# Solución de la Evaluación

## Ejercicio 2

Se presentan 2 soluciones alternativas:
1-

```
/*
secuencia(+Dist,+N,?Sec) <-- Sec es una secuencia de N naturales que comienza en 0 y tal que las distancias entre todos los partes de puntos se encuentran en el conjunto con repeticiones Dist
*/

secuencia(Dist,N,Candidata):-
        maxL(Dist,DMax),
        N2 is N-2,
        DM1 is DMax-1,
        generaCandidata(1,DM1,CandidataParcial,N2),
        append([0|CandidataParcial],[DMax],Candidata),
        chequeaCandidata(Candidata,Dist).

% maxL(+L,?M) <-- M es el máximo de la lista L de Naturales


maxL([X|Xs],M):-
        maxL(Xs,Mpar),
        max(X,Mpar,M).
maxL([],0).

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

/*
generaCandidata(I,F,Secuencia,N) <-- Secuencia es una secuencia de N naturales no repetidos con valores entre I y F
*/



generaCandidata(I,J,[I|R],N):-
        N > 0,
        I1 is I+1,
        I1 =< J,
        N1 is N-1,
        generaCandidata(I1,J,R,N1).

generaCandidata(I,J,R,N):-
        N > 0,
        I1 is I+1,
        I1 =< J,
        generaCandidata(I1,J,R,N).

generaCandidata(_I,_J,[],0).


/* chequeaCandidata(Secuencia,Distancias) <-- Secuencia conotiene un conjunto de enteros cuyas distancias 2 a 2 se encuentran en Distancias
*/


chequeaCandidata(Candi,Sec):-
        findall(d(C),(member(A,Candi),member(B,Candi),A<B, C is B-A),Dist),
        representadas(Dist,Sec).
```

```
representadas([d(D)|R],Sec):-
        select(D,Sec,Sec1),
        representadas(R,Sec1).
representadas([],[]).

select(X,[X|R],R).
select(Y,[X|R],[X|R1]):-
        select(Y,R,R1).
```

---

2

```
%
% pdp(+DX, +N, -X)
%
pdp(DX, N, [0,M|X]) :-
        max(DX, M, R),
        N2 is N-2,
        combinacion(R, X, N2),
        conjunto_distancia([0,M|X], DX1),
        iguales(DX1, DX).

%
% combinacion(+X, -Y, +N) es verdadero si Y es un subconjunto de N elementos de X
%
combinacion(_, [], 0).

combinacion([X|Xs], [X|Ys], N) :-
        N1 is N-1,
        combinacion(Xs, Ys, N1).

combinacion([_|Xs], Ys, N) :-
        combinacion(Xs, Ys, N).

iguales(L1, L2) :-
        ordenar(L1, SL1),
        ordenar(L2, SL1).

conjunto_distancia([], []).
conjunto_distancia([X|Xs], Y) :-
        distancias_a_elemento(X, Xs, D),
        conjunto_distancia(Xs, Z),
        concatenar(D, Z, Y).


distancias_a_elemento(_, [], []).
distancias_a_elemento(X, [Y|Ys], [D|Ds]):-
        distancia(X, Y, D),
        distancias_a_elemento(X, Ys, Ds).

distancia(X, Y, D) :-
        X >= Y,
        D is X-Y.

distancia(X, Y, D) :-
        Y > X,
        D is Y-X.

largo([],0).
largo([_|Xs], L):-
        largo(Xs, L1),
        L is L1 + 1.

concatenar([], L, L).
concatenar([X|Xs], L, [X|Ys]) :-
        concatenar(Xs, L, Ys).
```

```
max([X], X, []).

max([X|Xs], X, Xs) :-
        max(Xs, M, _),
        X >= M.

max([X|Xs], M, [X|Ys]) :-
        max(Xs, M, Ys),
        M > X.

ordenar([],[]).

ordenar(Xs, [M|SYs]) :-
        max(Xs, M, Ys),
        ordenar(Ys, SYs).
```

