UTF-8

Marias y Arañas en ISO-Prolog

Nicolas Cossio Miravalles, B190082

Table of Contents

code

Este módulo define operaciones con 'Marias y Arañas'

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 termino 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 han escrito los siguientes predicados: o

Predicado 1.1 pots(M,N,Ps):

Devuelve en `Ps` una lista con las potencias de `M` que son menores o iguales que `N`, en orden descendente. `M` y `N` son enteros.

```
pots(1,_1,[1]) :- !.
pots(M,N,Ps) :-
    pots_helper(M,1,N,[1],Ps).
```

Dado que todo número tiene como potencia el 1, hemos empezado la lista en dicho número, que pasamos a una función auxiliar `pots_helper/5`:

```
pots_helper(M,LastPower,Max,Powers,Powers) :-
    LastPower*M>Max,
    !.
pots_helper(M,LastPower,Max,Powers,Res) :-
    NewPower is LastPower*M,
    pots_helper(M,NewPower,Max,[NewPower|Powers],Res).
```

La función hace lo siguiente:

Devuelve en `Res` una lista con las potencias de `M` que son menores o iguales que `Max`, en orden descendente. `M`es la base de las potencias y `Max` es el numeero que marca el máximo a la que estas pueden llegar, ambos son enteros.

`ExpLastPower` es el la última potencia calculada. La forma de resolver este enunciado ha sido mediante programacion dinamica, guardando el resultado en la lista y dandolo como parametro en la llamada recursiva, lo que permite poder reusarlo para calcular la siguiente potencia, multiplicando nada más que por la base `M`. Esto se hace hasta mirando en cada llamada si la siguiente potencia que se va a calcular es superior al `Max`, cuando esto ocurre se hace un corte que pone `Powers`, que es la lista que se lleva en cada llamada, a `Res`, que es el resultado final.

Ejemplos de uso:

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

Predicado 1.2 mpart(M,N,P):

La solución que he planteado utiliza una función auxiliar `mpart_backtrack(Powers, Counter, Solution)`:

```
mpart_backtrack(_1,0,[]).
mpart_backtrack([FirstPower|RestPowers],Counter,[FirstPower|PRest]) :-
    Counter>0,
    NewCounter is Counter-FirstPower,
    mpart_backtrack([FirstPower|RestPowers],NewCounter,PRest).
mpart_backtrack([_1|RestPowers],Counter,P) :-
    Counter>0,
    mpart_backtrack(RestPowers,Counter,P).
```

Esta sentencia/s se encarga de insertar las potencias `Powers` dadas por `pots`, tantas veces como haga falta en función de un contador `Counter`, que ha sido inicializado al número que se quiere particionarN, este contador se va decrementando en función de la potencia (la cabeza de `Powers`) que se inserta. La gracia de la implementación es la llamada recursiva que pasa la cola de la solución como la siguiente solución a hacer, manteniendo la lista de potencias en la llamada.

Esto por supuesto no daria todas las soluciones, es por esto que hace falta una cláusula que nos permita hacer backtracking, para calcular esos casos intermedios en el arbol de soluciones. El backtracking ocurre en la esta cláusula, donde se hace la llamada recursiva que da el resto de soluciones intermedias, dando esta vez la lista de potencias recortada, ya que la cabeza (potencia actual) ya se ha usado y no se puede/debe añadir más a la lista.

Se llega al caso base cuando la lista de sumandos de potencias esta vacia y el contador es 0, indicando que se ha llegado a una solución.

Dados M y N, que son enteros, devuelve en P todas las particiones M-arias de N, representadas como listas de enteros. Las soluciones deben ir ordenadas con las listas más cortas primero.

```
mpart(M,N,P) :-
    pots(M,N,Powers),
    mpart_backtrack(Powers,N,P).
 ? -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

?- mpart(5, 123, X).

X = [25,25,25,25,5,5,5,5,1,1,1] ? ;

X = [25,25,25,25,5,5,5,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,25,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,25,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;
```

```
X = [25,25,25,25,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,5,5,5,5,5,5,5,5,5,1,1,1] ? ;

X = [25,25,25,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;;

X = [25,25,25,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,25,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,25,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,25,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,25,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1] ? ;

X = [25,25,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1] ? ;

X = [25,25,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,25,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

X = [25,25,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,25,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

