```
% ---------------------------------------- 1 ------------------------------
-------------

padre(juan, carlos). %juan es el padre de carlos, carlos es el padre de diego
y de daniel, juan es el abuelo de carlos y daniel
padre(juan, luis). %juan es el padre de luis
padre(carlos, daniel).
padre(carlos, diego).
padre(luis, pablo).
padre(luis, manuel).
padre(luis, ramiro).

abuelo(X,Y) :- padre(X,Z), padre(Z,Y). %X es el padre de z, z es es el papa de
y

%  i. ¾Cuál el resultado de la consulta abuelo(X, manuel)? -> padre(X,Y)
padre(Y,manuel), el papa de manuel es luis, luego padre (x,luis)
%es juan, por lo tanto la x se unifica a juan.



% ii. A partir del predicado binario padre, denir en Prolog los predicados
binarios: hijo, hermano y descendiente.

hijo(Y, X) :- padre(X, Y).
hermano(X, Y) :- padre(Z, X), padre(Z, Y), X\=Y.

descendiente(X, Y):-padre(Y, X).
descendiente(X, Y):-padre(Z, X), descendiente(Z, Y).

% descendiente(manuel, juan) → ☑ true
% descendiente(diego, juan) → ☑ true
% descendiente(juan, X) → ❌ (nadie, porque juan no tiene padres)



% iii. Dibujar el árbol de búsqueda de Prolog para la consulta
descendiente(Alguien, juan).
% iv. ¾Qué consulta habría que hacer para encontrar a los nietos de juan? ?-
abuelo(juan Y).

% v. ¾Cómo se puede denir una consulta para conocer a todos los hermanos de
pablo? ?- hermano(pablo, Y).
```

```
% vi. Considerar el agregado del siguiente hecho y regla:
ancestro(X, Y):- padre(X, Y).
ancestro(X, Y) :- ancestro(Z, Y), padre(X, Z).
% y la base de conocimiento del ítem anterior.
% vii. Explicar la respuesta a la consulta ancestro(juan, X). ¾Qué sucede si
se pide más de un resultado?->se obtiene el arbol genealogico de
%juan, ya qeues abuelo y padre de todos, pero pone que juan es su peropio
ancestro



% viii. Sugerir una solución al problema hallado en los puntos anteriores
reescribiendo el programa de ances




% ---------------------------------------- 3 ------------------------------
--------------



natural(0).
natural(suc(X)) :- natural(X).

menorOIgual(X, X) :- natural(X).
menorOIgual(X, suc(Y)) :- menorOIgual(X, Y).



% i. Explicar qué sucede al realizar la consulta menorOIgual(0,X).
% natural(0).
% natural(suc(X)) :- natural(X).

% menorOIgual(X, suc(Y)) :- menorOIgual(X, Y).
% menorOIgual(X, X) :- natural(X).



% ii. Describir las circunstancias en las que puede colgarse un programa en
Prolog. Es decir, ejecutarse innitamente sin arrojar soluciones.



% iii. Corregir la denición de menorOIgual para que funcione adecuadamente.->
aca le estoy pidiendo al pred menorOIgual que me diga todos los numeros
% mayores o iguales a 0, por como esta definida la funcion es 0, suc(0),
suc(suc(0)), suc(suc(suc(0)))...
```

```
%  ?- menorOIgual(0, X).
% X = 0 ;
% X = suc(0) ;
% X = suc(suc(0)) ;
% X = suc(suc(suc(0))) ;
% X = suc(suc(suc(suc(0)))) ;
% X = suc(suc(suc(suc(suc(0))))) ;
% X = suc(suc(suc(suc(suc(suc(0)))))) ;
% X = suc(suc(suc(suc(suc(suc(suc(0))))))) ;
% X = suc(suc(suc(suc(suc(suc(suc(suc(0)))))))) ;
% X = suc(suc(suc(suc(suc(suc(suc(suc(suc(0))))))))) ;
% X = suc(suc(suc(suc(suc(suc(suc(suc(suc(suc(...)))))))))) ...

% ---------------------------------------- 4 Listas ----------------------
---------------------

juntar([],[],[]). %C1
juntar([],L,L). %C2
juntar(L,[],L). %C3
juntar([H1|L1],L2,[H1|L3]):- juntar(L1,L2,L3).
%tomo el primer elemento de l1, lo pongo al principio de mi lista return l3 /
[H1|L3]
% luego sigo juntando el resto hasta llegar a la lista l2, ¿cuando arranca a
copiar la segunda lista?
% CUANDO SE TERMINA DE COPIAR TODA LA LISTA L1, O SEA HASTA QUE L1 QUEDA VACIA
[]
%aca cae en el caso C2,“Para juntar [H|T] con L2, el resultado será una lista
que empieza con H y sigue con lo que salga de juntar T con L2.”
%Y eso, por sí solo, ya garantiza que el primer elemento (H) se mantiene por
delante del resto del resultado.

% ---------------------------------------- 5 Listas ----------------------
---------------------
 % +X -> DEBE ESTAR INSTANCIADO
 % -X -> NO DEBE ESTARLO
 % ?X -> PUEDE O NO ESTARLO

% instanciada -> ya tiene un valor, si no lo esta es que esta libre, algo la
tiene que unificar.