## Ejercicio 3

a. Para cada par de expresiones indique si existe una sustitución que las unifica. En caso afirmativo dé un unificador más general.

| | | | |
|---|---|---|---|
| i) | t([],a) | t([X\|Xs]) | No unifican |
| ii) | p(X) | p(f(X)) | No unifican |
| iii) | s(X,f(X),Y) | s(f(Z),Z,Z) | No unifican |
| iv) | a(f(Y),g(Z)) | a(X,V) | Θ={X/f(Y),V/g(Z)} |
| v) | b(f(Y), W, g(Z)) | b(V, X, V) | No unifican |
| vi) | c(a, X, f(g(Y))) | c(Z, h(Z,W), f(W)) | Θ={Z/a,X/h(a,g(Y)),W /g(Y)} |

b. Defina respuesta correcta y respuesta computada. Dé un ejemplo de un programa Prolog, una consulta y una respuesta correcta que no es computada.

i. Respuesta computada: Dado un programa lógico P y un objetivo G, llamamos respuesta computada a la sustitución para las variables de G resultante de aplicar resolución SLD a P U {G}
ii. Respuesta correcta: Dado un programa lógico P y una consulta C, una sustitución Θ para las variables en C es una respuesta correcta si P implica lógicamente CΘ.
iii. Sea P = {p(X):-q(a). q(X).}, C=p(Y). Θ={Y/a} es una respuesta correcta que no es computada.

c. Sea el siguiente programa lógico:

```
p(T,a).
p(S,S) :- r(S).
q(f(W,W),W).
q(a,f(b)).
r(Z).
a(X,Y) :- p(X,Y),q(X,f(Y)).
```

Construya el árbol SLD para el objetivo ← a(A,B).

← a(A,B)
- ← p(X1,Y1),q(X1,f(Y1)) --- {A/X1, B/Y1}
  - ← q(T1,f(a)) --- {X1/T1,Y1/a}
    - □ --- {T1/f(f(a),f(a))} / Θ = {A/f(f(a),f(a)),B/a}
  - ← r(S1),q(S1,f(S1)) --- {X1/S1,Y1/S1}
    - ← q(Z1,f(Z1)) --- {S1/Z1}
      - **x** --- falla

