Programación Lógica

Laboratorio 1

Fundamentos de Prolog

Integrantes

Nombre	CI	Correo
Gonzalo Marco	4.854.892-9	gonzalo.marco.mohotse@fing.edu.uy
Germán Ouviña	4.823.566-1	german.ouvina@fing.edu.uy
Ramiro Pombo	4.725.542-8	ramiro.pombo@fing.edu.uy

Índice

1. Ejercicio 1 - Predicados varios **3**

1.1. Predicados auxiliares 3

1.2. Predicados principales 3

2. Ejercicio 2 - Criba de Eratóstenes **7**

2.1. Predicados auxiliares 7

2.2. Predicados principales 7

3. Ejercicio 3 - Caminitos **9**

3.1. Predicados auxiliares 9

3.2. Predicados principales 10

1. Ejercicio 1 - Predicados varios

A continuación se agregan los cabezales y una breve descripción de los predicados implementados para el ejercicio 1, separando en predicados principales (aquellos predicados especificados por la letra) y auxiliares (aquellos que se utilizaron a modo de facilitar la implementación de los predicados principales).

Para los predicados principales, se adjuntan al final de la sección capturas de las medidas de performance tomadas al ejecutarlos en un escenario genérico de prueba.

1.1. Predicados auxiliares

Los predicados implementados fueron los siguientes:

- `crear_fila(+N, ?E, ?F)`
  Retorna una lista `F` de largo `N` y con el elemento `E` en cada celda.
- `generar_transpuesta(+A, +L, +N, ?C)`
  Genera una matriz `C` a partir de la matriz `A` con cantidad de filas `N`, usando un acumulador `L` comenzando en 1 y yendo hasta `N`. La matriz `C` es la transpuesta de `A`.
- `largo(+L, ?N)`
  Comprueba si la lista `L` tiene `N` elementos.

1.2. Predicados principales

Los predicados principales implementados fueron los siguientes:

- `elegir(?X,?L1,?L2)`
  Comprueba que la lista `L2` sea el resultado de eliminar el elemento `X` de la lista `L1`. En la definición del predicado se diferencian dos escenarios:
  - Cuando `L1 = [X|L2]`, es decir, cuando se encuentra `X` en la primera lista (y el resto es igual a la otra lista) el predicado devuelve `true`. Se toma como paso base de la recursión generada en el siguiente escenario.
  - Cuando no se encuentra `X` al inicio de `L1`, se «itera» sobre los elementos de la misma, comprobando que los elementos de `L2` coincidan con los de `L1` mientras la cabeza de `L1` sea distinta a `X`. Una vez hecho esto, se procede a llamar al predicado recursivamente con las colas de ambas listas. En el caso que `X` se encuentre en `L1`, eventualmente se alcanza el paso base (predicado anterior), devolviendo `true`. En el caso contrario, el predicado finaliza devolviendo `false`.
- `elegirN(+L,+L1,+N,?L2)`
  Comprueba que la lista `L2` sea el resultado de eliminar `N` elementos de la lista `L1`, apareciendo dichos elementos en `L`. En la definición del predicado se diferencian dos escenarios:
  - Cuando `L = []`, no importa que contiene la lista `L1`, pero es igual a `L2` y el predicado devuelve `true`. Se toma como paso base de la recursión generada en el siguiente escenario.
  - Cuando `L = [H|T]`, se «itera» sobre los elementos de la misma, ejecutando el predicado `elegir` usando el elemento `H` y las listas `L1` y `L`. Una vez hecho esto, se procede a llamar al predicado recursivamente con la cola `T`, hasta alcanzar el paso base (predicado anterior), devolviendo `true`. En el caso contrario, el predicado finaliza devolviendo `false`.
- `suma(+L,?S)`
  Se ejecuta recursivamente, inicializando `S = 0` en el paso base (es decir, `L = []`) y acumulando la suma hasta completar la lista.

