Segunda práctica (Programación en ISO-Prolog)

Fecha de entrega: ~~30 de mayo~~ 10 de junio de 2022

Marias y Arañas

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 el alumno repasará los conceptos fundamentales estudiados en el bloque temático de programación con ISO-Prolog. Por ello, además de las instrucciones generales anteriores, como instrucciones específicas para esta práctica se recuerda que sí se pueden usar los predicados predefinidos y de las librerías de Prolog, incluida aritmética de Prolog, corte, etc.

Parte 1 (5 puntos). Particiones M-arias de un número.

Una partición de un número entero positivo `N` se define como un conjunto de números enteros positivos que suman `N`, escritos en orden descendente. Por ejemplo, para el número 10 tenemos:

```
10 = 4+3+2+1
```

Una partición es M-aria si cada término de dicha partición es una potencia de M. Por ejemplo, las particiones 3-arias de 9 son:

```
9
3+3+3
3+3+1+1+1
3+1+1+1+1+1+1
1+1+1+1+1+1+1+1+1
```

El objetivo de esta parte es escribir un predicado `maria/3` tal que el tercer argumento es el número de particiones `M`-arias del segundo argumento siendo `M` el primer argumento. Por ejemplo, `?- maria(3,9,M).` debe devolver `M=5`.

Para ello, se escribirán los siguientes predicados:

Predicado 1.1 (1.5 puntos) Definir `pots(M,N,Ps):` que, dados `M` y `N` enteros, devuelve en `Ps` una lista con las potencias de ~~`N`~~ `M` que son menores o iguales que ~~`M`~~ `N`, en orden descendente. Ejemplos:

```
?- pots(3,9,Ps).
Ps = [9,3,1] ? ;
no
?- pots(5,123,Ps).
Ps = [25,5,1] ? ;
no
```

Predicado 1.2 (2 puntos) Definir `mpart(M,N,P):` tal que, dados `M` y `N` enteros, devuelve en `P` por backtracking todas las particiones `M`-arias de `N`, *representadas como listas de enteros*. Las soluciones deben ir ordenadas con las listas más cortas primero, como en el ejemplo. Ejemplo:

```
?- mpart(3,9,P).
P = [9] ? ;
P = [3,3,3] ? ;
P = [3,3,1,1,1] ? ;
P = [3,1,1,1,1,1,1] ? ;
P = [1,1,1,1,1,1,1,1,1] ? ;
no
```

Predicado 1.3 (1.5 puntos) Definir `maria(M,N,NPart):` tal que el tercer argumento es el número de particiones `M`-arias del segundo argumento siendo `M` el primer argumento. A continuación se presentan varios ejemplos con la pregunta y la salida esperada. El programa debe entregar la solución en un tiempo razonable. En la corrección se probarán ejemplos de tamaños crecientes.

```
?- maria(3,9,M).
M = 5.

?- maria(3,47,M).
M = 63.

?- maria(5,123,M).
M = 75.

?- maria(7,4321).
M = 144236.
```

Parte 2 (3.5 puntos). Arañas de expansión de un grafo.

Una araña de expansión de un grafo se define como un árbol de expansión que además es una araña. Un árbol de expansión es un sub-grafo que es un árbol y que contiene todos los vértices del grafo. Una araña es un grafo con como máximo un vértice cuyo grado —número de aristas incidentes a él— es 3 o más. Por tanto, una araña de expansión de un grafo es un sub-grafo que es un árbol, contiene todos los vértices del grafo, y como máximo un sólo vértice de grado 3 o más.

No todos los grafos contienen una araña de expansión. Se pide al alumno que escriba un predicado `aranya/0` que tenga éxito únicamente una vez si el grafo proporcionado contiene una araña de expansión, y que falle de forma finita —es decir, sin bucles infinitos— en caso contrario.

El grafo se representa como una lista de estructuras `arista/2` que tienen como argumentos los nodos que forman dicha arista. Estas estructuras se guardarán como hechos llamando al predicado `guardar_grafo/1`.

En este problema se asumirá que los grafos de entrada son conexos y no dirigidos, por lo que para representar la conexión entre dos vértices a y b, incluiremos en la base de hechos el hecho arista(a,b) o el hecho arista(b,a), pero no ambos.

A continuación se presentan dos ejemplos con la pregunta y la salida esperada. Este primer grafo:

```
?- guardar_grafo(
  [ arista(d,a),
    arista(d,b),
    arista(d,c),
    arista(d,e),
    arista(a,x),
    arista(b,y),
    arista(c,z),
    arista(a,b),
    arista(y,c) ]).
yes
```

sí contiene una araña de expansión::

```
?- aranya.
yes
```

Este grafo en cambio:

```
?- guardar_grafo(
  [ arista(a,e),
    arista(b,e),
    arista(e,f),
    arista(f,d),
    arista(f,c) ]).
```

no contiene una araña de expansión:

```
?- aranya.
no
```

Predicado 2.1 (1.5 puntos) Definir el predicado `guardar_grafo(G):` tal que, dado un grafo representado como una lista de aristas, con el formato anterior, deje asertados en la base de datos como hechos del predicado `arista/2` los elementos de `G`. Para ello, se debe definir como *dinámico* el predicado `arista/2` en el programa. Al llamar al predicado `guardar_grafo(G)` se debe *borrar cualquier hecho* que hubiera guardado anteriormente de este predicado.

Por ejemplo, llamando a `?- guardar_grafo([arista(a,e), arista(b,e), arista(e,f), arista(f,d), arista(f,c)]).` deben quedar añadidos al programa los hechos:

```
arista(a,e).
arista(b,e).
arista(e,f).
arista(f,d).
arista(f,c).
```

Predicado 2.2 (2 puntos) Definir el predicado `aranya/0:` tal que, dado un grafo `G` *guardado en la base datos*, tenga éxito únicamente una vez si el grafo proporcionado contiene una araña de expansión, y que falle de forma finita —es decir, sin bucles infinitos— en caso contrario.

Parte 3 (1.5 puntos). MEMORIA/MANUAL (fichero `code.pdf`)

Se debe entregar también un fichero `code.pdf` con la memoria de la práctica. La forma más sencilla y eficiente de hacerlo es utilizando la herramienta `lpdoc` (ver la siguiente sección y las instrucciones generales de las prácticas).

PUNTOS ADICIONALES (para subir nota):

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

  Se darán puntos adicionales a las prácticas que realicen la documentación de dichos 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` del sistema Ciao. En este caso se puede entregar este documento como memoria, y se recomienda por tanto escribir este manual de forma que incluya el contenido que se solicita para la memoria. Las aserciones aserciones y comentarios `lpdoc` deben estar incluidos en el fichero `code.pl` para que puedan valorados.

- 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 deben estar incluidos en el mismo fichero `code.pl` con el código para que puedan ser valorados.

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.