```
% ejercicios practica

%concatenar(?L1, ?L2, ?L3)
concatenar([], L2, L3) :- L2 = L3.
concatenar([E|L1], L2, [E3|L3]) :- E = E3, concatenar(L1, L2, L3).


%last(?L, ?U)
last([E],U) :- E = U.
last([_|L], U) :- last(L,U).

%reverse(+L, -L2)
reverse([], []).
reverse(_, []) :- false.
reverse([], _) :- false.
reverse([E|L], L2) :- reverse(L, Rec), concatenar(Rec, [E], L2).

%maxlista(+L, -M)
maxLista([E], E).
maxLista([E|L], M) :- maxLista(L, U), M is max(E, U).

%prefijo(?P, +L)
prefijo([], _).
prefijo([E|P], L) :- nth1(1,L,Contenido), Contenido = E, concatenar([E], Rec, L), prefijo(P, Rec).



%sufijo(?S, +L)
sufijo(L, L).
sufijo(S, [_|L]) :- sufijo(S,L).

%sublista(?S, +L)
sublista(S, L) :- prefijo(P, L), sufijo(S,P).

%pertenece(?X, +L)
pertenece(X, [X|_]).
pertenece(X, [_|L]) :- var(X), pertenece(X,L).
pertenece(X, [E|L]) :- nonvar(X), X \= E, pertenece(X,L).

%aplanar(+X,-Y)
aplanar([], []).
aplanar([E|X],Y) :- aplanar(E, L), aplanar(X, Rec), concatenar(L, Rec, Y).
aplanar(E,Y) :- not(is_list(E)), Y = [E].



%palindromo(+L, -L1)
palindromo(L, L1) :- reverse(L, Rev), concatenar(L, Rev, L1).

%doble(?L, ?L2)      MAL
doble([], []).
doble([E|L], L2) :- doble(L,Rec), concatenar([E,E], Rec , L2).
```

```
%iesimo(?I, +L, -X)f
iesimo(1,[E|_],X) :- X is E.
iesimo(I,[_|L],X) :-  iesimo(R, L, X), I is R+1.

%desde(+X,?Y)
desde2(X,Y) :- var(Y), desde(X,Y).
desde2(X,Y) :- nonvar(Y), X<Y.

desde(X, X).
desde(X, Y) :- N is X+1, desde(N, Y).

%interseccion(+L1, +L2, -L3)
interseccion([], _, []).
interseccion([E|L], L2, L3) :- pertenece(E, L2), not(pertenece(E, L)), interseccion(L, L2, Rec), concatenar(Rec,
[E], L3).
interseccion([E|L], L2, L3) :- pertenece(E, L2), pertenece(E, L), interseccion(L, L2, Rec) ,concatenar([], Rec, L3).
interseccion([E|L], L2, L3) :- not(pertenece(E, L2)), interseccion(L, L2, Rec) ,concatenar([], Rec, L3).

interseccionEnOrder(L,L2,L3) :- reverse(L,Rev), interseccion(Rev,L2,L3).

%split(+N, +L, ?L1, ?L2)
split(0, L, L1, L2) :- L1 = [], L2 = L.
split(N, [E|L], L1, L2) :- Pred is N-1, split(Pred, L, Rec1, Rec2), concatenar([E], Rec1, L1), L2 = Rec2.


%borrar(+ListaOriginal, +X, -ListaSinXs)
borrar([],_,[]).
borrar([E|L],X,L2) :- E \= X, borrar(L,X,Rec), concatenar([E],Rec,L2).
borrar([X|L],X,L2) :- borrar(L,X,L2).

%sacarDuplicados(+L1, -L2)
sacarDuplicados(L,L2) :- interseccionEnOrder(L,L,L2).


%reparto(+L, +N, -LListas)
reparto(L,1,LListas) :- [L] = LListas.
reparto(L,N,LListas) :- N \= 1, length(L,Size), between(0,Size,X), split(X,L,L1,L2), Pred is N-1,
reparto(L1,Pred, Rec), concatenar(Rec, [L2], LListas).


%sinVacias(+L,?L2)
sinVacias([],[]).
sinVacias([E|L],L2) :- pertenece([],E), sinVacias(L,L2).
sinVacias([E|L],L2) :- not(pertenece([],E)), sinVacias(L,Rec), concatenar([E],Rec, L2).

%repartoSinVacias(+L, -LListas)
%repartoSinVacias(L, LListas) :- length(L,Size), between(1,Size,X), reparto(L,X,Res), sinVacias(Res,Dup),
sacarDuplicados(Dup, LListas).
repartoSinVacias(L, LListas) :- length(L,Size), between(1,Size,X), reparto(L,X,Res), not(pertenece([], Res)),
sacarDuplicados(Res, LListas).
