a starting configuration of a triangle peg solitaire game to a particular ending configuration.

Print the series of moves that leads to the solution, or a message stating that no solution exists.

Every starting and ending configuration will have at least one peg.

Notes

* This is *not* the game described on the board in the image provided.
* Legal moves are parallel to one of the sides of the equilateral triangle. You must "jump" a pin in a legal direction. The pin that was jumped, is removed from the board.
* You must develop the algorithm and solution yourself. This is not a test of your "Google skills".
* We will be using a board with six pegs per side.

The original and final configurations will be provided as text input, from the keyboard (copy & paste) in the following form. Determine if the final configuration can be reached from the initial configuration (and show the moves).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P  PP  PPP  POPP  PPPPP  PPPPPP  O  OO  OPO  OOOO  OOPPP  OOOOOO  *-- A capital P indicates a peg at that location. A capital O indicates the position is open.* | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 0,0 |  |  |  |  |  | | 1,0 | 1,1 |  |  |  |  | | 2,0 | 2,1 | 2,2 |  |  |  | | 3.0 | 3,1 | 3,2 | 3,3 |  |  | | 4,0 | 4,1 | 4,2 | 4,3 | 4,4 |  | | 5,0 | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 |   Use this numbering scheme for displaying moves. Legal directions for moves. |
| The peg at (3,2) could move to  (1,0) if (2,1) is occupied & (1,0) not  (3,0) if (3,1) is occupied & (3,0) not  (5,2) if (4,2) is occupied & (5,2) not  (5,4) if (4,3) is occupied & (5,4) not | The peg at (4,4) could move to   (2,2) if (3,3) is occupied & (2,2) not  (4,2) if (4,3) is occupied & (4,2) not |

**Output must be of the form:**

1. No solution exists, or
2. SRC: (1,1) DEST:(3,1) //including all the moves

**Obvious base cases:**

* The current configuration matches the “final configuration” that was read in.
* No legal move can be made.
* There are now fewer pins on the board than in the “final configuration” that was read in.

**Design:**

If I can solve the problem, print out the moves used

|  |
| --- |
| For each row  For each column  If there is a peg in this spot  If I can jump up and solve the problem return true;  If I can jump left and solve the problem return true;  If I can jump right and solve the problem return true;  If I can jump down-right and solve the problem return true;  If I can jump up-left and solve the problem return true;  If I can jump down and solve the problem return true;  Return false; // no solution |

**Hints:**

* For more on recursive backtracking: <https://en.wikipedia.org/wiki/Backtracking>
* See PowerPoint notes (ask).
* Make sure you try and record the test cases.
  + Your sequence of moves will only match mine only if you test for moves in the exact same order as me (or there is only one solution or we just get lucky).

**Submit:**

Your code and a completed test sheet on the Java drive.

Note that this program does NOT use file I/O, it uses keyboard input – though obviously copy/paste makes sense.

**Due:** Apr 12