UTF-8

Segunda practica (Programacion en ISO-Prolog)

Carlos Pozo Serrano, B190234

Table of Contents

code

Pruebas:

1. pots/3:

```
?- pots(0,100,Ps).

Ps = [1] ? .

no
?- pots(7,4321,Ps).

Ps = [2401,343,49,7,1] ? .

no
?- pots(2,10000,Ps).

Ps = [8192,4096,2048,1024,512,256,128,64,32,16,8,4,2,1] ? .

no
?-
```

2. mpart/3:

```
?- mpart(0,100,P).

P = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
     1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
     1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
     1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] ? .

no
?- mpart(2,200,P).

P = [128,64,8] ? .

P = [128,64,4,4] ? .

P = [128,32,32,8] ? .

P = [64,64,64,8] ? .

P = [128,64,4,2,2] ? .

P = [128,32,32,4,4] ?

yes
```

```
?- mpart(5,1000,P).

P = [625,125,125,125] ? .

P = [625,125,125,25,25,25,25,25] ? .

P = [125,125,125,125,125,125,125,125] ? .

P = [625,125,125,25,25,25,25,5,5,5,5,5] ? .

P = [625,125,25,25,25,25,25,25,25,25,25,25] ? .

P = [125,125,125,125,125,125,125,25,25,25,25,25] ? .

P = [625,125,125,25,25,25,25,5,5,5,5,1,1,1,1,1] ?

yes
?-
```

3. maria/3:

```
?- maria(0,100,NPart).

NPart = 1 ? .

no
?- maria(2,200,NPart).

NPart = 205658 ? .

no
?- maria(5,1000,NPart).

NPart = 14373 ? .

no
?-
```

4. guardar_grafo/1:

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

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

yes
?- guardar_grafo([arista(a,b),arista(a,c),
   arista(a,d),arista(a,e),arista(b,f),arista(f,g),
```

```
   arista(b,h),arista(a,h),arista(f,a),arista(j,b)]).

yes
?-
```

5. aranya/1:

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

yes
?- aranya.

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

yes
?- aranya.

no
?- guardar_grafo([arista(a,b),arista(a,c),
   arista(a,d),arista(a,e),arista(b,f),arista(f,g),
   arista(b,h),arista(a,h),arista(f,a),arista(j,b)]).

yes
?- aranya.

yes
?-
```

Usage and interface

- **Library usage:**
  `:- use_module(/mnt/c/Users/carlo/Desktop/Prolog/Pr2/code.pl).`
- **Exports:**
  - *Predicates:*
    `pots/3`, `aux_pots/4`, `sol/2`, `mpart/3`, `calculate/2`, `sum/3`, `num/3`, `maria/3`, `arista/2`, `guardar_grafo/1`, `ggAux/1`, `aranya/0`, `check_aranya/2`, `comprobar/3`, `todos_nodos/1`, `arst/1`, `aristo/2`, `anadir/3`, `borrar/3`.
  - *Multifiles:*
    `Σcall_in_module/2`.

Documentation on exports

pots/3: PREDICATE

Predicado 1.1: Este predicado resulve los casos base, `M` 0 o 1, en caso de no poder resolverlo de esta forma delega la resolucion en `aux_pots/4`, tras lo cual invierte la lista devuelta para que este en orden decreciente de sumandos.

```
pots(0,_1,[1]) :- !.
pots(1,_1,[1]) :- !.
pots(A,B,C) :-
    aux_pots(A,B,1,D),
    reverse([1|D],C).
```

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

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

aux_pots/4: PREDICATE

Este predicado resulve los casos base, el resultado de multiplicar `R` por `M` seria superior a `N`, o `R` y `N` son iguales,en otro caso multiplica `R` por `M` y llama recursivamente con el resultado obtenido.

```
aux_pots(M,N,R,[]) :-
    C is N//R,
    C<M,
    !.
aux_pots(_1,N,N,[N]) :- !.
aux_pots(M,N,R,[E|C]) :-
    E is R*M,
    aux_pots(M,N,E,C).
```

Usage: `aux_pots(M,N,R,Ps)`

Dados `M`, `N` y `R` enteros, devuelve en `Ps` una lista con las potencias de `M` que son menores o iguales que `N`, en orden ascendednte.

sol/2: PREDICATE

Este predicado sirve para almacenar las distintas soluciones del predicado `mpart/3` y guarda el numero de elementos junto con la lista resultado. The predicate is of type *dynamic*.