- `matriz(+M,+N,+E,?A)`
  Comprueba que `A` sea una matriz de `M` filas y `N` columnas, donde todas las entradas de la misma son el elemento `E`. La forma en la que `A` se representa es una lista con `M` listas de largo `N`, es decir, una lista de filas. Para realizar el chequeo se ejecuta el predicado `matriz` recursivamente, utilizando el predicado auxiliar `generar_fila` en cada llamada. Se distinguen dos escenarios:
  - Cuando `M = 1, A = [F]`, se ejecuta el predicado `generar_fila` sobre `F` para comprobar que efectivamente es una fila y en caso de que así sea, el predicado devuelve `true`. Se toma como paso base de la recursión generada en el siguiente escenario.
  - Cuando `M > 1, A = [F|T]`, se «itera» sobre los elementos de la matriz, es decir, sobre sus filas. Al igual que en el paso base, se ejecuta `generar_fila` sobre `F` y luego se llama `matriz` recursivamente, reduciendo `M` y quitando `F`. Si en algún momento `generar_fila` retorna `false` o no se encuentra la cantidad de filas esperada, el predicado `matriz` retorna `false`.

- `valor_celda(+I,+J,+A,?E)`
  Comprueba que el valor de la celda en la fila `I` y columna `J` de la matriz `A` sea `E`. Para ello, se ejecuta recursivamente de una manera similar al predicado `matriz`, distinguiendo dos escenarios:
  - Cuando `I = 1`, se ejecuta el predicado auxiliar `enesimo` sobre la primera fila de la matriz, para obtener el valor de la celda ubicado en la columna `N`. Se toma como paso base de la recursión generada en el siguiente escenario.
  - Cuando `I > 1`, se «itera» sobre las filas de la matriz, disminuyendo `I` en 1 por cada paso. Si `I > M`, el predicado nunca llegará al paso base y se retorna `false`. En caso contrario, se ejecuta el paso base y se devuelve `true` asumiendo que se encuentra el elemento en la columna correspondiente.

- `fila(+N,+M,?F)`
  Comprueba que la matriz `M` tenga como `N`-ésima fila a la lista `F`. Su funcionamiento es análogo al del predicado `enésimo`, ya que las matrices son listas de filas.

- `col(+N,+M,?C)`
  Comprueba que la matriz `M` tenga como `N`-ésima columna a la lista `C`. Dado que las matrices son listas de filas, la construcción de una lista columna es más compleja. En resumen, se recorre cada fila de la matriz `M` recursivamente, obteniendo el `N`-ésimo elemento a través del predicado `enesimo`. Una vez se recorren todas las filas, se construye la lista columna `C`.

- `transpuesta(+M,?T)`
  Comprueba que la matriz `T` sea la transpuesta de `M`. Para ello, se utilizan el predicado `largo` para saber cuantas filas tiene `M` y el predicado auxiliar `generar_transpuesta`. Este último sigue una lógica similar a la del predicado `col`; se recorre fila por fila de `M` recursivamente, construyendo la matriz `T` como una lista de columnas de `M`.

- `numeros(+Inicio,+Fin,?Lista)`
  Comprueba que la lista `Lista` se componga por los números entre `Inicio` y `Fin`. Para que el predicado no falle, se debe cumplir que `Inicio < Fin`. Se distinguen dos escenarios en su funcionamiento:
  - Cuando `Inicio = Fin`, se obtiene `Lista = [Inicio]`.
  - Cuando `Inicio < Fin`, se «itera» ejecutando recursivamente sobre el predicado `numeros`, agregando el valor de `Inicio` a `Lista` y llamando a `numeros` con `Inicio = Inicio + 1`. Así hasta alcanzar el escenario anterior.

Habiendo reseñado todos los predicados implementados y no habiendo distinguido anomalías en ninguno de los test de performance, se adjunta captura de varias ejecuciones tipo para cada uno de ellos, ejecutando a su vez el predicado `time` para controlar su eficiencia.

Se destaca la utilidad del predicado `time` al momento de determinar el número de inferencias lógicas de un predicado en concreto. Durante el proceso de desarrollo resultó útil para encontrar predicados definidos de forma subóptima.

