Segunda práctica (Programación en ISO-Prolog)

Fecha de entrega: **28 de mayo de 2023**

Recorridos de valor mínimo en tableros de operaciones matemáticas

> Antes de empezar la práctica es **muy importante** empezar por leer las **Instrucciones generales para la realización y entrega las prácticas**, así como la **Preguntas frecuentes sobre Ciao, Deliverit, LPdoc, etc.**, disponibles en Moodle.

Instrucciones específicas

En esta práctica se debe usar aritmética de ISO-Prolog (predicados `is/2`, `>/2`, etc.) y se pueden usar otros predicados predefinidos de ISO-Prolog tales como los de de agregación, inspección de estructuras, etc.

Enunciado de la práctica (8.5 puntos)

Se dispone de un **tablero** NxN donde cada casilla está etiquetada con un operador aritmético binario y un operando entero. Comenzando desde una casilla elegida de forma arbitraria, y con el valor inicial 0 (que pasa a ser el valor actual), se **recorre** el tablero visitando cada una de las casillas exactamente una vez. Por cada casilla visitada se debe realizar la operación indicada utilizando el valor actual como operando izquierdo, y el valor entero que aparece en la casilla como operando derecho. El resultado de esta operación se utilizará como operando izquierdo para la operación asociada a la siguiente casilla visitada, y así sucesivamente. El **valor** final obtenido (es decir, el que se obtiene tras completar el recorrido) depende evidentemente del recorrido realizado.

Tableros: El tablero se representa mediante una lista de celdas definidas mediante estructuras `cell/2` con cabecera `cell(Pos,Op)`. El argumento `Pos` debe tomar valores `pos(Row,Col)` donde las variables `Row` y `Col` identifican la fila y la columna del tablero en la que está ubicada la celda en cuestión. Estas variables únicamente pueden tomar valores entre 1 y N. Por otro lado, la variable `Op` representa a la **operación** matemática asociada a la celda. Estas operaciones se explicitan mediante la estructura `op/2` con cabecera `op(Op,Val)`, donde `Op` es un operador aritmético escogido de entre los pertenecientes al conjunto $\{+, -, *, //\}$, y `Val` es un número entero. A continuación se muestra un ejemplo de tablero junto con la estructura de datos que la representa: [1]

[1] Dado que son términos largos, os recomendamos almacenarlos junto con vuestro programa en predicados `board1(...)`, `board2(...)`, etc. como hacemos a continuación.

*(-3)	-1	-4	-555
-3	+2000	*133	-444
*0	*155	//2	+20
-2	-1000	-9	*4

```
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))]).
```

Nótese que en este ejemplo la lista de estructuras que representa el tablero está ordenada de izquierda a derecha y de arriba a abajo. El alumno no debe asumir que esto será siempre así, ya que cualquier permutación de esta lista constituye un tablero igualmente válido y equivalente al original.

Direcciones: las direcciones de tránsito de una casilla a otra se identifican por las siguientes constantes: `n` (norte), `s` (sur), `e` (este), `o` (oeste), `no` (noroeste), `ne` (noreste), `so` (suroeste) y `se` (sureste),

Direcciones Permitidas: Para realizar el tránsito de una casilla a la siguiente es necesario escoger una dirección de entre las disponibles en una lista denominada `DireccionesPermitidas`. Esta lista contiene una serie de estructuras `dir/2` con cabecera `dir(Dir,Num)`, donde `Dir` es una de las direcciones anteriores y `Num` es el número máximo de veces que puede utilizarse dicha dirección durante el cálculo de un recorrido. Por ejemplo, con la lista:
`DireccionesPermitidas =[dir(n,3), dir(s,4), dir(o,2), dir(se,10)]`
solamente estaría permitido transitar desde la casilla actual a la casilla ubicada al norte, sur, oeste o sureste de esta siempre y cuando la casilla exista (es decir, que se encuentre dentro de los límites del tablero). Asimismo, durante todo el recorrido, como máximo se puede transitar a las direcciones `n`, `s`, `o` y `se` en 3, 4, 2 y 10 ocasiones respectivamente.

