ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

Μάθημα : Λογικός Προγραμματισμός

Τελική Εργασία

Μαλούκου Αγορή

ΑΜ : ΜΠΠΛ 20046

Πειραιάς

Φεβρουάριος 2021

Περιεχόμενα

ΕΙΣΑΓΩΓΗ

1. Βασικές έννοιες στην Prolog

Στην εργασία αυτή υλοποιούμε ασκήσεις Λογικού Προγραμματισμού χρησιμοποιώντας βασικές έννοιες της Prolog.

Πρόγραμμα στην Prolog είναι ένα σύνολο προτάσεων που περιγράφουν τα δεδομένα του προβλήματος και τις σχέσεις μεταξύ τους.

Τα παρακάτω παραδείγματα αποτελούν ένα πρόγραμμα Prolog:

woman(roula).

man(giannis).

Με αυτές τις εντολές δηλώνουμε ότι η Roula είναι γυναίκα κι ότι ο Giannis είναι άντρας. Κάθε μία από αυτές τις εντολές λέγεται ***γεγονός*** (fact) και αποτελείται από ένα ***κατηγόρημα*** (predicate) που στο συγκεκριμένο παράδειγμα είναι τα woman και man και ένα ***όρισμα*** (argument) που είναι τα roula και giannis.

Η Prolog έχει δύο επίπεδα λογικής: ***TRUE*** και ***FALSE*** και αντίστοιχα επιστρέφει 'Yes' αν η πρόταση είναι αληθής και 'No' αν η πρόταση είναι ψευδής.

Στην Prolog επίσης χρησιμοποιούνται λογικοί τελεστές. Το ',' αντιστοιχεί στο AND, το ';' στο OR και το '\+' στο NOT.

Οι μεταβλητές ξεκινούν πάντα με το πρώτο γράμμα κεφαλαίο, ενώ οι σταθερές με μικρό.

2. Τρόπος εκτέλεσης των ερωτήσεων στην Prolog

Η ερώτηση μπορεί να είναι είτε απλή (περιέχει μία μόνο κλήση) :

?-woman(X).

είτε σύνθετη (αποτελείται από σύζευξη πολλαπλών κλήσεων) :

?-woman(X), man(Y).

Οι κλήσεις της ερώτησης επιλέγονται με την σειρά από αριστερά προς τα δεξιά. Για κάθε κλήση της ερώτησης, ο μηχανισμός ελέγχου αναλαμβάνει να βρει μια πρόταση του προγράμματος της οποίας η κεφαλή έχει το ίδιο κατηγόρημα και τάξη με την επιλεγμένη κλήση. Η ερώτηση θεωρείται ότι απαντήθηκε, όταν απαντηθούν επιτυχώς όλες οι κλήσεις της. Εφόσον οι κλήσεις της ερώτησης περιέχουν κοινές

μεταβλητές, θα πρέπει αυτές να πάρουν την ίδια τιμή. Αν δεν υπάρχει άλλη κλήση στην ερώτηση του χρήστη τότε η εκτέλεση του προγράμματος τερματίζεται με επιτυχία και επιστρέφονται στον χρήστη οι τιμές των μεταβλητών που περιείχε η αρχική ερώτηση. Αυτό προφανώς δεν γίνεται στην αρχική ερώτηση του χρήστη (η οποία δεν είναι κενή), αλλά είναι μια κατάσταση που προκύπτει από την επαναληπτική εφαρμογή των βημάτων εκτέλεσης.

Όταν η κλήση που εξετάζεται ενοποιείται με ένα από τα γεγονότα του προγράμματος τότε αυτή ικανοποιείται και απομακρύνεται από την ερώτηση.

Παράδειγμα:

Γεγονός: friends(giannis, roula).

friends(roula, stella).

Ερώτηση: ?-friends(X, Y).

Για την αντικατάσταση {X/giannis, Y/roula} η κλήση με την πρόταση (γεγονός) γίνονται ταυτόσημα και η κλήση ικανοποιείται και απομακρύνεται. Παραμένει κενή η ερώτηση (χωρίς κλήσεις), άρα η διαδικασία απόδειξης έχει τερματίσει επιτυχημένα.

```
?- friends(X, Y).
X = giannis,
Y = roula ;
X = roula,
Y = stella.

?-
```

Αν η τρέχουσα κλήση ενοποιείται με κάποιον κανόνα, τότε αυτή απομακρύνεται από την ερώτηση και τη θέση της παίρνει το σώμα του κανόνα αυτού. Για την ικανοποίηση της αρχικής κλήσης είναι απαραίτητη η ικανοποίηση των κλήσεων του σώματος του κανόνα που την αντικατέστησε.

Παράδειγμα:

Γεγονός: friends(giannis, roula).

friends(roula, stella).

Κανόνας: known(X, Y) :- friends(X, Z), friends(Z, Y).

(Ο X γνωρίζει τον Y εάν ο X έχει φίλο τον Z και ο Z έχει φίλο τον Y).

Ερώτηση: ?-known(giannis, A).

Για {X/giannis, Y/A} η κλήση με την κεφαλή της πρότασης (κανόνας) γίνονται ταυτόσημα και η κλήση ικανοποιείται αν ικανοποιηθούν οι προϋποθέσεις (το σώμα) του κανόνα. Γίνεται αντικατάσταση της κλήσης στην ερώτηση με το σώμα.

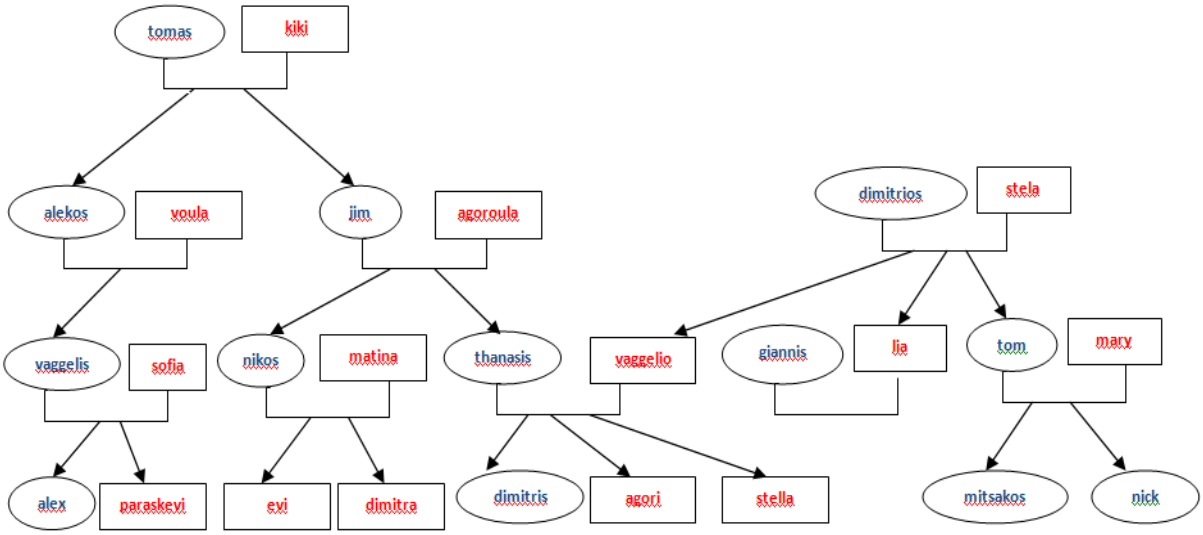
```
?- known(giannis, A).
A = stella.

?-
```

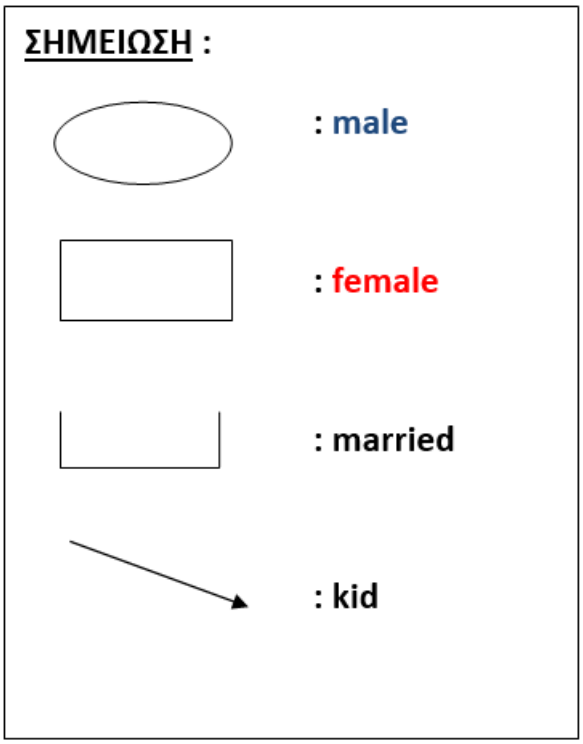
Αν υπάρχουν περισσότερες της μιας προτάσεις με τις οποίες μπορεί να ενοποιηθεί η κλήση, τότε ενοποιείται με την πρώτη. Ονομάζεται backtracking, (οπισθοδρόμηση) και αντιπροσωπεύει πιθανές εναλλακτικές "απαντήσεις" στην κλήση. Σε περίπτωση αποτυχίας εύρεσης λύσης ή σε περίπτωση που ο χρήστης ζητά και άλλη λύση, ο μηχανισμός οπισθοδρόμησης επιστρέφει στο τελευταίο σημείο οπισθοδρόμησης ακυρώνοντας τα υπολογιστικά βήματα που έπονται και αναζητά στις επόμενες προτάσεις κάποια που να μπορεί να ενοποιηθεί με την κλήση.

Αν με την εξάντληση της διαδικασίας αυτής δεν καταλήξουμε σε ερώτηση χωρίς κλήσεις, τότε η απάντηση στην αρχική ερώτηση είναι αρνητική ή αλλιώς αποτυχία. Σε αντίθετη περίπτωση η απάντηση στην ερώτηση είναι θετική ή αλλιώς επιτυχής. Συνοδεύεται από τις αναθέσεις τιμών στις μεταβλητές της αρχικής ερώτησης που προέκυψαν από τις διαδοχικές ταυτοποιήσεις.

Θέμα 1ο. Γενεαλογικό Δένδρο

Το γενεαλογικό δένδρο για την επίλυση της άσκησης απεικονίζεται στο παρακάτω διάγραμμα :





Θα υλοποιήσω τα predicates:

/*----------*/

parents(Child, Father, Mother).

male(Child).

/*----------*/

Το παιδί Child έχει πατέρα Father και μητέρα Mother.

parents(agori, thanasis, vaggelio).
parents(stella, thanasis, vaggelio).
parents(dimitris, thanasis, vaggelio).
parents(thanasis, jim, agoroula).
parents(vaggelio, dimitrios, stela).
parents(mitsakos, tom, mary).
parents(nick, tom, mary).
parents(tom, dimitrios, stela).
parents(lia, dimitrios, stela).
parents(evi, nikos, matina).
parents(dimitra, nikos, matina).
parents(nikos, jim, agoroula).
parents(alex, vaggelis, sofia).
parents(paraskevi, vaggelis, sofia).
parents(vaggelis, alekos, vivi).
parents(jim, tomas, kiki).
parents(alekos, tomas, kiki).

