CSCE 322 - Fall 2010

Assignment 4: Logic Programming with Prolog

Due Date: **Friday, December 3rd before class starts**

Task

In this assigment you'll develop Prolog programs to assess the rail services in a region. In part 1, you'll design a program to handle queries regarding service coverage by repair trains. In part 2, your program should handle queries to determine the paths from a fire station to the site of a disaster.

Part One:

`noRepairCoverage(FuelCap, Region, Cell)`

This query succeeds when there is a track piece in Cell within Region that is not reachable from a rail yard within FuelCap. A Region is a grid represented by a list of lists of integers (one list per row in the region). The values in the region correspond to terrain and structures so, for example, the value 2 represents a rail yard. A Cell is a list of two integers describing the row and column of a location in a Region.

Pieces of railroad track are naturally repaired by repair trains. These repair trains depart from rail yards (labeled as 2) and travel over railroad tracks (labeled as 1). The train starts with FuelCap units of fuel, and depletes 1 unit of fuel for every cell it travels. If a train has 0 units of fuel, it can no longer travel. However, every time the train is adjacent to a fueling station (labeled as 3), the train's fuel is restored to FuelCap. When we say a train is adjacent to a fueling station, we mean that it is 1 cell away from the station in a cardinal direction (up, down, left, or right). A train can use a fueling station more than once. Table 1 lists the symbols that may appear in the region.

A Cell has no repair coverage if and only if it is a piece of railroad track and a repair train cannot reach the cell by departing from any rail yard.

3	1	1	1	1	0
0	1	0	0	0	0
0	1	3	0	0	1
2	1	0	2	1	1

When quering `noRepairCoverage(2, ThisRegion, Cell)` with *ThisRegion* bound to the above region, we might see the following:

```
[1, 5]
true ;
[3, 6]
true ;
fail.
```

This indicates that the cells at 1,5 and 3,6 cannot be reached from any rail yard. Note that the order of your results is not important. (Remember that lists in Prolog start at 1!)

Integer	Indicates
0	Empty terrain
1	Track piece
2	Rail yard
3	Fueling station
4	Fire station
5	Fire location

Table 1: Region Legend.

Part Two:

`routeToFire(Region, Station, Path)`

This query would succeed when a path between any of the fire stations and the location adjacent to the site of a fire can be found. A given region may have any number of fire stations, but only ONE potential site for a fire.

The path directions should be indicated by 'u', 'd', 'l', and 'r', which correspond to up, down, left, and right. As before, there is no diagonal movement. Since fire response in these regions happens via railroad, fire-fighting trains may only travel on railroad lines (1), and originate from fire stations (4). The objective is to find a path to a cell which is adjacent to a cell where a potential disaster is occurring (5).

Fire-fighting trains will always have enough fuel to reach their destination. Also note that in the interest of minimizing response time, a fire-fighting train should not return to a cell which it has already traveled through.

A sample scenario appears below. A possible path for the station at row 6, column 2 is 'urrr'. The station at row 4, column 1 has two possible paths: 'urrrr' and 'uruurrrdd'.

0	1	1	1	1
0	1	0	0	1
1	1	1	1	1
4	0	0	0	5
0	1	1	1	1
0	4	0	0	0

If we prompted Prolog `routeToFire(MyRegion, Station, Path)` with *MyRegion* already bound as above, we might expect the following:

```
Station = [4, 1],
Path = [u, r, r, r, r];
Station = [4, 1],
Path = [u, r, u, u, r, r, r, d, d];
Station = [6, 2],
Path = [u, r, r, r].
```

As before, the order of these results is not important.

Environment and Testing

Your code must run in *pl* on cse.

As before, you will receive 5 tests for each part in advance of the due date. In addition to the 5 tests given before, another 5 tests will be run against each part of your code for grading.

The test cases are provided in the files *testcases.pl*. You may test your work by running *pl* and issuing the following commands:

```
[testcases].
main1part1.
```

To use different test cases, simply replace the '1' in 'main1' with another number from 1 to 5, and the '1' in 'part1' with a number from 1 to 2. In order to see every possible binding for results, enter a semicolon after every result until no more are found.

Grading

This assignment is worth 15% of your final grade. The breakdown for this assignment is as follows:

- (50%) Successful completion of Part 1.
- (45%) Successful completion of Part 2.
- (5%) Brief documentation included in a README.TXT file. Your readme file should include any assumptions that you made (e.g., how to handle cases that were not covered in the provided specifications) and a disclosure and explanation of the sources you used (e.g., friends, web site, etc) to help you complete this assignment.

Please check the course policy about assignments. Keep in mind that "Exchange of ideas and techniques is encouraged but you must turn in original work and acknowledge the help you receive from others" and that "Each student is given an automatic extension of five (5) calendar days," but some of you have already consumed some of them.

Submission

Use handin (http://cse.unl.edu/˜handin) to submit all your documents:

1. Your Prolog source files. Label these program files *assn4_part1.pl* and *assn4_part2.pl*.
2. A readme file on how to compile and run your code. Label this file *README.TXT*.