**MUTA**

Per realitzar aquest exercici el que hem plantejat és anar provant les paraules de la “base de dades”, però només s’acceptaran com a candidates les que tinguin una lletra de diferència amb la paraula original. Un cop acceptada aquesta paraula com a candidata es provarà de trobar un camí de paraules entre la nova paraula acceptada i la paraula final.

Per tal de que no vagi provant paraules que s’allunyen de la paraula final i reduir més la cerca, primer intentarem les paraules que s’apropen més a la paraula final, és a dir, que tenen més paraules semblants amb la final, si té més lletres diferents que la paraula anterior ja no es deixa continuar. Si no es troba cap paraula que s’apropi a la paraula final, se’n busca una que al menys tingui el mateix nombre de lletres diferents.

Per mirar la diferència entre dos paraules utilitzem el predicat string\_chars(), on el segon paràmetre és una llista amb els caràcters de la paraula, i després utilitzem el predicat diferencia() per mirar quantes lletres diferents hi ha entre les dues paraules.

% diferencia(X, Y, N). Es compleix quan N és el nombre d’elements diferents entre X i Y. Pressuposa que les dues llistes tenen la mateixa mida.

diferencia([], [], 0).

diferencia([X|XS], [Y|YS], N):- X == Y, diferencia(XS, YS, N), !.

diferencia([X|XS], [Y|YS], N):- X \== Y, diferencia(XS, YS, Np), N is Np+1, !.

Per fer el predicat muta() el primer que fem és mirar la diferència entre les dues paraules i cridar a i\_muta() amb les dues paraules passades, una llista buida i la diferència que hi ha entre les dues paraules.

muta(PI, PF):- string\_chars(PI, CPI), string\_chars(PF, CPF), diferencia(CPI, CPF, N), i\_muta(PI, PF, [], N).

% i\_muta(PI, PF, LP, LastDif). Es compleix quan PI és la paraula inicial de la mutació, PF la paraula final de la

% mutació, LP és la llista de paraules per les que ja s’ha passat i LastDif és la última diferència entre PI i PF

i\_muta(PI, PF, LP, \_):- string\_chars(PI, CPI),

                      string\_chars(PF, CPF),

                      diferencia(CPI, CPF, N),

                      N =:= 0, % Si la diferència és 0 significa que s’ha arribat a la paraula final

                      reverse([PF|LP], LF), length(LF, Llarg),

                      write("Canvis: "), writeln(LF), write("Llargada: "), writeln(Llarg).

% Es pot considerar com un “if” amb la condició de que la nova paraula tingui diferència 1 i la diferència amb la paraula

% final sigui més petita (ens apropem a ella)

i\_muta(PI, PF, LP, LastDif):- string\_chars(PI, CPI),

                              w4(X),

                              not(member(X, LP)), % Assegurar que la nova paraula triada no forma part de la llista de

                              string\_chars(X, CX), % paraules per la que s’ha passat, sinó segurament s’entraria en un

                              diferencia(CPI, CX, N), % bucle entre dues paraules

                              N =:= 1,

                              string\_chars(PF, CPF),

                              diferencia(CX, CPF, Np),

                              Np < LastDif, % Assegurar que la diferencia entre la nova paraula i la final és mes petita

                              append([PI], LP, L), % Es dona la paraula com a bona i s’afegeix a la llista

                              i\_muta(X, PF, L, Np). % Finalment es torna a cridar al predicat amb la paraula acceptada

% com a la nova paraula inicial, la mateixa paraula final, la nova

% llista i la nova diferència entre paraules

% Es pot considerar com un “if” amb la condició de que la nova paraula tingui diferència 1 i la diferència amb la paraula

% final sigui igual a l’anterior paraula, el predicat és idèntic a l’anterior excepte LastDif =:= Np

i\_muta(PI, PF, LP, LastDif):- string\_chars(PI, CPI),

                              w4(X),

                              not(member(X, LP)),

                              string\_chars(X, CX),

                              diferencia(CPI, CX, N),

                              N =:= 1,

                              string\_chars(PF, CPF),

                              diferencia(CX, CPF, Np),

                              LastDif =:= Np,

                              append([PI], LP, L),

                              i\_muta(X, PF, L, Np).