Αν το παιδί είναι αγόρι ικανοποιεί το γεγονός male.

male(dimitris).
male(thanasis).
male(mitsakos).
male(nick).
male(tom).
male(nikos).
male(alex).
male(vaggelis).
male(jim).
male(alekos).

1ος Κανόνας

/*----------*/

/* siblings(X, Y). % X, Y αδέλφια */

/*----------*/

Ο X και ο Y είναι αδέλφια εάν έχουν ίδιο πατέρα (A) ή ίδια μητέρα (B) και ταυτόχρονα οι X και Y αποτελούν διαφορετικά πρόσωπα.

siblings(X, Y) :- parents(X, A, B), parents(Y ,A, B), X\=Y.

siblings(X, Y) :- parents(X, A, B), parents(Y, A, C), B\=C.

siblings(X, Y) :- parents(X, A, B), parents(Y, C, B), A\=C.

Ερώτηση: ?- siblings(agori, Z).

```
?- siblings(agori, Z).
Z = stella ;
Z = dimitris ;
false.

?- siblings(thanasis, Z).
Z = nikos ;
false.

?-
```

2ος Κανόνας

/*----------*/

/* father(Father, Child). % Father είναι πατέρας του Child */

/*----------*/

Ο Father είναι πατέρας του Child, εάν ο Child έχει πατέρα τον Father, ανεξάρτητα από την μητέρα. Δηλαδή αρκεί να ικανοποιεί το γεγονός parents:

father(Father, Child) :- parents(Child, Father, _).

Ερώτηση: ?- father(Father, agori).

```
?- father(Father, agori).
Father = thanasis.

?-
```

3ος Κανόνας

/*----------*/

/* mother(Mother, Child). % Mother είναι μητέρα του Child */

/*----------*/

Η Mother είναι μητέρα του Child, εάν ο Child έχει μητέρα την Mother, ανεξάρτητα από τον πατέρα. Δηλαδή αρκεί να ικανοποιεί το γεγονός parents:

mother(Mother, Child) :- parents(Child, _, Mother).

Ερώτηση: ?- mother(Mother, agori).

```
?- mother(Mother, agori).
Mother = vaggelio.

?-
```

4ος Κανόνας

/*----------*/

/* grandfather(GFather, Child). % GFather είναι παππούς του Child */

/*----------*/

Ο GFather είναι παππούς του Child, εάν ο Child έχει πατέρα τον Father και ο Father έχει πατέρα τον GFather **ή** εάν ο Child έχει μητέρα την Mother και η Mother έχει πατέρα τον GFather.

grandfather(GFather, Child) :-

parents(Child, Father, Mother),

(parents(Father, GFather, _) ; parents(Mother, GFather, _)).

Ερώτηση: ?- grandfather(GFather, agori).

```
?- grandfather(GFather, agori).
GFather = jim ;
GFather = dimitrios.

?-
```

5ος Κανόνας

```
/*-------------------------------------------------------------------------------------------------------------*/
/*      grandmother(GMother, Child).      % GMother είναι γιαγιά του Child      */
/*-------------------------------------------------------------------------------------------------------------*/
```

Η GMother είναι γιαγιά του Child, εάν ο Child έχει πατέρα τον Father και ο Father έχει μητέρα την GMother **ή** εάν ο Child έχει μητέρα την Mother και η Mother έχει μητέρα την GMother.

```
grandmother(GMother, Child) :-
        parents(Child, Father, Mother),
        (parents(Father, _, GMother) ; parents(Mother, _, GMother)).
```

Ερώτηση: ?- grandmother(GMother, agori).

```
?- grandmother(GMother, agori).
GMother = agoroula ;
GMother = stela.

?-
```

6ος Κανόνας

```
/*-------------------------------------------------------------------------------------------------------------*/
/*      all_kids(Kids, Father).    όλα τα παιδιά του Father σε μια λίστα Kids      */
/*-------------------------------------------------------------------------------------------------------------*/
```

Όλα τα παιδιά ενός πατέρα εμφανίζονται με χρήση του κατηγορήματος **findall**, η οποία επιστρέφει σε μία λίστα **Kids** όλα τα παιδιά που έχουν πατέρα τον Father.

```
all_kids(Kids, Father) :- findall(Kid, parents(Kid, Father, _), Kids).
```

Ερώτηση: ?- all_kids(Kids, thanasis).

```
?- all_kids(Kids, thanasis).
Kids = [agori, stella, dimitris].

?-
```

7ος Κανόνας

```
/*-------------------------------------------------------------------------------------------------------------*/
/*        first_cousins(X, Y).                    % X, Y πρώτα ξαδέλφια                    */
/*-------------------------------------------------------------------------------------------------------------*/
```

Ο X και ο Y είναι πρώτα ξαδέλφια εάν κάποιος από τους γονείς του X είναι αδελφός-ή με κάποιον από τους γονείς του Y με οποιοδήποτε συνδυασμό.

```
first_cousins(X, Y) :-
        parents(X, A, B), parents(Y, C, D),
        (siblings(A, C) ; siblings(A, D) ; siblings(B, C) ; siblings(B, D)).
```

Ερώτηση: ?- first_cousins(FCousin, agori).

```
?- first_cousins(FCousin, agori).
FCousin = mitsakos ;
FCousin = nick ;
FCousin = evi ;
FCousin = dimitra ;
false.

?-
```

8ος Κανόνας

```
/*-------------------------------------------------------------------------------------------------------------*/
/*        second_cousins(X, Y).                   % X, Y δεύτερα ξαδέλφια                  */
/*-------------------------------------------------------------------------------------------------------------*/
```

Ο X και ο Y είναι δεύτερα ξαδέλφια εάν κάποιος από τους γονείς του X είναι πρώτα ξαδέλφια με κάποιον από τους γονείς του Y με οποιοδήποτε συνδυασμό.

```
second_cousins(X, Y) :- parents(X, A, B), parents(Y, C, D),
                        (first_cousins(A, C); first_cousins(A, D);
                        first_cousins(B, C); first_cousins(B, D)).
```

Ερώτηση: ?- second_cousins(SCousin, agori).

```
?- second_cousins(SCousin, agori).
SCousin = alex ;
SCousin = paraskevi ;
false.

?-
```

9ος Κανόνας

/*----------*/

/* uncle(Uncle, X). % Uncle είναι θείος του/της X */

/*----------*/

Ο Uncle είναι θείος του/της X εάν έχει αδελφό-ή τον πατέρα (A) ή την μητέρα (B) του X και ικανοποιεί τον κανόνα male.

Σημείωση:

Η περίπτωση αυτή αφορά τους θείους εξ' αίματος.

uncle(Uncle, X) :-

parents(X, A, B), (siblings(A, Uncle) ; siblings(B, Uncle)), male(Uncle).

Ερώτηση: ?- uncle(Uncle, agori).

```
?- uncle(Uncle, agori).
Uncle = nikos ;
Uncle = tom ;
false.

?-
```

Ερώτηση: ?- uncle(Uncle, dimitra).

```
?- uncle(Uncle, dimitra).
Uncle = thanasis ;
false.

?-
```

10ος Κανόνας

/*----------*/

/* aunt(Aunt,X). % Aunt είναι θεία του/της X */

/*----------*/

Η Aunt είναι θεία του X εάν έχει αδελφό-ή τον πατέρα (A) ή την μητέρα (B) του X και δεν ικανοποιεί τον κανόνα male.

Σημείωση:

Η περίπτωση αυτή αφορά τις θείες εξ' αίματος.

aunt(Aunt, X) :-

parents(X, A, B), (siblings(A, Aunt) ; siblings(B, Aunt)), \+male(Aunt).

Ερώτηση: ?- aunt(Aunt, agori).

```
?- aunt(Aunt, agori).
Aunt = lia ;
false.

?-
```

Ερώτηση: ?- aunt(Aunt, mitsakos).

```
?- aunt(Aunt, mitsakos).
Aunt = vaggelio ;
Aunt = lia ;
false.

?-
```

<u>Συμπληρωματικοί Κανόνες</u>:

Στους κανόνες 9 και 10: uncle(Uncle, X), aunt(Aunt, X), θείος και θεία λογίζονται μόνο τα αδέλφια των γονέων του X. Για να χαρακτηρίσουμε θείο-θεία και τους συζύγους των εξ' αίματος θείων, πρέπει να ορίσουμε έναν νέο κανόνα married(A, B) και να συμπληρώσουμε νέα ορίσματα στο γεγονός male.

```
married(thanasis, vaggelio).
married(dimitrios, stela).
married(jim, agoroula).
married(vaggelis, sofia).
married(alekos, vivi).
married(tomas, kiki).
married(giannis, lia).
married(nikos, matina).
married(tom, mary).
male(giannis).
```

Ο Uncle είναι θείος του X εάν έχει παντρευτεί την αδελφή της μητέρας ή του πατέρα του X, δηλαδή αν η A είναι θεία του X και οι A, Uncle ικανοποιούν τον κανόνα married.

uncle2(Uncle, X) :- aunt(A, X), married(Uncle, A).

Ερώτηση: ?- uncle2(Uncle, agori).

```
?- uncle2(Uncle, agori).
Uncle = giannis ;
false.

?-
```

Για να συμπεριλάβουμε όλες τις περιπτώσεις ορίζουμε τον νέο κανόνα:

uncle_all(Uncle, X) :- uncle(Uncle, X) ; uncle2(Uncle,X).

Ερώτηση: ?- uncle_all(Uncle_all, agori).

```
?- uncle(Uncle, agori).
Uncle = nikos ;
Uncle = tom ;
false.

?- uncle2(Uncle2, agori).
Uncle2 = giannis ;
false.

?- uncle_all(Uncle_all, agori).
Uncle_all = nikos ;
Uncle_all = tom ;
Uncle_all = giannis ;
false.

?-
```

Αντίστοιχα ορίζεται και ο κανόνας aunt_all(Aunt, X).

aunt2(A,X) :- uncle(U, X), married(U, A).

Ερώτηση: ?- aunt2(Aunt, agori).

```
?- aunt2(Aunt, agori).
Aunt = matina ;
Aunt = mary ;
false.

?-
```

aunt_all(A,X) :- aunt(A, X) ; (uncle(U, X), married(U, A)).

