17th Prolog Programming Contest
17 July 2010, Edinburgh

The contest consists of 5 challenges. Each correct submission (at most one for each challenge) earns you one point. Incorrect submissions cost nothing, except that you might be mentioned in the award speech.

Efficiency of your programs is usually not important, but if your program fails to finish in a reasonable time, it will be considered incorrect. However, the efficiency of your submission to challenge *circ* will decide in case of a draw.

Submit a file for each solution. The file name must be the same as given in the header of the challenge - this file must be readable for the organisers. For instance, for the first problem, you make a file named `rqueens.pl`.

The file must not contain predicates whose names start with `iclp10` or `test` - do not use the dynamic database or other similar global stuff built-in predicates!

1 The Tartan of McLog *(tartan.pl)*

The tartan of the McLog clan at Loch Pro nicely scales up (with N) from small handkerchiefs over kilts to bedspreads. Write a `tartan/1` predicate that generates the tartan for given size `N` on the screen by the goal `?-tartan(N).` Do not add extra spaces to the left! Here are the patterns for size 1, 2, 3 and 7:

```
?- tartan(1).
*

?- tartan(2).
***
 *
***

?- tartan(3).
*   *
*****
* * *
*****
*   *
```

```
?- tartan(7).
*           *
*************
* *       * *
* ********* *
* * *   * * *
* * ***** * *
* * * * * * *
* * ***** * *
* * *   * * *
* ********* *
* *       * *
*************
*           *
```

2 Eric the Viking *(eric.pl)*

Eric, the one-eyed captain of the Viking dragon boat the "Red Cutter", is visiting the British Isles with his raiding party. He intends to pay the tree kings a visit (so called for their tree shaped castles) or at least their castles' treasuries.

Now, as far as Eric is concerned, the days of mindless looting and pillaging are a thing of the past. After all you've got only so many eyes to loose before the lights go out permanently. No, Eric's devised a plan for systematic looting and pillaging!

The party is to strike a tree castle when the king is out hunting with the majority of his men. That gives them V viking grunts[1] of relatively low-risk plundering. While Eric's crew distracts the token force at the gates, Eric uses his time to systematically search for booty.

There are only two types of room in a tree castle: forks `fork(`*`Left`*`,`*`Right`*`)` that lead to two more rooms, and treasuries `treasury(`*`T`*`)` with a bag of T golden sovereigns. Watch out: there is a guard behind every right door.

Now here's Eric approach. He first explores the wing of the castle behind the left door in a fork, before visiting the right door. He can carry out only one bag, so he takes the one with the most gold and drops any other on the way. Charging into a room (from any direction) takes 1 viking grunt, and incapacitating a guard takes also one viking grunt. Eric's got to be out of the castle when the time is up, possibly leaving part of the castle unexplored.

Write a predicate `eric(`C`,`V`,`T`)` that given the number of available viking grunts V and the castle layout C, succeeds and binds T to the number of golden sovereigns Eric carries away. The predicate fails if Eric does. Here are some example queries:

```
?- eric(treasury(10),2,T).
T = 10.

?- eric(treasury(10),1,T).
false.

?- eric(fork(treasury(10),treasury(20)),6,T).
T = 10.

?- eric(fork(treasury(10),treasury(20)),7,T).
T = 20.
```

[1] The Viking's measure of time.

3 Maglev Switching *(maglev.pl)*

Maglev trains are the future: driving from one station to the next has zero cost.[2] Stations are connected by (unidirectional[3]) tracks. Trains want to drive from the initial station of their journey, to the final station with minimal cost. Shortest path with 0 weight edges, you think? Not quite so.
Suppose that there is a direct track from station A to stations X, Y and Z. The station A is at any particular moment set to direct the first incoming train to say X, its preferred next station: every train arriving at A continues its journey to X, until the incoming train switches the preferred next station of A to Y or Z. Of course, the train that performed the switch, still goes on to X ! Changing the preferred next station has a cost, of 1.
We represent a station as a term like `station(a,b,[c,d])` meaning that station `a` has preferred next station `b`, and two more stations that can be reached from it, namely `c` and `d`. A set of stations is represented as a list of represented stations. You must write a predicate `maglev/4` that will be called with as first argument a set of stations, as second argument the start station, and as third argument the destination station. The last argument will be free and it must be unified with the minimal number of preferred next station changes needed to go from start to destination. Just to give a trivial example:

```
?- maglev([station(a,b,[c,d]), station(b,z,[a,c,d])],a,z,Cost).
Cost = 0.
```

Just one more thing ... the railway company is very nice to its passengers: it does not allow booking an impossible journey. I.e., `maglev/4` must finitely fail if that's the case.

4 Red Queens *(rqueens.pl)*

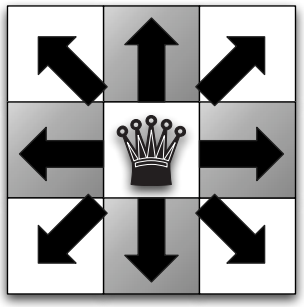
Red queens are more aggressive than their white and black sisters. They have a dominant streak too.
To have some piece and quiet, we should have as few as possible on our $N \times N$ chess board. None of the red queens should attack another (the way queens do), and no square should be uncovered[4].
Write a predicate `rqueens(`N`,`Q`)` that returns the least number of red queens Q that fit on an $N \times N$ board, satisfying the above rules.

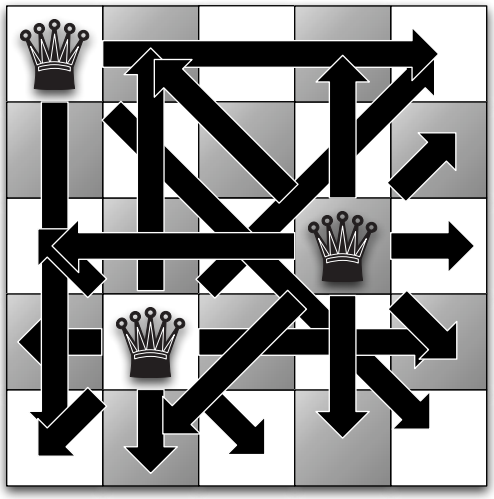
[2] Frictionless supermagnets, remember ?
[3] To reduce the risk of headon collisions
[4] and thus elicit a mad dash of conquest

```
?- rqueens(3, Q).
Q = 1.

?- rqueens(5, Q).
Q = 3.

?- rqueens(8, Q).
Q = 5.
```





5 Student Logic *(student.pl)*

> 'Your proof is full of flaws. You know what you're starting from, and where you want to end up, but inbetween it's a mess. I cannot award you any credits for it.' 'But sir, it's almost correct. You see, I've made only minimal amends to make it work out.'

Help the poor student make his case, working out the minimal number N of changes to the derivation to make a ground goal *Goal* provable. To do so, write a predicate `student(`*`Program`*`,`*`Goal`*`,`*`N`*`)`. *Program* is a list of Prolog rules. See the examples for the format; only user defined predicates are used.
A change is simply a swap of any two arguments in an (initial or intermediate) goal.

```
?- student([rule(p(a,b),[])], p(b,a), N).
N = 1.

?- student([rule(append([],L,L),[])
           ,rule(append([X|Xs],Ys,[X|Zs]), [append(Xs,Ys,Zs)])
           ], append([1],[2],[2,1]),N).
N = 1.

?- student([rule(append([],L,L),[])
           ,rule(append([X|Xs],Ys,[X|Zs]), [append(Xs,Ys,Zs)])
           ], append([1],[2,3],[2,1,3]),N).
N = 2.
```