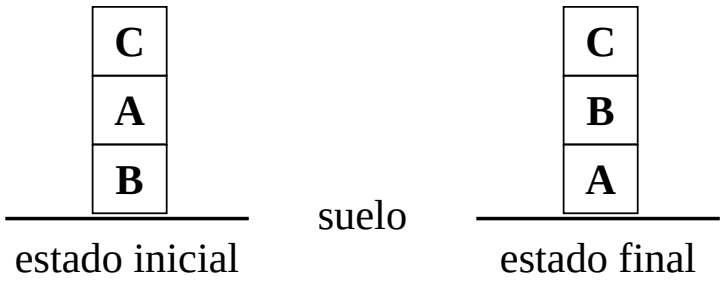
Ejercicio 1 [2.5 puntos]
Consideraremos el siguiente problema de espacio de estados:

Disponemos de tres bloques etiquetados con las letras **A**, **B** y **C** situados inicialmente como se indica en la figura y queremos moverlos hasta llegar al estado final mostrado:



Un movimiento consiste en tomar un bloque libre (es decir, que no tiene nada encima) y situarlo encima del "suelo" o encima de otro bloque libre.

Se pide:

1. Definir una heurística que permita encontrar una solución a este problema por el procedimiento de búsqueda en escalada.
2. Representar el árbol de búsqueda en escalada con dicha heurística, enumerando los nodos según se van analizando.

. . . . . .

Solución:
Apartado 1.1
Dos posibles heurísticas son las siguientes:

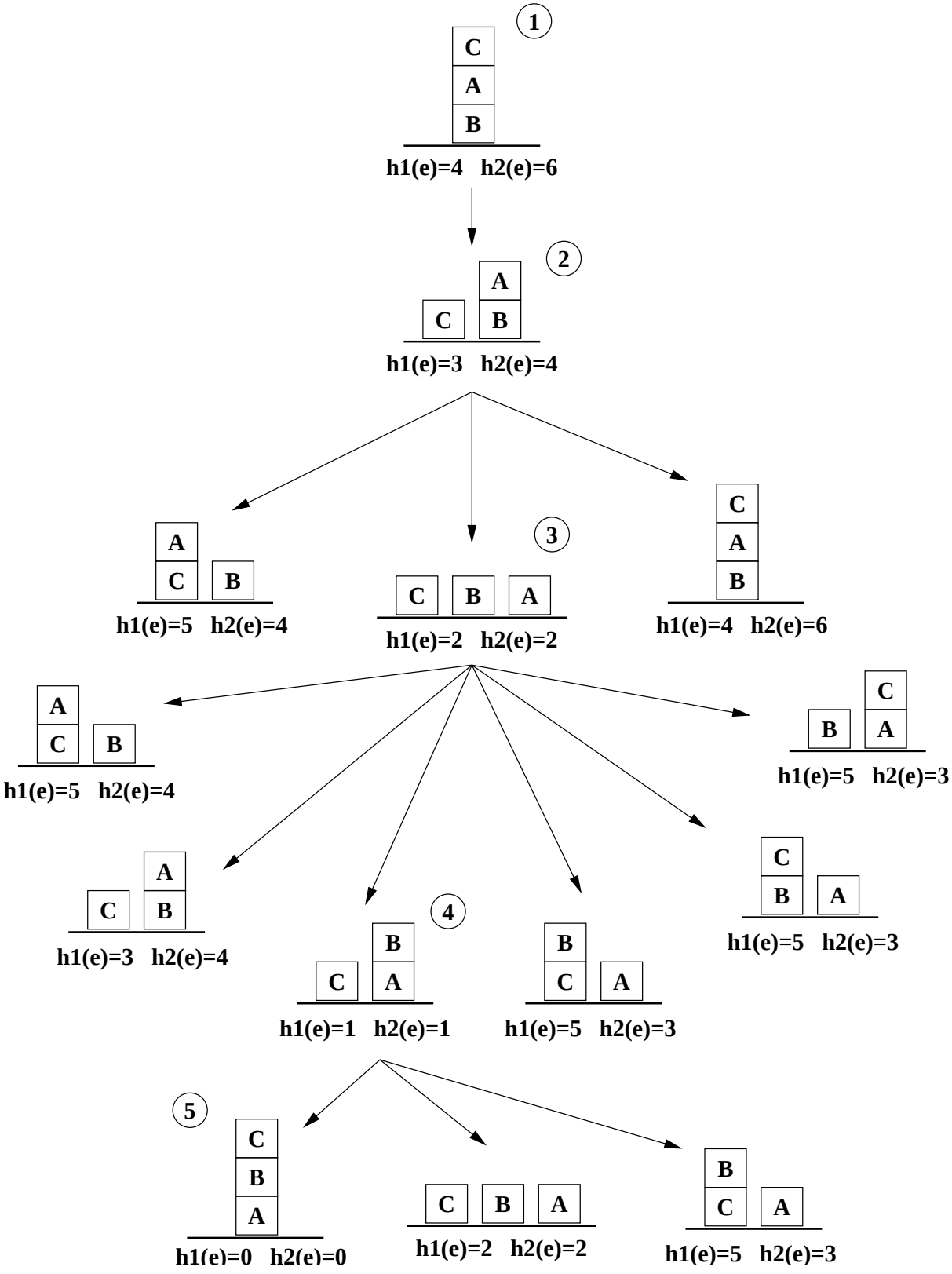
- Heurística `h1`: Se define esta heurística por casos, dado un estado e