mpart/3: PREDICATE

Predicado 1.2: Este predicado borra todos las soluciones almacenadas con anterioridad, a continuacion llama a `pots/3` para obtener la lista de las potencias, luego llama a `calculate/2` para generar las soluciones y almacenarlas y por ultimo se usa `num/3` y `sol/2` para recoger todas las soluciones almacenadas. Comentario: Esta solucion no es la mas optima, ya que para obtener tan solo una solucion el predicado ha de obtener todas las soluciones antes para asi poder devolverlas en orden. A pesar de ello no se ha encontrado otra solucion que cumpla con las condiciones que se piden.

```
mpart(M,N,P) :-
    retractall(sol(_1,_2)),
    pots(M,N,C),
    calculate(N,C),
```

```
    num(1,N,R),
    sol(R,P).
```

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

Dados M y N enteros, devuelve en P por backtracking todas las particiones `M`-arias de `N`, representadas como listas de enteros. Las soluciones son devultas con las listas mas cortas primero.

calculate/2: PREDICATE

Este predicado recoge todas las formas de obtener N a partir de una combinacion de elementos de C a traves del predicado sum/3 y a continuacion lo almacena en memoria a traves de los predicados length y assert, siendo almacenados como proposiciones sol. Se itera a traves de backtracking mientras haya soluciones, una vez se acaban estas termina.

```
calculate(N,C) :-
    sum(N,C,P),
    length(P,Length),
    assert(sol(Length,P)),
    fail.
calculate(_1,_2).
```

Usage: `calculate(N,C)`

Dados `N` entero y `C` una lista de enteros almacena en memoria todas las formas posibles de obtener `N` a partir de los elementos de `C`.

sum/3: PREDICATE

Este predicado resuelve el caso base de que N sea 0, en cualquier otro caso comprueba que N sea mayor que X, en ese caso almacena X al inicio de la lista que se devuelve como solucion y se llama a sum de forma recursiva. Por ultimo, hay otra regla que permite vaciar la lista C y llamar de forma recursiva, esto se usa tanto para obtener soluciones con numeros mas pequenos por bactracking, como para completar soluciones que requieren de numeros mas pequenos para poder llegar a N.

```
sum(0,_1,[]) :- !.
sum(N,[X|C],[D|L]) :-
    N>=X,
    D=X,
    R is N-X,
    sum(R,[X|C],L).
sum(N,[_1|C],L) :-
    N>0,
    sum(N,C,L).
```

Usage: `sum(N,C,P)`

Dados `N` entero y `C` una lista de enteros, devuelve en `P` una lista de enteros cuya suma es `N` y esta compuesta por elementos contenidos en `C`.

num/3: PREDICATE

Este predicado predicado resuelve el caso base en el que R es A si A es menor o igual que N, si se piden mas soluciones se llama recursivamente con A = A+1.

```
num(A,N,A) :-
    A=<N.
num(A,N,R) :-
    A<N,
    A1 is A+1,
    num(A1,N,R).
```

Usage: `num(A,N,R)`

Dados `A` y `N` enteros, devuelve en `R` un numero comprendido entre `A` y `N`.

maria/3: PREDICATE

Predicado 1.3: Este predicado ejecuta el predicado setof para obtener todas las solucionse que puede dar mpart, a continuacion se ejecuta length para obtener la longitud de la lista obtenida. Comentario: esta solucion podria ser optimizada utilizando una modificacion de calculate, para asi evitar el uso de memoria que este predicado requiere.

```
maria(M,N,NPart) :-
    setof(X,mpart(M,N,X),L),
    length(L,NPart).
```

Usage: `maria(M,N,NPart)`

`NPart` es el numero de particiones `M`-arias de `N`.

arista/2: PREDICATE

Este predicado sirve para almacenar el grafo en memoria. The predicate is of type *dynamic*.

guardar_grafo/1: PREDICATE

Predicado 2.1: Este predicado borra los hechos que haya guardados de arista y a continuacion llama a `ggAux/1` para recorrer la lista de aristas y almacenarlas en la base de datos.

