

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

28-5-2023

MEMORIA

Code.pl

Sergio Heras

GRADO MATEMÁTICAS E INFORMÁTICA (UPM)

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right, creating a dynamic, abstract graphic element.

Código

```
:- module(_,_,[classic,assertions,regtypes]).
:- use_module(library(lists)).
author_data('Heras', 'Alvarez', 'Sergio', 'C20M025').

board1([cell(pos(1 ,1),op(*,-3)),
        cell(pos(1 ,2),op(-,1)),
        cell(pos(1 ,3),op(-,4)),
        cell(pos(1 ,4),op(- ,555)),
        cell(pos(2 ,1),op(-,3)),
        cell(pos(2 ,2),op(+ ,2000)),
        cell(pos(2 ,3),op(* ,133)),
        cell(pos(2 ,4),op(- ,444)),
        cell(pos(3 ,1),op(*,0)),
        cell(pos(3 ,2),op(* ,155)),
        cell(pos(3 ,3),op(/ / ,2)),
        cell(pos(3 ,4),op(+ ,20)),
        cell(pos(4 ,1),op(-,2)),
        cell(pos(4 ,2),op(- ,1000)),
        cell(pos(4 ,3),op(-,9)),
        cell(pos(4 ,4),op(*,4))]).

% Direcciones permitidas
direccion(n).
direccion(s).
direccion(e).
direccion(o).
direccion(no).
direccion(ne).
direccion(so).
direccion(se).

% Predicado para obtener la posición resultante de moverse en una
% dirección desde una posición dada
efectuar_movimiento(pos(Row, Col), Dir, pos(Row2, Col2)) :-
    direccion(Dir),
    mover_posicion(Dir, Row, Col, Row2, Col2).

% Definición de las reglas de movimiento para cada dirección
mover_posicion(n, Row, Col, Row2, Col) :- Row2 is Row - 1.
mover_posicion(s, Row, Col, Row2, Col) :- Row2 is Row + 1.
mover_posicion(e, Row, Col, Row, Col2) :- Col2 is Col + 1.
mover_posicion(o, Row, Col, Row, Col2) :- Col2 is Col - 1.
mover_posicion(no, Row, Col, Row2, Col2) :- Row2 is Row - 1, Col2 is Col
- 1.
mover_posicion(ne, Row, Col, Row2, Col2) :- Row2 is Row - 1, Col2 is Col
+ 1.
mover_posicion(so, Row, Col, Row2, Col2) :- Row2 is Row + 1, Col2 is Col
- 1.
mover_posicion(se, Row, Col, Row2, Col2) :- Row2 is Row + 1, Col2 is Col
+ 1.
```

```

movimiento_valido(N, pos(Row,Col), Dir) :-
    efectuar_movimiento(pos(Row,Col), Dir, pos(Row2,Col2)),
    Row2 >= 1, Row2 <= N,
    Col2 >= 1, Col2 <= N.

select_cell(IPos, Op, Board, NewBoard) :-
    select(cell(IPos, Op), Board, NewBoard).

% Caso base, donde dir(Dir, 1) es la cabeza de la lista.
% En este caso, simplemente retornamos la cola de la lista
select_dir(Dir, [dir(Dir, 1) | RestoDirs], RestoDirs).

% Caso donde dir(Dir, Num) es la cabeza de la lista, y Num > 1.
% En este caso, restamos uno de Num y agregamos de nuevo a la lista
select_dir(Dir, [dir(Dir, Num) | RestoDirs], [dir(Dir, NewNum) |
RestoDirs]) :-
    Num > 1,
    NewNum is Num - 1.

% Caso donde la cabeza de la lista es diferente de dir(Dir, _).
% En este caso, dejamos la cabeza de la lista intacta y operamos sobre el
resto de la lista
select_dir(Dir, [dir(Dir2, Num) | RestoDirs], [dir(Dir2, Num) | NewDirs])
:-
    Dir \= Dir2,
    select_dir(Dir, RestoDirs, NewDirs).

aplicar_op(op(+, Operando), Valor, Valor2) :-
    Valor2 is Valor + Operando.
aplicar_op(op(-, Operando), Valor, Valor2) :-
    Valor2 is Valor - Operando.
aplicar_op(op(*, Operando), Valor, Valor2) :-
    Valor2 is Valor * Operando.
aplicar_op(op(/, Operando), Valor, Valor2) :-
    Valor2 is Valor // Operando.

generar_recorrido(Ipos, N, Board, DireccionesPermitidas, RecorridoFinal,
ValorFinal) :-
    select_cell(Ipos, op(Operador,Ivalor), Board,NewBoard),
    aplicar_op(op(Operador,Ivalor),0,Valor2),
    generar_recorrido_aux(Ipos, N, NewBoard,
DireccionesPermitidas,Valor2, [(Ipos, Valor2)], RecorridoFinal,
ValorFinal,[Ipos])).

generar_recorrido_aux(_, _, _, [],Ivalor, Recorrido, Recorrido,Ivalor,_).
generar_recorrido_aux(_, _, [], _,Ivalor, Recorrido, Recorrido,Ivalor,_).
generar_recorrido_aux(_, N, _, _,Ivalor, Recorrido, Recorrido,
Ivalor,Visited):-
    length(Visited,N*N).
generar_recorrido_aux(Ipos, N, Board, Dirs,Ivalor,RecorridoParcial,
RecorridoFinal, ValorFinal,Visited) :-

