Inteligencia Artificial

Práctica 3

Pareja 02

Oscar Gomez Borzdynski, José Ignacio Gómez García

12-4-2018

Ejercicio 1:
En este ejercicio implementamos un predicado que comprueba si un elemento está en una lista o en alguna de las sublistas de la misma.
Para que esto suceda tienen que cumplirse las siguientes condiciones:
Sea L la lista, RL la lista sin el primer elemento, PL el primer elemento de la lista y X el elemento a buscar:
· X = PL y PL no es lista
· X pertenece a la sublista PL
· X pertenece a RL
Si se cumplen => X pertenece a L

```
% EJERCICIO 1 %

% Es necesario comprobar que X no sea una lista para que %
% no se devuelvan listas como posibles candidatos. Tambien %
% es necesario mirar si X pertenece al primer elemento de la %
% lista, que podria ser una sublista %

pertenece_m(X, [X|_]) :- X \= [_|_].
pertenece_m(X, [Y|_]) :- pertenece_m(X,Y).
pertenece_m(X, [_|Rs]) :- pertenece_m(X, Rs).

/** <examples>
?- pertenece_m(X, [2,[1,3],[1,[4,5]]])
*/
```

Ejercicio 2:
En este ejercicio implementamos un predicado que comprueba que la lista L2 es la lista L1 invertida. Para ello utilizamos el predicado concatena, que genera una sola lista a partir de dos listas concatenadas.
Además, necesitaremos un predicado auxiliar al que llamaremos invierte_aux, usará un acumulador para invertir una lista y así poder compararla.
Para que esto se cumpla:
Sea ACC el acumulador, L la lista a invertir y I la lista invertida.
· Si ACC vacío y L = I
· Si ACC no vacío concateno PL a I, situándolo en primera posición y compruebo si la lista sigue estando invertida.

```
% EJERCICIO 2 %

% CONCATENA: %
concatena([], L, L).
concatena([X|L1], L2, [X|L3]) :- concatena(L1, L2, L3).

% INVIERTE: %
invierte(L1, L2) :- invierte_aux(L1, L2, []).
invierte_aux([], L2, L3) :- L3 = L2.
invierte_aux([X|L1], L2, L3) :-
   concatena([X], L3, A),
   invierte_aux(L1, L2, A).

/** <examples>
?- invierte([1, 2], L).
?- invierte([], L).
?- invierte([1, 2, 5, 4], L).
*/
```

Ejercicio 3:
En este ejercicio implementamos un predicado que inserta un elemento en una lista de manera ordenada. Los elementos seguirán un formato de par P-N donde N es el número que comparar.
Para que se cumpla:
Sea L la lista donde insertar, P-N el elemento a insertar, R el resultado
· SI L está vacía y R es [P-N]
· Si el N del primer par de RL es mayor que el N a insertar, concatenar el elemento con L y comparar con R
· Si el N del primer par de RL es menor que el N a insertar, insertamos P-N en RL

```
% EJERCICIO 3 %

% INSERTAR: %
insert([X-P], [], [X-P]).
insert([X-P], [A-P1|Rs], R) :-
   P =< P1,
   concatena([X-P], [A-P1|Rs], C),
   C = R.
insert([X-P], [A-P1|Rs], [B-P2|Ls]) :-
   P > P1,
   P2 = P1,
   A = B,
   insert([X-P], Rs, Ls).

/** <examples>
?- insert([a-6],[], X).
?- insert([a-6],[p-0], X).
?- insert([a-6],[p-0, g-7], X).
?- insert([a-6],[p-0, g-7, t-2], X).
*/
```

Ejercicio 4.1

En este ejercicio implementamos un predicado que cuenta el número de veces Xn que aparece un elemento X en una lista L.

Para que esto se cumpla:

· Si L está vacía y Xn es 0

· Si el primer elemento de L es X, sumo 1 a Xn y sigo contando en RL

· Si el primer elemento de L no es X, sigo contando en RL

```
% EJERCICIO 4.1 %

elem_count(_,[],0). %caso base

elem_count(X,[X|Rs], M) :-
   elem_count(X, Rs, N),
   M is N+1.

elem_count(X, [Y|Rs], M) :-
   X \= Y,
   elem_count(X, Rs, M).

/** <examples>
?- elem_count(1, [2,3], 1).
?- elem_count(1, [], 1).
?- elem_count(1, [1,2,3], 1).
?- elem_count(1, [1,2,3], 2).
?- elem_count(1, [1,2,3,1], X).
*/
```

