CS 481 — AI — Giraffe Cover Puzzle

Project — Giraffe Cover Puzzle

Introduction

This project is to write a Lisp program to find a sequence of moves for a chess giraffe that "covers" (visits all the squares) minimally on a 7x7chess board. A square can be visited only once in a minimal "cover".

A giraffe moves in two parts. The first part is to an adjoining square orthogonally (horizontally or vertically). The giraffe may stop its move after the first part, or continue with a second part of the move. The second part is to move diagonally further away from its original square. The two-part giraffe move is like a Knight (Horse) move in chess. The one-part giraffe move is like a Rook (Castle-Tower) moving only one square. Thus, a giraffe in the middle of the board will have 12 possible squares to move to; and fewer such possible squares when it is near a board edge. On a two-part move, the giraffe does not visit the one-part square, but is merely "passing through".

The giraffe starts on the upper-left corner square: coordinates (0 0). The lower right square has the coordinates (6 6). (Note, the commas in the coordinates are implied.)

Your path is a sequence of square coordinates. (Reverse order would be a good choice.) The current path is a "state", and your search for a correct path is a "state space" search.

The goal is to find a path of successive giraffe moves that visits the other 48 squares (a minimal cover). Then print out that path as a list, something like this "((0 0) (1 2) ...)", or its reverse.

This search should be done recursively. Your function should take a path/list, and try in succession each of the 12 different moves (12 kids), and calling yourself recursively with the path argument extended by one move.

When deciding to try a kid move, don't add a square to the path that is already on the path. (You can check this with "member".) Also, ensure that you pick kid squares that are legal moves from where the giraffe currently is (which is the "car" or "nth 0" of the reversed path).

Return nil if you have no more legal moves to try (you've tried all 12 kids). If a "kid call" returns nil, then you need to try the next kid move; else the "kid call" detected a full path and returned it. Thus, if you (your function is run on a kid) are called with a complete 49 move path, then return that path as a success indication, as described in the previous sentence – this is how you detect a "goal state".

It might be handy to pass as a second argument the length of the path so far (but this is an optimization).

Extra credit, 5%: find a minimal cover path that can also move back to the start square (0 0). This means finding a minimal cover path, and then rejecting it if the ending square isn't within 1 camel move of the start square. A path that also returns to its starting square is also called a "tour".

Team

The team may contain up to three members. Pick a three-letter name for the team. If there are ties, I'll think up something.

Readme File

You should provide a README.txt text file that includes the class and section, your (team) name, the project/program name, instructions for building, instructions for use, any extra features, and any known bugs to avoid. Be clear in your instruction on how to build and use the project by providing instructions a novice programmer would understand. Usage should include an example invocation.

A README would cover the following:

- •Program name
- •Your Name (authors, team)
- •Contact info (email)
- •Class number (481)
- •Intro (see the Introduction section, above)
- •External Requirements (eg, "none")
- •Build, Installation, and Setup (eg, Lisp type)
- •Usage
- •Extra Features (if any, none needed)
- •Bugs

Academic Rules

Correctly and properly attribute all third party material and references, if any, lest points be taken off.

Submission

Your submission must, at a minimum, include a plain ASCII text file called **`README.txt`** (e.g., title, contact info (of all team members), files list, installation/run info, bugs remaining, features added) all necessary source files to allow the submission to be built and run independently by the instructor. [For this project, no unusual files are expected.] Note, the instructor doesn't necessarily use your IDE or O.S.

All source code files must include a comment header identifying the author, author's contact info (please, no phone numbers), and a brief description of the file.

Do not include any IDE-specific files, object files, binary **executables**, or other superfluous files.

Place your submission files in a **folder named** `481-p1_XYZ` (for team named XYZ).

Then zip up this folder. Name the .zip file the **same as the folder name**.

Turn in by 11pm on the due date (in the bulletin-board post) by **sending me email** (see the Syllabus for the correct email address) with the zip file attached. The email subject title should also include **the folder name**. [NB, If your emailer will not email a .zip file, then change the file extension from .zip to .zap, attach that, and tell me so in the email.] Please include your name and campus ID (for each team member) at the end of the email (because some email addresses don't make this clear). If there is a problem with your project, don't put it in the email body – put it in the README.txt file. Do not provide a link to Dropbox, Gdrive, or other cloud storage.

Grading

- •75% for "compiling" and executing with no errors or warnings
- •10% for clean and well-documented code
- •10% for a clean and reasonable **`README`** file
- •5% for successfully following Submission rules