```

```

movimiento_valido(N,Ipos,Dir),
select_dir(Dir, Dirs,NewDirs),
efectuar_movimiento(Ipos, Dir, NextPos),
\+ member(NextPos,Visited),
select_cell(NextPos, Op, Board, NewBoard),
aplicar_op(Op, Ivalor , ValorParcial),
append(RecorridoParcial, [(NextPos, ValorParcial)],
RecorridoParcial2),
generar_recorrido_aux(NextPos, N, NewBoard,NewDirs,ValorParcial,
RecorridoParcial2, RecorridoFinal, ValorFinal,[NextPos|Visited]).
generar_recorrido_aux(Ipos, N, Board, [dir(_, 0) | RestoDirs],Ivalor,
RecorridoParcial, Recorrido, Valor,Visited) :-
    generar_recorrido_aux(Ipos, N, Board, RestoDirs,Ivalor,
RecorridoParcial, Recorrido, Valor,Visited).

generar_todos_recorridos(N, Board, Dirs, Recorridos) :-
    findall((Recorrido, Valor),
        (member(cell(Pos, _), Board),
        generar_recorrido(Pos, N, Board, Dirs, Recorrido, Valor)),
        Recorridos).
generar_recorridos(N, Board, Dirs, Recorrido, Valor) :-
    generar_todos_recorridos(N, Board, Dirs, Recorridos),
    member((Recorrido, Valor), Recorridos).

% Predicado para encontrar el mínimo de una lista
min_list([X], X).
min_list([H|T], Min) :-
    min_list(T, MinRest),
    H <= MinRest,
    Min = H.
min_list([H|T], Min) :-
    min_list(T, MinRest),
    H > MinRest,
    Min = MinRest.

% Predicado para contar el número de veces que aparece un valor en una
lista de tuplas
count_valores_minimos([], _, 0).
count_valores_minimos([(_, Valor)|T], ValorMinimo, Numero) :-
    Valor == ValorMinimo,
    count_valores_minimos(T, ValorMinimo, NumeroResto),
    Numero is NumeroResto + 1.
count_valores_minimos([(_, Valor)|T], ValorMinimo, Numero) :-
    Valor \= ValorMinimo,
    count_valores_minimos(T, ValorMinimo, NumeroResto),
    Numero is NumeroResto.

tablero(N, Board, Dirs, ValorMinimo, NumeroDeRutasConValorMinimo) :-
    generar_recorridos(N, Board, Dirs, Recorridos,ValorFinal),
    findall(Valor, member((_, Valor), Recorridos), Valores),
    min_list(Valores, ValorMinimo),
    count_valores_minimos(Recorridos, ValorMinimo,
NumeroDeRutasConValorMinimo).

