Assignment #4
CS 3060 Programming Languages, Spring 2021
Instructor: S. Roy

Prolog #2

Due Date: Feb 27 @ 11.59 PM.
Total Points: 50 points

Directions: Using the source provided via Gitlab `https://gitlab.com/sanroy/sp21-cs3060-hw/`, complete the assignment below. The process for completing this assignment should be as follows:

1. You already forked the Repository "sanroy/sp21-cs3060-hw" to a repository "yourId/sp21-cs3060-hw" under your username. If not, do it now.
2. Get a copy of hw4 folder in "sanroy/sp21-cs3060-hw" repository as a hw4 folder in your repository "yourId/sp21-cs3060-hw"
3. Complete the assignment, committing changes to git. Each task code should be in a separate file. As an example, task1.pl for Task 1.
4. Push all commits to your Gitlab repository
5. If you have done yet done so, add TA (gitlab id: prabeshpaudel and Roy (gitlab id: sanroy) as a **maintainer** of your Gitlab repository

Tasks:

1. **Task #1:)** (15 points) Consider the following knowledge base as in the previous assignment:

   ```
   hasDirectConn(newOrleans, chicago).
   hasDirectConn(philadelphia, newOrleans).
   hasDirectConn(columbus, philadelphia).
   hasDirectConn(sanFrancisco, columbus).
   hasDirectConn(detroit, sanFrancisco).
   hasDirectConn(toledo, detroit).
   hasDirectConn(houston, sanFrancisco).
   ```

   You already wrote a recursive rule `hasConn/2` that tells us whether we can travel by plane from one town A to another town B.

   This time you write a recursive rule `findRoute/2` that should find the route (with zero or more airports as intermediate hops) from town A to town B. Your rule not only finds the route, it also prints the route (showing each hop seperated by a dash or so; e.g. a route from toledo to sanFrancisco can be printed as sanFrancisco <- detroit <- toledo) on the computer screen. If any such route does not exist, prolog should respond "false" or "no" (as appropriate in your Prolog dialect). Run at least 2 querries and show the output (in README) you get from Prolog. `Writing README carries 2 point.`

2. **Task #2a:)** (10 points) Write a Prolog rule `myDelete(List1, A, List2)` which deletes the first occurrence of item A from List1 to produce result List2. As an example, running query `myDelete([b,c,d,b,c,b], c, Result).` should give us [b, d, b, c, b] as *Result*. Then, extend your Prolog rule so that it asks the user for the name of the output file so that it can present the result in that output file. Test your implementation of the rule with at least 2 querries and show the results in README. `Writing README carries 2 points.`

   Task #2b:) (10 points) Write a Prolog rule `myGcd(A, B, G)` which computes gcd G of positive integers A and B. As an example, running query `myGcd(11, 3, Result).` should give us 1 as *Result*. Test your implementation of the rule with at least 2 querries and show the results in README. Then, in the README

file, draw the recursion tree for the computation of `myGcd(12,3,X)`. You may use the *trace* facilty to track the whole computation. To get an example of recursion tree, refer to the recursion tree of the factorial computation in Prolog Lecture ppt2. Furthermore, you may see some tips on gcd computation at the end of Prolog lecture ppt3. `Writing README carries 2 points.`

3. **Task #3:)** (15 points) Write Prolog code which can solve any given 9x9 Sudoku puzzle. Note that the textbook has solution for the 4x4 Sudoku puzzle. You need to extend the textbook's code (i.e., sudoku4.pl on Page 104-105) and make it work for the 9x9 puzzle. Test your implementation with at least 2 querries and show the results in README. `Writing README carries 1 point.` Note: you may need an alternative to library rules fd_all_different and fd_domain depending on your prolog system, and an alternative of these rules are available at the end of the prolog lecture3 ppt.