Lista de ejercicios de Prolog (Práctica 2)

1. Escribe un predicado `prod(L,P)` que signifique: "`P` es el producto de los elementos de la lista de enteros dada `L`". Debe poder generar la `P` y también comprobar una `P` dada.

2. Escribe un predicado `pescalar(L1,L2,P)` que signifique: "`P` es el producto escalar de los vectores `L1` y `L2`", donde los vectores se representan como listas de enteros. El predicado debe fallar si los dos vectores tienen longitudes distintas.

3. Representando conjuntos con listas sin repeticiones, escribe predicados para las operaciones de intersección y unión de conjuntos dados.

4. Usando `append`, escribe un predicado para calcular el último elemento de una lista dada, y otro para calcular la lista inversa de una lista dada.

5. Escribe un predicado `fib(N,F)` que signifique: "`F` es el `N`-ésimo número de Fibonacci para la `N` dada". Estos números se definen así: $fib(1) = 1$, $fib(2) = 1$, y si $N > 2$ entonces $fib(N) = fib(N-1) + fib(N-2)$.

6. Escribe un predicado `dados(P,N,L)` que signifique: "la lista `L` expresa una manera de sumar `P` puntos lanzando `N` dados". Por ejemplo: si `P` es 5 y `N` es 2, una solución sería `[1,4]` (nótese que la longitud de `L` es `N`). Tanto `P` como `N` vienen instanciados. El predicado debe ser capaz de generar todas las soluciones posibles.

7. Escribe un predicado `suma_demas(L)` que, dada una lista de enteros `L`, se satisface si existe algún elemento en `L` que es igual a la suma de los demás elementos de `L`, y falla en caso contrario.

8. Escribe un predicado `suma_ants(L)` que, dada una lista de enteros `L`, se satisface si existe algún elemento en `L` que es igual a la suma de los elementos anteriores a él en `L`, y falla en caso contrario.

9. Escribe un predicado `card(L)` que, dada una lista de enteros `L`, escriba la lista que, para cada elemento de `L`, dice cuántas veces aparece este elemento en `L`. Por ejemplo, si hacemos la consulta `card( [1,2,1,5,1,3,3,7] )` el intérprete escribirá:

   `[[1,3],[2,1],[5,1],[3,2],[7,1]]`.

10. Escribe un predicado `esta_ordenada(L)` que signifique: "la lista `L` de números enteros está ordenada de menor a mayor". Por ejemplo, a la consulta:

    `?-esta_ordenada([3,45,67,83]).`

    el intérprete responde `yes`, y a la consulta:

    `?-esta_ordenada([3,67,45]).`

    responde `no`.

11. Escribe un predicado `ord(L1,L2)` que signifique: "`L2` es la lista de enteros `L1` ordenada de menor a mayor". Por ejemplo: si `L1` es `[4,5,3,3,2]` entonces `L2` será `[2,3,3,4,5]`. Hazlo en una línea, usando sólo los predicados `permutacion` y `esta_ordenada`.

12. Escribe un predicado `diccionario(A,N)` que, dado un alfabeto `A` de símbolos y un natural `N`, escriba todas las palabras de `N` símbolos, por orden alfabético (el orden alfabético es según el alfabeto `A` dado). Por ejemplo, `diccionario( [ga,chu,le],2)` escribirá:

    `gaga gachu gale chuga chuchu chule lega lechu lele`.

13. Escribe un predicado `palindromos(L)` que, dada una lista de letras `L`, escriba todas las permutaciones de sus elementos que sean palíndromos (capicúas). Por ejemplo, con la consulta `palindromos([a,a,c,c])` se escribe `[a,c,c,a]` y `[c,a,a,c]`.

14. Encuentra mediante un programa Prolog, usando el predicado `permutación`, qué 8 dígitos diferentes tenemos que asignar a las letras `S`, `E`, `N`, `D`, `M`, `O`, `R`, `Y`, de manera que se cumpla la suma siguiente:

```
  S E N D
+ M O R E
---------
M O N E Y
```

15. Escribe un predicado `simplifica` que pueda usarse en combinación con el programa de calcular derivadas.

16. Queremos obtener en Prolog un predicado `dom(L)` que, dada una lista `L` de fichas de dominó (en el formato de abajo), escriba una cadena de dominó usando *todas* las fichas de `L`, o escriba "no hay cadena" si no es posible. Por ejemplo,

