

Programación declarativa

Práctica 2: Marias y Arañas

Autor:

Daniel Corrales Falcó; b190410; d.corralesf@alumnos.upm.es

Índice:

- Código.....	3
○ Explicación de métodos.....	6
- Pruebas.....	8

CÓDIGO:

```
:-module(_,_,[classic,assertions,regtypes]).
```

```
author_data('Corrales','Falco','Daniel','b190410').
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%PREDICADOS  
AUXILIARES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Parte 1
```

```
    %1.1
```

```
%llenado de la lista
```

```
pots_aux(M,N,Ps,PsAux,I):-
```

```
    M1 is M*I,
```

```
    M1 <= N,
```

```
    pots_aux(M1,N,Ps,[M1 | PsAux],I).
```

```
%condicion de parada
```

```
pots_aux(M,N,Ps,PsAux,I):-
```

```
    M1 is M*I,
```

```
    M1 > N,
```

```
    Ps = PsAux.
```

```
    %1.2
```

```
mpart_aux(A,N,L):-
```

```
    member(X,A),
```

```
    R is (N-X),
```

```
    R = 0,
```

```
    L = [X].
```

```
mpart_aux(A,N,L):-
```

```
    member(X,A),
```

```
    R is (N-X),
```

```

R > 0,
mpart_aux(A,R,L1),
L = [X | L1].

ordenada([]).
ordenada([_]).
ordenada(Ps) :-
    Ps = [A,B | Z],
    A >= B,
    ordenada([B|Z]).

%Parte 2

    %2.1

guardar_grafo_aux([]).
guardar_grafo_aux([X|G]):-
    assert(X),
    guardar_grafo_aux(G).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%PREDICADOS
PRINCIPALES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Parte 1

    %1.1 -- Predicado pots

pots(0,_,[0]).
pots(1,_,[1]).
pots(_0,[ ]).
pots(_1,[1]).
pots(M,N,Ps):-
    1=<N,
    pots_aux(1,N,Ps,[1],M).

```

%1.2 -- Predicado mpart(M,N,P)

mpart(M,N,P):-

 pots(M,N,Ps),

 mpart_aux(Ps,N,P),

 ordenada(P).

%1.3 -- Predicado maria(M,N,NPart)

maria(M,N,NPart).

%segunda parte

%2.1 -- Predicado guardar_grafo

:-dynamic(arista/2).

guardar_grafo(G):-

 abolish(arista/2),

 guardar_grafo_aux(G).

%2.2 -- Predicado aranya

aranya.

- EXPLICACIÓN DE LOS MÉTODOS IMPLEMENTADOS:

1.1 – Predicado pots/3:

Para este predicado se nos pedía obtener la lista de múltiplos de un número que no superaran a otro. Para implementarlo puse varios casos base, ya que hay que poder operar por ejemplo potencias de 0, las cuales van a ser siempre 0, las de 1 que de forma análoga siempre será 1 y los casos en los que el número que establece el límite es 1, ya que solo habrá que devolver 1, representando la potencia de X^0 . Ahora bien, una vez definidos estos casos, en el propio predicado compruebo si el número límite es \geq a 1, y a continuación accedo a un predicado auxiliar, pots_aux/5.

Este predicado es recursivo y su tarea es llenar esa lista encontrando los múltiplos del número que pasamos a un inicio que sean menores que el número límite. El caso base es aquel en el que el número límite es menor que esa potencia, en cuyo caso, devolvemos la lista y terminamos las llamadas recursivas.

1.2 - Predicado mpart/3:

En este predicado se nos pide hallar las particiones M-arias de un número N. Para implementar su correcto funcionamiento, primero realizo una llamada a la función implementada anteriormente, el predicado pots, para así obtener una lista de los múltiplos de M menores que N. A continuación, realizo una llamada a un predicado auxiliar recursivo mpart_aux/3, al cual le paso N, la lista de múltiplos de M menores que N, y una lista vacía.

En este predicado auxiliar realizo la construcción de las distintas posibles particiones M-arias de N. En este mi proceso es el siguiente, saco la cabeza de la lista de múltiplos, se la resto a N, y a partir de aquí tengo 2 casos, uno en el que el resultado de la resta es positivo y otro en el que el resultado es 0. En el primer caso hago una llamada recursiva, pues puedo seguir quitando valores y a su vez, añado ese valor a la lista a devolver. En el segundo caso simplemente añado ese valor a la lista y devuelvo, es el caso base de parada.