Ερώτηση: ?- aunt_all(Aunt, agori).

```
?- aunt(Aunt, agori).
Aunt = lia ;
false.

?- aunt2(Aunt2, agori).
Aunt2 = matina ;
Aunt2 = mary ;
false.

?- aunt_all(Aunt_all, agori).
Aunt_all = lia ;
Aunt_all = matina ;
Aunt_all = mary ;
false.

?-
```

Θέμα 2ο. Μικρά προγράμματα με λίστες

Λίστα είναι μια διατεταγμένη ακολουθία από οποιονδήποτε αριθμό στοιχείων. Τα στοιχεία αυτά μπορεί να είναι απλοί όροι, σύνθετοι όροι ή ακόμη κι άλλες λίστες. Τα στοιχεία μίας λίστας περικλείονται σε αγκύλες ([]) και χωρίζονται μεταξύ τους με κόμμα.

Μία λίστα μπορεί να είναι:

- Κενή: μία δομή χωρίς όρους που συμβολίζεται με []
- Μία δομή με δύο όρους: την κεφαλή (head) που είναι το πρώτο στοιχείο της λίστας και την ουρά (tail) που είναι το υπόλοιπο τμήμα της λίστας.

Αναδρομικότητα είναι η δυνατότητα που παρέχουν οι γλώσσες προγραμματισμού στις συναρτήσεις τους να καλούν τον εαυτό τους. Η τεχνική αυτή αποτελεί ένα από τα βασικά χαρακτηριστικά της Prolog.

1ο Κατηγόρημα

```
/*-------------------------------------------------------------------------------------------------------------*/
/*        sumlist(List, S).        % List = λίστα αριθμών, S = άθροισμά τους                    */
/*-------------------------------------------------------------------------------------------------------------*/
```

Άθροισμα S, των στοιχείων μιας λίστας List:

Για να υπολογίσουμε το άθροισμα των αριθμών που περιέχονται σε μια λίστα List, δημιουργούμε έναν αναδρομικό κανόνα ο οποίος προσθέτει ένα ένα τα στοιχεία της λίστας έως ότου φθάσει στο τέλος της και επιστρέφει αυτό το άθροισμα σε μία μεταβλητή S.

sumList([], 0).

Τερματική συνθήκη: το άθροισμα των στοιχείων της κενής λίστας ισούται με 0.

sumList([X|Xs], S) :- sumList(Xs, K), S is K+X.

Αναδρομή: το άθροισμα S των στοιχείων μιας λίστας (που δεν είναι κενή) ισούται με την τιμή του πρώτου στοιχείου αθροιζόμενο με το άθροισμα των υπολοίπων στοιχείων Xs της λίστας, οπότε το S ισούται με το άθροισμα K+X.

```
sumList([], 0).
sumList([X|Xs], S) :- sumList(Xs, K), S is K+X.
```

Ερώτηση: ?- sumList([1, 2, 3, 4, 5, 6], Sum).

```
?- sumlist([1, 2, 3, 4, 5, 6], Sum).
Sum = 21.

?- sumlist([10, 100, 1000, 10000], Sum).
Sum = 11110.

?-
```

2ο Κατηγόρημα

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*        multList(List, M). % List = λίστα αριθμών, M = γινόμενό τους                                              */
/*-------------------------------------------------------------------------------------------------------------------*/
```

Γινόμενο Μ, των στοιχείων μιας λίστας List:

Για να υπολογίσουμε το γινόμενο των αριθμών που περιέχονται σε μια λίστα List, δημιουργούμε έναν αναδρομικό κανόνα ο οποίος πολλαπλασιάζει ένα ένα τα στοιχεία της λίστας έως ότου φθάσει στο τέλος της και επιστρέφει αυτό το γινόμενο αυτό σε μία μεταβλητή Μ.

multList([], 1).

Τερματική συνθήκη: το γινόμενο των στοιχείων μιας κενής λίστας ισούται με 1.

multList([X|Xs], M) :- multList(Xs, K), M is K*X.

Αναδρομή: το γινόμενο Μ των στοιχείων μιας λίστας (που δεν είναι κενή) ισούται με την τιμή του πρώτου στοιχείου πολλαπλασιαζόμενο με το γινόμενο των υπολοίπων στοιχείων της λίστας.

multList([], 1).
multList([X|Xs], M) :- multList(Xs, K), M is K*X.

Ερώτηση: ?- multList([3, 4, 5], M).

```
?- multList([3, 4, 5], M).
M = 60.

?- multList([1, 2, 3, 0], M).
M = 0.

?-
```

3ο Κατηγόρημα

/*----------*/

/* listlength(List, L). % List = λίστα, L = μήκος λίστας */

/*----------*/

Το L είναι το μήκος μιας λίστας List:

Για να υπολογίσουμε το μήκος L μιας λίστας List δημιουργούμε έναν αναδρομικό κανόνα ο οποίος όσο υπάρχουν στοιχεία στην λίστα προσθέτει + 1 ώσπου να καταλήξει σε κενή λίστα η οποία θα επιστρέψει 0.

listlength([], 0).

Τερματική συνθήκη: το μήκος της κενής λίστας είναι 0.

listlength([_|Xs], S) :- listlength(Xs, K), S is K+1.

Αναδρομή: το μήκος L των στοιχείων μιας λίστας είναι κατά ένα μεγαλύτερο από το μήκος της ουράς της.

listlength([], 0).

listlength([_|Xs], S) :- listlength(Xs, K), S is K+1.

Ερώτηση: ?- listlength([4,5,6,7,8,[a,b,c]], L).

```
?- listlength([4,5,6,7,8,[a,b,c]], L).
L = 6.

?- listlength([[a,b,c],[a],[b],[c]], L).
L = 4.

?-
```

4ο Κατηγόρημα

/*----------*/

/* first_element(List, F). % List = λίστα, F = πρώτο στοιχείο της λίστας */

/*----------*/

Το F είναι το πρώτο στοιχείο μιας λίστας List:

Το πρώτο στοιχείο F μιας λίστας [X|_] είναι το X.

first_element([X|_], X).

Ερώτηση: ?- first_element([7,8,9], F).

```
?- first_element([7,8,9], F).
F = 7.

?- first_element([[a],[7,8,9]], F).
F = [a].

?-
```

5ο Κατηγόρημα

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*        last_element(List,L). % List = λίστα, L = Τελευταίο στοιχείο της λίστας        */
/*-------------------------------------------------------------------------------------------------------------------*/
```

Το L είναι το τελευταίο στοιχείο μιας λίστας List:

last_element([X], X) :- !.

Τερματική συνθήκη: αν μια λίστα περιέχει ένα μόνο στοιχείο, τότε αυτό είναι και το τελευταίο. Βάζω **cut (!)** γιατί θέλω να σταματήσει το backtracking μόλις βρίσκει το στοιχείο.

last_element([_|Xs], Y) :- last_element(Xs, Y).

Αναδρομή: το τελευταίο στοιχείο μιας λίστας είναι το τελευταίο στοιχείο της ουράς της.

last_element([X], X) :- !.

last_element([_|Xs], Y) :- last_element(Xs, Y).

Ερώτηση: ?- last_element([a,b,c,d], L).

```
?- last_element([a,b,c,d], L).
L = d.

?- last_element([a], L).
L = a.

?-
```

6ο Κατηγόρημα

/*----------*/

/* nth_element(List, N, E). % Επιστρέφει στο E το N-οστό στοιχείο της λίστας List */

/*----------*/

Το E είναι το στοιχείο στη N-οστή θέση μιας λίστας List:

Ο τρόπος που ορίζω το nth_element είναι: nth_element(input, input, output). Το πρώτο input είναι η λίστα List που θέλω να εφαρμόσει το κατηγόρημα, το δεύτερο input είναι η θέση N που δίνω για να δω ποιο στοιχείο βρίσκεται εκεί και το output είναι το ζητούμενο στοιχείο E.

nth_element([X|_], 1, X) :- !.

Τερματική συνθήκη: το στοιχείο που βρίσκεται στην πρώτη θέση είναι αυτό της κεφαλής της λίστας. Βάζω **cut (!)** γιατί θέλω να σταματήσει το backtracking μόλις βρίσκει το στοιχείο.

nth_element([_|Xs], N, E) :- M is N-1, nth_element(Xs, M, E).

Αναδρομή: το ζητούμενο στοιχείο E στη N-οστή δοθείσα θέση μιας λίστας, είναι αυτό που θα βρίσκεται στην κεφαλή της λίστας Xs, καθώς το N θα μειώνεται M = (N-1) φορές.

nth_element([X|_], 1, X) :- !.

nth_element([_|Xs], N, E) :- M is N-1, nth_element(Xs, M, E).

Ερώτηση: ?- nth_element([a,b,b,f,g], 4, E).

```
?- nth_element([a,b,b,f,g], 4, E).
E = f.

?- nth_element([a,b,b,f,g], 2, E).
E = b.

?-
```

7ο Κατηγόρημα

/*----------*/

/* element_position(List, E, N). % Επιστρέφει στο N την θέση του στοιχείου E της λίστας List */

/*----------*/

Το δοθέν στοιχείο Ε βρίσκεται στην Ν θέση μιας λίστας List:

Ο τρόπος που ορίζω το element_position είναι: element_position(input, input, output). Το πρώτο input είναι η λίστα List που θέλω να εφαρμόσει το κατηγόρημα, το δεύτερο input είναι το στοιχείο Ε της λίστας που θέλω να ψάξω σε ποια ή ποιες θέσεις βρίσκεται και το output είναι η θέση ή οι θέσεις Ν που βρίσκεται το στοιχείο της λίστας που έδωσα σαν input.

element_position([X|_], X, 1).

Τερματική συνθήκη: στην 1η θέση βρίσκεται το στοιχείο της κεφαλής της λίστας. Στην τερματική συνθήκη δεν βάζω cut (!) γιατί μπορεί να θέλω να βρω κι άλλα στοιχεία.

element_position([_|Xs], X, N) :- element_position(Xs, X, N1), N is N1 + 1.

Αναδρομή: το δοθέν στοιχείο Χ μιας λίστας βρίσκεται σε θέση κατά μια μονάδα μεγαλύτερη από το πλήθος κλήσης της συνάρτησης για το υπόλοιπο της λίστας Xs, ώστε το Χ να συμπίπτει με την κεφαλή της λίστας Xs.

element_position([X|_], X, 1).

element_position([_|Xs], X, N):- element_position(Xs, X, N1), N is N1 + 1.

Ερώτηση: ?- element_position([a,b,c,c,d,e,f,c], c, N).

```
?- element_position([a,b,c,c,d,e,f,c], c, N).
N = 3 ;
N = 4 ;
N = 8 ;
false.

?- element_position([a], a, N).
N = 1 ;
false.

?-
```

8ο Κατηγόρημα

/*----------*/

/* my_member(X, List). % X είναι στοιχείο της λίστας List */

/*----------*/

Το X είναι στοιχείο μιας λίστας List:

my_member(X, [X|_]).

Τερματική συνθήκη: ένα στοιχείο X είναι μέλος μιας λίστας εάν είναι η κεφαλή της λίστας.

my_member(X, [_|Xs]) :- my_member(X, Xs).

