CSCI 36200: Data Structures

Programming Assignment 1

Instructor: Dr. Snehasis Mukhopadhyay **Due date: February 15, 2018**

The overall objective of this exercise is to get familiar with the use of several data structures including arrays, linked lists, and stacks. The basic problem to be solved is what is commonly known as the "Knight's Tour" problem among puzzle and chess enthusiasts. Briefly stated, the problem is to move the knight, beginning from any given square on the chess-board, successively to all 64 squares, touching each square once and only once. For a given initial position, the solution is represented by placing the numbers 0, 1, …, 63 in an 8×8 array encoding the chessboard. The numbers placed in the array indicate the order in which the corresponding squares are visited.

One can write a recursive procedure to solve the above problem. However, you are required not to do that and you will not get credit for doing so. Instead, you should write an iterative procedure with 'back-tracking' involving a stack. The board state at each instant is an 8×8 array of integers all of whose elements are initially set at −1. As you move to a square, you replace the corresponding array element by the sequence number of the move made (such as 1 for the square visited after the first move, 2 for the second move *etc.*). At each instant the board state with any other necessary information is pushed on to the stack. This is to recover from the case when the knight is trapped somewhere, with no unvisited square to move to, without finding the complete solution. In such a case, you back-track along the traversed path by popping the states from the stack, until you find a square from which you can make a valid move. While back-tracking you should also unmark the the squares (*i.e.*, put −1 at the corresponding elements of the current board state) so that you can visit those squares again.

If, at a particular instant, the knight's position on the board is given by (i,j), there may be at most eight possible moves for the knight which will move it to one of the squares $(i-2,j+1),(i-1,j+2),(i+1,j+2),(i+2,j+1),(i+2,j-1),(i+1,j-2),(i-1,j-2),(i-2,j-1)$. However, in some cases when the knight is located near the edge of the board, some of these moves are not valid since they may move the knight off the board. Also, remember that in the successful trajectory, the knight can visit a square only once.

You do not need to know this for implementing your project, but the procedure corresponds to a depth-first search of a tree which is used to generate systematically (and laboriously) one path after another on the board until you find the correct path. If at every square you choose one move out of the available valid ones in a fixed simple manner (say, always the lowest numbered move), you will find that the program, for most of the initial conditions, takes an inordinate amount of time to terminate. (Satisfy yourself that this is the case). This is because there are too many possible paths to search through before the solution can be found. And there is a solution for every initial position of the knight. Can you guesstimate how many moves of the knight will be made in the worst case before the solution is found? If you cannot, do not worry, since we will develop the theory necessary for such estimation later in the course in the context of trees.

This combinatorial explosion problem associated with an exhaustive search process is a fundamental one that we encounter over and over again in Computer Science, particularly in the area of Artificial Intelligence. One of the approaches to overcome this is to develop heuristic rules, if possible, that

somehow guide the search in the right direction. The overall objective is to find the solution with as few searches as possible and in as general a setting as possible. For the knight's tour problem, one such heuristic rule was provided by J. C. Warnsdoff in 1823. His rule is that the knight should move, at each instant, given several choices, to the square from which there are the fewest possible exits to squares not already visited.

If you choose the knight's move from the beginning, at every instant, following Warnsdoff's rule, you will find that almost always the solution is found in one shot without any backtracking at all! Then we do not need to use any stack. However, one of the objectives of this project is to get practice with the use of a stack data structure. Hence, I suggest that you choose moves using Warndoff's heuristic for the first 32 (out of the total of 63 moves in the solution). But then you turn off the heuristics, and choose moves in an exhaustive manner. This will require the use of some back-tracking (and, hence, a stack), while at the same time resulting in a manageable search time.

Once again, you will not get full credit if you follow Warnsdoff's rule throughout and do not use a stack.

The project also requires the use of a singly linked list to store a number of choices of initial conditions for the knight. Before solving the problem for any initial condition, the program should prompt the user incrementally to enter the initial conditions, but should not ask for the number of initial positions that the user wishes to enter. At the end of this data entry mode, the program should display all the positions entered. It should then further prompt to see if the user wants to add, delete, or modify the list of initial positions.

For each initial position, the program would then solve the knight's tour problem. The result in each case would be an 8 × 8 matrix holding the order of the visits. Store all the results in an array of these matrices.

To summarize, the project has the following two components:

(i) Implementation of a singly linked list (with associated operations) to store the user inputsconcerning the initial positions of the knight.
(ii) Solving the knight's tour problem for each initial position using a stack, backtracking, andusing Warnsdoff's heuristic for the first 32 moves.
The two components should work together. All the messages, and/or the transcript of a session demonstrating the successful operation of the complete program, should be saved in the output file.

You should submit the source code and the output file. It should be possible to compile and exceute the source code on one of our SUN UNIX machines. You are required also to submit a project report consisting of: (a) the project description (in your words), (b) the source code, (c) all outputs including a transcript of the session, (d) any other comments and conclusions that you may prefer to include. The report, excluding the source code and outputs, should be roughly about five printed pages.

You will be graded for the project on the following points: (i) correctness of the program, (ii) organization of the program, (iii) readability of the program, and (iv) your report. Your code will be tested to verify correctness.

As you can see this project has many components. Start early and have fun!