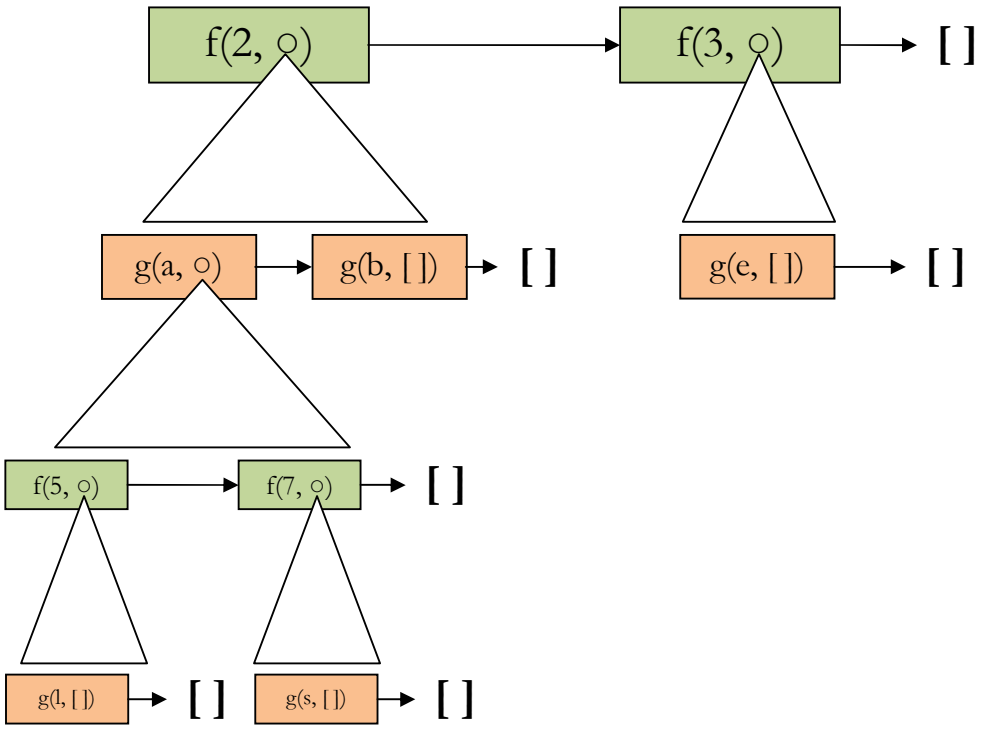
## Ejercicio 4

**Parte 1)**

Estructura = [f(2,[g(a, []),g(b,[])]), f(3,[g(d,[]),g(e,[])])] es una estructura válida.

En general:
- Estructura es una lista de elementos de la forma f(Clave, Valores).
- No se chequea la estructura de Clave, así que puede ser cualquier término.
- Valores es una lista de elementos de la forma g(Valor, SubEstructura).
- Tampoco se chequea la estructura de Valor, así que puede ser cualquier término.
- SubEstructura tiene la misma estructura que Estructura.

Una posible interpretación de la estructura es que se trate de una representación de un árbol finitario donde los nodos de niveles impares son Claves y los de niveles pares son Valores.




**Parte 2)**

La consulta

? pred([f(2,[g(a, []),g(b,[])]), f(3,[g(d,[]),g(e,[])])], Claves, Valores)

devuelve

Claves = [2]
Valores = [a]
;
Claves = [2]
Valores = [b]
;
Claves = [3]

```
Valores = [d]
;
Claves = [3]
Valores = [e]
;
No
```

**Parte 3, a)**

Para que el CUT sea verde:

- Si el objetivo fuera utilizar el programa con el 2do argumento sin instanciar, entonces las listas de Claves deberían tener un único elemento.
- Si el objetivo fuera utilizar el programa con el 2do argumento instanciado, entonces cada lista de Claves no debería repetir la misma Clave (una vez que encuentra la primera, no sigue buscando).

Debido a que la letra especifica que el programa puede utilizarse con el 2do argumento instanciado o no, nos quedamos con la solución más restrictiva (es decir, la primera).

**Parte 3, b)**

Estructura = [f(2,[g(a, []),g(b,[])]), f(3,[g(d,[]),g(e,[])])] es una estructura válida en el programa original. Al agregar el CUT, las soluciones son:

```
Claves = [2]
Valores = [a]
;
Claves = [3]
Valores = [d]
;
```

**Parte 3, c)**

El CUT es rojo ya que modifica el significado del programa.

**Parte 4)**

```
colectar(Estructura, Pares) :- findall((C, V), pred(Estructura, C, V), Pares).
```

## Ejercicio 5

```
?- subterm(S,a(Z,b([a,2,3]))).
        S = a(Z, b([a, 2, 3])) ;
        S = Z ;
        S = b([a, 2, 3]) ;
        S = [a, 2, 3] ;
        S = a ;
        S = [2, 3] ;
        S = 2 ;
        S = [3] ;
        S = 3 ;
        S = [] ;
        fail.
```

---

```
?- subterm(a,f(b,X)).
X = a ;
fail.

subterm(Term,Term).
subterm(Sub,Term):-
```

```
        compound(Term),
        Term=..[_F|Args],
        subtermList(Sub,Args).

subtermList(Sub,[Arg|_Args]):-
        nonvar(Arg),subterm(Sub,Arg).
subtermList(Sub,[_Arg|Args]):-
        subtermList(Sub,Args).
```