University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Mathematics

Program 2: Recursion

Purpose

The programming assignment will provide an exercise in using recursion. The problem is to solve a path-finding problem in two dimensional space. It will require the usage of recurrence as well as good class design.

Path Finding Problem

A robot is positioned on an integral point in a two-dimensional coordinate grid (x_r, y_r). There is a treasure that has been placed at a point in the same grid at (x_t, y_t). All x's and y's are integers. The robot can move up (North), down (South), left (West), or right (East). Commands can be given to the robot to move one position in one of the four direction. That is, "E" moves a robot one slot East (to the right) so if the robot was on position (3, 4), it would now be on (4, 4). The command N would move the robot one position north so a robot at position (4, 4) would be at (4, 5).

Because the robot cannot move diagonally, the shortest distance between a robot at (x_r, y_r) and a treasure at (x_t, y_t) is

$$| x_r - x_t | + | y_r - y_t | = \text{ShortestPossibleDistance}$$

Write a **recursive** program which determines all the unique shortest possible paths from the robot initial position to the treasure with the following stipulation: The robot may never move in the same direction more than a specified number of steps in a row, determined by the input parameter *MaxStepsDir*. (*MaxStepsDir = 0*, would correspond to having no restrictions on the number of steps in same direction.) Also, there may be blocked positions at which the robot can not pass.

You will build one class for the grid, called board from here on, and one class for the robot. The board class would set the blocked positions on the grid, and the robot class would interact with the board and find the shortest path from the initial position to the treasure.

The input to the Robot constructor will be 5 positive integers: *MaxStepsDir*, the starting position of the robot (x_r, y_r), and the position of the treasure (x_t, y_t). These will be specified as

an input of five integers in the main program. For instance, an input of Robot R(2,1,3,-2,4) corresponds to the robot restricted to 2 consecutive moves in any direction, starting at position (1, 3) and needing to get to position (-2, 4). Do not worry about error conditions on input as we will assume the input is well formed.

The input to the Board constructor would be the size of the board and a vector of integers, ordered as (x_p, y_p), that are the blocked positions on the board. For example, the input Board B(8,8,{1,1,3,1,4,1}) corresponds to an 8-by-8 board with the positions (1,1), (3,1), and (4,1) are those that the robot can not visit.

The output of the program should be the listing of all the unique shortest possible paths followed by the number of unique paths. A path should be output as a string of characters with each character corresponding to a direction the Robot should move. For instance, NNENE corresponds to having the robot move North, North, East, North and East. This would be one answer to the input: 3 3 5 6, which corresponds to (3,3) -> (5,6) (if no positions were blocked on that path, and 2 consecutive moves were allowed).

For the sake of efficiency, do not make two separate recursive calls in your program (one to count and one to print). Make sure that one recursive call handles both.

For the input Robot R(0,1,2,3,5), and Board B(7,7,{}), which corresponds to (1,2) -> (3,5), with no restrictions on the number of consecutive steps, and no blocked positions on the 7-by-7 board, the output should be:

NNNEE
NNENE
NNEEN
NENNE
NENEN
NEENN
ENNNE
ENNEN
ENENN
EENNN
Number of paths: 10

For the input Robot R(3,1,4,4,1) and Board B(6,6,{1,2,2,4,3,3,3,4,4,2,4,4}), which corresponds to *MaxStepsDir = 3,* starting position at (1,4) and treasure at (4,1), on a 6-by-6 board and the following coordinates blocked, (1,2), (2,4) , (3,3) ,(3,4) ,(4,2), and (4,4), the output should be:

SESESE
SESSEE
Number of paths: 2

Please turn-In: a zip file with all .cpp/.h files, and a.out built on Linux.