```

Explicaciones / razonamientos

- **Efectuar_movimiento/3**

Es un predicado implementado para simular el funcionamiento de un movimiento en un tablero 2D. Toma una posición dada en forma de una tupla `pos(Row, Col)` y una dirección `Dir`, y devuelve una nueva posición `pos(Row2, Col2)` que representa el resultado de moverse en la dirección `Dir` desde la posición original. El predicado `direccion(Dir)` simplemente garantiza que la dirección dada es válida.

`mover_posicion` es un conjunto de predicados que definen cómo cambian las coordenadas de una posición cuando se mueve en una dirección específica:

`n` (norte): la fila (`Row`) disminuye en 1 (movimiento hacia arriba en el tablero).

`s` (sur): la fila (`Row`) aumenta en 1 (movimiento hacia abajo en el tablero).

`e` (este): la columna (`Col`) aumenta en 1 (movimiento hacia la derecha en el tablero).

`o` (oeste): la columna (`Col`) disminuye en 1 (movimiento hacia la izquierda en el tablero).

`no` (noroeste): la fila (`Row`) disminuye en 1 y la columna (`Col`) disminuye en 1 (movimiento diagonal hacia arriba y a la izquierda en el tablero).

`ne` (noreste): la fila (`Row`) disminuye en 1 y la columna (`Col`) aumenta en 1 (movimiento diagonal hacia arriba y a la derecha en el tablero).

`so` (suroeste): la fila (`Row`) aumenta en 1 y la columna (`Col`) disminuye en 1 (movimiento diagonal hacia abajo y a la izquierda en el tablero).

`se` (sureste): la fila (`Row`) aumenta en 1 y la columna (`Col`) aumenta en 1 (movimiento diagonal hacia abajo y a la derecha en el tablero).

Así, si quieres moverte desde `pos(2,2)` en la dirección `n`, `efectuar_movimiento` te dará `pos(1,2)` como resultado.

- **Movimiento_valido/3:**

El predicado se usa para verificar si un movimiento en una dirección específica desde una posición dada es válido en un tablero de dimensión `N`.

Para hacer esto, toma tres argumentos:

`N`: la dimensión del tablero.

`pos(Row, Col)`: la posición actual en el tablero desde la que se está intentando mover.

Dir: la dirección en la que se intenta mover.

El cuerpo del predicado primero usa **efectuar_movimiento** para calcular la nueva posición **pos(Row2,Col2)** que resultaría de moverse en la dirección **Dir** desde la posición actual **pos(Row,Col)**.

Luego, verifica que las coordenadas de la nueva posición estén dentro de los límites del tablero. Las filas y las columnas deben estar dentro del rango de 1 a **N**, que son los límites del tablero.

Si la nueva posición se encuentra fuera de estos límites, el movimiento no es válido y el predicado falla. Si la nueva posición está dentro de los límites, el movimiento es válido y el predicado se cumple.

Por lo tanto, este predicado garantiza que todos los movimientos realizados en el tablero se mantengan dentro de sus límites.

- **Select_cell/4:**

El predicado se usa para seleccionar una celda específica del tablero y devolver el tablero sin esa celda seleccionada.

Los argumentos que recibe son:

IPos: la posición de la celda que se quiere seleccionar en el tablero.

Op: el operador y el valor de la celda seleccionada.

Board: el tablero original.

NewBoard: el tablero resultante después de seleccionar la celda.

El cuerpo del predicado utiliza el predicado predefinido **select/3** de Prolog.

select/3 toma un elemento, una lista y una lista resultante. Intenta eliminar el elemento de la lista original y devuelve la lista resultante.

En este caso, el "elemento" es la celda completa en la posición **IPos** con el operador **Op**, y la "lista" es el tablero **Board**.

Entonces, **select/3** busca la celda **cell(IPos, Op)** en el tablero **Board**, y si la encuentra, la elimina de **Board** para producir **NewBoard**.

Por lo tanto, **select_cell** selecciona una celda del tablero al eliminarla, y devuelve el tablero modificado.

- **Select_dir/3:**

El predicado se utiliza para manejar la selección de una dirección de una lista de direcciones permitidas y para modificar la lista de direcciones de acuerdo con las reglas del juego. La lista

de direcciones permitidas se da en el formato `[dir(Dir, Num) | RestoDirs]`, donde `Dir` es una dirección y `Num` es el número de veces que se puede mover en esa dirección.