Por último se menciona que para predicados como `suma` o `numeros`, se aprecia el aumento lineal de la cantidad de inferencias en relación al aumento de elementos en las listas, estando relacionados de forma directamente proporcional. Para otros predicados fue más difícil encontrar tales relaciones, por lo que se adjuntan pruebas básicas.

```
% ELEGIR
% ------------------------------------------------------------------------------
time((elegir(1,[1,2,1,2,1],L), fail;true)).
% 7 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((elegir(1,[1,2,1,2,1,2,2,1,1,2,2,1,1,2,2,2,1,1,1,2],L), fail;true)).
% 23 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------
```

```
% ELEGIR N
% ------------------------------------------------------------------------------
time((elegirN([2,5],[1,2,3,4,5],2,L), fail;true)).
% 15 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((elegirN([2,5,7,5,9],[1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9],5,L), fail;true)).
% 412 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------
```

```
% SUMA
% ------------------------------------------------------------------------------
time((suma([10],S), fail;true)).
% 3 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((suma([10,10],S), fail;true)).
% 5 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((suma([100,-20,32,-248,2932,334,5443,1,2,0,2334,12,29,343,100,10],S), fail;true)).
% 33 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------
```

```
% MATRIZ
% ------------------------------------------------------------------------------
time((matriz(1,1,1,M), fail;true)).
% 5 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((matriz(7,12,5,M), fail;true)).
% 183 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------
```

```
% VALOR CELDA
% ------------------------------------------------------------------------------------
time((valor_celda(1,1,[[1]],E), fail;true)).
% 5 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((valor_celda(3,2,[[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]],E), fail;true)).
% 11 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------------
```

```
% COLA
% ------------------------------------------------------------------------------------
time((col(2,[[8,-10,1],[5,4,2], [7,9,3]],[-10,4,9]), fail;true)).
% 7 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------------

% FILA
% ------------------------------------------------------------------------------------
time((fila(1,[[8,-10,1],[5,4,2], [7,9,3]],C), fail;true)).
% 3 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------------
```

```
% TRANSPUESTA
% ------------------------------------------------------------------------------------
time((transpuesta([[1,2],[3,4]],T), fail;true)).
% 13 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((transpuesta([[1,2,3,4],[5,6,7,8]],T), fail;true)).
% 19 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
% ------------------------------------------------------------------------------------
```

```
% NUMEROS
% ------------------------------------------------------------------------------------
time((numeros(0,0,L), fail;true)).
% 3 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((numeros(0,1,L), fail;true)).
% 5 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
time((numeros(0,10000,L), fail;true)).
% 20,003 inferences, 0.000 CPU in 0.002 seconds (0% CPU, Infinite Lips)
% ------------------------------------------------------------------------------------
```

2. Ejercicio 2 - Criba de Eratóstenes

A continuación se agregan los cabezales y una breve descripción de los predicados implementados para el ejercicio 2, separando en predicados principales (aquellos predicados especificados por la letra) y auxiliares (aquellos que se utilizaron a modo de facilitar la implementación de los predicados principales).

Para los predicados principales, se adjuntan capturas de las medidas de performance tomadas al ejecutarlos en varios escenarios.

2.1. Predicados auxiliares

Los predicados implementados fueron los siguientes:

- `borrarH(+H, +L1, ?L2)`
  La lista `L2` resulta de eliminar todas las ocurrencias del elemento `H` en la lista `L1`.
- `borrar(+L, +L1, ?L2)`
  La lista `L2` resulta de eliminar los elementos de `L` que se encuentren en `L1`.
- `calculo_multiplos(+H, +N, +L, ?L2)`
  La lista `L2` contiene todos los múltiplos de `H`, mayores a `H` y menores a `N`.
- `depurarH(+H, +L, ?L2)`
  La lista `L2` resulta de eliminar todos los numeros múultiplos de `H` en la lista `L`.
- `depurar(+L1, ?L2)`
  La lista `L2` es la lista `L1` que solo contiene a los primos.

2.2. Predicados principales

En este caso, el predicado principal es solamente uno, para el cual se adjuntan casos de prueba y medidas de performance a continuación:

`primos(+N, ?P)`
`P` es la lista de todos los primos mayores a 1 y menores que `N`.
Observar que para los casos no base se busca seguir el algoritmo de la *Criba de Eratóstenes*: primero se generara una lista con todos los números entre 2 y N (utilizando para ello, el predicado `numeros`), y luego se irún eliminando aquellos que sean multiplos de aquellos que ya se determinaron como primos (por medio del predicado `depurar`).

Habiendo reseñado todos los predicados implementados y no habiendo distinguido anomalías en ninguno de los test de performance, se adjuntan en la siguiente página capturas de ejecuciones tipo para el predicado `primos`, controlando resultado y eficiencia a través del predicado `time`.

```
%Caso Correcto
- time(primos(92,L)).
% 10,192 inferences, 0.016 CPU in 0.005 seconds (313% CPU, 652288 Lips)
L = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89] ;
% 270 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
```

```
%Caso incorrecto
time(primos(-31,L)).
% 3 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
```

```
%Test de stress
time(primos(3784,L)).
% 20,104,006 inferences, 1.797 CPU in 1.958 seconds (92% CPU, 11188316 Lips)
L = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131,
137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277,
281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617,
619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991,
997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129,
1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297,
1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471,
1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613,
1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787,
1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979,
1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131,
2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309,
2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467,
2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671,
2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803,
2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001,
3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203,
3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371,
3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547,
3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719,
3727, 3733, 3739, 3761, 3767, 3769, 3779] ;
% 16,144 inferences, 0.016 CPU in 0.018 seconds (87% CPU, 1033216 Lips)
false.
```

