8. ΕΦΑΡΜΟΓΕΣ

8.1. Γεωγραφικά Διαμερίσματα

Να γραφεί πρόγραμμα σε PROLOG για την αναπαράσταση πληροφοριών που σχετίζονται με το γεωγραφικό διαμέρισμα της Μακεδονίας. Οι πληροφορίες αυτές θα πρέπει να αναφέρονται:

α. Στον αριθμό των νομών της Μακεδονίας.
β. Στις πρωτεύουσες των νομών.
γ. Στο εμβαδόν του κάθε νομού.
δ. Στον Πληθυσμό του κάθε νομού.
ε. κ.ο.κ.

Να σχηματιστούν οι παρακάτω ερωτήσεις:

1. Είναι η Θεσσαλονίκη νομός της Μακεδονίας;
2. Ποιοί είναι οι νομοί της Μακεδονίας;
3. Ποιά είναι η πρωτεύουσα του νομού Χαλκιδικής;
4. Ποιοί είναι οι νομοί της Μακεδονίας με τις αντίστοιχες πρωτεύουσες
5. Ποιός νομός της Μακεδονίας έχει πληθυσμό 240.000 κατ.;

.....

6. Ποιά είναι η πυκνότητα πληθυσμού στο νομό Καβάλας;

```
part_of(makedonia, n_thessalonikis).
part_of(makedonia, n_pierias).
part_of(makedonia, n_imathias).
part_of(makedonia, n_kilkis).
part_of(makedonia, n_kavalas).
part_of(makedonia, n_serres).
part_of(makedonia, n_kastorias).
part_of(makedonia, n_florinas).
```

```
part_of(makedonia, n_kozanis).
part_of(makedonia, n_pellis).
part_of(makedonia, n_xalkidikis).
part_of(makedonia, n_dramas).

capital_of(n_thessalonikis, thessaloniki).
capital_of(n_pierias, katerini).
capital_of(n_imathias, veria).
capital_of(n_kilkis, kilkis).
capital_of(n_kavalas, kavala).
capital_of(n_serres, serres).
capital_of(n_kastorias, kastoria).
capital_of(n_florinas, florina).
capital_of(n_kozanis, kozani).
capital_of(n_pellis, edessa).
capital_of(n_xalkidikis, polygiros).
capital_of(n_dramas, drama).

population_of(n_thessalonikis, 1500).    /* σε χιλιάδες */
population_of(n_pierias, 200).
population_of(n_imathias, 210).
population_of(n_kilkis, 195).
population_of(n_kavalas, 220).
population_of(n_serres, 240).
population_of(n_kastorias, 160).
population_of(n_florinas, 125).
population_of(n_kozanis, 135).
population_of(n_pellis, 140).
population_of(n_xalkidikis, 190).
population_of(n_dramas, 165).

area_of(n_thessalonikis, 3000). /* σε τετραγωνικά χιλιόμετρα */
area_of(n_pierias, 3100).
area_of(n_imathias, 2800).
area_of(n_kilkis, 2700).
```

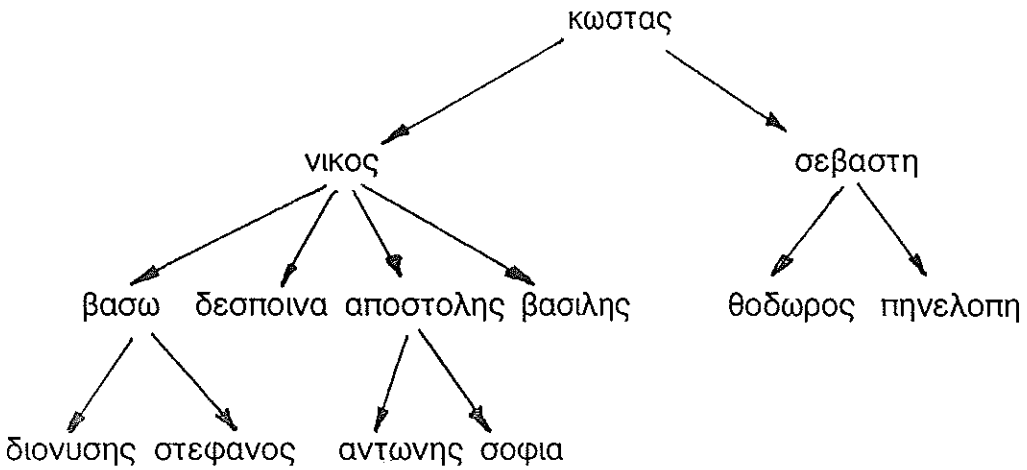
```
area_of(n_kavalas, 2750).
area_of(n_serres, 3050).
area_of(n_kastorias, 2600).
area_of(n_florinas, 2500).
area_of(n_kozanis, 2550).
area_of(n_pellis, 2450).
area_of(n_xalkidikis, 2850).
area_of(n_dramas, 2750).

density_of(X,Y) :-
    population_of(X,P),
    area_of(X,E),
    Y is (P/E)*1000 .
```

