Observaciones:

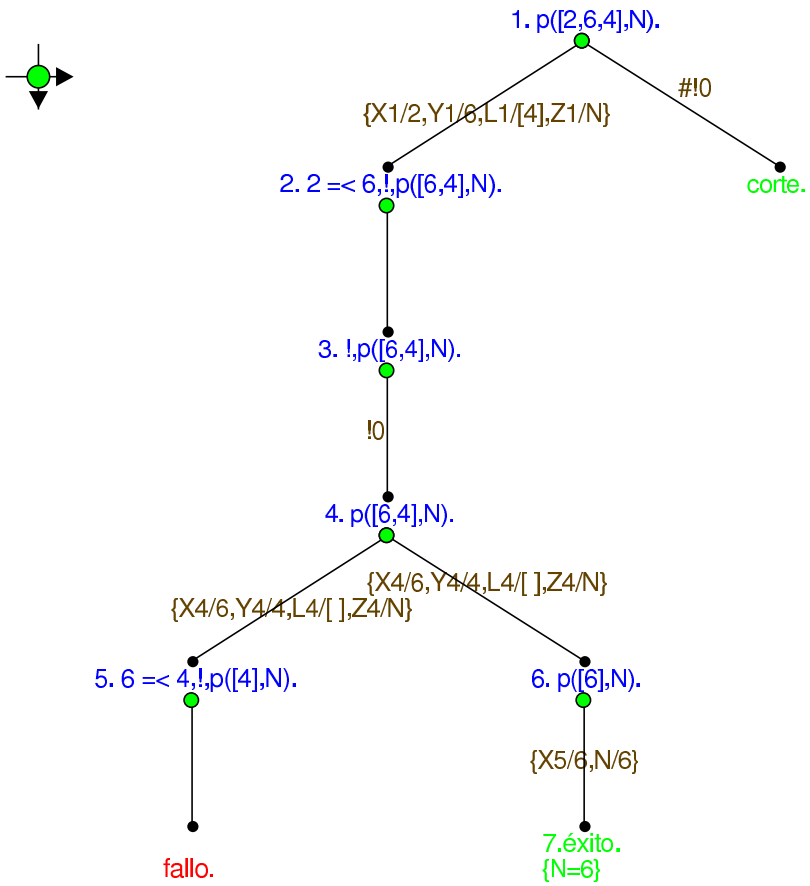
1. *En la evaluación se tendrá en cuenta la corrección, simplicidad y eficiencia de las respuestas.*
2. *Hay que describir las definiciones auxiliares (menos las del sistema).*

Ejercicio 1 (2.5 puntos) *Se considera el siguiente programa lógico*

```
p([X],X).
p([X,Y|L],Z) :-
   X =< Y, !,
   p([Y|L],Z).
p([X,Y|L],Z) :-
   p([X|L],Z).
```

1. *Escribir el árbol de resolución y las repuestas correspondientes al programa y a la pregunta*
   `?- p([2,6,4],N).`
2. *Dar una definición no recursiva de la relación* `p`.

Solución:
El árbol de resolución se representa en la figura La respuesta obtenida es `N = 6`.



La relación `p(L,X)` se verifica si `X` es el máximo elemento de la lista `L`. Una definición no recursiva de `p` es

```
p_1(L,X) :-
   member(X,L),
   not((member(Y,L), X < Y)), !.
```

Ejercicio 2 (2.5 puntos) *Definir la relación* `subconjunto_suma(+L1,+N,?L2)` *que se verifique si* `L2` *es un subconjunto de* `L1` *tal que la suma de los elementos de* `L2` *es* `N`*. Por ejemplo,*

```
?- subconjunto_suma([3,10,4,7,2,1],7,L).
L = [3, 4] ;
L = [4, 2, 1] ;
L = [7] ;
No
?- subconjunto_suma([1,2,3],0,L).
L = [] ;
No
?- subconjunto_suma([1,2,6],5,L).
No
```

Solución:
Presentaremos tres definiciones y comparemos su eficiencia.
La primera definición usa las relaciones `subconjunto` y `suma_lista`.

```
subconjunto_suma_1(L1,N,L2) :-
   subconjunto(L1,L2),
   suma_lista(L2,N).
```