- `select_dir(Dir, [dir(Dir, 1) | RestoDirs], RestoDirs)`: Este es el caso base en el que la dirección `Dir` que queremos seleccionar es la cabeza de la lista y sólo se puede mover en esa dirección una vez (`Num` es 1). En este caso, simplemente retornamos el resto de la lista `RestoDirs`.
- `select_dir(Dir, [dir(Dir, Num) | RestoDirs], [dir(Dir, NewNum) | RestoDirs])`: Este caso maneja la situación en la que la dirección `Dir` que queremos seleccionar es la cabeza de la lista y podemos movernos en esa dirección más de una vez (`Num` es mayor que 1). En este caso, restamos uno a `Num` para obtener `NewNum` y construimos una nueva lista con `dir(Dir, NewNum)` en la cabeza y el `RestoDirs` sin cambios.
- `select_dir(Dir, [dir(Dir2, Num) | RestoDirs], [dir(Dir2, Num) | NewDirs])`: Este caso maneja la situación en la que la dirección `Dir` que queremos seleccionar no es la cabeza de la lista. Aquí, `Dir2` es la dirección en la cabeza de la lista y es diferente de `Dir`. En este caso, dejamos la cabeza de la lista `dir(Dir2, Num)` intacta y aplicamos recursivamente `select_dir` al `RestoDirs`.

Estas cláusulas juntas permiten seleccionar una dirección de la lista de direcciones permitidas y actualizar esa lista según las reglas del juego.

- **Aplicar_op/3:**

Este predicado se utiliza para aplicar una operación a un valor dado.

Cada cláusula representa una operación diferente. Los argumentos para `aplicar_op` son `op(Operator, Operand)`, `Value`, y `NewValue`. `op(Operator, Operand)` representa la operación a aplicar, donde `Operator` puede ser uno de los operadores permitidos (+, -, *, o //), y `Operand` es el valor con el que se realizará la operación. `Value` es el valor actual al que se aplicará la operación, y `NewValue` es el resultado de aplicar la operación.

Las cláusulas son las siguientes:

`aplicar_op(op(+, Operand), Value, NewValue)`: Esta cláusula aplica la operación de suma. `NewValue` es el resultado de sumar `Operand` a `Value`.

`aplicar_op(op(-, Operand), Value, NewValue)`: Esta cláusula aplica la operación de resta. `NewValue` es el resultado de restar `Operand` a `Value`.

`aplicar_op(op(*, Operand), Value, NewValue)`: Esta cláusula aplica la operación de multiplicación. `NewValue` es el resultado de multiplicar `Value` por `Operand`.

`aplicar_op(op(//, Operand), Value, NewValue)`: Esta cláusula aplica la operación de división entera. `NewValue` es el resultado de dividir `Value` por `Operand`, redondeando hacia abajo a la entera más cercana si es necesario.

Estas cláusulas permiten que el programa aplique operaciones matemáticas al valor actual mientras se recorre el tablero, de acuerdo con las operaciones definidas en cada celda.

- **Generar_recorrido/6:**

Estos predicados se utilizan para generar todas las rutas posibles en el tablero dada una posición inicial y una serie de direcciones permitidas.

El predicado `generar_recorrido` toma una posición inicial (`Ipos`), la dimensión del tablero (`N`), el tablero en sí mismo (`Board`), las direcciones permitidas (`DireccionesPermitidas`), y devuelve el `RecorridoFinal` y el `ValorFinal` de la ruta generada. Primero, selecciona la celda en la posición inicial y aplica la operación de esa celda a 0, obteniendo `Valor2`. Luego, llama a `generar_recorrido_aux` para continuar generando el recorrido desde `Ipos`.

`generar_recorrido_aux` es un predicado auxiliar que realiza la recursión para generar la ruta. Tiene varios casos base:

Si no hay más direcciones permitidas o celdas en el tablero, devuelve el recorrido parcial actual como el recorrido final y el valor acumulado como el valor final.

Si el número de celdas visitadas es igual al total de celdas en el tablero ($N*N$), también devuelve el recorrido y el valor actual.