Αναδρομή: ένα στοιχείο X είναι μέλος μιας λίστας εάν είναι μέλος της ουράς της λίστας, δηλαδή αν είναι ένα από τα υπόλοιπα στοιχεία μετά την κεφαλή της λίστας.

my_member(X, [X|_]).

my_member(X, [_|Xs]) :- my_member(X, Xs).

Ερώτηση: ?- my_member(b, [a,b,c]).

```
?- my_member(b, [a,b,c]).
true ;
false.

?- my_member(b, [a,b,c,b,d,e,f,b]).
true ;
true ;
true ;
false.

?-
```

9ο Κατηγόρημα

/*----------*/

/*my_select(X, List, Rest). % X είναι στοιχείο της λίστας List, Rest η υπόλοιπη λίστα */

/*----------*/

Το X είναι στοιχείο μιας λίστας List και το Rest η υπόλοιπη λίστα:

Η λίστα Rest είναι το αποτέλεσμα της κατάργησης μιας εμφάνισης του X από τη λίστα List.

my_select(X, [X|Xs], Xs).

Το στοιχείο X είναι η κεφαλή της λίστας και αφαιρείται από αυτήν, δίνοντας το υπόλοιπο της Xs.

my_select(X, [Y|Ys], [Y|Zs]) :- my_select(X, Ys, Zs).

Αναδρομή: το στοιχείο X αφαιρείται από την λίστα [Y|Ys] και δίνει την λίστα [Y|Zs].

my_select(X, [X|Xs], Xs).

my_select(X, [Y|Ys], [Y|Zs]) :- my_select(X, Ys, Zs).

Ερώτηση: ?- my_select(X, [4,6,8], R).

```
?- my_select(X, [4,6,8], R).
X = 4,
R = [6, 8] ;
X = 6,
R = [4, 8] ;
X = 8,
R = [4, 6] ;
false.

?-
```

10ο Κατηγόρημα

/*----------*/

/* my_append(L1, L2, R). % R είναι το αποτέλεσμα της 'ένωσης' των λιστών L1, L2 */

/*----------*/

Το R είναι το αποτέλεσμα της ένωσης των λιστών L1, L2:

my_append([], List2, List2).

Τερματική συνθήκη: η συνένωση της κενής λίστας με οποιαδήποτε λίστα List2 έχει ως αποτέλεσμα την ίδια την λίστα List2.

my_append([X|Tail], List2, [X|List3]) :- my_append(Tail, List2, List3).

Αναδρομή: αν η πρώτη λίστα δεν είναι κενή η συνένωση με οποιαδήποτε λίστα είναι μια νέα λίστα η οποία έχει κεφαλή την κεφαλή της πρώτης λίστας και ουρά την συνένωση της ουράς της πρώτης λίστας με την δεύτερη λίστα.

my_append([], List2, List2).

my_append([X|Tail], List2, [X|List3]) :- my_append(Tail, List2, List3).

Ερώτηση: ?- my_append([a,b,c,d], [e,f,g], R).

```
?- my_append([a,b,c,d], [e,f,g], R).
R = [a, b, c, d, e, f, g].

?- my_append([a,b,c,d], [], R).
R = [a, b, c, d].

?- my_append([], [], R).
R = [].

?-
```

Θέμα 3ο. Πράξεις σε χάρτη

Ένας χάρτης αποτελεί μια λίστα από άλλες λίστες.

Παράδειγμα :

map ([[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]]).

Διαφορετικά η ανωτέρω λίστα γράφεται :

```
        %X   1     2     3     4     5     Y

map([
             [0,   0,    0,    0,    0],   %1
             [1,   0,    0,    1,    1],   %2
             [1,   1,    0,    1,    1],   %3
             [1,   1,    1,    1,    1],   %4
             [1,   1,    1,    1,    1],   %5
             ]).
```

Το X εκπροσωπεί τα στοιχεία κάθε εσωτερικής λίστας (στήλες).

Το Y εκπροσωπεί τα στοιχεία της λίστας map (γραμμές).

1ο Κατηγόρημα

```
/*-------------------------------------------------------------------------------------------------------------*/
/*xymapelement(Map,X,Y,E).        % το E είναι το περιεχόμενο του χάρτη Map στη θέση X,Y                    */
/*-------------------------------------------------------------------------------------------------------------*/
xymapelement(input, input, input, output).
```

Δίνω σαν δεδομένα τον χάρτη Map και τις συντεταγμένες X,Y και παίρνω ως αποτέλεσμα E το περιεχόμενο του χάρτη Map στην θέση X,Y.

Αρχικά θα πρέπει να συντάξουμε έναν κανόνα που επιστρέφει το στοιχείο X, μιας δοθείσας θέσης N σε μια λίστα.

Αυτός ο κανόνας έχει ήδη αναλυθεί στο 2ο Θέμα « nth_element(List, N, X). ».

nth_element ([X|_], 1, X) :- !.

nth_element([_|L], N, X) :- N1 is N-1, nth_element (L, N1, X).

Ερώτηση: ?- nth_element([1,2,3,4,5], 3, X).

```
?- nth_element([1,2,3,4,5], 3, X).
X = 3.

?-
```

Εφαρμόζοντας τον παραπάνω κανόνα, αρχικά μπορούμε να βρούμε σε ποιο στοιχείο – λίστα Y βρισκόμαστε στην λίστα map (A).

Το ζητούμενο στοιχείο E βρίσκεται στην θέση X (στήλη) του στοιχείου – λίστας Y (B).

xymapelement(Map, X, Y, E):-

nth_element(Map, Y, Line), %(A)

nth_element(Line, X, E). %(B)

Ερώτηση: ?- map(M), xymapelement(M,1,4,E).

```
?- map(M), xymapelement(M,1,4,E).
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

E = 1.

?- map(M), xymapelement(M,3,3,E).
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

E = 0.

?-
```

2ο Κατηγόρημα

/*----------*/

/*mapelementposition(Map, E, X, Y). % βρίσκει τις συντεταγμένες X,Y του E στο χάρτη Map */

/*----------*/

Με δοθέν στοιχείο E στον χάρτη map, εντοπίζουμε τις συντεταγμένες X, Y.

Αρχικά θα πρέπει να συντάξουμε έναν κανόνα που θα επιστρέφει την θέση N ενός δοθέντος στοιχείου E σε μια λίστα.

Αυτός ο κανόνας έχει αναλυθεί στο 2ο Θέμα « element_position(List, E, N). ».

element_position([X|_], X, 1).

element_position([_|Xs], X, N) :- element_position(Xs, X, N1), N is N1 + 1.

Ερώτηση: ?- element_position([1, 4, 6, 7], 6, N).

```
?- element_position([1,4,6,7], 6, N).
N = 3 ;
false.

?-
```

Εφαρμόζοντας τον παραπάνω κανόνα, αρχικά μπορούμε να βρούμε σε ποιο στοιχείο – λίστα Y βρίσκουμε το στοιχείο E στην λίστα map.

Το ζητούμενο στοιχείο E βρίσκεται στην θέση X (στήλη) του στοιχείου – λίστας Y.

mapelementposition([Map, E, X, Y) :-

element_position(Map, Line, Y),

element_position(Line, E, X).

Ερώτηση: ?- map(M), mapelementposition(M, 0, X, Y).

```
?- map(M), mapelementposition(M, 0, X, Y).
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

X = Y, Y = 1 ;
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

X = 2,
Y = 1 ;
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

X = 3,
Y = 1 ;
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

X = 4,
Y = 1
```

3ο Κατηγόρημα

/*----------*/

/* printlist(List) % τυπώνει οριζόντια τα στοιχεία της λιστας List */

/*----------*/

Για να τυπώσουμε οριζόντια τα στοιχεία μιας λίστας εφαρμόζουμε τον εξής κανόνα:

printlist([]) :- nl. !.

Τερματική συνθήκη: αν η λίστα είναι κενή δεν εκτυπώνω τίποτα.

printlist([X|Xs]) :- write(X), write(' '), printlist(Xs).

Αναδρομή: διαφορετικά εκτυπώνεται η κεφαλή της υπόλοιπης λίστας έως ότου η λίστα Xs γίνει κενή.

printlist([]) :- nl, !.

printlist([X|Xs]) :- write(X), write(' '), printlist(Xs).

Έστω η λίστα: lista([1,2,3,4,5,6]).

Ερώτηση: ?- lista(L), printlist(L).

```
?- lista(L), printlist(L).
1 2 3 4 5 6
L = [1, 2, 3, 4, 5, 6].

?-
```

4ο Κατηγόρημα

/*----------*/

/* printmap(Map) % τυπώνει όμορφα τον χάρτη Map */

/*----------*/

Για να τυπώσουμε τα στοιχεία ενός χάρτη όμορφα εφαρμόζουμε τον εξής κανόνα:

printmap([]) :- nl, !.

Τερματική συνθήκη: αν ο χάρτης είναι κενός δεν εκτυπώνω τίποτα.

printmap([Line|Rest]) :- printlist (Line), printmap(Rest).

Αναδρομή: διαφορετικά, με κλήση της printlist, εκτυπώνεται κάθε στοιχείο – λίστα του χάρτη.

printmap([]) :- nl , !.

printmap([Line|Rest]) :- printlist (Line), printmap(Rest).

Ερώτηση: ?- map(Map), printmap(Map).

```
?- map(Map), printmap(Map).
0 0 0 0 0
1 0 0 1 1
1 1 0 1 1
1 1 1 1 1
1 1 1 1 1

Map = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]
].

?-
```

Για να αποφύγουμε την εκτύπωση του χάρτη map σε οριζόντια λίστα ορίζουμε τον κανόνα:

print_map :- map(Map), printmap(Map).

Ερώτηση: ?- print_map.

```
?- print_map.
0 0 0 0 0
1 0 0 1 1
1 1 0 1 1
1 1 1 1 1
1 1 1 1 1

true.

?-
```

5ο Κατηγόρημα

/*----------*/

/* map_maxXY(Map,MaxX,MaxY) % επιστρέφει τα MaxX και MaxY του χάρτη Map */

/*----------*/

Προσδιορισμός ορίων χάρτη (maxX, maxY):

mapmaxXY(Map, MaxX,MaxY).

Ο κανόνας που προσδιορίζει το μήκος μιας λίστας έχει ορισθεί στο 2ο Θέμα και είναι ο «listlength(List, L). ».

listlength([], 0).

listlength([_|Xs], S) :- listlength(Xs, K), S is K+1.

Το πρώτο στοιχείο μιας λίστας προσδιορίζεται από τον κανόνα:

firstElement([X|_], X).

Συνδυάζοντας τους παραπάνω κανόνες και με χρήση αναδρομής μπορούμε να βρούμε το maxX και maxY.

map_maxXY(M, X, Y):-

firstElement(M, F),

listlength(F, X),

listlength(M, Y).

Ερώτηση: ?- map(M), map_maxXY(M, X, Y).

```
?- map(M), map_maxXY(M, X, Y).
M = [[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1|...]],

X = Y, Y = 5.

?-
```

Για να εμφανίζονται πιο όμορφα τα αποτελέσματα, αλλάζουμε λίγο τον κανόνα:

xymap :-

map(Map),

map_maxXY(Map, MaxX, MaxY),

write("MaxX= "), write(MaxX), nl,

write("MaxY= "), write(MaxY), nl.