Una vez hallada la lista de particiones, llamo a la función ordenada, la cual ordena las particiones para eliminar posibles repeticiones.

1.3 – Predicado maría/3:

Sin implementar

2.1 – Predicado guardar grafo/1:

En este predicado se pedía eliminar las instancias de un predicado dinámico “arista/2”, que representan aristas de un grafo, que ya estuvieran en la base de hechos, y meter en la base de hechos las nuevas aristas proporcionadas por la lista de entrada G. Esto fue realizado de la siguiente manera: lo primero era eliminar las instancias ya existentes de aristas, lo cual fue resuelto de una forma sencilla mediante el uso del predicado abolish(Predicado), el cual elimina de la base de hechos las instancias del predicado que se le pase por valor. Para esto, previamente he definido como predicado dinámico “arista/2”. Finalmente, solo queda añadir las nuevas aristas, para lo cual uso un predicado recursivo auxiliar, al cual se le pasa la lista de aristas, tiene como caso base que reciba una lista vacía y finalmente su funcionamiento es el que sigue. Saca la cabeza de la lista de aristas, realiza un assert para añadirla a la base de hechos y se llama de forma recursiva con el resto de la lista.

2.2 – Predicado aranya/0:

Sin implementar

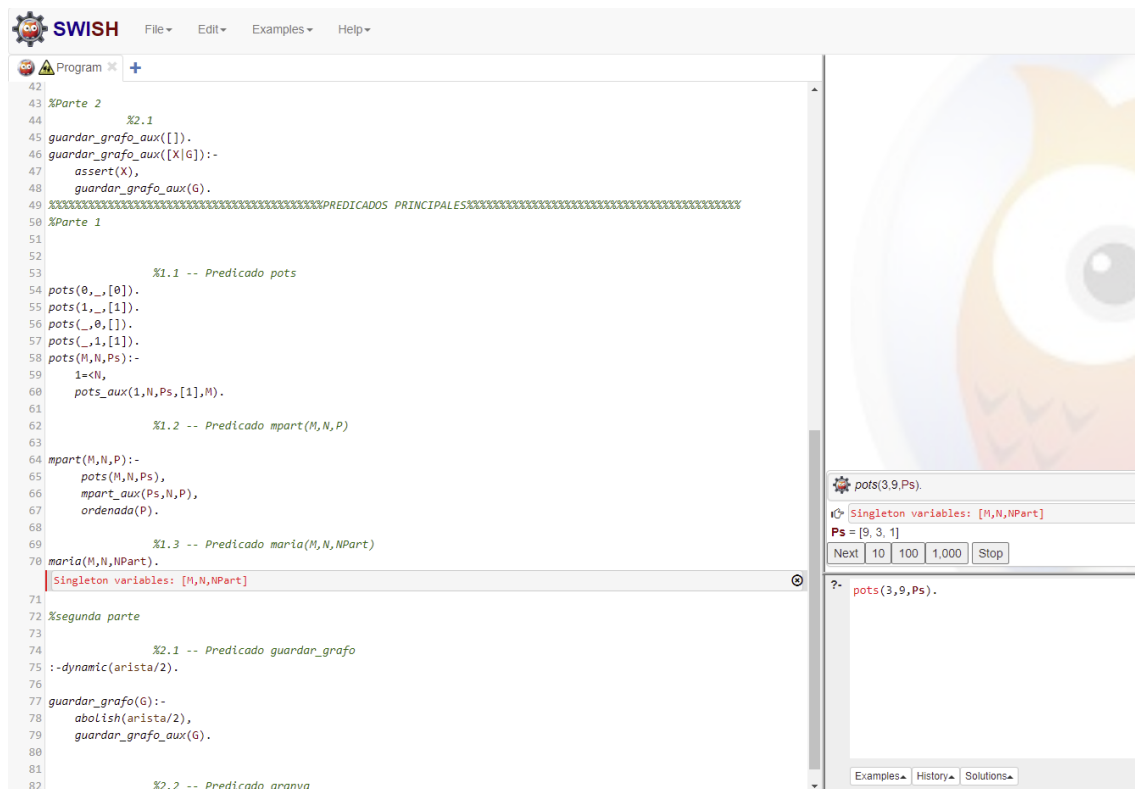
PRUEBAS:

Para realizar las pruebas he utilizado la página → <https://swish.swi-prolog.org/>

A su vez, para llamar a los métodos se necesitan los siguientes datos:

- 1- `pots(M,N, Ps)` → Siendo M el valor del cual queremos hallar los múltiplos, N el valor límite el cual no pueden superar esos múltiplos y finalmente Ps, la lista que se va a devolver, se hace la llamada con una lista vacía.

a. Prueba 1: `pots(3,9,Ps)`.