3. Ejercicio 3 - Caminitos

A continuación se agregan los cabezales y una breve descripción de los predicados implementados para el ejercicio 3, separando en predicados principales (aquellos predicados especificados por la letra) y auxiliares (aquellos que se utilizaron a modo de facilitar la implementación de los predicados principales).

Para el predicado principal (que en este caso es uno), se adjuntan al final de la sección capturas de las medidas de performance tomadas al ejecutarlo en distintos escenarios de prueba.

Se destaca que se desarrolló una versión preliminar del predicado en cuestión, la cual separa claramente la generacion del chequeo. No obstante, al tomar medidas de performance se consideró que la misma estaba muy por debajo de los estándares necesarios y se desarrolló la versión actual, la cual busca unificar en cierta manera ambos procesos. Las medidas de performance tomadas son en referencia a esta versión, aunque también se adjunta una corrida para la versión no eficiente, a modo de comparación.

3.1. Predicados auxiliares

Los predicados implementados fueron los siguientes:

- `tablero(+N, +Ac, ?T)`
  `T` es una lista de $N \times N$ con todas las celdas del tablero de $N \times N$.
  `Ac` es una lista usada como acumulador durante la ejecución.
- `pertenece(?X, ?L)`
  Devuelve `true` si `X` pertenece a la lista `L`.
- `es_adyacente(+A, ?B, +N)`
  Chequea que dos celdas del tablero sean adyacentes.
  Sea `N` el límite de la tabla, y sean `A` y `B` dos posiciones de la forma `pos(Fila, Columna)`, si `B` es una celda adyacente a `A`, comparten fila o columna con distancia de 1. De esta manera, se comprueban las siguientes condiciones:
  - Si están en la misma fila, se chequea que esten a difernecia 1 en las columnas
  - Si están en la misma columna, se chequea que esten a difernecia 1 en las filas
- `siguiente_paso(+L, ?H)`
  `H` es la cabeza de la lista `L`.
- `secuencia(+Inicial, +Celdas, +Muros, +N, +Acumulador, ?Caminito)`
  `Caminito` es una secuencia de posiciones de la forma `pos(Fila, Columna)`, dentro de un tablero $N \times N$. Dicha secuencia comienza en la posición `Inicial` y termina en la posición `Final`.
  El camino se construye celda por celda, y a medida que se avanza, `Celdas` contiene las posibles posiciones a incluir en el camino, ya que hasta el momento no se agregaron.
  Observar que para optimizar la construccion del camino, se recorre desde `Final` a `Inicio`. De esta forma se puede ir agregando las nuevas posiciones en la lista de forma directa.
  Por último, las condiciones que se comprueban para seguir la ejecución o no, son las siguientes:
  - Se llegó hasta Inicio, por lo tanto se encontró un camino posible.
  - El próximo elemento a agregar es una celda adyacente a la que se encuentre en el cabezal de `Acumulador`.
  - El próximo elemento a agregar no es un muro.
  - El próximo elemento a agregar es una de las celdas posibles (se evita pasar dos veces por la misma casilla).

3.2. Predicados principales

A continuación se adjunta el predicado principal implementado para esta tarea.

- `caminito(+N,+Muros,+Inicial,+Final,?Caminito)`
  `Caminito` es una secuencia de posiciones de la forma `pos(Fila, Columna)`, correspondiente a un camino entre la casilla `Inicial` y la casilla `Final`. La lista de `Muros` está especificada también como una lista de casillas.

  Se invoca al predicado `Secuencia` con el acumulador iniciado con el elemento `Final`, asegurándose que el camino termine con esa celda, mientras que el predicado comprobará que el mismo tenga en el cabezal la casilla `Inicial`. Además, se le pasa la lista `Celdas`, la cual consiste de todas las celdas posibles del tablero (obtenidas a través del predicado `Tablero`), a la cual se le quito la celda `Final`.

  Por último, las condiciones que se comprueban para seguir la ejecución o no, son las siguientes:
  - La celda `Inicial` no es un muro.
  - La celda `Final` no es un muro.