% "debe estar instanciada" es que necesito que tenga un valor para poder
avanzar.

% Denir los siguientes predicados sobre listas usando append:
% i. last(?L, ?U), donde U es el último elemento de la lista L.
last(L, U) :- append(_, [U], L).
```

```
% ii. reverse(+L, ?R), donde R contiene los mismos elementos que L, pero en
orden inverso.
% Ejemplo: reverse([a,b,c], [c,b,a]).
% Mostrar el árbol de búsqueda para el ejemplo dado.

reverse([],[]).
reverse([H|T], R) :- reverse(T, RT), append(RT, [H], R).

% iii. prefijo(?P, +L) donde P es prejo de la lista L.
% iv. sufijo(?S, +L), donde S es sujo de la lista L.
% v. sublista(?S, +L), donde S es sublista de L.
% vi. pertenece(?X, +L), que es verdadero sii el elemento X se encuentra en la
lista L. (Este predicado ya viene
% denido en Prolog y se llama member).

%----------------------------- 6 ---------------------------.
% idea = se que puedo tener la situacion de que la cabeza es un atomo o una
lista;
%si es un atomo -> me aseguro que no sea una lista y se que H va a pertenecer
tanto como a la
%primer lista como a la segunda, luego aplano el resto de las listas.
%si es lista -> necesito aplanar la cabeza y la cola, esto si lo junto me va a
generar la lista L que quiero (append con HA y TA).
aplanar([], []).
aplanar([H|T], [H|R]):- not((is_list(H))), aplanar(T,R).
aplanar([H|T], L):- is_list(H), aplanar(H, HA), aplanar(T, TA), append(HA, TA,
L).



%---------------------------- 7 -----------------------------

interseccion([], _, []).
interseccion([H|T], L2, [H|R]):- member(H, L2), interseccion(T,L2,R).
interseccion([H|T], L2, R):- not((member(H, L2))),interseccion(T,L2,R).

partir1(N, L, L1, L2):- length(L1, N), append(L1, L2, L).
partir2(N, L, L1, L2):- length(L, M), length(L1, N), MmN is M-N, length(L2,
MmN), append(L1, L2, L).

borrar([], _, []).
borrar([H|T], H, L):- borrar(T, H, L).
borrar([H|T], N, [H|L]):- H\=N, borrar(T, N, L).
```

```
sacarDuplicados([], []).
sacarDuplicados([H|T], [H|L]):- not((member(H,T))), sacarDuplicados(T,L).
sacarDuplicados([H|T], L):- member(H,T), sacarDuplicados(T,L).

permutacion([],[]).
permutacion(L1, L2):- length(L1, N), length(L2, N), interseccion(L1, L2, L1).

reparto([], 0, []).
reparto(L, N, [X|R]):- N>=1, N1 is N-1, append(X, Lrec, L), reparto(Lrec, N1,
R).


hayListasVacias(L):- member([],L).

repartoSinVacias([],[]).
repartoSinVacias(L,[H|R]):- append(H, Lrec, L), H\=[],  repartoSinVacias(Lrec,
R).


%---------------------- 8 ---------------

subsecuencia([],[]).
subsecuencia([X|Xs],[X|Zs]):- subsecuencia(Xs, Zs).
subsecuencia([_|Xs], Zs):- subsecuencia(Xs, Zs).

parteQueSuma(L,S,P):- subsecuencia(L, P), sum_list(P, S).

%---------------------- 9 ---------------

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

%---------------- 10 --------------


intercalar([], L2, L2).
intercalar(L1, [], L1).
intercalar([X|Xs], [Y|Ys], [X, Y | R]):- intercalar(Xs, Ys, R).


%---------------- 11 --------------
```

```
vacio(nil).

raiz(nil, nil).
raiz(bin(_, R, _), R).

altura(nil, 0).
altura(bin(I, _, D), A):- altura(I, AI), altura(D, AD), A is max(AI, AD)+1.

cantidadDeNodos(nil,0).
cantidadDeNodos(bin(I,_,D), N):- cantidadDeNodos(I, IN),
                                 cantidadDeNodos(D, DN),
                                 N is IN+DN+1.

%--------------- 12 -------------
inorder(nil,[]).
inorder(bin(I, V, D),L):- inorder(I, LI),
                          inorder(D, LD),
                          append(LI,[V],LIV),
                          append(LIV, LD, L).


arbolConInorder([],nil).
arbolConInorder(L,AB):- append(LI, [X|LD], L), arbolConInorder(LI, AI),
arbolConInorder(LD, AD), AB= bin(AI, X, AD).

aBB(nil).
aBB(B):-inorder(B, L), msort(L, L).

insertar(E, nil, bin(nil, E, nil)).
insertar(E, bin(BI, R, BD), bin(A2, R, BD)) :- insertar(E, BI, A2).
insertar(E, bin(BI, R, BD), bin(BI, R, A2)) :- insertar(E, BD, A2).

aBBInsertar(X, T1, T2) :- insertar(X, T1, T2), aBB(T2).
```