The screenshot shows the SWISH Prolog environment. The main window displays a Prolog program with the following code:

```
42
43 %Parte 2
44 %2.1
45 guardar_grafo_aux([]).
46 guardar_grafo_aux([X|G]):-
47     assert(X),
48     guardar_grafo_aux(G).
49 %XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
50 %Parte 1
51
52
53 %1.1 -- Predicado pots
54 pots(0,_,[0]).
55 pots(1,_,[1]).
56 pots(_,0,[1]).
57 pots(_,1,[1]).
58 pots(M,N,Ps):-
59     1<=N,
60     pots_aux(1,N,Ps,[1],M).
61
62 %1.2 -- Predicado mpart(M,N,P)
63
64 mpart(M,N,P):-
65     pots(M,N,Ps),
66     mpart_aux(Ps,N,P),
67     ordenada(P).
68
69 %1.3 -- Predicado maria(M,N,NPart)
70 maria(M,N,NPart).
71
72 %Singleton variables: [M,N,NPart]
73
74 %segunda parte
75
76 %2.1 -- Predicado guardar_grafo
77 :-dynamic(arista/2).
78
79 guardar_grafo(G):-
80     abolish(arista/2),
81     guardar_grafo_aux(G).
82
83 %2.2 -- Predicado aranya
```

The right-hand side of the interface shows the execution results for the query `pots(3,9,Ps)`. It indicates that the query was successful and provides the solution `Ps = [9, 3, 1]`. Below the solution, there are buttons for 'Next', '10', '100', '1,000', and 'Stop'. At the bottom of the right-hand side, there are tabs for 'Examples', 'History', and 'Solutions'.

b. Prueba 2: pots(3,47,Ps).

The image shows the SWISH Prolog IDE interface. The left pane contains the Prolog code, and the right pane shows the execution results for the query `pots(3,47,Ps)`.

```

5 %Parte 1
6
7 %1.1
8 %llenado de la lista
9 pots_aux(M,N,Ps,PsAux,I):-
10     M1 is M*I,
11     M1 <= N,
12     pots_aux(M1,N,Ps,[M1|PsAux],I).
13
14 %condicion de parada
15 pots_aux(M,N,Ps,PsAux,I):-
16     M1 is M*I,
17     M1 > N,
18     Ps = PsAux.
19
20 %1.2
21 mpart_aux(A,N,L):-
22     member(X,A),
23     R is (N-X),
24     R > 0,
25     L = [X].
26
27 mpart_aux(A,N,L):-
28     member(X,A),
29     R is (N-X),
30     R > 0,
31     mpart_aux(A,R,L1),
32     L = [X | L1].
33
34
35 ordenada([]).
36 ordenada([_]).
37 ordenada(Ps):-
38     Ps = [A,B | Z],
39     A >= B,
40     ordenada([B|Z]).
41
42
43 %Parte 2
44 %2.1
45 guardar_grafo_aux([]).
46 guardar_grafo_aux([X|G]):-

```

The right pane shows the execution results for the query `pots(3,47,Ps)`. It displays the singleton variables `[M,N,NPart]` and the value of `Ps` as `[9, 3, 1]`. Below this, there are buttons for `Next`, `10`, `100`, `1,000`, and `Stop`. The bottom of the right pane shows the query `?- pots(3,47,Ps).` and buttons for `Examples`, `History`, and `Solutions`.

c. Prueba 3: pots(2,256,Ps).

