Guía de Trabajos Prácticos XI – Programación Lógica III

1) Dadas dos palabras, representadas como listas de caracteres, evaluar la semejanza entre las mismas. Para esto se verificarán, por posición, las letras de las dos palabras, cada coincidencia sumará un punto y cada vez que las letras no coincidan se restará un punto.

ejemplo:
semejanza([h,o,l,a], [h,o,l,o], S). --> S = 2.
semejanza([m,e,s,a], [m,e,s,a,d,a], S). --> S = 2.
semejanza([s,o,l,a], [m,o,n,a], S). --> S = 0.
semejanza([s,o,l], [c,a,s,a], S). --> S = -4.

Dado un diccionario en forma de lista de palabras, se quiere buscar una palabra cualquiera, para lo cual se deberá chequear que la misma pertenezca al diccionario. Si existe se debe devolver una lista de un elemento con la palabra buscada, si no existe, se debe devolver una lista con las alternativas a la palabra buscada y su valoración de semejanza.

Las alternativas devueltas en caso de no existir la palabra en el diccionario, serán tomadas del mismo en virtud de la semejanza con la palabra buscada. Se tomarán como semejantes aquellas palabras cuyo valor de semejanza sea mayor a cero.

Para el desarrollo de ésta segunda parte del programa, se puede usar el predicado predefinido "atom_chars(C, L)" que toma un átomo (C) y lo transforma en una lista de literales (L): atom_chars(hola, L). --> L = [h, o, l, a].

La definición del diccionario en principio es:
dic([sanar, hola, sabana, sabalo, prueba, computadora, cartera, mate, termo, mesa, silla, sarna]).

buscar(hola, L). --> L = [hola]
buscar(holo, L). --> L = [[hola, 2]]
buscar(saban, L). --> L = [[sanar, 1], [sabana, 4], [sabalo, 2]]

```
dic([sanar, hola, sabana, sabalo, prueba, computadora, cartera, mate, termo, mesa, silla, sarna]).

compara([], [], 0):- !.
compara([], L, N):- !, length(L, N1), N is -N1.
compara(L, [], N):- !, length(L, N1), N is -N1.
compara([X|L], [X|L1], N):- !, compara(L, L1, N1), N is N1 + 1.
compara([_|L], [_|L1], N):- !, compara(L, L1, N1), N is N1 - 1.

buscar(X, [X]):- dic(D), member(X, D), !.
buscar(X, L):- dic(D), buscar(X, D, L).
```

```
buscar(_, [], []):- !.

buscar(X, D, [[P, N]|L]):-
                        D = [P|R],
                        atom_chars(X, Xc),
                        atom_chars(P, Pc),
                        compara(Xc, Pc, N),
                        N > 0, !,
                        buscar(X, R, L).

buscar(X, D, L):-       D = [P|R],
                        atom_chars(X, Xc),
                        atom_chars(P, Pc),
                        compara(Xc, Pc, N),
                        N =< 0,
                        buscar(X, R, L).
```

2) Escriba un predicado en prolog de aridad 6 que implemente la función reemplazar, la cual recibirá los siguientes parámetros:

1- una lista de elementos en donde se reemplazará
2- el elemento a reemplazar en la primer lista
3- el elemento de reemplazo
4- el cuarto parámetro será a partir de que instancia encontrada comienza a reemplazar, debe validarse que su valor sea >= 1
5- el quinto parámetro será cuantos reemplazos como máximo se harán, siendo -1 (cero) el valor para indicar que reemplace todas las instancias que encuentre. Debe validarse que su valor sea >= -1
6- El último parámetro debe unificar con la lista resultante

Nota: Se debe implementar la funcionalidad, no se puede usar un predicado predefinido que la implemente.
Ej:
reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 0, -1, L).
El valor de inicio debe ser mayor o igual a 1
Fail.

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 1, -2, L).
La cantidad de reemplazos debe ser mayor o igual a -1
Fail.

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 1, -1, L).
L = [a, a, 2, 2, 3, 3, 2, 2, a, a]

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 1, 1, L).
L = [a, 1, 2, 2, 3, 3, 2, 2, 1, 1]

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 2, 2, L).
L = [1, a, 2, 2, 3, 3, 2, 2, a, 1]

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 2, -1, L).
L = [1, a, 2, 2, 3, 3, 2, 2, a, a]

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 1, a, 5, -1, L).
L = [1, 1, 2, 2, 3, 3, 2, 2, 1, 1]

reemplazar([1, 1, 2, 2, 3, 3, 2, 2, 1, 1], 4, a, 1, -1, L).
L = [1, 1, 2, 2, 3, 3, 2, 2, 1, 1]

```
reemplazar(_, _, _, X, _, _):-
        X < 1, !,
        write('El valor de inicio debe ser mayor a 0'), fail.
reemplazar(_, _, _, _, X, _):-
        X < -1, !,
        write('La cantidad de reemplazos debe ser mayor o igual a -1'), fail.
reemplazar(L, Xo, Xd, Desde, Cant, Resu) :-
        reem(L, Xo, Xd, Desde, Cant, Resu).
```

```
reem([], _, _, _, _, []) :- !.
reem(L, _, _, _, 0, L) :- !.

reem([X|L], Xo, Xd, Desde, Cant, [X|Resu]) :-
                X \= Xo, !,
                reem(L, Xo, Xd, Desde, Cant, Resu).

reem([Xo|L], Xo, Xd, Desde, Cant, [Xo|Resu]) :-
                Desde > 1,
                Desde1 is Desde - 1, !,
                reem(L, Xo, Xd, Desde1, Cant, Resu).

reem([Xo|L], Xo, Xd, Desde, Cant, [Xd|Resu]) :-
                Desde = 1,
                (Cant > 0 -> Cant1 is Cant -1; Cant1 = Cant),
                reem(L, Xo, Xd, Desde, Cant1, Resu).
```