Para el caso recursivo, `generar_recorrido_aux` toma la posición actual (`Ipos`), verifica si hay algún movimiento válido a partir de esa posición, selecciona una dirección de las direcciones permitidas, realiza el movimiento en esa dirección para obtener la siguiente posición (`NextPos`), verifica que `NextPos` no esté en la lista de posiciones ya visitadas, selecciona la celda en `NextPos` y aplica su operación al valor acumulado. Añade la nueva posición y su valor correspondiente al recorrido parcial, y luego llama a sí mismo recursivamente con la nueva posición, el nuevo tablero, las nuevas direcciones permitidas, el nuevo valor, el nuevo recorrido parcial, y la lista actualizada de posiciones visitadas.

También hay un caso especial en `generar_recorrido_aux` para manejar direcciones que se han agotado (es decir, direcciones con un contador de 0). En este caso, simplemente ignora esa dirección y procede con el resto de las direcciones permitidas.

En conjunto, estos predicados generan todas las rutas posibles en el tablero dada una posición inicial y un conjunto de direcciones permitidas, siguiendo las reglas del juego.

- **Generar_recorridos/5:**

El predicado `generar_todos_recorridos` genera todas las rutas posibles en el tablero. Para hacer esto, usa el predicado `findall` para recoger todos los pares de (`Recorrido`, `Valor`) donde `Recorrido` es una ruta posible y `Valor` es su valor correspondiente. Para cada celda en el tablero, se llama al predicado `generar_recorrido` para generar una ruta a partir de la posición de esa celda. Todos los pares (`Recorrido`, `Valor`) resultantes se recogen en la lista `Recorridos`.

El predicado `generar_recorridos` simplemente llama a `generar_todos_recorridos` para generar todas las rutas posibles y luego selecciona una ruta y su valor correspondiente de la lista de todas las rutas. Esto es útil si quieres generar todas las rutas posibles una vez y luego explorarlas de alguna manera. En este caso, simplemente se selecciona una ruta y su valor de la lista de todas las rutas posibles.

Entonces, en esencia, `generar_todos_recorridos` es responsable de generar todas las rutas posibles y `generar_recorridos` se utiliza para seleccionar una ruta específica y su valor correspondiente de la lista de todas las rutas posibles.

- **Tablero/5**

Estos son tres predicados diferentes que interactúan juntos para el funcionamiento del predicado `tablero`.

`min_list`: este es un predicado estándar en Prolog que encuentra el valor mínimo en una lista. Aquí se ha implementado de manera recursiva. En el caso base, cuando la lista contiene solo un elemento, ese elemento es el mínimo. En los casos recursivos, se divide la lista en su cabeza (`H`) y cola (`T`), se encuentra el mínimo en la cola (`MinRest`), y si la cabeza es menor o igual a ese mínimo, se establece la cabeza como el nuevo mínimo. Si la cabeza es mayor que `MinRest`, entonces `MinRest` permanece como el mínimo.

`count_valores_minimos`: este predicado cuenta el número de veces que un valor mínimo aparece en una lista de tuplas. En el caso base, si la lista está vacía, entonces el conteo es 0. En los casos recursivos, se divide la lista en su cabeza (una tupla `(_, Valor)`) y cola. Si `Valor` es igual al `ValorMinimo`, se incrementa el contador en 1 y se sigue con el resto de la lista. Si `Valor` es diferente a `ValorMinimo`, no se incrementa el contador y se sigue con el resto de la lista.

`tablero`: este es el predicado principal que utiliza los otros dos predicados para resolver el problema. Primero, genera todas las rutas posibles en el tablero con `generar_recorridos`. Luego, encuentra todos los valores correspondientes a estas rutas y los guarda en la lista `Valores`. A continuación, se usa `min_list` para encontrar el valor mínimo en esta lista, que se guarda como `ValorMinimo`. Finalmente, se usa `count_valores_minimos` para contar cuántas veces aparece este valor mínimo en la lista de rutas generadas, y este conteo se guarda como `NumeroDeRutasConValorMinimo`.