Cabe también aclarar que podría suceder que no sea posible completar ningún recorrido con ciertas listas de direcciones permitidas independientemente de cual sea la casilla de comienzo.

Apartado 1 (6 puntos)

El objetivo de este primer apartado es implementar un predicado que genere recorridos en tableros. Se pide resolver el problema de forma gradual. Primero se deben implementar los siguientes predicados:

- (0.5 puntos) `efectuar_movimiento(Pos,Dir,Pos2)` : `Pos2` es la posición resultante de moverse desde `Pos` en la dirección indicada por `Dir`.
- (0.5 puntos) `movimiento_valido(N,Pos,Dir` : debe tener éxito si la posición resultado de moverse desde `Pos` en la dirección indicada por `Dir` representa una posición válida en un tablero de NxN.
- (0.5 puntos) `select_cell(IPos,Op,Board,NewBoard)`: extrae (se puede consultar el predicado Prolog `select/3` como inspiración) la celda con posición `IPos` de la lista `Board` obteniendo `NewBoard` (sin dicha celda) y unificando `Op` con la operación asociada a la respectiva celda.
- (0.5 puntos) `select_dir(Dir,Dirs,NewDirs)`: extrae (se puede consultar el predicado Prolog `select/3` como inspiración) una dirección `Dir` de las permitidas en `Dirs`, obteniendo en `NewDirs` la lista de direcciones permitidas que restan. Esta puede ser la misma lista `Dirs` pero con el número de aplicaciones de la dirección seleccionada disminuido en uno, o, si esta fuera la última aplicación permitida, sin ese elemento.
- (0.5 puntos) `aplicar_op(Op, Valor, Valor2)` : dado `Op=op(Operador,Operando)`, aplica la operación indicada en `Valor` para obtener `Valor2`.

Finalmente, para terminar este apartado se debe escribir el predicado `generar_recorrido/6` (3.5 puntos). Este predicado está formalizado mediante la cabecera:

```
generar_recorrido(Ipos,N,Board,DireccionesPermitidas,
                  Recorrido,Valor)
```

donde `Ipos` representa posición (columna y fila en forma de término `pos(Irow,Icol)`) de la celda desde la que se inicia el recorrido del tablero `Board` de dimensión `N` teniendo en cuenta las direcciones permitidas indicadas por `DireccionesPermitidas`. Por otro lado, `Recorrido` es una lista de tuplas `(Pos, ValorActual)`, donde la variable `Pos` hace referencia a las estructuras `pos/2` con cabecera `pos(Row,Col)` que representan la posición de las celdas que forman parte del recorrido generado cuyo valor final es `Valor`. Asimismo, la variable `ValorActual` representa al valor actual del recorrido tras la realización de la operación indicada en la celda correspondiente. Nótese que este predicado debe devolver por vuelta hacia atrás todos los recorridos que parten desde la casilla señalada por la posición `Ipos` usando únicamente las direcciones reseñadas en la lista `DireccionesPermitidas`. A modo de ejemplo, se muestra el resultado de la ejecución de una consulta que busca obtener por vuelta hacia atrás todos

los recorridos (en este caso un total de 27) posibles que pueden construirse partiendo desde la casilla `(2, 2)` del tablero de ejemplo mostrado anteriormente, usando para ello la lista de direcciones permitidas `[dir(n,5),dir(s,6),dir(e,7),dir(o,4)]`.

```
?- 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).
```

Esta que se muestra a continuación es una de las 27 respuestas que devuelve Prolog tras la ejecución de la pregunta anterior:

```
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(4,3),-1011),
             (pos(4,4),-4044),
             (pos(3,4),-4024),
             (pos(2,4),-4468),
             (pos(1,4),-5023),
             (pos(1,3),-5027),
             (pos(2,3),-668591),
             (pos(3,3),-334295),
             (pos(3,2),-51815725)],
Valor = -51815725 ?
```