The image shows the SWISH Prolog IDE interface. The left pane contains the Prolog code, and the right pane shows the execution results for the query `pots(2,256,Ps)`.

```

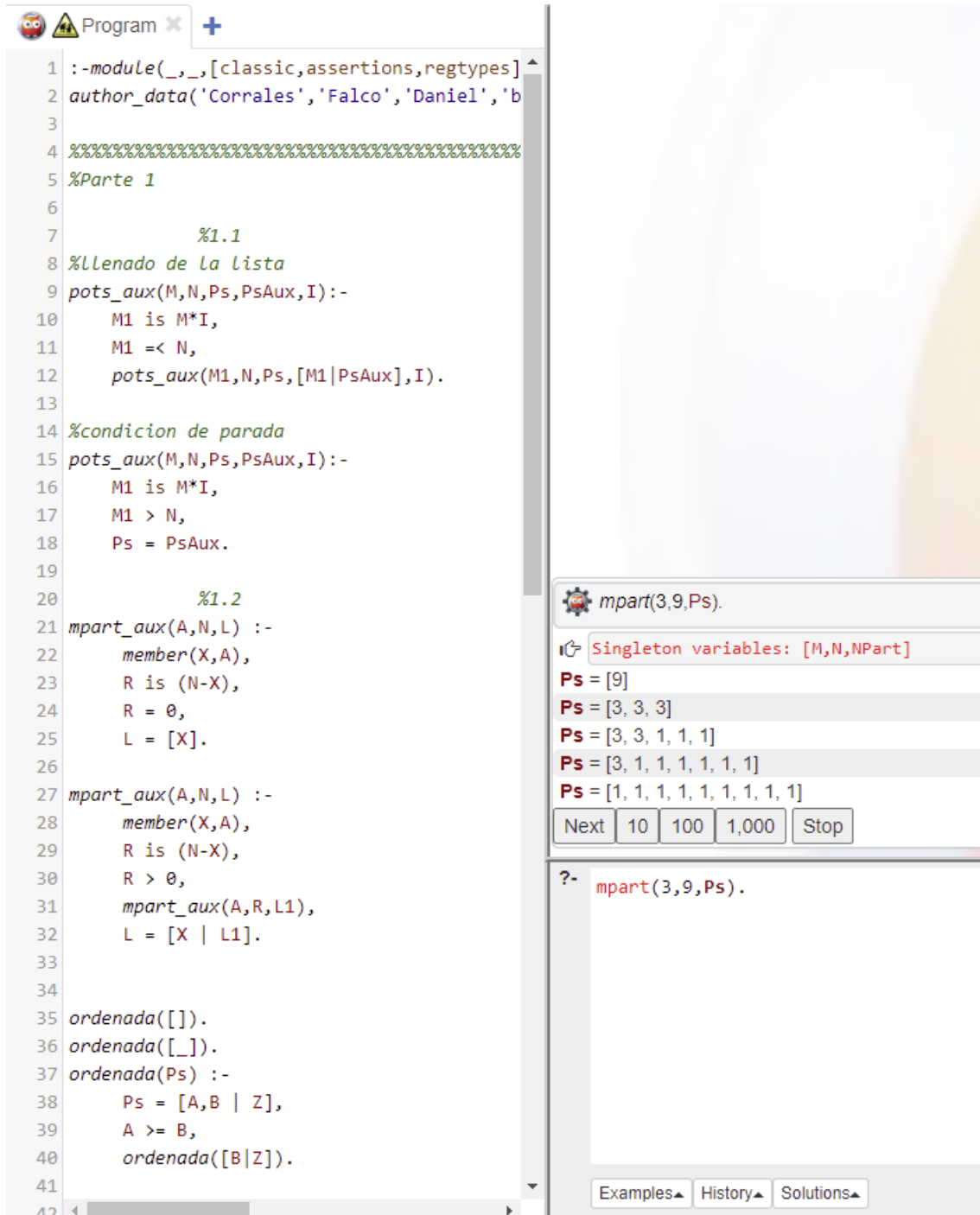
5 %Parte 1
6
7 %1.1
8 %llenado de la lista
9 pots_aux(M,N,Ps,PsAux,I):-
10     M1 is M*I,
11     M1 <= N,
12     pots_aux(M1,N,Ps,[M1|PsAux],I).
13
14 %condicion de parada
15 pots_aux(M,N,Ps,PsAux,I):-
16     M1 is M*I,
17     M1 > N,
18     Ps = PsAux.
19
20 %1.2
21 mpart_aux(A,N,L):-
22     member(X,A),
23     R is (N-X),
24     R > 0,
25     L = [X].
26
27 mpart_aux(A,N,L):-
28     member(X,A),
29     R is (N-X),
30     R > 0,
31     mpart_aux(A,R,L1),
32     L = [X | L1].
33
34
35 ordenada([]).
36 ordenada([_]).
37 ordenada(Ps):-
38     Ps = [A,B | Z],
39     A >= B,
40     ordenada([B|Z]).
41
42
43 %Parte 2
44 %2.1
45 guardar_grafo_aux([]).
46 guardar_grafo_aux([X|G]):-

```

The right pane shows the execution results for the query `pots(2,256,Ps)`. It displays the singleton variables `[M,N,NPart]` and the value of `Ps` as `[256, 128, 64, 32, 16, 8, 4, 2, 1]`. Below this, there are buttons for `Next`, `10`, `100`, `1,000`, and `Stop`. The bottom of the right pane shows the query `?- pots(2,256,Ps).` and buttons for `Examples`, `History`, and `Solutions`.