Consultas / Respuestas

```
?- efectuar_movimiento(pos(3,3), n, NewPos).
```

```
NewPos = pos(2,3) ?
```

```
yes
```

```
?- movimiento_valido(4, pos(4,3), n).
```

```
yes
```

```
?- movimiento_valido(4, pos(4,3), s).
```

```
no
```

```
?- board1(Board),select_cell(pos(1,1),op(*,-3),Board,NewBoard).
```

```
Board = [cell(pos(1,1),op(*,-3)),cell(pos(1,2),op(-,1)),cell(pos(1,3),op(-,4)),cell(pos(1,4),op(-,555)),cell(pos(2,1),op(-,3)),cell(pos(2,2),op(+,2000)),cell(pos(2,3),op(*,133)),cell(pos(2,4),op(-,444)),cell(pos(3,1),op(*,0)),cell(pos(3,2),op(*,155)),cell(pos(3,3),op(/,2)),cell(pos(3,4),op(+,20)),cell(pos(4,1),op(-,2)),cell(pos(4,2),op(-,1000)),cell(pos(4,3),op(-,9)),cell(pos(4,4),op(*,4))],
```

```
NewBoard = [cell(pos(1,2),op(-,1)),cell(pos(1,3),op(-,4)),cell(pos(1,4),op(-,555)),cell(pos(2,1),op(-,3)),cell(pos(2,2),op(+,2000)),cell(pos(2,3),op(*,133)),cell(pos(2,4),op(-,444)),cell(pos(3,1),op(*,0)),cell(pos(3,2),op(*,155)),cell(pos(3,3),op(/,2)),cell(pos(3,4),op(+,20)),cell(pos(4,1),op(-,2)),cell(pos(4,2),op(-,1000)),cell(pos(4,3),op(-,9)),cell(pos(4,4),op(*,4))] ?
```

```
yes
```

```
?- select_dir(n, [dir(n, 3), dir(e, 2), dir(s, 1), dir(o, 5)], NewDirs).
```

```
NewDirs = [dir(n,2),dir(e,2),dir(s,1),dir(o,5)] ?
```

```
yes
```

```
?- select_dir(no, [dir(n, 3), dir(e, 2), dir(s, 1), dir(o, 5)], NewDirs).
```

```
no
```

```
?- aplicar_op(op(+, 5), 10, Valor2).
```

```
Valor2 = 15 ?
```

```
yes
?- aplicar_op(op(*, 5), 4, Valor2).
```

```
Valor2 = 20 ?
```

```
yes
```

```
?- board1(_Board),
   _Dirs = [dir(n ,5) ,dir(s ,6) ,dir(e ,7) ,dir(o ,4)],
   generar_recorrido(pos(2 ,2) ,4,_Board ,_Dirs ,Recorrido ,Valor).
```

```
Recorrido = [(pos(2,2),2000),(pos(1,2),1999),(pos(1,1),-5997),(pos(2,1),-
6000),(pos(3,1),0),(pos(4,1),-2),(pos(4,2),-1002),(pos(3,2),-
155310),(pos(3,3),-77655),(pos(2,3),-10328115),(pos(1,3),-
10328119),(pos(1,4),-10328674),(pos(2,4),-10329118),(pos(3,4),-
10329098),(pos(4,4),-41316392),(pos(4,3),-41316401)],
Valor = -41316401 ?
```

```
yes
```

```
?- board1(_Board),
   _Dirs = [dir(n ,5) ,dir(s ,6) ,dir(e ,7) ,dir(o ,4)],
   generar_recorridos(4,_Board ,_Dirs ,R,V).
```

```
R = [(pos(1,1),0),(pos(2,1),-3),(pos(3,1),0),(pos(4,1),-2),(pos(4,2),-
1002),(pos(3,2),-155310),(pos(2,2),-153310),(pos(1,2),-
153311),(pos(1,3),-153315),(pos(1,4),-153870),(pos(2,4),-
154314),(pos(3,4),-154294),(pos(4,4),-617176),(pos(4,3),-
617185),(pos(3,3),-308592),(pos(2,3),-41042736)],
V = -41042736 ?
```

```
yes
```

```
?- board1(_Board),
   _Dirs = [dir(n ,5) ,dir(s ,6) ,dir(e ,7) ,dir(o ,4)],
   tablero(4,_Board ,_Dirs ,VM ,NR).
```

```
NR = 1,
VM = -41042736 ?
```

```
yes
```