```

```
%intercalar(+L1, +L2, ?L3)
intercalar(L1, [], L3) :- L3 = L1.
intercalar([], L2, L3) :- L3 = L2.
intercalar([E1|L1], [E2|L2], L3) :- intercalar(L1,L2,Rec), concatenar([E1,E2], Rec, L3).

bin(_,_,_).

altura(nil,0).
altura(bin(Izq,_,Der), N) :- altura(Izq, AltIzq), altura(Der, AltDer), MaxAlt is max(AltDer, AltIzq), N is
MaxAlt+1.


%inorder(+AB,-Lista)
inorder(nil, []).
inorder(bin(Izq,V,Der), L) :- inorder(Izq, Rec1), inorder(Der, Rec2), concatenar(Rec1, [V], Un), concatenar(Un,
Rec2, L).


%arbolConInorder(+Lista,-AB)
arbolConInorder([],nil).
arbolConInorder(Lista,AB) :- length(Lista,Size), between(1,Size,X), split(X, Lista, Izq, Der), last(Izq, Raiz),
concatenar(RecIzq, [Raiz], Izq), arbolConInorder(RecIzq, ABIzq), arbolConInorder(Der, ABDer), AB =
bin(ABIzq, Raiz, ABDer).

%coprimos(-X,-Y)
coprimos(X,Y) :- desde(1,A), between(1,A,B), 1 is gcd(A,B), X = A, Y = B.

%fliplength(?Longitud, ?Lista)
fliplength(N, L) :- length(L, N).

%matriz(?Matriz, ?Filas, ?Columnas)
matriz(M, F, C) :- length(M, F), maplist(fliplength(C), M).

%contenido(+?Tablero, ?Fila, ?Columna, ?Contenido)
contenido(Tablero, Fila, Columna, Contenido) :- nth1(Fila, Tablero, L), nth1(Columna, L, Contenido).

sumaLista(R,1,L) :- L = [R].
sumaLista(R,Size,L) :- Size>1, between(0,R,X), Pred is Size-1, sumaLista(X,Pred,Rec), Res is R-X,
concatenar([Res], Rec, L).

%cuadradoSemiLatino(+N, -XS)
cuadradoSemiLatino(N, XS) :- desde2(0,X), matriz(XS,N,N), maplist(sumaLista(X,N), XS).

%sacarUnoExacto(+L,+X,?R)
sacarUnoExacto([E|L], E, R) :- R = L.
sacarUnoExacto([E|L], E, R) :- pertenece(E,L), sacarUnoExacto(L,E,Rec), concatenar([E], Rec, R).
sacarUnoExacto([A|L], B, R) :- A \= B, sacarUnoExacto(L,B,Rec), concatenar([A], Rec, R).

cadena([],_,0).
cadena([E|S],A,N) :- N \= 0, member(E, A), Pred is N-1, cadena(S,A,Pred).
```

```
separar([E|[E2|S]],S1,A,B,S2) :- E = A, E2 = B, S = S2, S1 = [].
separar([E|S],S1,A,B,S2) :- separar(S, NewS1, A, B, S2), concatenar([E], NewS1, S1).

listaConOtroOrden([], []).
listaConOtroOrden([E|L1], L2) :- member(E, L2), sacarUnoExacto(L2,E,SinUno), listaConOtroOrden(L1,
SinUno).

diferenciaConjuntos([], _, []).
diferenciaConjuntos([E|L], L2, Res) :- member(E,L2), diferenciaConjuntos(L,L2,Res).
diferenciaConjuntos([E|L], L2, Res) :- not(member(E,L2)), diferenciaConjuntos(L,L2,Rec), concatenar([E], Rec,
Res).

receta(pizzaMozzarellaCasera, 30, [harina, levadura, sal, aceite, tomate, mozzarella]).
receta(otraPizzaMozzarellaCasera, 30, [levadura, sal, aceite, harina, tomate, mozzarella]).
ingrediente(harina).
ingrediente(levadura).

sonSimilares(P1, P2) :- receta(P1, Tiempo1, Ingredientes1), receta(P2, Tiempo1, Ingredientes2),
listaConOtroOrden(Ingredientes1, Ingredientes2).

queMeFalta(I, P, F) :- receta(P, _, Ingredientes), diferenciaConjuntos(Ingredientes, I, F).

tengo([]).
tengo([E|L]) :- ingrediente(E), tengo(L).

puedoPreparar(Plato) :- receta(Plato, _, Ingredientes), tengo(Ingredientes).

masRapido(P) :- receta(P, Tiempo1, _), puedoPreparar(P), not((receta(P2, Tiempo2, _), puedoPreparar(P2),
Tiempo2<Tiempo1)).
```