Ερώτηση: ?- xymap.

```
?- xymap.
MaxX= 5
MaxY= 5
true.

?-
```

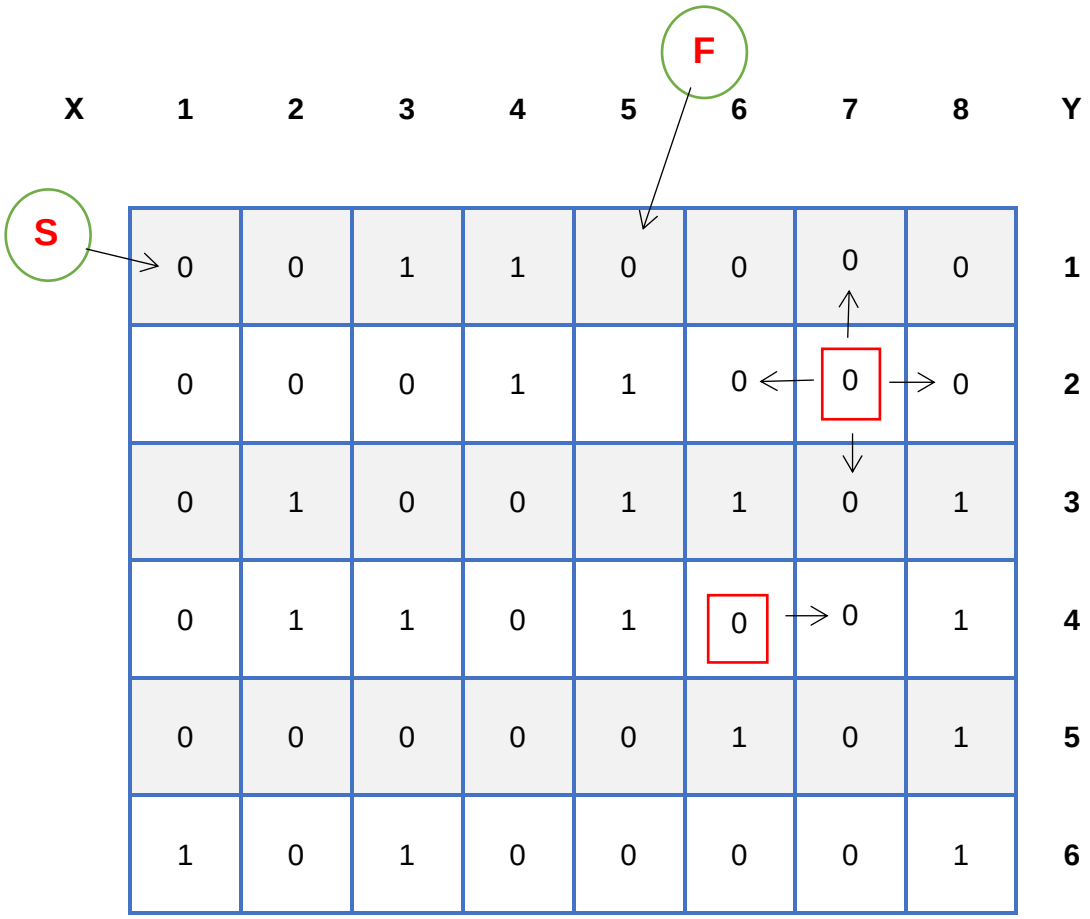
<u>Σημείωση:</u>

Στα παραπάνω ερωτήματα χρησιμοποιήσαμε κανόνες που έχουμε ήδη ορίσει σε προηγούμενα θέματα. Για τον λόγο αυτό, στον κώδικά μας, φτιάχνουμε μια «Βιβλιοθήκη» όπου γράφουμε όλους αυτούς τους κανόνες για να μπορούμε να τους χρησιμοποιούμε όποτε χρειαστεί.

```
/*-----------------------------------------------------------------------------------------------------------*/
/*                                          LIBRARY                                                          */
/*-----------------------------------------------------------------------------------------------------------*/

nth_element ([X|_], 1, X) :- !.
nth_element([_|L], N, X) :- N1 is N-1, nth_element (L, N1, X).

element_position([X|_], X, 1).
element_position([_|Xs], X, N) :-
                                    element_position(Xs, X, N1), N is N1 + 1.

listlength([], 0).
listlength([_|Xs], S) :- listlength(Xs, K), S is K+1.

firstElement([X|_], X).
```

Θέμα 4ο. Depth First Search σε χάρτη

Υποθέτουμε ότι βρισκόμαστε σε έναν λαβύρινθο, ο οποίος ουσιαστικά αποτελεί έναν δισδιάστατο πίνακα με τα εξής χαρακτηριστικά:



1. Τιμές έχουμε το μηδέν (0) και το ένα (1).
2. Τα κελιά που έχουν τιμή 0 θεωρούνται διάδρομοι.
3. Τα κελιά με τιμή 1 θεωρούνται εμπόδια.
4. Δεν μπορούμε να βγούμε εκτός των ορίων του λαβύρινθου.
5. Για να διασχίσουμε τον λαβύρινθο μπορούμε να ξεκινήσουμε από κάποιο κελί με τιμή 0 (Start) και να τελειώσουμε σε κάποιο κελί με τιμή 0 (Finish) την διαδρομή μας (Path).
6. Το επόμενο έγκυρο κελί που μπορούμε να κινηθούμε είναι αυτό που βρίσκεται επάνω, κάτω, δεξιά ή αριστερά του τρέχοντος κελιού, έχει τιμή 0 και είναι εντός των ορίων του πίνακα.
7. Το κελί που διασχίζουμε δεν πρέπει να το συναντάμε δεύτερη φορά.

*Αν βρεθώ στο κελί (6, 4) μπορώ να μετακινηθώ μόνο προς τα δεξιά στο κελί (7, 4).

** Αν βρεθώ στο κελί (7, 2) μπορώ να μετακινηθώ είτε δεξιά (8, 2) είτε αριστερά (6, 2) είτε πάνω (7, 1) είτε κάτω (7, 3).

Ο χάρτης του λαβύρινθου αποτελείται από 6 γραμμές και 8 στήλες ενός δισδιάστατου πίνακα.

```
    %X  1   2   3   4   5   6   7   8   Y

map([
        [0,  0,  1,  1,  0,  0,  0,  0],  %1
        [0,  0,  0,  1,  1,  0,  0,  0],  %2
        [0,  1,  0,  0,  1,  1,  0,  1],  %3
        [0,  1,  1,  0,  1,  0,  0,  1],  %4
        [0,  0,  0,  0,  0,  1,  0,  1],  %5
        [1,  0,  1,  0,  0,  0,  0,  1],  %6
        ]).

start( (1, 1) ).
finish( (5, 1) ).
```

Ο κανόνας που επιστρέφει το στοιχείο X μιας δοθείσας θέσης N σε μια λίστα έχει ορισθεί στο 3ο Θέμα και είναι ο εξής:

```
nth_element([X|_], 1, X) :- !.
nth_element([_|L], N, X) :- N1 is N-1, nth_element(L, N1, X).
```

Το E είναι το περιεχόμενο του χάρτη Map στη θέση X, Y. Ο κανόνας έχει ήδη αναλυθεί στο 3ο Θέμα.

```
xymapelement(Map, X, Y, E) :-
                        nth_element(Map, Y, Line),
                        nth_element(Line, X, E).
```

```
?- map(M), xymapelement(M,1,4,E).
M = [[0, 0, 1, 1, 0, 0, 0, 0], [0, 0, 0, 1, 1, 0, 0|...], [0, 1, 0, 0, 1, 1|...], [0, 1, 1,
 0, 1|...], [0, 0, 0, 0|...], [1, 0, 1|...]],
E = 0.

?-
```

Οι έγκυρες κινήσεις εντός του λαβυρίνθου είναι :

1. Κάτω :

   move((X, Y), (NX, NY)) :- NX is X, NY is Y+1.

2. Πάνω :

   move((X, Y), (NX, NY)) :- NX is X, NY is Y-1.

3. Δεξιά :

   move((X, Y), (NX, NY)) :- NX is X+1, NY is Y.

4. Αριστερά :

   move((X, Y), (NX, NY)) :- NX is X-1, NY is Y.

5. Εντός των ορίων του χάρτη, δηλαδή τα X και Y πρέπει να είναι θετικά και να μην ξεπερνούν τις μέγιστες τιμές :

   0 < X , X =< 8, 0 < Y, Y =< 6.

6. Η τιμή πρέπει να είναι 0 (το 1 είναι τοίχος) :

   valid(M, (X, Y)) :-

   0 < X, X =< 8, 0 < Y, Y =< 6,

   xymapelement(M, X, Y, E), E = 0, !.

```
?- map(M), valid(M,(1,1)).
M = [[0, 0, 1, 1, 0, 0, 0, 0], [0, 0, 0, 1, 1, 0, 0|...], [0, 1, 0, 0, 1, 1|...], [0, 1, 1,
 0, 1|...], [0, 0, 0, 0|...], [1, 0, 1|...]].

?- map(M), valid(M,(1,6)).
false.

?-
```

Αν η τιμή δεν είναι 0 επιστρέφει false.

Ο κανόνας που υπολογίζει την διαδρομή από ένα έγκυρο σημείο εκκίνησης σε ένα άλλο έγκυρο σημείο είναι :

findpath(M, S, F, Visited, [S|P]) :-

move(S, Next),

\+ (member(Next, Visited)),

valid(M, Next),

findpath(M, Next, F ,[Next|Visited], P).

όπου :

- M: ο χάρτης
- S: έγκυρο σημείο εκκίνησης (start),
- F: έγκυρο σημείο τερματισμού (finish),
- Visited: λίστα με τα σημεία που έχω ήδη επισκεφθεί
- move(S, Next): το επόμενο στοιχείο που θα επισκεφθώ
- \+ (member(Next, Visited)): το επόμενο στοιχείο δεν θα πρέπει να είναι μέλος της λίστας Visited. Η member είναι προκαθορισμένη συνάρτηση στην prolog.
- valid(M, Next): το επόμενο στοιχείο θα πρέπει να είναι έγκυρο (όπως ορίσθηκε παραπάνω)
- [S|P]: η επιστρεφόμενη λίστα με στοιχεία τις θέσεις της ζητούμενης διαδρομής. Η λίστα έχει κεφαλή το start (S).

Εάν ξεκινήσω από ένα οποιοδήποτε έγκυρο σημείο και καταλήξω ξανά σε αυτό, το μονοπάτι θα αποτελείται μόνο από αυτή την θέση.

findpath(_,F,F,_,[F]).

Ο κανόνας που καλεί την findpath(M, S, F, Visited, [S|P]) πιο κομψά γράφεται:

path_from_to(P) :-

start(S), finish(F), map(M),

findpath(M, S, F ,[S] ,P).