La relación `subconjunto(+L1,?L2)` se verifica si `L2` es un subconjunto de `L1`. Por ejemplo,

```
?- subconjunto([a,b,c,d],[b,d]).
Yes
?- subconjunto([a,b,c,d],[b,f]).
No
?- subconjunto([a,b,c],L).
L = [a, b, c] ;
L = [a, b] ;
L = [a, c] ;
L = [a] ;
L = [b, c] ;
L = [b] ;
L = [c] ;
L = [] ;
No
```

```
subconjunto([],[]).
subconjunto([X|L1],[X|L2]) :-
   subconjunto(L1,L2).
subconjunto([_|L1],L2) :-
   subconjunto(L1,L2).
```

La relación `suma_lista(+L,?X)` se verifica si `X` es la suma de los elementos de la lista de números `L`. Por ejemplo,

```
? - suma_lista([1,3,5],X).
X = 9
```

```
suma_lista([],0).
suma_lista([X|L],Y) :-
   suma_lista(L,Y1),
   Y is X+Y1.
```

En la segunda definición se adapta la definición de `subconjunto` teniendo en cuenta la suma de sus elementos.

```
subconjunto_suma_2([],0,[]).
subconjunto_suma_2([X|L1],N,[X|L2]) :-
   N >= X,
   N1 is N-X,
   subconjunto_suma_2(L1,N1,L2).
subconjunto_suma_2([_|L1],N,L2) :-
   subconjunto_suma_2(L1,N,L2).
```

En la tercera definición se define de forma dinámica la relación `subconjuntos(+L1,+N.-S)` que se verifica si `S` es la lista de subconjuntos de `S` tales que la suma se sus elementos es `N`.

```
:- dynamic subconjuntos_suma_calculados_3/3.

subconjunto_suma_3(L1,N,L2) :-
   subconjuntos_suma_calculados_3(L1,N,S), !,
   member(L2,S).
subconjunto_suma_3(L1,N,L2) :-
   findall(L,subconjunto_suma_3_aux(L1,N,L),S),
   asserta(subconjuntos_suma_calculados_3(L1,N,S)),
   member(L2,S).

subconjunto_suma_3_aux([],0,[]).
subconjunto_suma_3_aux([X|L1],N,[X|L2]) :-
   N >= X,
   N1 is N-X,
   subconjunto_suma_3(L1,N1,L2).
subconjunto_suma_3_aux([_|L1],N,L2) :-
   subconjunto_suma_3(L1,N,L2).
```

Para comparar la eficiencia se realizan los siguientes experimentos.

```
?- numlist(1,20,_L1), time(findall(L,subconjunto_suma_1(_L1,10,L),_S)).
% 25,165,844 inferences, 9.09 CPU in 9.19 seconds
?- numlist(1,20,_L1), time(findall(L,subconjunto_suma_2(_L1,10,L),_S)).
% 1,974 inferences, 0.00 CPU in 0.00 seconds
?- numlist(1,20,_L1), time(findall(L,subconjunto_suma_3(_L1,10,L),_S)).
% 3,242 inferences, 0.01 CPU in 0.01 seconds

?- numlist(1,140,_L1), time(findall(L,subconjunto_suma_2(_L1,70,L),_S)).
```

```
% 104,855,949 inferences, 47.90 CPU in 58.77 seconds
?- numlist(1,140,_L1), time(findall(L,subconjunto_suma_3(_L1,70,L),_S)).
% 593,122 inferences, 18.91 CPU in 21.54 seconds
```

Ejercicio 3 (2.5 puntos) *Definir la relación* `valor(+E,+A,-V)` *que se verifique si* `V` *es el valor de la expresión aritmética* `E` *asignándole a las variables de* `E` *su valor correspondiente en la lista de asignaciones* `A`. *Por ejemplo,*

```
?- valor(2*x+y,[x-5,z-11,y-4],V).
V = 14
?- valor(2*(x^2+y^2),[x-3,y-4,z-9],V).
V = 50
?- valor(2*(sin(x)^y+cos(x)^y),[x-3,y-2],V).
V = 2.0
?- valor(2+y,[x-3],V).
No
```

Solución:
La definición de `valor` es

```
valor(E,_,E) :-
   number(E).
valor(E,A,V) :-
   member(E-V,A).
valor(E,A,V) :-
   not(atomic(E)),
   E =.. [O|L1],
   valor_aux(L1,A,L2),
   E_2 =.. [O|L2],
   V is E_2.
```

`valor_aux(+L1,+A,-L2)` se verifica si `L2` es la lista de los valores de las expresiones de `L1` respecto de la asignación `A`.

```
valor_aux([],_,[]).
valor_aux([E1|L1],A,[V1|L2]) :-
   valor(E1,A,V1),
   valor_aux(L1,A,L2).
```