La solució que dóna aquest predicat no és la millor, és podria considerar un algorisme *greedy*, ja que sempre intenta apropar-se el màxim possible cap a la paraula final.

Per fer un algorisme que trobi la millor solució només se’ns ha ocorregut aprofitar el predicat findall() per trobar totes les solucions i després mirar quina és la millor comptant el nombre de passes que es fa. El que si hem fet és posar la restricció de que un cop es trobi un camí, cap altre camí que es trobi podrà ser major que aquest primer camí trobat entre les dues paraules, per exemple si es troba un camí entre les dues paraules de 6, qualsevol que sigui més gran de 6 serà automàticament descartat. Sigui dit que amb la solució del findall() no hem comprovat que funcioni ja que amb les paraules que hem provat no ens ha acabat mai, però creiem que teòricament hauria de funcionar.

Aquest predicat l’hem anomenat fmuta() i utilitza el trobaUn() que bàsicament és el mateix que i\_muta(), excepte que en aquest cas es “retorna” la llista trobada i s’utilitza per determinar la màxima llargada que podran tenir la resta de llistes. Després s’utilitza el trobaMesCurt() juntament amb el findall() per trobar totes les més curtes, finalment el mesCurta() per trobar la més curta de totes les retornades per el findall().

% mesCurta(X, LlCurta). Es compleix quan X és una llista de llistes i LlCurta és la llista més curta de X

mesCurta([X], X).

mesCurta([X,Y|XS], LlCurta):- length(X, LX), length(Y, LY), LX =< LY, mesCurta([X|XS], LlCurta).

mesCurta([X,Y|XS], LlCurta):- length(X, LX), length(Y, LY), LY < LX, mesCurta([Y|XS], LlCurta).

fmuta(PI, PF):- string\_chars(PI, CPI),

                string\_chars(PF, CPF),

                diferencia(CPI, CPF, Dif),

                trobaUn(PI, PF, [], Dif, LFTmp), % Es troba la primera solució

                length(LFTmp, MaxLlarg), % I es fa servir la llargada per la resta de possibles llistes

                findall(LF, trobaMesCurt(PI, PF, [], Dif, MaxLlarg-1, LF), Llistes),

                mesCurta([LFTmp|Llistes], MC), write(MC).

% trobaUn(PI, PF, LP, LastDif, LF). Es compleix quan PI és la paraula inicial de la mutació, PF la paraula final de la

% mutació, LP és la llista de paraules per les que ja s’ha passat, LastDif és la última diferència entre PI i PF i LF és

% la llista final trobada.

trobaUn(PI, PF, LP, \_, LF):- string\_chars(PI, CPI),

                             string\_chars(PF, CPF),

                             diferencia(CPI, CPF, N),

                             N =:= 0,

                             reverse([PF|LP], LF), !.

trobaUn(PI, PF, LP, LastDif, LF):- string\_chars(PI, CPI),

                            w4(X),

                            not(member(X, LP)),

                            string\_chars(X, CX),

                            diferencia(CPI, CX, N),

                            N =:= 1,

                            string\_chars(PF, CPF),

                            diferencia(CX, CPF, Np),

                            Np < LastDif, % Assegures que la diferencia entre la nova paraula i la final sigui mes petita

                            append([PI], LP, L),

                            trobaUn(X, PF, L, Np, LF).

trobaUn(PI, PF, LP, LastDif, LF):- string\_chars(PI, CPI),

                            w4(X),

                            not(member(X, LP)),

                            string\_chars(X, CX),

                            diferencia(CPI, CX, N),

                            N =:= 1,

                            string\_chars(PF, CPF),

                            diferencia(CX, CPF, Np),

                            LastDif =:= Np,

                            append([PI], LP, L),

                            trobaUn(X, PF, L, Np, LF).