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

Usage: `guardar_grafo(G)`

Dado `G` un grafo representado como una lista de aristas, deja asertados en la base de datos como hechos del predicado `arista/2` los elementos de G. Al llamar a este predicado se borra cualquier hecho que hubiera guardado anteriormente de este predicado.

ggAux/1: PREDICATE

Este predicado comprueba que todos los elementos de una lista son aristas.

```
ggAux([X|G]) :-
    functor(X,F,_1),
    F=arista,
    assert(X),
    ggAux(G).
ggAux([]).
```

Usage: `ggAux(G)`

Dado `G` un grafo representado como una lista de aristas, deja asertados en la base de datos como hechos del predicado arista/2 los elementos de G.

aranya/0: PREDICATE

Este predicado llama a todos_nodos/1 para almacenar en una lista todos los nodos del grafo, y a continuacion llama a check_aranya/2 para resolver el problema.

```
aranya :-
    todos_nodos(S),
    check_aranya(S,S).
```

Usage:

Predicado 2.2: Dado un grafo `G` guardado en la base datos, comprueba que este conntiene una araa de expansion, y en caso contrario falle de forma finita

check_aranya/2: PREDICATE

Este predicado borrar para eliminar el nodo maestro de la lista de nodos a contener por la araa, ya esta contenido, y a continuacion se intenta hallar una araa de expansion con raiz ese nodo a traves de `comprobar/3`, si tiene exito se cortan el resto de ramas, sino se borra el primer elemento de la lista de candidatos a maestro y se vuelve a llamar a este predicado.

```
check_aranya([D|_1],F) :-
    borrar(F,D,F2),
    comprobar([D],D,F2),
    !.
check_aranya([_1|L],F) :-
    check_aranya(L,F).
```

Usage: `check_aranya(L,F)`

Dados dos conjuntos de nodos `L` y `F` trata de encontrar una araa de expansion con raiz un nodo de `L` que contenga a todos los nodos de `F`

comprobar/3: PREDICATE

Este predicado elige un elemento de la lista L (nodos aadidos) y uno de la lista F (nodos por visitar) y comprueba que haya una arista que los una, a continuacion borra el nodo escogido de la lista F, y si el nodo escogido de la lista F no es el nodo maestro, D, lo borra de la lista L, solo puede tener dos aristas cada nodo, finalmente aade el nodo sacado de la lista F a la lista L y se llama de forma recursiva. El predicado finaliza cuando la lista de nodos por visitar esta vacia.

```
comprobar(L,D,F) :-
    member(X,F),
    member(A,L),
    aristo(X,A),
    borrar(F,X,R),
    ( A=D ->
        L1=L
    ; borrar(L,A,L1)
    ),
    anadir(L1,X,L2),
    comprobar(L2,D,R).
comprobar(_1,_2,[]).
```

Usage: `comprobar(L,D,F)`

Dados dos conjuntos de nodos `L` y `F` trata de encontrar una araa de expansion con raiz un nodo de `D`

todos_nodos/1: PREDICATE

Este predicado llama a set of para hallar todas las soluciones de arst.

```
todos_nodos(S) :-
    setof(A,arst(A),S).
```

Usage: `todos_nodos(S)`

Este predicado retorna en `S` una lista que contiene todos los nodos contenidos en el grafo almacenado en memoria

arst/1: PREDICATE

Este predicado comprueba si este nodo es parte de una arista (a la izquierda o a la derecha) .

```
arst(A) :-
    ( arista(A,_1)
    ; arista(_2,A)
    ).
```

Usage: `arst(A)`

Tiene exito si `A` es un nodo del grafo almacenado en memoria

aristo/2: PREDICATE

Este predicado comprueba si estos nodos forman una arista (en cualquiera de los dos sentidos) .

```
aristo(A,B) :-
    ( arista(A,B)
    ; arista(B,A)
    ).
```

Usage: `aristo(A,B)`

Tiene exito si `A` y `B` forman una arista del grafo almacenado en memoria

anadir/3: PREDICATE

Este predicado resuelve los casos base de que X ya este contenido en la lista o que la lista desde la que aadir ya este vacia, en caso contrario se llama de forma recursiva .

```
anadir([X|L],X,[X|L]) :- !.
anadir([A|L],X,[A|R]) :-
    anadir(L,X,R).
anadir([],X,[X]).
```

Usage: `anadir(L,X,R)`

Este predicado aade `X` a la lista `L` y lo devuelve en `R`. Si ya pertenecia a `L`, no lo aade

borrar/3: PREDICATE

Este predicado resuelve el caso base de hallar X lista y devuelve la lista sin incluir el elemento, en caso contrario se llama de forma recursiva .

```
anadir([X|L],X,[X|L]) :- !.
anadir([A|L],X,[A|R]) :-
    anadir(L,X,R).
anadir([],X,[X]).
```

Usage: `borrar(L,X,R)`

Este predicado borra `X` de la lista `L` y lo devuelve en `R`. Se presupone que esta contenido en ella

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`.

– *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)