    ```
    ?- dom( [ f(3,4), f(2,3), f(1,6), f(2,2), f(4,2), f(2,1) ] ).
    ```

    escribe la cadena correcta:

    ```
    [ f(2,3), f(3,4), f(4,2), f(2,2), f(2,1), f(1,6) ].
    ```

    También podemos *girar* alguna ficha como `f(N,M)`, reemplazándola por `f(M,N)`. Así, para:

    ```
    ?- dom ([ f(4,3), f(2,3), f(1,6), f(2,2), f(2,4), f(2,1) ]).
    ```

    sólo hay cadena si se gira alguna ficha (por ejemplo, hay la misma cadena que antes).

    El siguiente programa Prolog aún no tiene en cuenta los posibles giros de fichas, ni tiene implementado el predicado `ok(P)`, que significa: "`P` es una cadena de dominó correcta (tal cual, sin necesidad ya de girar ninguna ficha)":

    ```
    p([],[]).
    p(L,[X|P]) :- select(X,L,R), p(R,P).

    dom(L) :- p(L,P), ok(P), write(P), nl.
    dom(_) :- write('no hay cadena'), nl.
    ```

    (a) ¿Qué significa el predicado `p(L,P)` para una lista `L` dada?

    (b) Escribe el predicado `ok(P)` que falta.

    (c) Extiende el predicado `p` para que el programa también pueda hacer cadenas girando alguna de las fichas de la entrada.

17. Complete the following backtracking procedure for SAT in Prolog. Program everything, except the predicate `readclauses(F)`, which reads a list of clauses, where each clause is a list of integers. For example, $p_3 \vee \neg p_6 \vee p_2$ is represented by `[3,-6,2]`. Do things as simple as possible.

    ```
    p:- readclauses(F), sat([],F).
    p:- write('UNSAT'),nl.
    ```

```
sat(I,[]):- write('IT IS SATISFIABLE. Model:  '), write(I),nl,!.
sat(I,F):-
decision_lit(F,Lit), % Select unit clause if any; otherwise, an arbitrary one.
simplif(Lit,F,F1), % Simplifies F. Warning:  may fail and cause backtracking
sat( ...  , ...  ).
```

18. Consider two groups of 10 people each. In the first group, as expected, the percentage of people with lung cancer among smokers is higher than among non-smokers. In the second group, the same is the case. But if we consider the 20 people of the two groups together, then the situation is the opposite: the proportion of people with lung cancer is higher among non-smokers than among smokers! Can this be true? Write a little Prolog program to find it out.

19. Supongamos que tenemos una máquina que dispone de monedas de valores `[X1,...Xn]` y tiene que devolver una cantidad `C` de cambio utilizando el **mínimo número** de monedas. Escribe un programa Prolog `maq(L,C,M)` que, dada la lista de monedas `L` y la cantidad `C`, genere en `M` la lista de monedas a devolver de cada tipo. Por ejemplo, si `L` es `[1,2,5,13,17,35,157]`, y `C` es 361, entonces una respuesta es `[0,0,0,1,2,0,2]` (5 monedas).

    Note: greedy algorithms (starting with the largest coin, etc.) do not always work!

20. Write in Prolog a predicate `flatten(L,F)` that "flattens" (cast.: "aplana") the list `F` as in the example:
    `?-flatten( [a,b,[c,[d],e,[]],f,[g,h]], F ).`
    `F=[a,b,c,d,e,f,g,h]?`

21. Escribe un predicado Prolog `log(B,N,L)` que calcula la parte entera `L` del logaritmo en base `B` de `N`, donde `B` y `N` son naturales positivos dados. Por ejemplo, `?- log(2,1020,L).` escribe `L=9?` Podéis usar la exponenciación, como en `125 is 5**3`. El programa (completo) no debe ocupar más de 3 lineas.

22. Supongamos que `N` estudiantes (identificados por un número entre 1 y `N`) se quieren matricular de LI, pero sólo hay espacio para `M`, con $M < N$. Además nos dan una lista `L` de pares de estos estudiantes que son incompatibles entre sí (por ejemplo, porque siempre se copian). Queremos obtener un programa Prolog `li(N,M,L,S)` que, para `N`, `M` y `L` dados, genere un subconjunto `S` con `M` de los `N` estudiantes tal que si $[x, y] \in$ `L` entonces $\{x, y\} \not\subseteq$ `S`. Por ejemplo, una solución de

    `li( 20, 16, [[8,11],[8,15],[11,6],[4,9],[18,13],[7,9],[16,8],[18,10],[6,17],[8,20]], S )`
    es `[20,19,17,16,15,14,13,12,11,10,7,5,4,3,2,1]` .

