CS 325

Homework #4: Triangular Peg Solitaire (100 pts)

In this assignment, your job is to implement a solution to Triangle Peg Solitaire or the “Triangle Peg Game” containing **15** holes. Specifically, Peg Solitaire works as follows:

- The game’s board is an equilateral triangle with 15 holes (5 holes on each side):

JUMP E H TEE AND MOVE IT
LEAVE IR OR M YOU'RE T PLAIN
"EG-NO- MOOSE"
RACKER ARREL D COUN Y STOR

- Initially, the game starts with pegs in *all* holes **except one**.

- The goal is to “jump” pegs *one at a time* until only **one peg** remains.
    - Each peg can jump over any immediately adjacent peg, but *only* if there is an open space where the jumping peg “lands”.
    - Each peg you jump over must be *removed*.
    - Pegs cannot jump diagonally.

- Visit http://www.mathsisfun.com/games/triangle-peg-solitaire/ to play an interactive version online (for free).

Your C++ source code must meet the following requirements:

1. Takes **two** *command line arguments*:
    a. *emptyPegLoc* – number between 1 and 15 specifying the initial empty hole
    b. *printBoard* – either “true” or “false”.
        - If **false**, only output the **path** that solves the game
        - If **true**: in addition to outputting the **path**, you must also output the triangular board after each “jump”. (**0** represents a hole, **1** reps a peg)

2. Must **error check** the command line arguments.

3. Must abide by the rules of Peg Solitaire (i.e., no illegal jumps).

4. Must output the path as a series of “A -> B” strings, indicating that a peg was jumped “from” location **A** “to” location **B.**

5. Peg locations must be as follows:

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

<u>**HINTS:**</u>

- This is a **difficult** and **non-trivial** assignment -> start early and work incrementally!
- Draw lots of pictures.
    - Given a hole and adjacent pegs, what are the possible moves?

- How can Triangular Peg Solitaire be represented as a graph?
- adjacency matrix or adjacency list?
- What are the nodes in the graph?
- Depth-first traversal or Breadth-first traversal?
- pass by reference
- custom helper function to print board
- reverse iterator
- helper function to locate hard-coded peg location numbers
- int board[5][5]
    - Watch out for 2D array boundaries!!
    - E.g., hole at position 1

0	1	1	1	1
1	1	1	1	-1
1	1	1	-1	-1
1	1	-1	-1	-1
1	-1	-1	-1	-1

E.g., hard-coded peg locations

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

EXAMPLES:

```
UNIX > ./hw4
usage: ./a.out emptyPegLocation printBoard
  where emptyPegLocation is between 1 and 15 (inclusive)
  and printBoard is either 'true' or 'false'
         1
       2   3
     4   5   6
   7   8   9  10
11  12  13  14  15

UNIX > ./hw4 1 false
  1  ->   1
  4  ->   1
  9  ->   2
 12  ->   5
 11  ->   4
  3  ->   8
 10  ->   3
  1  ->   6
  2  ->   7
  7  ->   9
 14  ->  12
  6  ->  13
 12  ->  14
 15  ->  13

UNIX> ./hw4 13 false
 13  ->  13
  6  ->  13
  1  ->   6
  4  ->   1
 10  ->   3
  1  ->   6
 13  ->   4
  7  ->   2
  2  ->   9
 15  ->  13
 12  ->  14
  6  ->  13
 14  ->  12
 11  ->  13
```

```
UNIX> ./hw4 1 true
  1  ->   1
         0
       1   1
     1   1   1
   1   1   1   1
 1   1   1   1   1
====================
  4  ->   1
         1
       0   1
     0   1   1
   1   1   1   1
 1   1   1   1   1
====================
  9  ->   2
         1
       1   1
     0   0   1
   1   1   0   1
 1   1   1   1   1
====================
 12  ->   5
         1
       1   1
     0   1   1
   1   0   0   1
 1   0   1   1   1
====================
 11  ->   4
         1
       1   1
     1   1   1
   0   0   0   1
 0   0   1   1   1
====================
  3  ->   8
         1
       1   0
     1   0   1
   0   1   0   1
 0   0   1   1   1
====================
 10  ->   3
         1
       1   1
     1   0   0
   0   1   0   0
 0   0   1   1   1
====================
  1  ->   6
         0
       1   0
     1   0   1
   0   1   0   0
 0   0   1   1   1
====================
 2  ->   7
         0
       0   0
     0   0   1
   1   1   0   0
 0   0   1   1   1
====================
  7  ->   9
         0
       0   0
     0   0   1
   0   0   1   0
 0   0   1   1   1
====================
 14  ->  12
         0
       0   0
     0   0   1
   0   0   1   0
 0   1   0   0   1
====================
  6  ->  13
         0
       0   0
     0   0   0
   0   0   0   0
 0   1   1   0   1
====================
 12  ->  14
         0
       0   0
     0   0   0
   0   0   0   0
 0   0   0   1   1
====================
 15  ->  13
         0
       0   0
     0   0   0
   0   0   0   0
 0   0   1   0   0
====================
```