```
%4.
%juntar(?Lista1,?Lista2,?Lista3)
juntar([], YS, YS).
juntar([X|XS], YS, [X|Res]) :- juntar(XS, YS, Res).

%5. I.
%last(?L, ?U)
last(L, U) :- append(_, U, L).

%II.
%reverse(+L, -L1)
reverse([], []).
reverse([L|Ls], Y) :- reverse(Ls, Y2), append(Y2, [L], Y).

%IV.
%pertenece(?X, +L).
pertenece(X, L) :- append(_, [X|_], L).

%6.
eslista([]).
eslista([_|_]).
%aplanar(+Xs, -Ys). y si x no es una lista que pasa?
aplanar([], []).
aplanar([X|Xs], Ys) :- eslista(X), aplanar(X, X2), aplanar(Xs, Y2), append(X2, Y2 ,Ys).
aplanar([X|Xs], [X|Ys]) :- \+ is_list(X), aplanar(Xs, Ys).

%7. I.
%palindromo(+L, ?L1).
palindromo(L , L1) :- reverse(L, Lr), append(L, Lr, L1).

%8. I.
%interseccion(+L1, +L2, -L3).
interseccion([], _, []).
interseccion([X|Xs], Ys,[X|R]) :- member(X, Ys), interseccion(Xs, Ys, R), \+ member(X, R).
interseccion([X|Xs], Ys,R) :- \+ member(X, Ys), interseccion(Xs, Ys, R).

partir(0, L, [], L).
partir(N, [L1|L], [L1|L1s], L2) :- N2 is N-1, partir(N2, L, L1s, L2), append(L1s, L2, L).

%II.
%borrar(+ListaOriginal, +X, -ListaSinXs).
borrar([], _, []).
borrar([X|Xs], X, R) :- borrar(Xs, X, R).
borrar([Y|Xs], X, [Y|R]) :- Y \= X, borrar(Xs, X, R).

%III.
%sacarDuplicados(+L1, -L2).
sacarDuplicados([L1|L1s], L2) :- member(L1, L2), sacarDuplicados(L1s, L2).
```

```
sacarDuplicados([L1|L1s], [L1|L2]) :- \+ member(L1, L2), sacarDuplicados(L1s, L2).

%IV.
%permutacion(+L1, ?L2). usando insertar
insertar(X, [], [X]).
insertar(X, [Y|Ys], [X, Y|Ys]).
insertar(X, [Y|Ys], [Y|Zs]) :- insertar(X, Ys, Zs).

permutacion([], []).
permutacion([X|Xs], Ys) :- permutacion(Xs, Ys2), insertar(X, Ys2, Ys).

%V.
%reparto(+L, +N, -LListas)
reparto(L, 1, [L]).
reparto(L, N, [Ls|Llista]) :- N > 1, N1 is N-1, contiene(Ls, L), append(Ls, Resto, L),
reparto(Resto, N1, Llista).

contiene([],_).
contiene([X|Xs], [X|Ys]) :- contiene(Xs, Ys).

%VI. repartoSinVacías(+L, -LListas)
contiene2([X], [X|_]).
contiene2([X|Xs], [X|Ys]) :- contiene2(Xs, Ys).

novacia([_|_]).

repartoSV(L, [L]) :- novacia(L).
repartoSV(L, [Ls|Llista]) :- contiene2(Ls, L), append(Ls, Resto, L), repartoSV(Resto, Llista).

%9. elementosTomadosEnOrden(+L,+N,-Elementos) NO ANDA

elementosTomadosEnOrden(_, 0, []).
elementosTomadosEnOrden(Xs, N, L) :- elems(Xs, X), append2(Xs, X, Resto), N1 is N-1,
        elementosTomadosEnOrden2(Resto, N1, L2, X), append(X, L2, L).

elementosTomadosEnOrden2(_, 0, [], _).
elementosTomadosEnOrden2([X|Xs], N, [X|L], Elem) :- N > 0, X > Elem, N1 is N-1,
        elementosTomadosEnOrden2(Xs, N1, L, X).
elementosTomadosEnOrden2([X|Xs], N, L, Elem) :- N > 0, Elem > X,
        elementosTomadosEnOrden2(Xs, N, L, Elem).

elems([X|_], X).
elems([_|Xs], Elem) :- elems(Xs, Elem).

append2(Xs, E, Resto) :- append(L, Resto, Xs), append(_,[E], L).

%10.
desde(X,X).
```

```
desde(X,Y) :- N is X+1, desde(N,Y).
%I. desde(+X, -Y) explota pa la mierda
%II. desde2(+X, ?Y).

%desde2(X,X).