Apartado 2 (1.25 puntos)

A continuación, el alumno debe escribir el predicado `generar_recorridos/5` con cabecera `generar_recorridos(N, Board, DireccionesPermitidas, Recorrido, Valor)`. Este predicado genera por vuelta hacia atrás el valor `Valor` del recorrido `Recorrido` correspondiente a todos los recorridos posibles en el tablero `Board` de dimensión `N` partiendo *desde cualquier casilla*, y teniendo en cuenta las direcciones permitidas en la lista `DireccionesPermitidas`. Este predicado facilitará al alumno el averiguar en el siguiente apartado cual es el valor mínimo de entre los proporcionados por todos los recorridos posibles. El recorrido devuelto en la variable `Recorrido` debe seguir el mismo formato que en el predicado anterior. A continuación se muestra un ejemplo de uso del citado predicado.

```
?- 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(4,3),-1011),
      (pos(4,4),-4044),
      (pos(3,4),-4024),
      (pos(2,4),-4468),
      (pos(1,4),-5023),
      (pos(1,3),-5027),
      (pos(1,2),-5028),
      (pos(2,2),-3028),
      (pos(3,2),-469340),
      (pos(3,3),-234670),
      (pos(2,3),-31211110)],
V = -31211110 ?
```

Al igual que en el ejemplo anterior, aquí solamente se muestra una única respuesta. No obstante, el alumno debe tener en cuenta que este ejemplo devuelve un total de 362 soluciones por vuelta hacia atrás. El predicado implementado por el alumno debe devolver por vuelta hacia atrás todas las soluciones posibles.

Apartado 3 (1.25 puntos)

Para finalizar, se pide al alumno que escriba un predicado `tablero/5` descrito mediante la cabecera:

```
tablero(N, Tablero, DireccionesPermitidas, ValorMinimo,
   NumeroDeRutasConValorMinimo)
```

que se hace cierto si `ValorMinimo` unifica con el mínimo valor final que es posible obtener teniendo en cuenta todas las rutas posibles comenzando en una casilla cualquiera del tablero de dimensión `N`, y utilizando únicamente los movimientos indicados en la lista `DireccionesPermitidas` para transitar entre las diferentes casillas del tablero. Además, la variable `NumeroDeRutasConValorMinimo` debe unificarse con el número de rutas existentes que permiten obtener el valor mínimo indicado por `ValorMinimo`. Si no existiese ninguna ruta posible, el programa debe fallar. Se recomienda al alumno que repase los predicados de agregación estudiados en las clases de teoría porque pueden serle de mucha utilidad en la implementación de este predicado. A continuación se muestra un ejemplo de uso de este predicado.

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

NR = 1,
VM = -246992940 ? ;
no
```

2. MEMORIA/MANUAL (1.5 puntos, fichero `code.pdf`)

(Ver las instrucciones generales para las prácticas.)

3. PUNTOS ADICIONALES (para subir nota):

- Ejercicio complementario en '*programación alfabetizada*' ('literate programming'):

  Se darán puntos adicionales a las prácticas que realicen la documentación de los predicados insertando en el código aserciones y comentarios del lenguaje Ciao Prolog, y entregando como memoria o parte de ella el manual generado automáticamente a partir de dicho código, usando la herramienta `lpdoc` (incluida con Ciao Prolog). En este caso se puede entregar este documento como memoria, y se recomienda por tanto escribir este manual de forma que incluye el contenido que se solicita para la memoria.

- Ejercicio complementario en *codificación de casos de prueba*:

  También se valorará el uso de aserciones `test` que definan casos de prueba para comprobar el funcionamiento de los predicados. Estos casos de test se deben incluir en el mismo fichero con el código.

Hemos dejado en Moodle instrucciones específicas sobre cómo hacer todo esto y un ejemplo de código que tiene ya comentarios y tests, para practicar corriendo `lpdoc` sobre él y ejecutando los tests.