% trobaMesCurt(PI, PF, LP, LastDif, MaxLlarg LF). Es compleix quan PI és la paraula inicial de la mutació, PF la paraula

% final de la mutació, LP és la llista de paraules per les que ja s’ha passat, LastDif és la última diferència entre PI i

% PF, MaxLlarg és la màxima llargada que pot tenir la llista i LF és la llista final trobada.

trobaMesCurt(PI, PF, LP, \_, MaxLlarg, LF):- string\_chars(PI, CPI),

                            string\_chars(PF, CPF),

                            diferencia(CPI, CPF, N),

                            N =:= 0,

                            length(LP, Len),

                            Len < MaxLlarg, % Assegurar que la nova llista és més petita que el màxim

                            reverse([PF|LP], LF).

trobaMesCurt(PI, PF, LP, LastDif, MaxLlarg, LF):- string\_chars(PI, CPI),

                            w4(X),

                            not(member(X, LP)),

                            string\_chars(X, CX),

                            diferencia(CPI, CX, N),

                            N =:= 1,

                            string\_chars(PF, CPF),

                            diferencia(CX, CPF, Np),

                            Np < LastDif, % Assegures que la diferencia entre la nova paraula i la final sigui mes petita

                            length(LP, Len),

                            Len < MaxLlarg, % Assegurar que la nova llista és més petita que el màxim

                            append([PI], LP, L),

                            trobaMesCurt(X, PF, L, Np, MaxLlarg, LF).

trobaMesCurt(PI, PF, LP, LastDif, MaxLlarg, LF):- string\_chars(PI, CPI),

                            w4(X),

                            not(member(X, LP)),

                            string\_chars(X, CX),

                            diferencia(CPI, CX, N),

                            N =:= 1,

                            string\_chars(PF, CPF),

                            diferencia(CX, CPF, Np),

                            LastDif =:= Np,

                            length(LP, Len),

                            Len < MaxLlarg, % Assegurar que la nova llista és més petita que el màxim

                            append([PI], LP, L),

                            trobaMesCurt(X, PF, L, Np, MaxLlarg, LF).

La idea del trobaMesCurt() era que si trobava una llista més curta, actualitzés el MaxLlarg per així anar reduint el màxim de la llista i reduir el nombre d’opcions però no hem aconseguit fer aquesta actualització perquè això és molt idea de variable i no unificava.

**BATUTS**

La base de dades sobre la que estem treballant és la següent:

establiment( % -> Promig 2.66666

    best\_batuts, [alan,john,mary],

    [

        batut(berry,    [orange, blueberry, strawberry], 2),

        batut(tropical, [orange, banana, mango, guava], 3),

        batut(blue,     [banana, blueberry], 3)

    ]

).

establiment( % -> Promig 2.5

    all\_batuts, [keith,mary],

    [

        batut(pinacolada,   [orange, pineapple, coconut], 2),

        batut(green,        [orange, banana, kiwi], 5),

        batut(purple,       [orange, blueberry, strawberry], 2),

        batut(smooth,       [orange, banana, mango], 1)

    ]

).

establiment( % -> Promig 2.2

    batuts\_galore, [heath,john,michelle],

    [

        batut(combo1, [strawberry, orange, banana], 2),

        batut(combo2, [banana, orange], 5),

        batut(combo3, [orange, peach, banana], 2),

        batut(combo4, [guava, mango, papaya, orange], 1),

        batut(combo5, [grapefruit, banana, pear], 1)

    ]

).

establiment( % -> Promig 3.0

    roses\_batuts, [marc,roger,carmen],

    [

        batut(fresc1, [peach, lemon, milk], 2.5),

        batut(fresc2, [cherry, orange], 3),

        batut(fresc3, [apple, strawberry, orange, milk], 4),

        batut(fresc4, [chocolate, banana, milk], 2),

        batut(fresc5, [watermelon, pear, yogurt, peach], 3.5)

    ]

).

establiment( % -> Promig 1.94

    batuts\_barats, [pep,laia,rosana],

    [

        batut(fresc1, [peach, lemon, milk], 2),

        batut(fresc2, [cherry], 1),

        batut(fresc3, [strawberry, orange, milk], 1.2),

        batut(fresc4, [chocolate, milk, strawberry], 1.5),

        batut(fresc5, [watermelon, pear, yogurt, peach], 4)

    ]

).

establiment( % -> Promig 3.54

    weird\_batuts, [josep,alex,sergi,cristina],

    [

        batut(weird1, [peach, lemon, milk, cherry], 3),

        batut(weird2, [cherry, watermelon, chocolate, yogurt], 2.7),

        batut(weird3, [strawberry, orange, milk, peach], 2.5),

        batut(weird4, [chocolate, milk, strawberry, apple], 3.6),

        batut(weird5, [watermelon, pear, yogurt, peach, lemon, milk], 6.2),

        batut(weird6, [orange, pear, peach, lemon, yogurt], 3.6),

        batut(weird7, [cherry, orange, milk, strawberry, lemon], 3.2)

    ]

).