Ejercicio 4 (2.5 puntos) *En el problema de las jarras* `M-N-X` *se dispone de*

- *una jarra de capacidad* `M` *litros,*
- *una jarra de capacidad* `N` *litros,*
- *un grifo que permite llenar las jarras de agua (las jarras no tienen marcas de medición),*
- *un lavabo donde vaciar las jarras.*

El problema consiste en averiguar cómo se puede lograr tener `X` *litros de agua en alguna de las jarras empezando con las dos jarras vacías. Los tipos de acciones que se permiten son*

- *Llenar la primera jarra con el grifo.*
- *Vaciar la primera jarra en el lavabo.*
- *Llenar la segunda jarra con el grifo.*

- *Vaciar la segunda jarra en el lavabo.*
- *Vaciar la primera jarra en la segunda.*
- *Llenar la segunda jarra con la primera.*
- *Vaciar la segunda jarra en la primera.*
- *Llenar la primera jarra con la segunda.*

donde la primera jarra es la que tiene `M` *litro de capacidad y la segunda es la que tiene* `N`*.*

La soluciones del problema de las jarras `M-N-X` *pueden representarse mediante listas de pares* `A-B` *donde* `A` *es el contenido de la primera jarra y* `B` *el de la segunda. Por ejemplo, una solución del problema* `4-3-2` *es* `[0-0, 0-3, 3-0, 3-3, 4-2]`*; es decir, se comienza con las jarras vacías (*`0-0`*), se llena la segunda con el grifo (*`0-3`*), se vacía la segunda en la primera (*`3-0`*), se llena la segunda con el grifo (*`3-3`*) y se llena la primera con la segunda (*`4-2`*).*

Definir la relación `solución_jarras(+M,+N,+X,-S)` *que se verifique si* `S` *es una solución del problema de las jarras* `M-N-X`*. Por ejemplo,*

```
?- solución_jarras(4,3,2,L).
L = [0-0, 4-0, 4-3, 0-3, 3-0, 3-3, 2-4]
L = [0-0, 4-0, 1-3, 4-3, 0-3, 3-0, 3-3, 4-2]
Yes
?- solución_jarras(9,4,3,L).
L = [0-0, 9-0, 9-4, 0-4, 4-0, 4-4, 8-0, 8-4, 9-3]
Yes
?- solución_jarras(6,3,2,L).
No
```

Solución:
La definición de `solución_jarras` es

```
solución_jarras(M,N,X,S) :-
   solución_jarras_aux(M,N,X,[0-0],S).

solución_jarras_aux(_,_,X,[E|L],S) :-
   una_jarra_tiene(X,E), !,
   reverse([E|L],S).
solución_jarras_aux(M,N,X,[A-B|L],S) :-
   siguiente(M,N,A-B,C-D),
   not(member(C-D,[A-B|L])),
   solución_jarras_aux(M,N,X,[C-D,A-B|L],S).
```

La relación `una_jarra_tiene(+X,+E)` se verifica si una de las jarras de `E` tiene `X` litros.

```
una_jarra_tiene(X,X-_).
una_jarra_tiene(X,_-X).
```

La relación `siguiente(+M,+N,+E1,-E2)` se verifique si `E2` es un estado obtenido aplicándole al estado `E1` una de las siguientes acciones en el problema de las jarras `M-N-X`:

1 Llenar la primera jarra con el grifo.

2 Vaciar la primera jarra en el lavabo.

3 Llenar la segunda jarra con el grifo.

4 Vaciar la segunda jarra en el lavabo.

5 Vaciar la primera jarra en la segunda.

6 Llenar la segunda jarra con la primera.

7 Vaciar la segunda jarra en la primera.

8 Llenar la primera jarra con la segunda.

```
siguiente(M,_,A-B,M-B) :- A < M.                    % 1
siguiente(_,_,A-B,0-B) :- A > 0.                    % 2
siguiente(_,N,A-B,A-N) :- B < N.                    % 3
siguiente(_,_,A-B,A-0) :- B > 0.                    % 4
siguiente(_,N,A-B,0-C) :- A+B =< N, C is A+B.       % 5
siguiente(_,N,A-B,C-N) :- A+B >  N, C is A-(N-B).   % 6
siguiente(M,_,A-B,C-0) :- A+B =< M, C is A+B.       % 7
siguiente(M,_,A-B,M-C) :- A+B >  M, C is B-(M-A).   % 8
```