2- mpart(M,N,P)→ Con M el valor de la partición M-aria, N el valor que queremos particionar y P, la lista a llenar con la partición (lista vacía a la hora de realizar la llamada).

a. Prueba 1: mpart(3,9,Ps).



The image shows a Prolog IDE window with a code editor on the left and a console on the right. The code editor contains the following Prolog code:

```

1 :-module(_,_,[classic,assertions,regtypes])
2 author_data('Corrales','Falco','Daniel','b
3
4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5 %Parte 1
6
7 %1.1
8 %llenado de la lista
9 pots_aux(M,N,Ps,PsAux,I):-
10     M1 is M*I,
11     M1 <= N,
12     pots_aux(M1,N,Ps,[M1|PsAux],I).
13
14 %condicion de parada
15 pots_aux(M,N,Ps,PsAux,I):-
16     M1 is M*I,
17     M1 > N,
18     Ps = PsAux.
19
20 %1.2
21 mpart_aux(A,N,L) :-
22     member(X,A),
23     R is (N-X),
24     R = 0,
25     L = [X].
26
27 mpart_aux(A,N,L) :-
28     member(X,A),
29     R is (N-X),
30     R > 0,
31     mpart_aux(A,R,L1),
32     L = [X | L1].
33
34
35 ordenada([]).
36 ordenada([_]).
37 ordenada(Ps) :-
38     Ps = [A,B | Z],
39     A >= B,
40     ordenada([B|Z]).
41
42

```

The console on the right shows the execution of the query `mpart(3,9,Ps).`. It displays the singleton variables `[M,N,NPart]` and the following solutions for `Ps`:

```

Ps = [9]
Ps = [3, 3, 3]
Ps = [3, 3, 1, 1, 1]
Ps = [3, 1, 1, 1, 1, 1]
Ps = [1, 1, 1, 1, 1, 1, 1, 1, 1]

```

Below the solutions, there are buttons for `Next`, `10`, `100`, `1,000`, and `Stop`. At the bottom of the console, there are buttons for `Examples`, `History`, and `Solutions`.

b. Prueba 2: mpart(3,47,Ps).

The screenshot shows a Prolog IDE with a program on the left and its execution results on the right.

Program Code:

```

1 :-module(_,_,[classic,assertions,regtypes])
2 author_data('Corrales','Falco','Daniel','b')
3
4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5 %Parte 1
6
7 %1.1
8 %llenado de la lista
9 pots_aux(M,N,Ps,PsAux,I):-
10     M1 is M*I,
11     M1 <= N,
12     pots_aux(M1,N,Ps,[M1|PsAux],I).
13
14 %condicion de parada
15 pots_aux(M,N,Ps,PsAux,I):-
16     M1 is M*I,
17     M1 > N,
18     Ps = PsAux.
19
20 %1.2
21 mpart_aux(A,N,L):-
22     member(X,A),
23     R is (N-X),
24     R = 0,
25     L = [X].
26
27 mpart_aux(A,N,L):-
28     member(X,A),
29     R is (N-X),
30     R > 0,
31     mpart_aux(A,R,L1),
32     L = [X | L1].
33
34
35 ordenada([]).
36 ordenada([_]).
37 ordenada(Ps):-
38     Ps = [A,B | Z],
39     A >= B,
40     ordenada([B|Z]).
41
42

```

Execution Results:

The first execution is for the query `mpart(3,9,Ps).`. It shows the following results for `Ps`:

- `Ps = [9]`
- `Ps = [3, 3, 3]`
- `Ps = [3, 3, 1, 1, 1]`
- `Ps = [3, 1, 1, 1, 1, 1, 1]`
- `Ps = [1, 1, 1, 1, 1, 1, 1, 1, 1]`

The second execution is for the query `mpart(3,47,Ps).`. It shows the following results for `Ps`:

- `Ps = [27, 9, 9, 1, 1]`
- `Ps = [27, 9, 3, 3, 3, 1, 1]`
- `Ps = [27, 9, 3, 3, 1, 1, 1, 1, 1]`
- `Ps = [27, 9, 3, 1, 1, 1, 1, 1, 1, 1]`
- `Ps = [27, 9, 1, 1, 1, 1, 1, 1, 1, 1, 1]`
- `Ps = [27, 3, 3, 3, 3, 3, 3, 1, 1]`
- `Ps = [27, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1]`
- `Ps = [27, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1]`
- `Ps = [27, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]`
- `Ps = [27, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]`
- `Ps = [27, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]`
- `Ps = [9, 9, 9, 9, 9, 1, 1]`
- `Ps = [9, 9, 9, 9, 3, 3, 3, 1, 1]`

En este caso, hay tantas posibles combinaciones que solo incluyo unas pocas opciones, sin embargo se podía seguir.

c. Prueba 3: mpart(2,256,Ps).

The screenshot shows the SWISH Prolog IDE interface. On the left, the Prolog code is displayed in a text editor. The code defines a module, author data, and several predicates including `pots_aux`, `mpart_aux`, and `ordenada`. The main query being executed is `mpart(2,256,Ps).`. On the right, the execution results are shown, displaying a long list of solutions for `Ps`. The solutions are lists of integers, starting with `[27, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]` and ending with `[128, 64, 32, 16, 4, 2, 2, 2]`. The interface includes a menu bar (File, Edit, Examples, Help) and a status bar at the bottom with buttons for Examples, History, and Solutions.

```

1 :-module(_,_,[classic,assertions,regtypes])
2 author_data('Corrales','Falco','Daniel','b
3
4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5 %Parte 1
6
7         %1.1
8 %Llenado de La Lista
9 pots_aux(M,N,Ps,PsAux,I):-
10     M1 is M*I,
11     M1 =< N,
12     pots_aux(M1,N,Ps,[M1|PsAux],I).
13
14 %condicion de parada
15 pots_aux(M,N,Ps,PsAux,I):-
16     M1 is M*I,
17     M1 > N,
18     Ps = PsAux.
19
20         %1.2
21 mpart_aux(A,N,L) :-
22     member(X,A),
23     R is (N-X),
24     R = 0,
25     L = [X].
26
27 mpart_aux(A,N,L) :-
28     member(X,A),
29     R is (N-X),
30     R > 0,
31     mpart_aux(A,R,L1),
32     L = [X | L1].
33
34
35 ordenada([]).
36 ordenada([_]).
37 ordenada(Ps) :-
38     Ps = [A,B | Z],
39     A >= B,
40     ordenada([B|Z]).
41
42

```

Execution results for `mpart(2,256,Ps).`:

```

Ps = [27, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Ps = [27, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Ps = [9, 9, 9, 9, 9, 1, 1]
Ps = [9, 9, 9, 9, 3, 3, 3, 1, 1]

```

Next 10 100 1,000 Stop

Singleton variables: [M,N,NPart]

```

Ps = [256]
Ps = [128, 128]
Ps = [128, 64, 64]
Ps = [128, 64, 32, 32]
Ps = [128, 64, 32, 16, 16]
Ps = [128, 64, 32, 16, 8, 8]
Ps = [128, 64, 32, 16, 8, 4, 4]
Ps = [128, 64, 32, 16, 8, 4, 2, 2]
Ps = [128, 64, 32, 16, 8, 4, 2, 1, 1]
Ps = [128, 64, 32, 16, 8, 4, 1, 1, 1, 1]
Ps = [128, 64, 32, 16, 8, 2, 2, 2, 2]
Ps = [128, 64, 32, 16, 8, 2, 2, 2, 1, 1]
Ps = [128, 64, 32, 16, 8, 2, 2, 1, 1, 1, 1]
Ps = [128, 64, 32, 16, 8, 1, 1, 1, 1, 1, 1, 1]
Ps = [128, 64, 32, 16, 4, 4, 4, 4]
Ps = [128, 64, 32, 16, 4, 4, 4, 2, 2]
Ps = [128, 64, 32, 16, 4, 4, 4, 2, 1, 1]
Ps = [128, 64, 32, 16, 4, 4, 4, 1, 1, 1, 1]
Ps = [128, 64, 32, 16, 4, 4, 2, 2, 2, 2]

```

Next 10 100 1,000 Stop

?- mpart(2,256,Ps).

Examples History Solutions

Ocorre como en el caso anterior, demasiadas opciones, sin embargo se podría seguir ejecutando.

- 3- guardar_grafo(G)→Siendo G la lista de nuevas aristas (arista/2) a añadir a la base de datos.