```
X = [25,25,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1] ? ;

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1] ? ;

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? ;

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,11

X = [25,5,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

X = [25,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

.......

no
```

Predicado 1.3 maria(M, N, NPartes):

La solución que he planteado es muy simple. Se usa la función findall para agregar todos los resultados dados al llamar a la función `mparts(M,N,X)`, esto devuelve en X una lista de listas con todos los resultados agregados. Despues se llama a la función `length(X, NPartes)` del paquete

aggregates para calcular facilmente cuantas soluciones se tiene que es la solución de la partición M-aria de N.

```
maria(M,N,NParts) :-
    findall(Partition,mpart(M,N,Partition),X),
    length(X,NParts).

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

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

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

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

?- maria(3, 1000, X).
X = 1295579 ?

?- maria(3, 1000, X).
X = 1295579 ?
```

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

Una araña de expansión de un grafo es un subgrafo que contiene todos los vértices de este menos, con grado máximo 3. No todos los grafos contienen una araña de expansión. A continuación se describe un predicado `aranya/0` que tenga éxito únicamente una vez si el grafo proporcionado contiene una araña de expansión.

El grafo se representa como una lista de estructurasarista/2que 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.

Predicado 2.1 guardar_grafo(G):

La solución planteada consiste en borrar cualquier sentencia de la base de hechos que siga el predicado `arista/2`. Para ello se usa la función `dropall(X,arista(_,_)`. Después se pasa a una función auxiliar que coge de la cabeza de la lista la arista a insertar en la base de hechos y la aserta mediante `assert(X)`. Para que esto funcione claro hemos tenido que definir antes el predicado `arista/2` como dinámico. Finalmente se hace una llamada recursiva a la función auxiliar con la cola de la lista de aristas. El caso base es cuando se llega a una lista vacia.

```
guardar_grafo(G) :-
    clean_edges,
    save_graph(G).
clean_edges :-
    retractall(arista(_1,_2)).
```

```
save_graph([]).
save_graph([Edge|ResEdges]) :-
    assert(Edge),
    save_graph(ResEdges).

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

yes
?-

X = a,
Y = e ? ;

X = b,
Y = e ? ;

X = e,
Y = f ? ;

X = f,
Y = d ? ;

X = f,
Y = c ? ;

no
?- guardar_grafo([arista(1,2), arista(2,3),
arista(4,5), arista(6,7), arista(8,9)]).

yes
?- arista(X, Y).

X = 1,
Y = 2 ? ;

X = 2,
Y = 3 ? ;

X = 4,
Y = 5 ? ;

'X = 6,
Y = 7 ? ;

X = 8,
Y = 9 ? ;