%desde2(X,Y) :- Y > X, N is X+1, desde2(N,Y). %deberia terminar pero no lo hace

%desde2(+X, +Y, -R)
desde2(X, Y, X) :- nonvar(Y), Y > X.
desde2(X, Y, X) :- var(Y).
desde2(X, Y, R) :- var(Y), X1 is X+1, desde2(X1, Y, R).
desde2(X, Y, R) :- nonvar(Y), Y > X, X1 is X+1, desde2(X1, Y, R).

%12. Arboles
%nil.
%bin(I, V, D).

vacio(nil).
novacio(bin(_, _, _)).

raiz(bin(_, V, _), V).

altura(nil, 0).
altura(bin(I, _, D), X) :- altura(I, Y), altura(D, Z), max(Y, Z, R), X is R+1.

max(X, Y, X) :- X >= Y.
max(X, Y, Y) :- Y > X.

cantidadNodos(nil, 0).
cantidadNodos(bin(I, _, D), Res) :- cantidadNodos(I, X), cantidadNodos(D, Y), Res is
X+Y+1.

%13.
%I. inorder(+AB, -Lista)
inorder(nil, []).
inorder(bin(I, V, D), L) :- inorder(I, L1), inorder(D, L2), append(L1, L2, L3), append(L3, [V], L).

%II. arbolConInorder(+Lista, -AB). OBTIENE TODOS LOS AB POSIBLES
arbolConInorder([], nil).
arbolConInorder(Ls, bin(I, V, D)) :- append(Ls1, [V], Ls), append(X, Y, Ls1),
arbolConInorder(X, I), arbolConInorder(Y, D).

%III. aBB(+T)
aBB(nil).
aBB(bin(nil, _, nil)).
aBB(bin(I, V, D)) :- raiz(I, X), raiz(D, Y), V >= X, Y > V.

%IV. aBBInsertar(+X, +T1, -T2)
```

```
aBBInsertar(X, nil, bin(nil, X, nil)).

aBBInsertar(X, bin(I,V,D), bin(I, V, T2)) :- aBB(bin(I,V,D)), X > V, aBBInsertar(X, D, T2).
aBBInsertar(X, bin(I,V,D), bin(T2, V, D)) :- aBB(bin(I,V,D)), V >= X, aBBInsertar(X, D, T2).

%GENERATE AND TEST
%14. coprimos(-X,-Y)
coprimos(X, Y) :- desde(1, N), suman(Y, X, N), gcd(X,Y) =:= 1.

suman(X, Y, S) :- S1 is S-1, between(1, S1, X), Y is S-X.


%15
%I. cuadradoSemiLatino(+N, -XS)
largo([], 0).
largo([_|Ls], N) :-N > 0, N1 is N-1, largo(Ls, N1).


suman([], 0).
suman([L|Ls], X) :- between(0, X, L), X1 is X-L, suman(Ls, X1).

cuadradoSemiLatino(N, [L|Ls]) :- desde(0, X), largo([L|Ls], N) ,largo(L, N), suman(L, X),
cuadradoSemiLatinoAux(X, N, Ls).

cuadradoSemiLatinoAux(_, _, []).
cuadradoSemiLatinoAux(X, N, [L|Ls]) :- largo(L, N), suman(L, X),
cuadradoSemiLatinoAux(X, N, Ls).

%II. cuadradoMagico(+N, -Xs).
cuadradoMagico(N, Xs) :- cuadradoSemiLatino(N, Xs), columnaSuma(Xs, X, 1), between(1,
N, Y), columnaSumaIgual(Xs, X, Y).

%between(1,N, X), iesimo(X, Xs, L).
columnaSuma([], 0, _).
columnaSuma([L|Ls], X, I) :- iesimo(I, L, Y), columnaSuma(Ls, K, I), X is Y + K.

columnaSumaIgual(Xs, X, Y) :- columnaSuma(Xs, X2, Y), X is X2.

%16.
%.I. esTriángulo(+T)
esTriángulo(tri(A,B,C)) :- A1 is B+C, A2 is abs(B-C),  A1 > A, A > A2,
                           B1 is A+C, B2 is abs(A-C),  B1 > B, B > B2,
                           C1 is B+A, C2 is abs(A-B),  C1 > C, C > C2.

%II. perímetro(?T,?P)
perimetro(tri(A,B,C), P) :- ground(tri(A,B,C)), esTriángulo(tri(A,B,C)), P is A+B+C.
perimetro(tri(A,B,C), P) :- not(ground(tri(A,B,C))), generarTri(tri(A,B,C), P), P is A+B+C.
```

```
generarTri(tri(A,B,C), P) :- desde2(1, P, A), between(1, A, B), between(1, B, C), esTriángulo(tri(A,B,C)).

%III. triángulo(-T)
triangulo(T) :- desde(1, P), perimetro(T, P).


