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).*

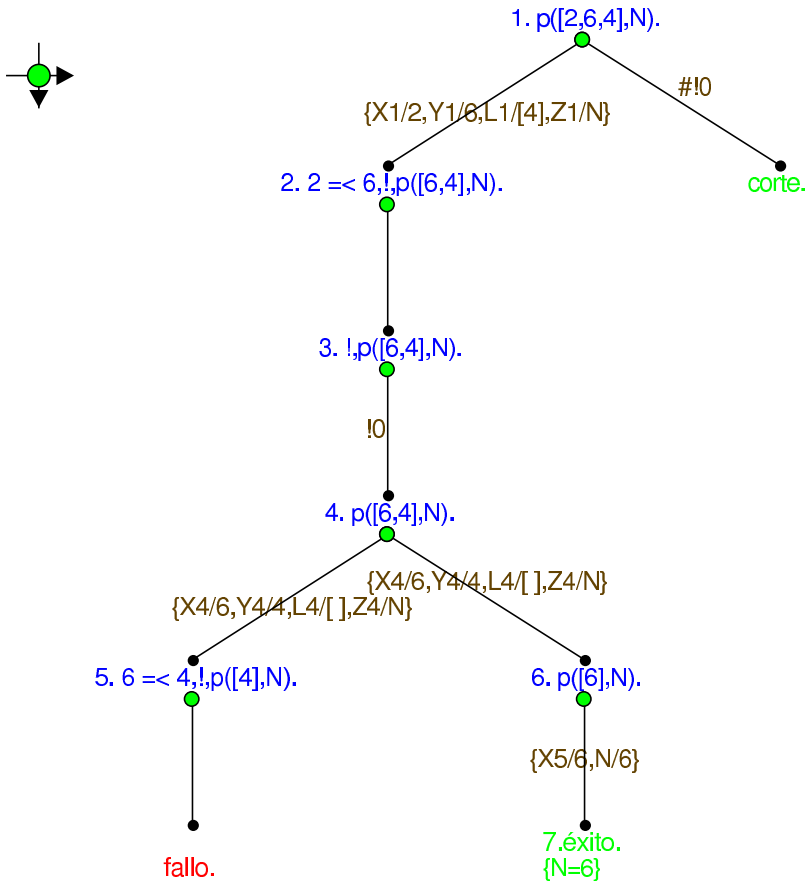
Ejercicio 1 (1 punto) *Se considera el siguiente programa lógico*

```
p([X],X).                              % R1
p([X,Y|L],Z) :- X =< Y, !, p([Y|L],Z). % R2
p([X,Y|L],Z) :- p([X|L],Z).            % R3
```

Escribir el árbol de resolución y las respuestas correspondientes al programa y a la pregunta

```
?- p([2,6,4],N).
```

Solución: El árbol de resolución es



La respuesta obtenida es `N = 6`.

Ejercicio 2 (1.5 puntos) *Definir la función*

```
esPermutación :: Eq a => [a] -> [a] -> Bool
```

tal que `(esPermutación xs ys)` *se verifique si* `xs` *es una permutación de* `ys`*. Por ejemplo,*

```
esPermutación [1,2,1] [2,1,1]  ⇝  True
esPermutación [1,2,1] [1,2,2]  ⇝  False
```

Solución:

```
esPermutación :: Eq a => [a] -> [a] -> Bool
esPermutación []     []     = True
esPermutación []     (y:ys) = False
esPermutación (x:xs) ys     = elem x ys && esPermutación xs (borra x ys)
```

donde `(borra x xs)` es la lista obtenida borrando una ocurrencia de `x` en la lista `xs`. Por ejemplo,

```
borra 1 [1,2,1]  ==>  [2,1]
borra 3 [1,2,1]  ==>  [1,2,1]
```

```
borra :: Eq a => a -> [a] -> [a]
borra x []                  = []
borra x (y:ys) | x == y     = ys
               | otherwise = y : borra x ys
```