no
```

Tests:

Se han planteado una serie de tests automaticos, que son los siguientes, y dan estos resultados.

:- test pots(M, N, Ps) : (M = 3, N = 9) => (Ps = [9,3,1]) + not_fails # 'Pots(3,9,X)->[9,3,1]'.

:- test pots(M, N, Ps) : (M = 5, N = 123) => (Ps = [25,5,1]) + not_fails # 'Pots(5,123,X)->[25,5,1]'.

:- test mpart(2,2,NP) => (NP = [2]; NP = [1,1]) + (try_sols(10),not_fails).

:- test mpart(3,9,NP) => (NP = [9]; NP = [3,3,3]; NP=[3,3,1,1,1]; NP = [3,1,1,1,1,1,1]) + (try_sols(4),not_fails) # 'mpart(3,9,X)-> [9]; [3,3,3]; [3,3,1,1,1]; [3,1,1,1,1,1,1]'.

:- test mpart(5,50,NP) => (NP = [25, 25]; NP = [25,5,5,5,5,5]; NP= [25,5,5,5,5,5,1,1,1,1,1]; NP = [25,5,5,5,1,1,1,1,1,1,1,1,1,1,1]) + (try_sols(4),not_fails) # 'mpart(5,50,X)-> [25, 25]; [25,5,5,5,5,5]; [25,5,5,5,5,5,1,1,1,1,1]; [25,5,5,5,5,1,1,1,1,1,1,1,1,1,1]'.

:- test pots(M, N, Ps) : (M = 3, N = 9) => (Ps = [9,3,1]) + not_fails # 'Pots(3,9,X)->[9,3,1]'.

:- test pots(M, N, Ps) : (M = 5, N = 123) => (Ps = [25,5,1]) + not_fails # 'Pots(5,123,X)->[25,5,1]'.

:- test maria(M, N, NParts) : (M = 3, N = 9) => (NParts = 5) + not_fails # 'maria(3,9,X)->5'.

:- test maria(M, N, NParts) : (M = 3, N = 47) => (NParts = 63) + not_fails # 'maria(3,47,X)->63'.

:- test maria(M, N, NParts) : (M = 5, N = 123) => (NParts = 75) + not_fails # 'maria(5,123,X)->75'.

test_guardar_grafo_1 :- guardar_grafo([arista(1, 2), arista(2, 3)]), arista(1, 2), arista(2, 3).

test_guardar_grafo_2 :- guardar_grafo([arista(5, 2), arista(7, 3)]), arista(1, 2), arista(2, 3).

test_guardar_grafo_3 :- test_guardar_grafo_1, guardar_grafo([arista(5, 2), arista(7, 3)]), arista(5, 2), arista(7, 3).

:- test test_guardar_grafo_1 + not_fails # 'guardar_grafo([arista(1, 2), arista(2, 3)]) -> no debe fallar, se comprueba si las aristas existen'.

:- test test_guardar_grafo_2 + fails # 'guardar_grafo([arista(5, 2), arista(7, 3)]) -> debe fallar, se comprueba si las aristas anteriores existen'.

:- test test_guardar_grafo_3 + not_fails # 'guardar_grafo([arista(1, 2), arista(2, 3)]), guardar_grafo([arista(5, 2), arista(7, 3)]) ->no debe fallar, se comprueba si las ultimas aristas metidas existen'.

PASSED: (lns 329-330) pots/3 'Pots(3,9,X)->[9,3,1]'.

PASSED: (lns 331-331) pots/3 'Pots(5,123,X)->[25,5,1]'.

PASSED: (lns 354-356) mpart/3.

PASSED: (lns 356-357) mpart/3 'mpart(3,9,X)-> [9]; [3,3,3]; [3,3,1,1,1]; [3,1,1,1,1,1,1]'.

PASSED: (lns 357-358) mpart/3 'mpart(5,50,X)-> [25, 25]; [25,5,5,5,5,5]; [25,5,5,5,5,1,1,1,1,1]; [25,5,5,5,1,1,1,1,1,1,1,1,1,1,1]'.

PASSED: (lns 370-371) maria/3 'maria(3,9,X)->5'.

PASSED: (lns 372-372) maria/3 'maria(3,47,X)->63'.

PASSED: (lns 373-373) maria/3 'maria(5,123,X)->75'.

PASSED: (lns 412-413) test_guardar_grafo_1/0 'guardar_grafo([arista(1, 2), arista(2, 3)]) -> no debe fallar, se comprueba si las aristas existen'.

PASSED: (lns 414-415) test_guardar_grafo_2/0 'guardar_grafo([arista(5, 2), arista(7, 3)]) -> debe fallar, se comprueba si las aristas anteriores existen'.

PASSED: (lns 416-417) test_guardar_grafo_3/0 'guardar_grafo([arista(1, 2), arista(2, 3)]), guardar_grafo([arista(5, 2), arista(7, 3)]) ->no debe fallar, se comprueba si las ultimas aristas metidas existen'.

Note: Total: Passed: 11 (100.00%) Failed: 0 (0.00%) Precond Failed: 0 (0.00%) Aborted: 0 (0.00%) Timeouts: 0 (0.00%) Total: 11 Run-Time Errors: 0

No he hecho el enunciado 2.2 por falta de tiempo.

Usage and interface

- **Library usage:**
  `:- use_module(/Users/nicolascossio/UPM/Prolog/iso-prolog/code.pl).`
- **Exports:**
  - *Predicates:*
    `author_data/4`, `pots_helper/5`, `pots/3`, `mpart/3`, `mpart_backtrack/3`, `maria/3`, `arista/2`, `guardar_grafo/1`, `clean_edges/0`, `save_graph/1`, `test_guardar_grafo_1/0`, `test_guardar_grafo_2/0`, `test_guardar_grafo_3/0`.
  - *Multifiles:*
    `Σcall_in_module/2`.

Documentation on exports

author_data/4: PREDICATE

No further documentation available for this predicate.

pots_helper/5: PREDICATE

Usage: `pots_helper(M,LastPower,Max,Powers,Res)`

Devuelve en `Res` una lista con las potencias de `M` que son menores o iguales que `Max`, en orden descendente. `M` y `Max` son enteros. `ExpLastPower` es el la última potencia calculada. La forma de resolver este enunciado ha sido mediante programacion dinamica, guardando el resultado en la lista y dandolo como parametro en la llamada recursiva, lo que permite poder reusarlo para calcular la siguiente potencia, multiplicando nada más que por la base `M`. `Powers` es la lista que se lleva en cada llamada y `Res` es el resultado final

pots/3: PREDICATE

Usage: `pots(M,N,Ps)`

Devuelve en `Ps` una lista con las potencias de `M` que son menores o iguales que `N`, en orden descendente. `M` y `N` son enteros,