$$h1(e) = \begin{cases} 4 & \text{si } e \text{ es } \begin{array}{|c|}\hline C \\ \hline A \\ \hline B \\ \hline \end{array} \\ 3 & \text{si } e \text{ es } \begin{array}{|c|c|}\hline & A \\ \hline C & B \\ \hline \end{array} \\ 2 & \text{si } e \text{ es } \begin{array}{|c|c|c|}\hline C & B & A \\ \hline \end{array} \\ 1 & \text{si } e \text{ es } \begin{array}{|c|c|}\hline & B \\ \hline C & A \\ \hline \end{array} \\ 0 & \text{si } e \text{ es } \begin{array}{|c|}\hline C \\ \hline B \\ \hline A \\ \hline \end{array} \\ 5 & \text{en otro caso} \end{cases}$$

- Heurística `h2`: Para todo bloque situado a una altura H (comenzando a contar desde 1), sumar H si dicho bloque o los que tiene situados debajo están en posición errónea con respecto al estado final.

Apartado 1.2

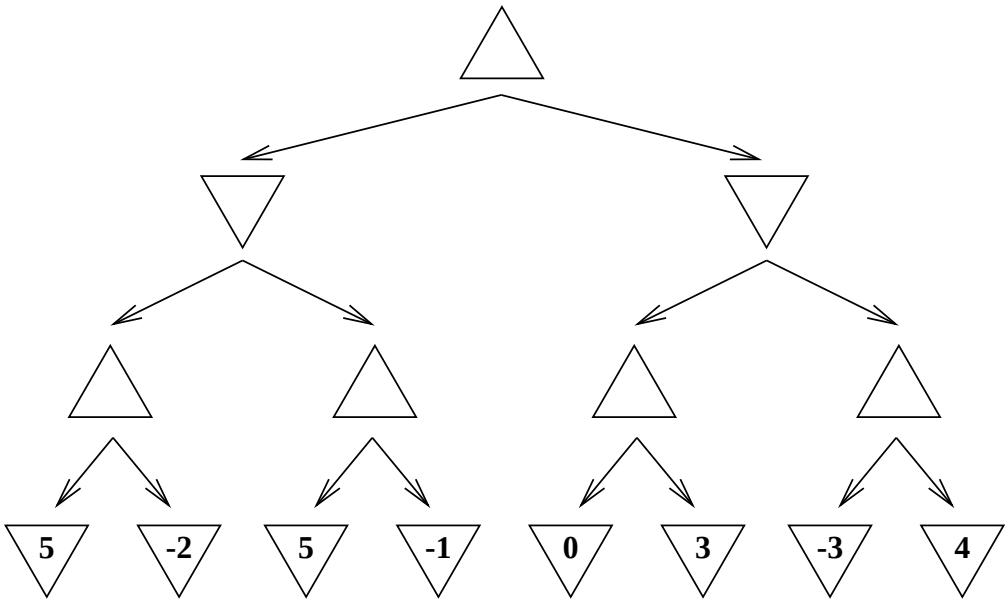
El árbol de búsqueda en escalada con dichas heurísticas queda como sigue:



Ejercicio 2 [2.5 puntos]

En los tres primeros apartados de este ejercicio consideraremos un juego con N operadores, que es analizado utilizando la estrategia alfa-beta hasta una profundidad de 3 (el nodo raíz está a profundidad 0), partiendo de un nodo MAX. En estas circustancias contestar a las siguientes preguntas:

- ¿Cuál es la cantidad mínima de nodos analizados, incluyendo las hojas?
- ¿En qué condiciones (valores de las cotas iniciales, valores de la función de evaluación estática en las hojas, ...) se corta la mayor cantidad del árbol?
- ¿En qué condiciones (valores de las cotas iniciales, valores de la función de evaluación estática en las hojas, ...) se puede cortar el primer hijo de un nodo MAX?
- Aplicar la estrategia alfa-beta con cotas iniciales $-\infty$ (α) y $+\infty$ (β) al siguiente árbol:

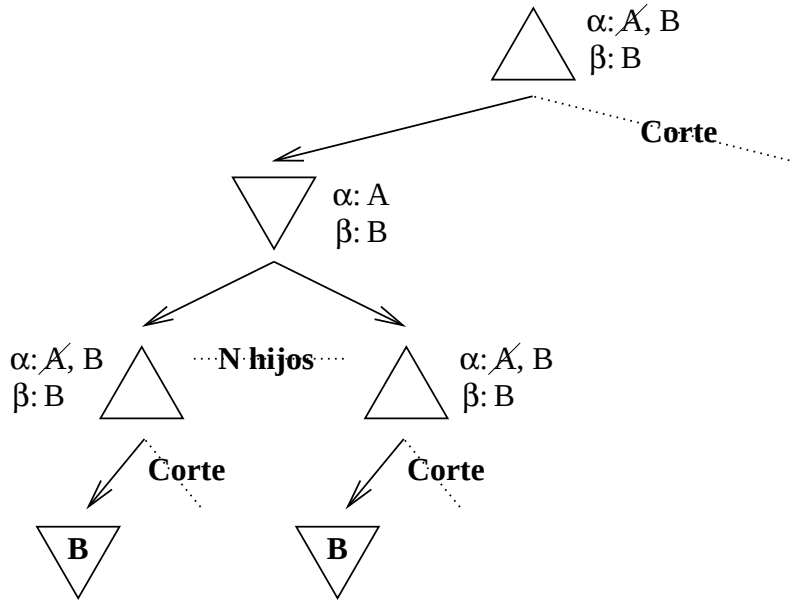


. . . . . .

Solución:

Apartado 2.1

La cantidad mínima de nodos analizados se alcanza cuando el límite superior del intervalo de búsqueda es finito y coincide con el valor de la función de evaluación estática en la primera hoja de todos y cada uno de los nodos MAX que hay a profundidad 2 en el primer hijo del nodo raíz. Ver dibujo:



con lo que el número mímino de nodos analizados es $2N + 2$.

