August 26, 2019

Computer Science 282
Programming Assignment #1

Due: September 11, 2019

You are to write a program that will implement Warnsdorff's algorithm for finding a generalized knight's tour on a chessboard. This link https://en.wikipedia.org/wiki/Knight%27s_tour provides some information about knight tours and the algorithm I want you to use.

You must implement the two classes with the methods shown starting on the next page. Do not make any changes to the classes, variables, names, etc. Just add code. Also, no backtracking or recursion should be used. Part of the input data will include the legal moves that can be made by the generalized knight. For example, for a normal knight three of the eight legal moves are (1, 2), (1, –2), and (2, 1). The test data file test1.txt shows a typical input file for the case of a normal knight. In general you will need to input the size of the board, the starting square for the knight, and the legal moves for the knight. Though I will sometimes provide links to various files, all files related to this and all other programs reside in the Files area of Canvas.

There often comes a time when the algorithm provides a choice of equally good moves. In such a case you must choose the move that comes last in the input list of legal moves.

This assignment is as much about following instructions as it is about programming. Be sure your methods do exactly what is expected of them and all rules mentioned below and in class are followed.

What follows are general rules that you must follow for this and, as applicable, all other programs you write in this class. Failure to follow these rules will cost you points.

1. I will eventually post a final test program. A small sample test program is already posted to help get you started. You are to turn in a printout of your program and a printout of the output obtained from running the final test program. Don't wait for my final test program to appear to start debugging your program. You should begin testing with your own test programs. Also, the final test program I post may not be the same test program I run on your programs. You cannot assume your program is correct just because it works on the posted test program.

2. In addition to handing in a listing of your program and a printout of your output, you must also email me both of your java classes in a **single** file named prog1.java. Do not email me your output or my test program. Be sure that my main (test) program is in a different file from your two classes. Also, do not make your classes "public". The subject of your email should be "Alison Riske – prog 1 – 2:00" if your name is Alison Riske and you attend the 2:00 class. Change 2:00 to 3:30 if that is the time of your class. Your prog1.java file should be an **attachment**. If you deviate from any of these rules you will lose points because doing so makes it much more difficult for me to manage and test your programs. For example, in the email subject pay attention to what is upper case, what is lower case, the location of the hyphen, etc.

3. The top of your prog1.java file (and all java code that you turn in to me in the future) should begin with comments that show: (i) Your name; (ii) Comp 282 and clear mention of what time your class meets; (iii) The assignment number; (iv) The date the assignment was handed in (which is not necessarily the same as the due date); (v) A brief description of what is contained in the file.

4. For all Java code you turn in be sure to:
   a) Choose clear and suggestive variable names.
   b) No part of your printed output should have lines wrapped to the first column of next line or disappearing off the end of the page.
   c) Use comments generously to describe how your methods work. Comments should appear inside your methods, too – not just at the top. That said, don't overdo the comments.
   d) There should be no more than one *return* statement in any method.
   e) Do not use *break* statements to exit loops.
   f) Do not use global variables. If you are not sure if you are using them, ask.
   g) Reminder: Programs should be 100% your own work, as stated in the syllabus.

```java
import java.io.*;
import java.util.*;

// Pair can be used as a row and column of a square on the board and
// also as increment values to represent a generalized knight's move
class Pair {
      private int row, col;

      // Constructor
      public Pair(int row, int col) { ...

      // Copy constructor
      public Pair(Pair p) { ...

      // Getters and Setters – a total of 4 methods

      // This is one of the four
      public void setRow(int row) {

      // It's almost always a good idea to override Java's default toString
      public String toString() {
            return "(" + String.valueOf(row) + ", " + String.valueOf(col) + ")";
      }
}

// The main class for the chessboard and methods needed to implement the
// algorithm for finding knight's tours
class KnightBoard {
      private int board[][];
      private int numRows, numCols;
      private Pair start;
      private ArrayList<Pair> move;

      // default constructor -- you might want to add a little to this
      //     and use it for debugging
      public KnightBoard() {
            board = new int[8][8];
            for (int row = 0; row < 8; row++)
                  for (int col = 0; col < 8; col++)
                        board[row][col] = 0;
            move = new ArrayList<Pair>();
            start = new Pair(-1, -1);
      }

      // constructor -- data comes from a file
      public KnightBoard(String fileName) throws IOException {
            // This is how mine starts, just to give you an idea
            // Also, this is an exception to the rule about not changing
            //  any of the code. If you are more comfortable accessing
            //  the file and/or reading the data a different way, feel
            //  free to do so.
            boolean inputError = false;
            int int1, int2;
            StringTokenizer input;
            BufferedReader inFile = new BufferedReader(new FileReader(fileName));
            String lineStart;
            String inputLine = inFile.readLine();
            input = new StringTokenizer(inputLine);
            lineStart = input.nextToken();  // read "board"
            lineStart = input.nextToken();  // read "size:"
            numRows = Integer.parseInt(input.nextToken());
            numCols = Integer.parseInt(input.nextToken());
            board = new int[numRows][numCols];
                            ...

      // copy constructor
      // Be sure to make a copy of everything. Do not have this KnightBoard
      //  point to anything in b.
      public KnightBoard(KnightBoard b) { ...

      // Look at my output in testTours to see what should be happening
      // Hint: use String.valueOf( num ) to convert int num to a String
      // You must be sure the columns line up properly as they do in my
      // output.
      public String toString() { ...

      // For easy checking of your answers
      public String toString2() {
            String result = new String();
            for (int row = 0; row < numRows; row++) {
                  for (int col = 0; col < numCols; col++) {
                        result = result + String.valueOf(board[row][col]);
                  }
            }
            return result;
      }

      // These are the 3 methods I use to get the job done. You do not have to use
      // these, but I found them useful. Ee sure to use the Pair class as much as
      // possible

      // Check if this is a legal square to move to, i.e., is it actually on
      // the board and has it not been entered yet
      private boolean tryMove(Pair sq) {

      // The number of legal moves from this square
      private int moveCt(Pair sq) {

      // sq is the square the knight is on. Update the square to its new
      // location based on the move and update the board to reflect this new move
      private void makeMove(Pair sq, Pair move) { ...


      // Enter the knight's moves into the board array
      // Here’s how mine begins. It would be nice if yours starts the same way.  :)
      public void solve() {
            int bestMove, i, bestMoveCt;
            Pair curSpot = new Pair(start), nextMove;
            boolean done = false;
            while (!done) { ...

      // Who are you? Put your name here.
      public static String myName() {
            return "Sam Loyd";
      }
}
```