Ejercicio 4.2:
En este ejercicio implementamos un predicado que devuelve una lista R donde aparecen el número de veces que aparecen los elementos de una lista L1 en una lista L2
Para que se cumpla:
· Si L1 está vacía y R es vacío
· Si L1 no vacía, cuento el número de veces que aparece PL1 en L2 y compruebo que es el resultado que aparece en R

```
% EJERCICIO 4.2 %

list_count([], A,  []) :- is_list(A).

list_count([X|Rs], L2, [X-C|Ls]) :-
   elem_count(X,L2,C),
   list_count(Rs,L2,Ls).

/** <examples>
?- list_count([b],[b,a,b,a,b],Xn).
?- list_count([b,a],[b,a,b,a,b],Xn).
?- list_count([b,a,c],[b,a,b,a,b],Xn).
?- list_count([],[b,a,b,a,b],Xn).
?- list_count([a],[],Xn).
*/
```

Ejercicio 5:
En este ejercicio implementamos un predicado que ordena una lista L1 a L2.
Para que esto suceda:
· Una lista vacía ordena a una lista vacía.
· Si el resultado de insertar el primer elemento de L1 en la lista ordenada de RL1 y coincide con L2

```
% EJERCICIO 5 %

sort_list([],[]).
sort_list([A|Rs], B) :-
   sort_list(Rs, L),
   insert([A], L, B).

/** <examples>
?- sort_list([p-0, a-6, g-7, t-2], X).
?- sort_list([p-0, a-6, g-7,p-9, t-2], X).
?- sort_list([p-0, a-6, g-7,p-9, t-2, 9-99], X).
*/
```

Ejercicio 6:
En este ejercicio creamos un predicado que crea un árbol T en función de una lista L de pares ordenados A-P.
Para que esto suceda:
· Si L tiene un solo elemento, T es tree(A, nil, nil)
. Si L tiene mas de un elemento: T será tree (1,(árbol de A), (árbol de RL))

% EJERCICIO 6 %

```
build_tree([A-_], tree(A,nil,nil)).
build_tree([A,B|Rs], X) :-
   build_tree([B|Rs], X1),
   build_tree([A], A1),
   X = tree(1, A1, X1).


/** <examples>
?- build_tree([p-0, a-6, g-7, p-9, t-2, 9-99], X).
?- build_tree([p-55, a-6, g-7, p-9, t-2, 9-99], X).
?- build_tree([p-55, a-6, g-2, p-1], X).
?- build_tree([a-11, b-6, c-2, d-1], X).
*/
```

Ejercicio 7.1:
En este ejercicio hemos implementado un predicado que codifique un elemento X según la estructura de árbol T.
Para que esto suceda:
· Si X es nodo raíz en T: []
· Si X es nodo izquierdo en T: [0]
· Else: [1]
Al concatenar todo conseguimos la secuencia deseada.

% EJERCICIO 7.1 %

```
encode_elem(A, [], tree(A,nil,nil)).
encode_elem(A, [0], tree(1,tree(A,nil,nil),_)).
encode_elem(A, X, tree(1, tree(B, _, _), G)) :-
   A \= B,
   encode_elem(A, X1, G),
   concatena([1], X1, X).

/** <examples>
?- encode_elem(a, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
?- encode_elem(b, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
?- encode_elem(c, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
?- encode_elem(d, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
*/
```

Ejercicio 7.2:
En este ejercicio implementamos un predicado que codifica una lista L de elementos con el mismo mecanismo que el ejercicio anterior respecto a un árbol T, generando R.
Para que esto suceda:
· Si L vacío y R vacío
· Si L no vacío codificar RL y el elemento PL, concatenarlos.

% EJERCICIO 7.2 %

```
encode_list([],[],_).
encode_list([A|Rs], B, T) :-
   encode_list(Rs, C, T),
   encode_elem(A, X, T),
   B = [X|C].

/** <examples>
?- encode_list([a], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
?- encode_list([a,a], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
?- encode_list([a,d,a], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
?- encode_list([a,d,a, q], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
*/
```

Ejercicio 8:
En este ejercicio hemos implementado un predicado que codifica un conjunto de letras en la lista L1 en L2.
Para ello:
· Comprobamos que todas las letras están en el diccionario.
· Contamos los elementos, los ordenamos de menor a mayor y lo invertimos. De esta manera conseguimos los elementos ordenados de mayor frecuencia a menor frecuencia.
· Construimos el árbol y codificamos la lista de letras con respecto al árbol creado.

```
% EJERCICIO 8 %

dictionary([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).

encode(A, B) :-
   dictionary(D),
   list_count(D, A, C),
   sort_list(C, S),
   invierte(S, I),
   build_tree(I, T),
   encode_list(A, B, T).

/** <examples>
?- encode([i,a],X).
?- encode([i,n,t,e,l,i,g,e,n,c,i,a,a,r,t,i,f,i,c,i,a,l],X).
?- encode([i,2,a],X).
*/
```