```
pots(1,_1,[1]) :- !.
pots(M,N,Ps) :-
    pots_helper(M,1,N,[1],Ps).
```

mpart/3: PREDICATE

Usage: `mpart(M,N,P)`

Dados `M` y `N`, que son enteros, devuelve en `P` todas las particiones M-arias de `N`, representadas como listas de enteros. Las soluciones van ordenadas con las listas más cortas primero.

```
mpart(M,N,P) :-
    pots(M,N,Powers),
    mpart_backtrack(Powers,N,P).
```

mpart_backtrack/3: PREDICATE

Usage: `mpart_backtrack(Powers,N,P)`

```
mpart_backtrack(_1,0,[]).
mpart_backtrack([FirstPower|RestPowers],Counter,[FirstPower|PRest]) :-
    Counter>0,
    NewCounter is Counter-FirstPower,
    mpart_backtrack([FirstPower|RestPowers],NewCounter,PRest).
mpart_backtrack([_1|RestPowers],Counter,P) :-
    Counter>0,
    mpart_backtrack(RestPowers,Counter,P).
```

maria/3: PREDICATE

Usage 1: `maria(M,N,NParts)`

```
maria(M,N,NParts) :-
    findall(Partition,mpart(M,N,Partition),X),
    length(X,NParts).
```

Usage 2: `maria(M,N,NParts)`

```
maria(M,N,NParts) :-
    findall(Partition,mpart(M,N,Partition),X),
    length(X,NParts).
```

arista/2: PREDICATE

No further documentation available for this predicate. The predicate is of type *dynamic*.

guardar_grafo/1: PREDICATE

Usage: `guardar_grafo(G)`

Guarda en la base de hechos el grafo `G` representado en una lista de arista/2. {guardar_grafo/1

clean_edges/0: PREDICATE

Usage:

Elimina de la base de hechos el grafo último guardado, representado en una lista de arista/2. {clean_edges/0

save_graph/1: PREDICATE

Usage: `save_graph(G)`

Función auxiliar de guardar_grafo/1, se encarga de guardar la arista que encabeza la lista G, llamandose recursivamente hasta que se queda sin aristas que guardar. {save_graph/1

test_guardar_grafo_1/0: PREDICATE

No further documentation available for this predicate.

test_guardar_grafo_2/0: PREDICATE

No further documentation available for this predicate.

test_guardar_grafo_3/0: PREDICATE

No further documentation available for this predicate.

Documentation on multifiles

Σcall_in_module/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

Documentation on imports

This module has the following direct dependencies:

– *Application modules:*

`operators`, `dcg_phrase_rt`, `datafacts_rt`, `dynamic_rt`, `classic_predicates`, `lists`, `aggregates`, `between`.

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `hiord_rt`, `stream_basic`, `io_basic`, `runtime_control`, `basic_props`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `classic`, `runtime_ops`, `dcg`, `dcg/dcg_phrase`, `dynamic`, `datafacts`, `assertions`, `assertions/assertions_basic`, `regtypes`.

References

(this section is empty)