CS 386 Artificial Intelligence Lab
11/08/2015, Tuesday
Lab Handout 4

General Instructions

- Create a folder named **YOURROLLNUM_lab3** (e.g. 131050001_lab3) in the home directory (or at a location of your choice). You need to submit a compressed (tar.gz or tgz only) copy of this folder at the end of this lab.
- This lab requires you to write prolog predicates for **two** problems. Each problem has 2 parts, a minimum required part and an additional part for extra bonus.
- The deadline for submission of the required part (or whatever you are able to complete) is end of today's lab (**Tuesday, 11/08/2015, 5:00 PM**). Solution for the additional/bonus part can be submitted till (**Sunday, 16/08/2015, 11:55 PM**). Separate links will be provided for both submissions. Your final submission should contain all the required (four) predicates.
- More specific submission instructions for each part are provided later.
- All problems must be solved using Prolog only.
- You **can use** in-built library functions for solving these problems. You may find predicates like `findall, random, and -> (if-then-else)` useful for solving these problems.
- Strictly follow the predicate names and conventions as described in the problem description and examples.
- Make sure that you read and understand the submission instructions before making a final submission. Your submission will be automatically graded.
- After evaluating your submission, we will share the test cases and result of your submission with you. Instructions regarding cribs will be notified soon.
- Add comments to your code. Write your Name/Roll number on top of every source. Explain your code briefly.
- You are encouraged to post your queries/doubts about the handout (or anything else) on the **moodle discussion forum** during the lab as well.
- The lab may contribute to a maximum of **2% of your final grade**.

<u>Problem 1</u>: p235 Problem

Consider numbers of the form $2^i * 3^j * 5^k$ - that is, all numbers whose only non-trivial prime factors are 2, 3 and 5. If you arrange them in ascending order, the list will look like below-

1, 2, 3, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, ...

Write a predicate `p235(N, Ans)` to calculate the N-th number of this form.

Example Queries:

```
?- p235(100,Ans).
Ans = 1600 .

?- p235(1000,Ans).
Ans = 51840000 .

?- p235(10000,Ans).
Ans = 288555831593533440 .
```

The name of the script should be *yourrollnumber_problem1.pl*

<u>Hints</u>:- You might require to increase the global stack size for higher values of N. This can be done by running swipl with a -G flag.

e.g. `swipl -G1024M`

Try using the in-built **`time(:Goal)`** predicate to check time used, number of logical inferences and the average number of lips (logical inferences per second).

Example:

```
?- time(p235(100,Ans)).
% 22,900 inferences, 0.007 CPU in 0.007 seconds (100% CPU,
3443322 Lips)
Ans = 1600 .
```

Additional Part

Try to generalize the above predicate by not hard-coding the list of primes to `[2,3,5]`. Write a predicate `findNth(ListOfPrimes, N, Ans)` where `ListOfPrimes` is `[2,3,5]` for above case. For example, the predicate can be called as `findNth([3,7,11,13], 20, Ans)` to find the 20th number of the form $3^i*7^j*11^k*20^l$.

Problem 2: Knight Problem

Let the squares of chess board be labelled [1,1] (Lower left) to [8,8] (upper right). Then the code below can be used to find all squares a knight can jump to in one step.

```
/* generate a move given start square [X,Y].*/
ktmove([X,Y], [X1,Y1]) :- c1(X,X1), c2(Y,Y1).
ktmove([X,Y], [X1,Y1]) :- c2(X,X1), c1(Y,Y1).

c1(X,X1) :- X > 1, X1 is X - 1.
c1(X,X1) :- X < 8, X1 is X + 1.
c2(X,X1) :- X > 2, X1 is X - 2.
c2(X,X1) :- X < 7, X1 is X + 2.
```

```
?- ktmove([2,5],Ans).
Ans = [1, 3] ;
Ans = [1, 7] ;
Ans = [3, 3] ;
Ans = [3, 7] ;
Ans = [4, 4] ;
Ans = [4, 6].

?- findall(Next, ktmove([2,5],Next), AllMoves).
AllMoves = [[1, 3], [1, 7], [3, 3], [3, 7], [4, 4], [4, 6]].
```

For this lab, you need to write a predicate `rWalk(StartSq, Walk)` where Walk is the list of squares visited by the Knight (no repeat) when it randomly picks one of the possible next steps it can move to from its current square without revisiting any square already seen. The knight stops when it gets stuck and the Walk ends. See sample output below.

Write also a predicate `rpt(N, StartSq)` which tries the random walk `N` times and checks how many steps it can go before blocking.

The name of the script should be *yourrollnumber_problem2.pl*

Example Queries:

```
?- rWalk([2,4],Ans).
Ans = [[2, 2], [1, 4], [2, 6], [4, 5], [3, 7], [2, 5], [1,
3], [3, 4], [4, 2], [5, 4], [6, 2], [7, 4], [8, 6], [6, 7],
[7, 5], [8, 3], [6, 4], [7, 2], [8, 4], [6, 5], [5, 3], [4,
1], [3, 3], [1, 2], [3, 1], [4, 3], [2, 4]] .

?- random(1, 30, Num).
Num = 24.

?- rpt(10,[2,4]).
39 steps before blocking.
25 steps before blocking.
31 steps before blocking.
37 steps before blocking.
32 steps before blocking.
15 steps before blocking.
32 steps before blocking.
34 steps before blocking.
30 steps before blocking.
45 steps before blocking.
true .
```

Additional Part

Write a new predicate `path(StartSq, EndSq, Path)` where Path is a shortest sequence of steps needed by the Knight to go from `StartSq` to `EndSq`.

Add the new predicates to the same file as mentioned above (*yourrollnumber_problem2.pl*)

Submission

At this point the folder yourrollnum_lab3 should contain two files:

1. yourrollnum_problem1.pl containing your solution to the p235 problem.
2. yourrollnum_problem2.pl containing your solution to the knight problem.

Compress the folder using the following command

```
tar -zcvf yourrollnum_lab3.tgz yourrollnum_lab3
```

Upload the generated **.tgz** file as your submission for this lab.

If you complete and submit all four predicates during today's lab, no separate submission is required on Sunday. However, your final submission should contain all the predicates.