Σημείο εκκίνησης : start((1 , 1)).

Σημείο τερματισμού : finish((1 , 1)).

```
?- path_from_to(P).
P = [(1, 1)] ;
false.

?-
```

Σημείο εκκίνησης : start((1 , 1)).

Σημείο τερματισμού : finish((1 , 5)).

```
?- path_from_to(P).
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5)] ;
P = [(1, 1),  (1, 2),  (2, 2),  (3, 2),  (3, 3),  (4, 3),  (4, 4),  (4, 5),  (..., ...)|...
] ;
P = [(1, 1),  (2, 1),  (2, 2),  (3, 2),  (3, 3),  (4, 3),  (4, 4),  (4, 5),  (..., ...)|...
] ;
P = [(1, 1),  (2, 1),  (2, 2),  (1, 2),  (1, 3),  (1, 4),  (1, 5)] ;
false.

?-
```

Εμφανίζονται πάρα πολλά διαφορετικά μονοπάτια.

Για να εκτυπώσουμε τα σημεία του μονοπατιού μας, συντάσσουμε τον κανόνα:

printPath.

printPath([]) :- nl, ! .

Τερματική συνθήκη: αν η λίστα είναι κενή μην κάνει τίποτα.

printPath([(X, Y)|L]) :- write(X), write(','), write(Y), write(' '), printPath(L).

Αναδρομή: διαφορετικά να γράψει το X και το Y της κεφαλής της κάθε λίστας.

Οι κανόνες printPath και path_from_to(P) μας δίνουν τον κανόνα:

path :- path_from_to(Path), printPath(Path), nl.

Σημείο εκκίνησης : start((1 , 1)).

Σημείο τερματισμού : finish((5 , 1)).

```
?- path_from_to(P).
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5),  (2, 5),  (3, 5),  (4, 5),  (..., ...)|...] ;
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5),  (2, 5),  (3, 5),  (4, 5),  (..., ...)|...] ;
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5),  (2, 5),  (3, 5),  (4, 5),  (..., ...)|...] .

?- path.
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7,2   7,1   6,1   5,1

true ;
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7,2   8,2   8,1   7,1   6,1   5,1

true ;
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7,2   6,2   6,1   5,1

true .

?-
```

Προεκτάσεις Θέματος:

Η αναζήτηση Depth First δεν επιστρέφει την καλύτερη λύση, δηλαδή το πιο σύντομο μονοπάτι αλλά όλες τις δυνατές λύσεις.

Ένας εναλλακτικός τρόπος εύρεσης μονοπατιού είναι ο κανόνας:

pathfinder(S,F,P) :-

map(M),

findpath(M,S,F,[S],P).

όπου S(start): σημείο εκκίνησης, F(finish): σημείο τερματισμού, P: επιστρεφόμενη λίστα με κάθε έγκυρο μονοπάτι.

```
?- pathfinder((1,1),(5,1),P).
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5),  (2, 5),  (3, 5),  (4, 5),  (..., ...)|...
] ;
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5),  (2, 5),  (3, 5),  (4, 5),  (..., ...)|...
] ;
P = [(1, 1),  (1, 2),  (1, 3),  (1, 4),  (1, 5),  (2, 5),  (3, 5),  (4, 5),  (..., ...)|...
] .

?-
```

Χρησιμοποιώντας τον κανόνα map_maxXY(Map, MaxX, MaxY) που αναλύθηκε στο 3ο Θέμα για την εύρεση των ορίων ενός χάρτη Map, μπορούμε να γράψουμε τον κανόνα valid ως εξής:

valid_XY(M, (X,Y)) :-

map_maxXY(M, MaxX, MaxY),

X>0, X=<MaxX, Y>0, Y=<MaxY,

map_element(M, X, Y, E), E = 0.

```
?- map(M), valid_XY(M, (6, 7)).
false.

?- map(M), valid_XY(M, (1, 1)).
M = [[0, 0, 1, 1, 0, 0, 0, 0], [0, 0, 0, 1, 1, 0, 0|...], [0, 1, 0, 0, 1, 1|...], [0, 1, 1,
 0, 1|...], [0, 0, 0, 0|...], [1, 0, 1|...]].

?-
```

Ο κανόνας path_from_to(Path) επιστρέφει όλους τους δυνατούς συνδυασμούς για ένα έγκυρο μονοπάτι.

Χρησιμοποιώντας τον κανόνα listlength(List, L) που αναλύθηκε στο 2ο Θέμα μπορούμε να γράψουμε τον κανόνα path ως εξής:

path_length :-

path_from_to(Path),

printPath(Path),

listlength(Path, Length),

write("Length: "), write(Length).

```
?- path_length.
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   7,1   6,1   5,1
Length: 19
true ;
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   8,2   8,1   7,1   6,1   5,1
Length: 21
true ;
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   6,2   6,1   5,1
Length: 19
true ;
```

Το μεγαλύτερο μονοπάτι είναι αυτό που έχει μήκος L = MaxX * MaxY, δηλαδή αυτό που περιλαμβάνει όλα τα κελιά του χάρτη.

Για να βρω την καλύτερη λύση θα πάρω ως αφετηρία το μονοπάτι με μήκος L και κάθε φορά θα καλώ αναδρομικά τον κανόνα path_from_to(Path) μειώνοντας το μήκος κατά 1, δηλαδή L = L – 1. Όταν βρεθεί το μικρότερο μονοπάτι χρησιμοποιώ **cut(!)** για να διακόψω την οπισθοδρόμηση.

better :-

map(M),

map_maxXY(M, MaxX, MaxY),

better_path(MaxX*MaxY).

better_path(L) :-

path_from_to(Path),

listlength(Path, Length),

Length =< L,

printPath(Path),write("Length: "), write(Length) ,nl,

L1 is Length-1, ! ,

better_path(L1).

```
?- path.
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   7,1   6,1   5,1

true ;
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   8,2   8,1   7,1   6,1   5,1


...


?- path_length.
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   7,1   6,1   5,1
Length: 19
true ;
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   8,2   8,1   7,1   6,1   5,1
Length: 21


...


?- better.
1,1   1,2   1,3   1,4   1,5   2,5   3,5   4,5   4,6   5,6   6,6   7,6   7,5   7,4   7,3   7
,2   7,1   6,1   5,1
Length: 19
false.

?-
```

Θέμα 5ο. Α Έκπληξη

Η εκτέλεση του ερωτήματος: ?- nomismata1(355,X). επιστρέφει τα παρακάτω αποτελέσματα:

```
?- nomismata1(355,X).
X = [0, 1, 8, 12, 2, 5, 5] ;
X = [0, 1, 8, 12, 2, 6, 3] ;
X = [0, 1, 8, 12, 3, 3, 4] ;
X = [0, 1, 8, 12, 3, 4, 2] ;
X = [0, 1, 8, 12, 3, 5, 0] ;
X = [0, 2, 6, 11, 2, 5, 5] ;
X = [0, 2, 6, 11, 2, 6, 3] ;
X = [0, 2, 6, 11, 3, 3, 4] ;
X = [0, 2, 6, 11, 3, 4, 2] ;
X = [0, 2, 6, 11, 3, 5, 0] ;
X = [0, 2, 6, 12, 0, 5, 5] ;
X = [0, 2, 6, 12, 0, 6, 3] ;
X = [0, 2, 6, 12, 1, 3, 4] ;
X = [0, 2, 6, 12, 1, 4, 2] ;
X = [0, 2, 6, 12, 1, 5, 0] ;
X = [0, 2, 6, 12, 2, 0, 5] ;
X = [0, 2, 6, 12, 2, 1, 3] ;
X = [0, 2, 6, 12, 2, 2, 1] ;
X = [0, 2, 6, 12, 3, 0, 0] ;
X = [0, 2, 7, 9, 2, 5, 5] ;
X = [0, 2, 7, 9, 2, 6, 3] ;
X = [0, 2, 7, 9, 3, 3, 4] ;
X = [0, 2, 7, 9, 3, 4, 2] ;
X = [0, 2, 7, 9, 3, 5, 0] ;
X = [0, 2, 7, 10, 0, 5, 5] ;
X = [0, 2, 7, 10, 0, 6, 3] ;
X = [0, 2, 7, 10, 1, 3, 4] ;
X = [0, 2, 7, 10, 1, 4, 2] ;
X = [0, 2, 7, 10, 1, 5, 0] ;
X = [0, 2, 7, 10, 2, 0, 5] ;
X = [0, 2, 7, 10, 2, 1, 3] ;
X = [0, 2, 7, 10, 2, 2, 1] ;
X = [0, 2, 7, 10, 3, 0, 0] ;
X = [0, 2, 7, 11, 0, 0, 5] ;
X = [0, 2, 7, 11, 0, 1, 3] ;
X = [0, 2, 7, 11, 0, 2, 1] ;
X = [0, 2, 7, 11, 1, 0, 0] ;
X = [0, 2, 8, 7, 2, 5, 5] ;
X = [0, 2, 8, 7, 2, 6, 3] ;
X = [0, 2, 8, 7, 3, 3, 4] ;
X = [0, 2, 8, 7, 3, 4, 2] ;
X = [0, 2, 8, 7, 3, 5, 0] ;
X = [0, 2, 8, 8, 0, 5, 5]
```

……

και η εκτέλεση του ερωτήματος ?- nomismata2(355,X).:

```
?- nomismata2(355,X).
X = [3, 1, 0, 0, 1, 0, 0] ;
X = [2, 1, 5, 0, 1, 0, 0] ;
X = [3, 0, 2, 1, 1, 0, 0] ;
X = [2, 2, 2, 1, 1, 0, 0] ;
X = [2, 0, 7, 1, 1, 0, 0] ;
X = [1, 2, 7, 1, 1, 0, 0] ;
X = [2, 1, 4, 2, 1, 0, 0] ;
X = [3, 0, 1, 3, 1, 0, 0] ;
X = [2, 2, 1, 3, 1, 0, 0] ;
X = [2, 0, 6, 3, 1, 0, 0] ;
X = [1, 2, 6, 3, 1, 0, 0] ;
X = [2, 1, 3, 4, 1, 0, 0] ;
X = [1, 1, 8, 4, 1, 0, 0] ;
X = [3, 0, 0, 5, 1, 0, 0] ;
X = [2, 2, 0, 5, 1, 0, 0] ;
X = [2, 0, 5, 5, 1, 0, 0] ;
X = [1, 2, 5, 5, 1, 0, 0] ;
X = [2, 1, 2, 6, 1, 0, 0] ;
X = [1, 1, 7, 6, 1, 0, 0] ;
X = [2, 0, 4, 7, 1, 0, 0] ;
X = [1, 2, 4, 7, 1, 0, 0] ;
X = [2, 1, 1, 8, 1, 0, 0] ;
X = [1, 1, 6, 8, 1, 0, 0] ;
X = [2, 0, 3, 9, 1, 0, 0] ;
X = [1, 2, 3, 9, 1, 0, 0] ;
X = [1, 0, 8, 9, 1, 0, 0] ;
X = [0, 2, 8, 9, 1, 0, 0] ;
X = [2, 1, 0, 10, 1, 0, 0] ;
X = [1, 1, 5, 10, 1, 0, 0] ;
X = [2, 0, 2, 11, 1, 0, 0] ;
X = [1, 2, 2, 11, 1, 0, 0] ;
X = [1, 0, 7, 11, 1, 0, 0] ;
X = [0, 2, 7, 11, 1, 0, 0] ;
X = [1, 1, 4, 12, 1, 0, 0] ;
X = [3, 0, 2, 0, 3, 0, 0] ;
X = [2, 2, 2, 0, 3, 0, 0] ;
X = [2, 0, 7, 0, 3, 0, 0] ;
X = [1, 2, 7, 0, 3, 0, 0] ;
X = [2, 1, 4, 1, 3, 0, 0] ;
X = [3, 0, 1, 2, 3, 0, 0] ;
X = [2, 2, 1, 2, 3, 0, 0] ;
X = [2, 0, 6, 2, 3, 0, 0] ;
X = [1, 2, 6, 2, 3, 0, 0] .....
```