%19.  corteMásParejo(+L,-L1,-L2)
corteMásParejo(L, I, D) :- append(I, D, L), sumlist(I, X), sum_list(D, Y), K is abs(X-Y), not(otroCorte(L, K)).

otroCorte(L, K):- append(I, D, L), sumlist(I, X), sum_list(D, Y), Z is abs(X-Y), K > Z.

%20.
eP(Y) :- Y > 500.
e2P(X) :- eP(X), not(e3P(X)).
e3P(X) :-  between(1, X, Y), eP(Y), X > Y.

%22.
% s(X) y e(X), siendo X el dato producido o ingresado.
% accion(+Proceso, ?Efecto, -SiguienteProceso), que es verdadero cuando el
% proceso Proceso puede realizar una acción con efecto Efecto, siendo ProcesoSiguiente el proceso al que
% reduce luego de realizar esa acción.

% simula(+Proceso1,+Proceso2)
% simula(Pro1, Pro2) :- accion(Pro1, s(X), Sig1), accion(Pro2, s(X), Sig2), simula(Sig1, Sig2).
% simula(Pro1, Pro2) :- accion(Pro1, e(X), Sig1), accion(Pro2, e(X), Sig2), simula(Sig1, Sig2).

% EJERCICIO RARO

%23. Cosas Dadas:
% aristas (a,b) y (b,a) iguales
% esNodo(+G,?X), dando un grafo nos dice si X es o no un nodo de G
% esArista(+G,?X,?Y) idem con aristas
% los nodos son iguales sii unifican

% I. caminoSimple(+G,+D,+H,?L)
% caminoSimple(G, H, H, []). HAY ALGUNA FORMA DE CREER LA LISTA VISITADOS SIN TENER QUE PASARLA COMO PARAMETRO?
% caminoSimple(G, D, H, [D|L]) :- avanzar(G ,D, Vecino), append(Vecino, Resto, L), caminoSimple(G, Vecino, H, Resto).

% avanzar(G, N, V) :- esArista(G, N, V).
```

```
% II. caminoHamiltoniano(+G,?L). ES ACEPTABLE DAR REPETIDOS? NECESITARIA UNA LISTA DE PARES DE NODOS PROBADOS
% caminoHamiltoniano(G, L) :- esNodo(G, D), esNodo(G, H), caminoSimple(G, D, H, L), todosNodos(G, L1, []), tieneTodos(G, L).

% todosNodos(G, [], L2) :- not(esNodo(G, N), not(member(L2, N))).
% todosNodos(G, L1, L2) :- esNodo(G, N), not(member(L2, N)), append([N], Resto, L), todosNodos(G, L1, [N|L2])

% tieneTodos(_, []). CUANDO DEFINO UNA FUNCION AUXILIAR, SI LLAMAS A SU CB NO VALE PERO PARA EL RECURSIVO SI?
% tieneTodos(L1, [L|Ls]) :- member(L, L1), tieneTodos(L1, Ls).

% III.
% esConexo(+G).
% esConexo(G) :-esNodo(G, Node), !, dfs(G, L, Node ,[]), todosNodos(G, L1, []), tieneTodos(L, L1).
%
% dfs(L = []) :- tieneTodos. good
% dfs(G, [N|L], N,Vis) :- not(tieneTodos(Vis blabla)), avanzarConTodos(G, N, V), not(member(V, Vis)), dfs(G, L, N, [V|Vis]).
%
% EL RECOMENDADO ERA MEJOR XD
% IV. esEstrella(+G)                    chequear si existe camino a todos los demas o si esta en todas las aristas?
% esEstrella(G):- esConexo(G), todosLosNodos(para cada nodo) ,not(esNodo(G, N2), caminoSimple(G,N,N2))
%
%
% 24.
% nil y bin(I, V, D)
% I. arbol(-A)
arbol(nil).
arbol(bin(I, _, D)) :- arbol(I), arbol(D). %Como soluciono que solo explore una rama?

%II.
nodoEnO(nil,_).
nodoEnO(bin(I, V,D), L) :- member(V, L), nodoEnO(I, L), nodoEnO(D, L).

%III. este era en realidad sinRepEn
nodosEn(nil, _).
nodosEn(bin(I, V, D), Ls) :-  quitar(V, Ls, Res), nodosEnResto(I, Res, Resto), nodosEn(D, Resto).

nodosEnResto(nil, R, R).
nodosEnResto(bin(I, V, D), Ls, R) :- quitar(V, Ls, Res), nodosEnResto(I, Res, R), nodosEn(D, R).
```

```
quitar(X, Ls, Res) :- append(R3, R2, Ls), append(R1, [X], R3), append(R1, R2, Res).
```