    Escribe una versión lo más sencilla que puedas, aunque sea ineficiente, del estilo "generar una solución (total) y después comprobar si es correcta".

23. Given a list of integers `L`, and a maximum sum `K`, write the subsets `Sm` of `L` such that:

    - `sum(Sm) =< K`, and
    - no element in $\texttt{L} \setminus \texttt{Sm}$ can be added to `Sm` without exceeding the sum `K`.

    For the example below, a correct output would be the following (or in another order):

    ```
    [2,5,-2,1]
    [2,-2,2,3,1]
    [2,-2,2,4]
    ```

```
[2,-2,4,1]
[5,-2,2,1]
[5,-2,3]
[7,-2,1]
[-2,2,4,1]
[-2,3,4,1]
```

Hint: you can use the predicate `sum_list(L, X)`, which is true if `X` is the sum of the numbers in `L`; e.g., `sum_list([1,2,3], 6)` holds.

Complete:

```
%% Example:
numbers([2,5,7,-2,2,9,3,4,1]).
maxSum(6).

%% subsetWithRest(L, Subset, Rest) holds
%% if Subset is a subset of L and Rest is the rest of the elements.
subsetWithRest([], [], []).
...

%% maxSubset(K, L, Sm) holds
%% if Sm is a subset of numbers of L such that
%% it sums at most K
%% and if we try to add any other element, the sum exceeds K.
maxSubset(K, L, Sm):-
    subsetWithRest(L, Sm, Rest),
    ...

main :-
    numbers(L),  maxSum(K),
    maxSubset(K, L, Sm),
    write(Sm), nl, fail.
main:- halt.
```

24. Given a graph declared as in the example below, write all its cliques of size at least `minCliqueSize`. Remember, a clique is a complete subgraph: a subset {texttttS of the vertices such that for all `U`,`V` in `S` there is an edge `U-V`.
For the example below, a correct output would be the following (or in another order):

```
[2,4,5,7,9]
[2,4,5,7]
[2,4,5,9]
[2,4,7,9]
[2,4,8,9]
[2,5,7,9]
[4,5,7,9]
```

Complete:

```
%%==== Example: ==========================================================
numVertices(10).
minCliqueSize(4).
vertices(Vs):- numVertices(N), findall(I,between(1,N,I),Vs).
```

```
vertex(V):- vertices(Vs), member(V,Vs).
edge(U,V):- edge1(U,V).
edge(U,V):- edge1(V,U).

edge1(9,8).
edge1(8,2).
edge1(7,4).
edge1(5,7).
edge1(4,2).
edge1(5,2).
edge1(2,7).
edge1(7,9).
edge1(2,9).
edge1(4,8).
edge1(4,9).
edge1(9,5).
edge1(4,5).
%%=============================================================

main:- ... subconjunto( Vs, S), ...  write(S), nl, fail.
main:- halt.

isClique(S):- ...
```

25. Complete the following predicate in prolog.

```
% nthRoot( N, K, R ) === "Given positive integers N and K,
                          the integer part of the Nth root of K is R".
% Example: the integer part of the 2th root (square root) of 16 is 4.
% Example: the integer part of the 3rd root (cubic root)  of  8 is 2.
% Example: the integer part of the 4th root               of 16 is 2.
% Example: the integer part of the 4th root               of 15 is 1.

nthRoot( N, K, R ):-  between( ... ...
```

26. Complete the following predicate in prolog.

```
% allSSSS(L) (allSquareSummingSubSequences) ===
%     "Given a sequence of positive integers L, write all non-empty subsequences of L
%      whose sum is a perfect square, in the following format":
% ?- allSSSS([6,3,4,5,6,9,8,5,2,3,4]).
% 9-[6,3]
% 49-[3,4,5,6,9,8,5,2,3,4]
% 4-[4]
% 9-[4,5]
% 9-[9]
% 9-[2,3,4]
% 4-[4]

allSSSS(L):- ...   write(Sum-SS), ....  .
```