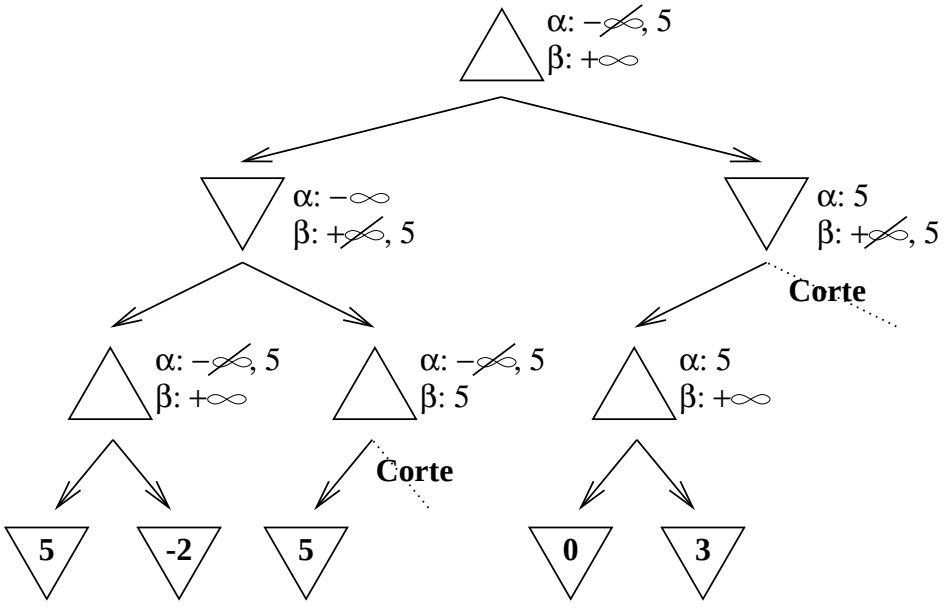
Apartado 2.2

La mayor cantidad del árbol se corta en la situación descrita en el apartado anterior.

Apartado 2.3

Nunca. Para cortar el primer hijo de un nodo MAX, sería necesario que $\alpha \geq \beta$ en dicho nodo, pero sus valores $[\alpha, \beta]$ provienen de un nodo MIN superior, en el que también se tendría $\alpha \geq \beta$, por lo que sus hijos habrían sido cortados y por tanto el nodo MAX no habría sido analizado.

Apartado 2.4



Ejercicio 3 [2.5 puntos]
Se considera el siguiente programa:

```
p(0,X,X).
p(s(X),Y,s(Z)) :- p(X,Y,Z).

q(0,X).
q(s(X),s(Y)) :- q(X,Y).
```

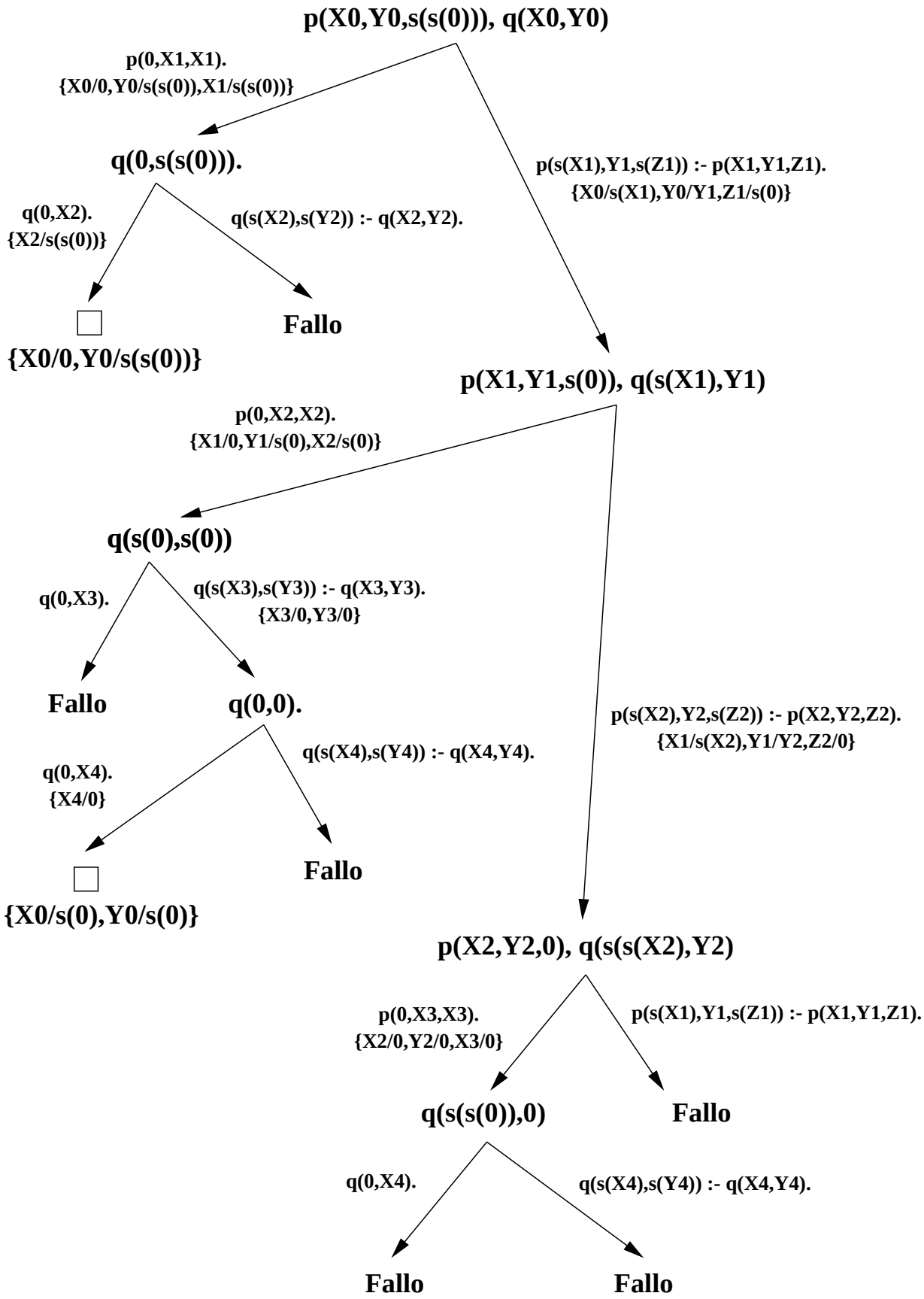
Construir el árbol de resolución SLD correspondiente a dicho programa y a la pregunta

```
?- p(X,Y,s(s(0))), q(X,Y).
```