Previo a los tests de performance, se adjunta una comparación entre la versión inicialmente desarrollada y la final, teniendo en cuenta que la primera se resolvía de una forma menos eficiente (separando la generación y el chequeo en etapas distintas). Si bien la primera versión no logró escalar con el tamaño del tablero, ejecutar esta comparación de tiempos y cantidad de inferencias permitió determinar con seguridad la inferioridad en performance de la versión inicial.

```
% Versión Inicial (No Eficiente)
?- time((caminito(3,[],pos(1,1),pos(3,3),Caminito), fail; true)).
% 199,746 inferences, 0.030 CPU in 0.030 seconds (100% CPU, 6682336 Lips)
true.

% Versión Final
time((caminito(3,[],pos(1,1),pos(3,3),Caminito), fail; true)).
% 2,659 inferences, 0.000 CPU in 0.000 seconds (100% CPU, 6554556 Lips)
true.
```

Habiendo reseñado todos los predicados implementados y no habiendo distinguido anomalías en ninguno de los test de performance, se adjuntan a continuación y en la siguiente página capturas de ejecuciones tipo para el predicado `caminito`, controlando resultado y eficiencia a través del predicado `time`. Se prueban diversos casos borde, así como un test de stress.

```
% Ejemplo dado
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(2,1),pos(3,1),pos(4,2),pos(4,4),pos(2,3),pos(3,4),pos(2,4)],pos(1,1),pos(5,1),Caminito)).
% 596 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Caminito = [pos(1, 1), pos(1, 2), pos(1, 3), pos(1, 4), pos(1, 5), pos(2, 5), pos(3, 5), pos(4, 5), pos(5, 5), pos(5, 4), pos(5, 3), pos(5, 2), pos(5, 1)] ;
% 1,091 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Caminito = [pos(1, 1), pos(1, 2), pos(2, 2), pos(3, 2), pos(3, 3), pos(4, 3), pos(5, 3), pos(5, 2), pos(5, 1)] ;
% 1,177 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
% Unico camino entre Incio y final
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(6,[pos(1,2), pos(1,3), pos(1,4), pos(1,5), pos(2,2), pos(2,3), pos(2,4), pos(2,4), pos(3,2), pos(3,3),
pos(3,4), pos(3,5), pos(4,2), pos(4,3), pos(4,4), pos(4,5),pos(5,2), pos(5,3), pos(5,4), pos(5,5)],pos(1,1),pos(6,6),C)).
% 680 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
C = [pos(1, 1), pos(2, 1), pos(3, 1), pos(4, 1), pos(5, 1), pos(6, 1), pos(6, 2), pos(6, 3), pos(6, 4), pos(6, 5), pos(6, 6)] ;
% 2,136 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
% Inicial atrapado entre muros
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(2,1),pos(1,2),pos(2,3), pos(3,2)],pos(2,2),pos(5,1),Caminito)).
% 112,318 inferences, 0.016 CPU in 0.008 seconds (205% CPU, 7188352 Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------

% Final atrapado entre muros
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(4,3),pos(4,5),pos(5,4), pos(3,4)],pos(2,2),pos(4,4),Caminito)).
% 120 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
% Inicial es un muro
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(1,1)],pos(1,1),pos(5,1),Caminito)).
% 10 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------

% Final es un muro
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(5,1)],pos(1,1),pos(5,1),Caminito)).
% 4 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
% Inicial no válido
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(1,1), pos(2,2),pos(3,3),pos(4,4),pos(5,5)],pos(6,4),pos(5,1),Caminito)).
% 5,340 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------

% Final no válido
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time(caminito(5,[pos(1,1), pos(2,2),pos(3,3),pos(4,4),pos(5,5)],pos(1,4),pos(5,8),Caminito)).
% 82 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
false.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
% Test de stress
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
time((caminito(9,[pos(2,1),pos(2,2),pos(2,3),pos(2,4), pos(2,7),pos(2,8), pos(2,9),pos(3,6),pos(4,6), pos(5,2),pos(5,6), pos(6,2), pos(6,6), pos(7,2),
pos(7,6),pos(7,8),pos(7,9), pos(8,1), pos(8,2), pos(8,3), pos(8,4)],pos(1,1),pos(5,9),Caminito), fail;true)).
% 475,551,508 inferences, 46.063 CPU in 46.070 seconds (100% CPU, 10324049 Lips)
true.
% ----------------------------------------------------------------------------------------------------------------------------------------------------------------
```