% mesDe(+N, E). Es satisfà si l’establiment E te més de N batuts

mesDe(N,E):- establiment(E,\_,L), length(L,Len), Len > N.

% elFa(B, E). Es satisfà si l’establiment E fa el batut B

elFa(B,E):- establiment(E,\_,L), member(batut(B,\_,\_),L).

% ratio(E, R). Es satisfà si l’establiment E te un ratio d’empleats per batuts d’R

ratio(E,R):- establiment(E, Empleats, Batuts), length(Empleats, LEmp), length(Batuts, LBat), R is LEmp/LBat.

% suma(LL, M). Es satisfà quan M és la suma dels preus de la llista de batuts LL.

suma([],0).

suma([batut(\_,\_,P)|LS], M):- suma(LS, Z), M is Z + P.

% promig(E, P). Es satisfà si el promig del preu dels batuts a l’establiment E és P

promig(E,P):- establiment(E,\_,L), length(L,Len), suma(L,M), Z is M/Len, Z = P.

% mesBarat(E). Es satisfà si l’establiment E te els batuts més barats en promig

mesBarat(E):- establiment(E, \_, \_), promig(E, Pe), establiment(X, \_, \_), X \= E, promig(X, Px), Pe > Px, !, fail. % A la mínima que n'hi hagi un de mes barat fallar

mesBarat(E):- establiment(E, \_, \_). % Si provant tots els altres establiment no s'ha complert l'anterior, aquesta retornarà true, NO ES POT CANVIAR D'ORDRE

Per fer el trobaBatuts() hem utilitzat els predicats auxiliars descrits a continuació:

% notInList(LX, D). Es satisfà quan algun dels elements de la llista LX està a la llista D

notInList([],\_).

notInList([L|LX],D):- notInList(LX,D), not(member(L,D)).

% inList(LX, D). Es satisfà quan tots els elemnts de la llista LX estan a la llista D

inList([],\_).

inList([L|LX],D):- inList(LX,D), member(L,D).

% fillList(E, Lb, D, I, L). Es satisfà quan L és una llista formada per parelles (E, NomBatut) que contenen

% els ingredients de la llista D i cap de la llista I. Lb és la llista de batuts de l'establiment E.

% Fa un recorregut de tots els batuts de l'establiment i comprovant si el batut compleix els requisits D i I.

fillList(\_,[],\_,\_,[]).

fillList(E,[batut(M,List,\_)|Lb],D,I,[E,M|L]):- fillList(E,Lb,D,I,L), notInList(List,I), inList(D,List), !.

fillList(E,[batut(\_,\_,\_)|Lb],D,I,L):- fillList(E,Lb,D,I,L).

% recurse(Establiments,L,D,I). Es satisfà quan L és una llista formada per llistes de parells (Establiment, NomBatut)

% de la llista d'establiments Establiments que contenen els ingredients que es demanan a la llista D

% i cap dels que es demanen a la llista I

recurse([],[],\_,\_).

recurse([X|Establiments],[Z|L],D,I):- recurse(Establiments,L,D,I), establiment(X,\_,Lb), fillList(X,Lb,D,I,Z).

Finalment per al trobaBatuts() el que fem és construir una llista amb tots els Establiments i després recórrer aquesta llista per anar comprovant quins dels batuts de cada establiment compleixen les condicions

% trobaBatuts(L,D,I). Es satisfà si L és una llista formada pels parells (Establiment,NomBatut)

% de tots els establiments-batuts que contenen els ingredients que es demanen a la llista D

% i cap dels que es diuen a la llista I.

trobaBatuts(L,D,I):- findall(Establiment, establiment(Establiment,\_,\_), List),

                     recurse(List,Z,D,I),

                     flatten(Z,L). % Converteix llista de llistes en una unica llista

**4LINIA**