Η κλήση του ερωτήματος **?- nomismata1(Poso, X).** ενοποιείται με τον κανόνα **nomismata1(Poso, [A100,A50,A20,A10,A5,A2,A1])** ,
απομακρύνεται από το ερώτημα και την θέση της παίρνει το σώμα του κανόνα:

member(A100, [0,1,2,3]),
member(A50, [0,1,2]),
member(A20, [0,1,2,3,4,5,6,7,8]),
member(A10, [0,1,2,3,4,5,6,7,8,9,10,11,12]),
member(A5, [0,1,2,3]),
member(A2, [0,1,2,3,4,5,6]),
member(A1, [0,1,2,3,4,5]),
Poso is A100*100+A50*50+A20*20+A10*10+A5*5+A2*2+A1*1.

- Το A100 αντιπροσωπεύει χαρτονόμισμα αξίας 100 χρηματικών μονάδων και μπορούμε κάθε φόρα να χρησιμοποιήσουμε 0, 1, 2 ή 3 χαρτονομίσματα.
- Το A50 αντιπροσωπεύει χαρτονόμισμα αξίας 50 χρηματικών μονάδων και μπορούμε να χρησιμοποιήσουμε 0, 1 ή 2 χαρτονομίσματα.
- Το A20 αντιπροσωπεύει χαρτονόμισμα αξίας 20 χρηματικών μονάδων και μπορούμε να χρησιμοποιήσουμε από 0 έως 8 χαρτονομίσματα.
- Το A10 αντιπροσωπεύει χαρτονόμισμα αξίας 10 χρηματικών μονάδων και μπορούμε να χρησιμοποιήσουμε από 0 έως 12 χαρτονομίσματα.
- Το A5 αντιπροσωπεύει χαρτονόμισμα αξίας 5 χρηματικών μονάδων και μπορούμε να χρησιμοποιήσουμε 0, 1, 2 ή 3 χαρτονομίσματα.
- Το A2 αντιπροσωπεύει νόμισμα αξίας 2 χρηματικών μονάδων και μπορούμε να χρησιμοποιήσουμε από 0 έως 6 νομίσματα.
- Το A1 αντιπροσωπεύει νόμισμα αξίας 1 χρηματικής μονάδας και μπορούμε να χρησιμοποιήσουμε από 0 έως 5 νομίσματα.

Το αποτέλεσμα που θα επιστραφεί πρέπει να ικανοποιεί το σώμα του κανόνα. Δηλαδή πρέπει η τιμή Poso να ισούται με την δοθείσα τιμή και τα στοιχεία της λίστας X θα πρέπει να αποτελούν συνδυασμό στοιχείων των λιστών του σώματος.

Αυτό που επιτελεί ο κανόνας αυτός είναι η εύρεση συνδυασμών κατακερματισμού ενός χρηματικού ποσού που δίνεται κατά την ερώτηση.

Η βασική λειτουργία είναι το **backtracking,** δηλαδή η οπισθοδρόμηση και αντιπροσωπεύει πιθανές εναλλακτικές απαντήσεις στην κλήση. Σε περίπτωση αποτυχίας εύρεσης της λύσης ή σε περίπτωση που ο χρήστης ζητά και άλλη λύση, ο μηχανισμός οπισθοδρόμησης επιστρέφει στο τελευταίο σημείο backtracking ακυρώνοντας τα υπολογιστικά βήματα που ακολουθούν για την αναζήτηση άλλης ικανοποιητικής λύσης.

Το αν η αναζήτηση θα γίνει από τις λιγότερες προς τις περισσότερες χρηματικές μονάδες ή το αντίστροφο, καθορίζεται από την σύνταξη του σώματος του κανόνα.

Στον κανόνα **nomismata1(Poso,[A100,A50,A20,A10,A5,A2,A1])**, η prolog ξεκινά τους συνδυασμούς από τις λιγότερες προς τις περισσότερες χρηματικές μονάδες, επιλέγοντας σε κάθε γεγονός member που συναντά από πάνω προς τα κάτω, τιμές που να ξεκινούν από το πρώτο στοιχείο κάθε λίστας και παράλληλα να ικανοποιούν την τιμή Poso. Όταν εξαντληθούν οι τιμές του τελευταίου γεγονότος, αναδρομικά επιλέγεται το δεύτερο στοιχείο του αμέσως προηγούμενου γεγονότος. Η διαδικασία συνεχίζεται μέχρι να επιλεγούν όλοι οι συνδυασμοί που ικανοποιούν την τιμή Poso.

Έστω ότι ζητάω τον κατακερματισμό του 1 ευρώ.

Ερώτηση: ?- nomismata1(1, X).

```
?- nomismata1(1,X).
X = [0, 0, 0, 0, 0, 0, 1] ;
false.

?-
```

Ο μόνος δυνατός συνδυασμός είναι: 1 * A1.

Αντίστοιχα για τα 2 ευρώ.

Ερώτηση: ?- nomismata1(2, X).

```
?- nomismata1(2,X).
X = [0, 0, 0, 0, 0, 0, 2] ;
X = [0, 0, 0, 0, 0, 1, 0] ;
false.

?-
```

Εδώ οι δυνατοί συνδυασμοί είναι 2: 2 * A1 **ή** 1 * A2

Αν κάνω το ίδιο για μεγαλύτερο ποσό οι συνδυασμοί θα αυξάνονται.

Ερώτηση: ?- nomismata1(10, X).

```
?- nomismata1(10,X).
X = [0, 0, 0, 0, 0, 3, 4] ;
X = [0, 0, 0, 0, 0, 4, 2] ;
X = [0, 0, 0, 0, 0, 5, 0] ;
X = [0, 0, 0, 0, 1, 0, 5] ;
X = [0, 0, 0, 0, 1, 1, 3] ;
X = [0, 0, 0, 0, 1, 2, 1] ;
X = [0, 0, 0, 0, 2, 0, 0] ;
X = [0, 0, 0, 1, 0, 0, 0] ;
false.

?-
```

Όπως είδαμε στην αρχή της άσκησης οι συνδυασμοί για την τιμή 355 είναι πολύ περισσότεροι.

```
?- nomismata1(355,X).
X = [0, 1, 8, 12, 2, 5, 5] ;
X = [0, 1, 8, 12, 2, 6, 3] ;
X = [0, 1, 8, 12, 3, 3, 4] ;
X = [0, 1, 8, 12, 3, 4, 2] ;
X = [0, 1, 8, 12, 3, 5, 0] ;
X = [0, 2, 6, 11, 2, 5, 5] ;
X = [0, 2, 6, 11, 2, 6, 3] ;
X = [0, 2, 6, 11, 3, 3, 4] ;
X = [0, 2, 6, 11, 3, 4, 2] ;
X = [0, 2, 6, 11, 3, 5, 0] ;
X = [0, 2, 6, 12, 0, 5, 5] ;
X = [0, 2, 6, 12, 0, 6, 3] ;
X = [0, 2, 6, 12, 1, 3, 4] ;
X = [0, 2, 6, 12, 1, 4, 2] ;
X = [0, 2, 6, 12, 1, 5, 0] ;
X = [0, 2, 6, 12, 2, 0, 5] ;
X = [0, 2, 6, 12, 2, 1, 3] ;
X = [0, 2, 6, 12, 2, 2, 1] ;
X = [0, 2, 6, 12, 3, 0, 0] ;
X = [0, 2, 7, 9, 2, 5, 5] ;
X = [0, 2, 7, 9, 2, 6, 3] ;
X = [0, 2, 7, 9, 3, 3, 4] ;
X = [0, 2, 7, 9, 3, 4, 2] ;
X = [0, 2, 7, 9, 3, 5, 0] ;
X = [0, 2, 7, 10, 0, 5, 5] ;
X = [0, 2, 7, 10, 0, 6, 3] ;
X = [0, 2, 7, 10, 1, 3, 4] ;
X = [0, 2, 7, 10, 1, 4, 2] ;
X = [0, 2, 7, 10, 1, 5, 0] ;
X = [0, 2, 7, 10, 2, 0, 5] ;
```

…

Στον δεύτερο κανόνα **nomismata2(Poso,[A100,A50,A20,A10,A5,A2,A1])**, συμβαίνει κάτι ανάλογο, με την διαφορά ότι εδώ οι συνδυασμοί επιλέγονται ώστε ο κερματισμός της χρηματικής μονάδας να γίνεται από τις περισσότερες προς τις λιγότερες μονάδες. Η λίστα A100 είναι η τελευταία σε αντίθεση με του πρώτου κανόνα που ήταν η πρώτη.