Ejercicio 3 (1.5 puntos) *En este ejercicio se usa la representación de los números naturales construidos con* `0` *y* `s`*; es decir, la serie*

```
0, S 0, S (S 0), S (S (S 0)), ...
```

representa a la serie de los números naturales

```
0, 1, 2, 3, ...
```

Para ello, definimos el siguiente tipo de datos

```
data Nat = 0 | S Nat
           deriving (Eq, Show)
```

Definir la función

```
entre :: Nat -> Nat -> [Nat]
```

tal que `(entre x y)` *es la lista de los números mayores o iguales que* `x` *y menores o iguales que* `y`*. Por ejemplo,*

```
entre (S 0)         (S (S (S 0))) ⇝  [S 0,S (S 0),S (S (S 0))]
entre (S (S (S 0))) (S 0)         ⇝  []
```

Solución:

```
entre :: Nat -> Nat -> [Nat]
entre x y
    | mayor x y = []
    | otherwise = x:(entre (S x) y)
```

donde `(mayor x y)` se verifica si el número natural `x` es mayor que el número natural `y`. Por ejemplo,

```
mayor (S 0) (S(S(0)))  ⇝  False
mayor (S(S(0))) (S 0)  ⇝  True
mayor (S 0) (S 0)      ⇝  False
```

```
mayor :: Nat -> Nat -> Bool
mayor (S _) 0     = True
mayor (S x) (S y) = mayor x y
mayor _     _     = False
```

Ejercicio 4 (1.5 puntos) *Definir el predicado* `rotaciones(+L1,-L2)` *que se verifique si* `L2` *es la lista cuyos elementos son las listas formadas por las sucesivas rotaciones de los elementos de la lista* `L1`*. Por ejemplo,*

```
?- rotaciones([1,2,3,4],L).
L = [[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]]
?- rotaciones([2,3,4,1],L).
L = [[2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3], [1, 2, 3, 4]]
```

(Nota: *No importa el orden de los elementos en* `L2`*.)*

Solución:

```
rotaciones(L1,L2) :-
   findall(L,es_rotación(L1,L),L2).
```

donde `es_rotación(+L1,-L2)` se verifica si `L2` es una rotación de `L1`. Por ejemplo,

```
?- es_rotación([1,2,3,4],L).
L = [1, 2, 3, 4] ;
L = [2, 3, 4, 1] ;
L = [3, 4, 1, 2] ;
L = [4, 1, 2, 3] ;
No
```

```
es_rotación(L1,L2)  :-
   append(L3,[X|L4],L1),
   append([X|L4],L3,L2).
```

Ejercicio 5 (1.5 puntos) *Consideremos el siguiente problema de las torres de Hanoi:*

- *Existen tres postes que llamaremos A, B y C.*
- *Hay N discos en el poste A ordenados por tamaño (el de arriba es el de menor tamaño).*
- *Los postes B y C están vacíos.*
- *Sólo puede moverse un disco a la vez y todos los discos deben de estar ensartados en algún poste.*
- *Ningún disco puede situarse sobre otro de menor tamaño.*

Definir la relación `hanoi(+N,+A,+B,+C,-L)` *que se verifica si* `L` *es la lista de movimientos para mover* `N` *discos desde el poste* `A` *al poste* `B` *usando el* `C` *como auxiliar. Por ejemplo,*

```
?- hanoi(4,a,b,c,L).
L = [a-c,a-b,c-b,a-c,b-a,b-c,a-c, a-b, c-b,c-a,b-a,c-b,a-c,a-b,c-b]
?- hanoi(3,a,c,b,L).
L = [a-c,a-b,c-b,a-c,b-a,b-c,a-c]
?- hanoi(3,c,b,a,L).
L = [c-b,c-a,b-a,c-b,a-c,a-b,c-b]
```

Solución:

```
hanoi(1,A,B,_C,[A-B]).
hanoi(N,A,B,C,L) :-
   N > 1,
   N1 is N-1,
   hanoi(N1,A,C,B,L1),
   hanoi(N1,C,B,A,L2),
   append(L1,[A-B|L2],L).
```