indicando las respuestas obtenidas.

. . . . . .

Solución:
El árbol de resolución SLD pedido es el siguiente:



Las soluciones son:

$$X = 0, Y = s(s(0)) \quad \text{y} \quad X = s(0), Y = s(0)$$

Ejercicio 4 [2.5 puntos]
Se considera el procedimiento general de búsqueda en Prolog definido por

```
busqueda(M,S) :-
   estado_inicial(E),
   busqueda(M,[[E]],S).

busqueda(_M,Abiertos,S) :-
   Abiertos = [[E|C]|_],
   estado_final(E),
   reverse([E|C],S).
busqueda(M,Abiertos,S) :-
   selecciona(M,Abiertos,N,R),
   sucesores(N,Sucesores),
   expande(M,R,Sucesores,NAbiertos),
   busqueda(M,NAbiertos,S).
```

Se pide:

1. Definir lar relación `sucesores(+N,?L)` que se verifique si `L` es la lista de los sucesores del nodo `N`.

2. Definir la relación `expande(+M,+L1,+Sucesores,?L2)` en el caso de la búsqueda en profundidad; es decir que se verifique si `M` es `profundidad` y `L2` es la lista obtenida expandiendo (según el método `M`) la lista de nodos `L1` con la lista de nodos `Sucesores`.

3. Definir la relación `selecciona(+M,+LN1,?N,?LN2)` en el caso de la búsqueda optimal; es decir, que se verifique si `M` es `optimal`, `N` es el nodo de la lista `LN1` seleccionado en el caso de la búsqueda optimal y `LN2` es la lista de los restantes nodos.

. . . . . .

Solución:
Apartado 4.1

```
sucesores([E|C],L) :-
  findall([E1,E|C],sucesor(E,E1),L).
```

Apartado 4.2

```
expande(profundidad,L1,Sucesores,L2) :-
  append(Sucesores,L1,L2).
```

Apartado 4.3

```
selecciona(optimal,LN1,N,LN2) :-
   selecciona_con_valor(optimal,LN1,N,LN2).

selecciona_con_valor(M,LN1,N,LN2) :-
   member(N,LN1),
   valor(M,N,V),
   not(member(N1,LN1),
       valor(M,N1,V1),
       V1 < V),
   select(LN1,N,LN2).

valor(optimal,N,V) :-
   coste_camino(N,V).

coste_camino([_E],0).
coste_camino([E2,E1|R],V) :-
   coste(E2,E1,V1),
   coste_camino([E1|R],V2),
   V is V1+V2.
```
