Departamento de Ciência de Computadores - FCUP
Second Test of Logic Programming
(Duration: 2h)

Date: December 12th, 2016

NB: These are possible solutions. Remember that each one may have a different way of solving the same problem. Solutions are in blue.

1) Given a list of sublists, where each sublist has two elements and represents an interval $[X, Y], X \leq Y$:

```
[[1,2],[5,7],[6,10],[12,15]]
```

Write a program to return the largest interval. You can assume that the sublists are sorted.

```
% largest_interval(+Lists,-LargInt).
largest_interval(List,[Min,Max]) :-
   flatten(List,FlatList),
   qsort(FlatList,[Min|SortedFlatList]),
   last([Min|SortedFlatList],Max).
```

This is a very naive and inefficient solution, but it works, and I scored in full solutions similar to this and whose code gave the correct solution.

2) Write a program that drops every n-th element from a list. For example:

```
?- drop([a,b,c,d,e,f,g,h,i,k],3,X).
X = [a,b,d,e,g,h,k]
```

```
% drop(+Elements,+IndiceInterval,-NewList).
drop(Elements,X-Y,NewList) :-
   drop(Elements,X-Y,1,NewList).

drop([_H|T],X-Y,X,NewT) :-
   Nelems is Y-X+1, % ignore next elements in the interval (after X)
   drop1(T,Nelems,NewT), !.
drop([H|T],X-Y,NotX,[H|NewT]) :- % keep elements before X
   X1 is NotX + 1,
   drop(T,X-Y,X1,NewT).

drop1(T,1,T).
drop1([_H|T],N,NewT) :- % ignore elements inside the interval
  N1 is N - 1,
  drop1(T,N1,NewT).
```

This program has two parts: one that copies the elements of the original list to the output list up to index X (second clause of drop/4), and the second one that ignores all elements from index X to index Y (drop1/3), returning the remaining elements of the list.

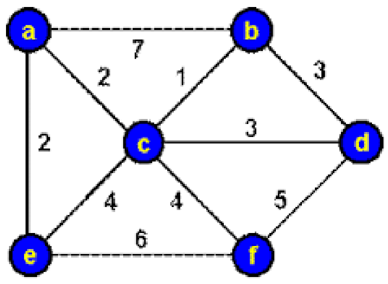
3) Goldbach's conjecture says that every positive even number greater than 2 is the sum of two prime numbers. For example, $28 = 5 + 23$.
It is one of the most famous facts in number theory that has not been proved to be correct in the general case. It has been numerically confirmed up to very large numbers (much larger than we can go with our Prolog system). Write a predicate to find the two prime numbers that sum up to a given even integer. For example:

```
?- goldbach(28, L).
L = [5,23]
```

```
gb(N) :-
   primes(N,PrimesList),
   del(Prime1,PrimesList,NewPrimesList),
   del(Prime2,NewPrimesList,_RemainingPrimes),
   N =:= Prime1 + Prime2.

primes(N,List) :-
   primes(2,N,List).

primes(N,N,[]).
primes(I,N,[I|T]) :-
   prime(I), !,
   I1 is I + 1,
   primes(I1,N,T).
primes(I,N,T) :-
   I1 is I + 1,
   primes(I1,N,T).

prime(I) :-
   prime(2,I).

prime(I,I) :- !.
prime(I,J) :- I > J, !.
prime(D,I) :-
   X is mod(I,D),
   \+ X = 0,
   D1 is D + 1,
   prime(D1,I).

del(X,[X|T],T).
del(X,[Y|T],[Y|T1]) :-
   del(X,T,T1).
```

Not efficient at all...But it works for big numbers that are goldbach. For big numbers that are not goldbach, it takes lots of time and memory to finish with a fail (it will try all possible combinations of the prime numbers - which has quadratic complexity.

4) Given the graph shown in the figure, where nodes are locations and edges are costs, write a program to find the shortest path between nodes “a” and “d”.



```
:- use_module(library(lists)).

s(a,b,7).
s(a,c,2).
s(a,e,2).
s(b,c,1).
s(b,d,3).
s(c,d,3).
s(c,e,4).
s(c,f,5).
s(d,f,5).

bfs(Initial,Final) :- solve([[[Initial,0]]],Final).

solve([[[N,C]|Path]|_],[[N,C]|Path]) :- goal(N), !.
solve([Path|Paths],Solution) :-
  extend(Path,NewPaths),
  append(Paths,NewPaths,Paths1), !,
  solve(Paths1,Solution).

extend([[Node,CNode]|Path],SortedNewPaths) :-
  bagof([[NewNode,NewC],[Node,CNode]|Path],
        C^(s(Node,NewNode,C), NewC is CNode + C),
        NewPaths),
  write(NewPaths),nl,
  mysort(NewPaths,SortedNewPaths), !.
extend(Path,_). % node has no successor.

goal(d).

mysort([],[]).
mysort([[[Node,X]|Path]|T],Sorted) :-
   partition(T,[[Node,X]|Path],L1,L2),
   mysort(L1,S1),
   mysort(L2,S2),
   append(S1,[[[Node,X]|Path]|S2],Sorted).

partition([],_,[],[]).
partition([[[Node1,Y]|Path1]|T],[[Node2,X]|Path2],[[[Node1,Y]|Path1]|L1],L2) :-
   Y =< X, !,
   partition(T,[[Node2,X]|Path2],L1,L2).
partition([[[Node1,Y]|Path1]|T],[[Node2,X]|Path2],L1,[[[Node1,Y]|Path1]|L2]) :-
   partition(T,[[Node2,X]|Path2],L1,L2).
```

5) Show two advantages of constraint logic programming over Prolog programming.
Examples:

1. It allows variables to have multiple values (domain).
2. It allows to write and solve mathematical expressions without requiring variables to be instantiated.

6) In cypher-arithmetic problems, each letter is associated to a number between 0 and 9. Each letter can be assigned a number that is distinct from all other letters. Write a program to solve the cypher-arithmetic problem `TWO + TWO = FOUR`, using `clp(fd)` syntax.

```
:- use_module(library(clpfd)).
solve(Vars) :-
   Vars = [T,W,O,F,U,R],
   Vars ins 0..9,
   all_different(Vars),
    100 * T + W * 10 + O +
    100 * T + W * 10 + O #=
   1000 * F + O * 100 + U * 10 + R,
   T #\= 0, F #\= 0,
   label(Vars).
```

7) What is the difference between instructions `get` and `put` in the Prolog abstract machine?

```
get instructions fetch values pointed by the registers A_i (arguments
that come from the current predicate call), and unify with the values
represented by the actual parameters used by the predicate head. For
example, if the instruction is get_constant nil,A2, the current
predicate is expecting a constant nil in argument 2 of the predicate
call. If A_2 is not pointing to nil or is not an unbound variable,
the execution of this instruction will fail.

put instructions prepare the stack to the executon of the
next predicate call. It initializes the registers A_i.
```