Αυτό φαίνεται στα παρακάτω παραδείγματα:

```
?- nomismata2(1,X).
X = [0, 0, 0, 0, 0, 0, 1] ;
false.

?- nomismata2(2,X).
X = [0, 0, 0, 0, 0, 1, 0] ;
X = [0, 0, 0, 0, 0, 0, 2] ;
false.

?- nomismata2(10,X).
X = [0, 0, 0, 1, 0, 0, 0] ;
X = [0, 0, 0, 0, 2, 0, 0] ;
X = [0, 0, 0, 0, 0, 5, 0] ;
X = [0, 0, 0, 0, 1, 2, 1] ;
X = [0, 0, 0, 0, 0, 4, 2] ;
X = [0, 0, 0, 0, 1, 1, 3] ;
X = [0, 0, 0, 0, 0, 3, 4] ;
X = [0, 0, 0, 0, 1, 0, 5] ;
false.

?-
```

```
?- nomismata2(355,X).
X = [3, 1, 0, 0, 1, 0, 0] ;
X = [2, 1, 5, 0, 1, 0, 0] ;
X = [3, 0, 2, 1, 1, 0, 0] ;
X = [2, 2, 2, 1, 1, 0, 0] ;
X = [2, 0, 7, 1, 1, 0, 0] ;
X = [1, 2, 7, 1, 1, 0, 0] ;
X = [2, 1, 4, 2, 1, 0, 0] ;
X = [3, 0, 1, 3, 1, 0, 0] ;
X = [2, 2, 1, 3, 1, 0, 0] ;
X = [2, 0, 6, 3, 1, 0, 0] ;
X = [1, 2, 6, 3, 1, 0, 0] ;
X = [2, 1, 3, 4, 1, 0, 0] ;
X = [1, 1, 8, 4, 1, 0, 0] ;
X = [3, 0, 0, 5, 1, 0, 0] ;
X = [2, 2, 0, 5, 1, 0, 0] ;
X = [2, 0, 5, 5, 1, 0, 0] ;
X = [1, 2, 5, 5, 1, 0, 0] ;
X = [2, 1, 2, 6, 1, 0, 0] ;
X = [1, 1, 7, 6, 1, 0, 0] ;
X = [2, 0, 4, 7, 1, 0, 0] ;
X = [1, 2, 4, 7, 1, 0, 0] ;
X = [2, 1, 1, 8, 1, 0, 0] ;
X = [1, 1, 6, 8, 1, 0, 0] ;
X = [2, 0, 3, 9, 1, 0, 0] ;
X = [1, 2, 3, 9, 1, 0, 0] ; ...
```

Θέμα 5ο. Β΄Εκπληξη

Η παράγωγος μιας συνάρτησης εκφράζει την μεταβολή της τιμής της συνάρτησης (μια συνάρτηση ή εξαρτημένη μεταβλητή) η οποία προσδιορίζεται από μια άλλη ποσότητα (ανεξάρτητη μεταβλητή).

Η παράγωγος υπολογίζεται από τον κανόνα:

differentiate(F, SDF) :- diff(F, DF), simpl(DF, SDF).

Στο πρώτο μέρος η παράγωγος της συνάρτησης F υπολογίζεται από τους κανόνες της μορφής:

diff(F, DF)

και στο δεύτερο μέρος απλοποιείται σε απλούστερη μορφή με τους κανόνες:

simpl(DF, SDF).

Οι παράγωγοι έχουν ως εξής:

1. diff(C, 0) :- integer(C). (c)' = 0

2. diff(x, 1). (x)' = 1

3. diff(F+G, DF+DG) :-
   diff(F, DF), diff(G, DG). (f(x) + g(x))' = f'(x) + g'(x)

4. diff(-F, -DF) :- diff(F, DF). (-1 * f(x))' = -1 * f '(x)

5. diff(F-G, DF-DG) :- diff(F, DF), diff(G, DG). (f(x) - g(x))' = f'(x) - g'(x)

6. diff(C*F, C*DF) :- integer(C), !, diff(F, DF). (c * f(x))' = c * f '(x)

7. diff(F*G, F*DG+G*DF) :-
   diff(F, DF), diff(G, DG). (f(x) * g(x))' = f '(x) * g(x) + f(x) * g'(x)

8. diff(F//G, (G*DFF*DG)/G^2) :- (f(x) / g(x))' =
   diff(F, DF), diff(G, DG). (f '(x) * g(x) + f(x) * g'(x)) / $(g(x))^2$

9. diff(F^C, C*F^(C-1)*DF)
:- integer(C), diff(F, DF). ([f(x)] v)' = v [f(x)] $^{v-1}$ * f '(x)

10. diff(sin(F), cos(F)*DF) :- diff(F, DF). (ημf(x))' = συνf(x) * f '(x)

11. diff(cos(F), -sin(F)*DF) :- diff(F, DF). (συνf(x))' = - ημf(x) * f '(x)

12. diff(log(F), (1/F)*DF) :- diff(F, DF). (lnf(x))' = (1/f (x)) * f '(x), f(x) > 0

13. diff(exp(F), exp(F)*DF) :- diff(F, DF). ($e^{f(x)}$)' = ($e^{f(x)}$) * f '(x)

Η diff επιστρέφει την παράγωγο της συνάρτησης χωρίς απλοποίηση:

```
?- diff(x^3+x^(2),R).
R = 3*x^(3-1)*1+2*x^(2-1)*1.

?-
```

Για την απλοποίηση των παραπάνω παραγώγων χρησιμοποιούμε τους κανόνες της μορφής :

simpl(F, SF) :- simplify(F, SF1), !, simpl(SF1, SF). (αναδρομή)

simpl(F, F). (τερματική συνθήκη)

Η simpl(F, SF) καλείται πρώτη και απλοποιεί αναδρομικά στο μεγαλύτερο δυνατό βαθμό την συνάρτηση F. Όταν αποτύχει η simpl(SF1, SF) διακόπτεται η οπισθοδρόμηση (!). Στο τέλος καλείται η simpl(F, F) όπου επιστρέφει την συνάρτηση F στην τελική απλοποιημένη της μορφή (SF) και τερματίζει την simpl. Δεν υπάρχει άλλη απλοποίηση για την F και τα ενδιάμεσα στάδια της απλοποίησης παραλείπονται, γι' αυτό η τερματική συνθήκη είναι μετά τον κανόνα της αναδρομής.

Αυτό φαίνεται στο παρακάτω παράδειγμα :

```
?- differentiate(x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x)), R).
R = 4*x^3+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x).

?-
```

Αν γράψω πρώτα την τερματική συνθήκη και μετά τον αναδρομικό κανόνα στο ίδιο παράδειγμα θα έχω το εξής αποτέλεσμα :

simpl(F, F). (τερματική συνθήκη)

simpl(F, SF) :- simplify(F, SF1), !, simpl(SF1, SF). (αναδρομή)

```
?- differentiate(x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x)), R).
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+3*(-sin(exp(2*x))*(exp(2*x)*(2*1))) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+3*(-sin(exp(2*x))*(exp(2*x)*2)) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+3*(-sin(exp(2*x))*exp(2*x)*2) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+3*(-sin(exp(2*x))*exp(2*x))*2 ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+6*(-sin(exp(2*x))*exp(2*x)) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1))+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^1)+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x)+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*2*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^(3-1))+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*(3*x^2)+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+2*3*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)*1+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^(4-1)+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
R = 4*x^3+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x) ;
false.

?-
```

Θα μου επιστρέφει δηλαδή και τα ενδιάμεσα στάδια της απλοποίησης που είναι μη επιθυμητό αποτέλεσμα.

1. **simplify(A+B, C) :- integer(A), integer(B),!, C is A+B.**

Το άθροισμα C δύο ακεραίων A, B είναι C = A+B.

2. **simplify(A-B, C) :- integer(A), integer(B),!, C is A-B.**

Η διαφορά C δύο ακεραίων A, B είναι C = A-B.

3. **simplify(A*B, C) :- integer(A), integer(B),!, C is A*B.**

Το γινόμενο C δύο ακεραίων A, B είναι C = A*B.

4. **simplify(0+F, F) :- !.**

   simplify(F+0, F) :- !.

Μια συνάρτηση αθροιζόμενη με το 0 δίνει τον εαυτό της.

5. **simplify(1*F, F) :- !.**

   simplify(F*1, F) :- !.

Μια συνάρτηση πολλαπλασιαζόμενη με 1 δίνει τον εαυτό της.

6. **simplify(0*_, F) :- !.**

   simplify(_*0, F) :- !.

Μια οποιαδήποτε οντότητα πολλαπλασιαζόμενη με 0 δίνει 0.

7. **simplify(F ^1, F) :- !.**

Μια συνάρτηση υψωμένη στην μονάδα δίνει τον εαυτό της.

8. **simplify(-F, -SF) :- simplify(F, SF).**

Για την απλοποίηση μιας συνάρτησης -F αρκεί να απλοποιηθεί η συνάρτηση F.

9. **simplify(F+G, F+SG) :- simplify(G, SG).**

simplify(F+G, SF+G) :- simplify(F, SF).

Για την απλοποίηση μιας συνάρτησης F αθροιζόμενη με μια συνάρτηση G αρκεί να απλοποιηθούν οι συναρτήσεις G και F.

Όμοια για την περίπτωση της αφαίρεσης, του πολλαπλασιασμού, της διαίρεσης και της ύψωσης σε δύναμη.

simplify(F-G, F-SG) :- simplify(G, SG).

simplify(F-G, SF-G) :- simplify(F, SF).

simplify(F*G, F*SG) :- simplify(G, SG).

simplify(F*G, SF*G) :- simplify(F, SF).

simplify(F//G, F//SG) :- simplify(G, SG).

simplify(F//G, SF//G) :- simplify(F, SF).

simplify(F^G, F^SG) :- simplify(G, SG).

simplify(F^G, SF^G) :- simplify(F, SF).

Σε όλες τις παραπάνω περιπτώσεις ορίζονται δύο κανόνες αντί για έναν. Αυτό συμβαίνει έτσι ώστε σε περίπτωση που αποτύχει η απλοποίηση μιας συνάρτησης να γίνει απλοποίηση στην άλλη συνάρτηση.

10. **simplify(F*(G*H), (F*G)*H).**

Προσεταιριστική ιδιότητα.

Η λειτουργικότητα της simpl φαίνεται στο παρακάτω παράδειγμα :

Με χρήση κανόνων απλοποίησης.

```
?- differentiate((x^4), R).
R = 4*x^3.

?- differentiate(x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x)), R).
R = 4*x^3+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x).

?-
```

Χωρίς χρήση κανόνων απλοποίησης στο ίδιο παράδειγμα.

```
?- differentiate((x^4), R).
R = 4*x^(4-1)*1.

?- differentiate(x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x)), R).
R = 4*x^(4-1)*1+2*(3*x^(... - ...)*1)+3*(2*x^(2-1)*1)+1+cos(2*x)*(2*1)+3*(-sin(exp(2*x))*(exp(2*x)*(2*
1))).

?-
```

Παραδείγματα παραγώγων:

```
?- differentiate(x^4, R).
R = 4*x^3.

?- differentiate(x^4+2*x^3+3*x^2, R).
R = 4*x^3+6*x^2+6*x.

?- differentiate(x^4+2*x^3+3*x^2+x+sin(2*x), R).
R = 4*x^3+6*x^2+6*x+1+cos(2*x)*2.

?- differentiate(x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x)), R).
R = 4*x^3+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x).

?- differentiate(log(exp(2*x)), R).
R = 1/exp(2*x)*exp(2*x)*2.

?-
```

Το παράδειγμα της άσκησης μπορούμε να το εκτυπώσουμε πιο κομψά με τον εξής κανόνα :

```
diff :-

        F=x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x)),

        write("F= "), write(F),nl, differentiate(F, R), write("R= "), write(R), nl.
```

```
?- diff.
F= x^4+2*x^3+3*x^2+x+sin(2*x)+3*cos(exp(2*x))
R= 4*x^3+6*x^2+6*x+1+cos(2*x)*2+6* -sin(exp(2*x))*exp(2*x)
true.

?-
```