/* Ερωτήσεις */

```
/* 1 */   ?- part_of(makedonia,n_thessaloniki).
/* 2 */   ?- part_of(makedonia,N).
/* 3 */   ?- capital_of(n_xalkidikis,C).
/* 4 */   ?- part_of(makedonia,N),capital_of(N,C).
/* 5 */   ?- population_of(N, 240), part_of(makedonia,N).
/* 6 */   ?- density(n_kavalas,D).
```

8.2. Γενεολογικό Δέντρο

Θεωρήστε το παρακάτω οικογενειακό δέντρο:



A. Αν το σύμβολο **-->** ερμηνεύεται σαν η σχέση **parent**, δημιουργήστε τα αντίστοιχα γεγονότα στην PROLOG που να περιγράφουν το παραπάνω δέντρο και στη συνέχεια ορίστε τη σχέση "**grandparent(X,Y)**" (ο/η **X** είναι παππούς/γιαγιά του/της **Y**). Στη συνέχεια σχηματίστε τις παρακάτω ερωτήσεις:

(α) ποιός είναι ο γονιός της πηνελοπης;

(β) εχει παιδιά ο βασιλης;

(γ) Ποιός είναι ο παππούς της σοφιας;

B. Επαυξήστε το πρόγραμμα σας με γεγονότα της μορφής: "**male(X)**" (ο **X** είναι γενους αρσενικού), "**female(X)**" (Η **X** είναι γένους θηλυκού) και χρησιμοποιήστετες για να ορίσετε τις σχέσεις "**father(X,Y)**", "**mother(X,Y)**" και "**sister(X,Y)**" (Η **X** είναι αδελφή του/της **Y**).

Προσοχή !!!

Τι απαντά το πρόγραμμα σας στην ερώτηση: **?- sister(σοφια, σοφια)** και γιατί;

/* **parent(X,Y)** : Ο/Η **X** είναι γονιός του/της **Y** */

```
parent(kostas,nikos).
parent(kostas,sevasti).
parent(sevasti,thodoros).
parent(sevasti,pinelopi).
parent(nikos,vasso).
parent(nikos,despina).
parent(nikos,apostolis).
parent(nikos,vassilis).
parent(vasso,dionisis).
parent(vasso,stefanos).
parent(apostolis,antonis).
parent(apostolis,sofia).

/* grandparent(X,Y) : Ο/Η X είναι παππούς/γιαγιά του/της Y */

grandparent(X,Y) :-
        parent(X,Z),
        parent(Z,Y).

/* Ερωτήσεις */

/* 1(α) */   ?- parent(X,pinelopi).
/* 1(β) */   ?- parent(vassilis, _).
/* 1(γ) */   ?- grandparent(X,sofia).

/* female(X) : Η X είναι γένους θηλυκού */

female(sevasti).
female(pinelopi).
female(vasso).
female(despina).
female(sofia).

/* male(X) : Ο X είναι γένους αρσενικού */
```

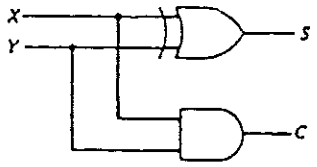
```
male(kostas).
male(nikos).
male(apostolis).
male(dionisis).
male(stefanos).
male(vassilis).
male(thodoros).

/* father(X,Y) : Ο X είναι πατέρας του/της Y. */

father(X,Y) :-
    parent(X,Y),
    male(X).

/* mother(X,Y) : Η X είναι μητέρα του/της Y. */

mother(X,Y) :-
    parent(X,Y),
    female(X).

/* sister(X,Y) : Η X είναι αδελφή του/της Y. */

sister(X,Y) :-
    parent(Z,X),
    parent(Z,Y),
    female(X).
```

8.3. Ψηφιακά Κυκλώματα

A. Ορίστε με τη βοήθεια φράσεων της PROLOG την (ψηφιακή) λειτουργία των λογικών πυλών **AND**, **OR**, **NOT**, **NAND**, **NOR** και **XOR**. Στη συνέχεια με τη βοήθεια των πυλών αυτών ορίστε ένα **μερικό αθροιστή** ***(half adder)*** και έναν **πλήρη αθροιστή** ***(full adder).***

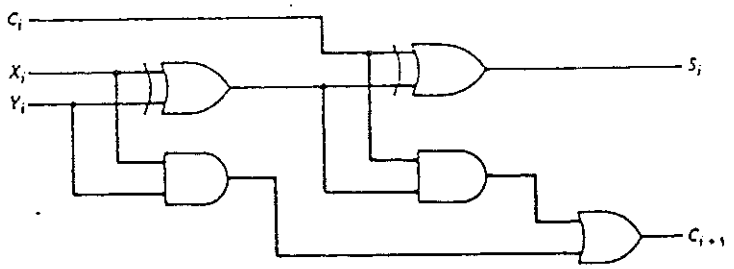
Δίνονται τα ψηφιακά κυκλώματα για το μερικό και τον πλήρη αθροιστή, καθώς και οι αντίστοιχοι πίνακες αληθείας για να γίνει σύγκριση αποτελεσμάτων.

X	Y	C	S
0	0	0	0
0	1	0	1
1	1	1	0
1	0	0	1



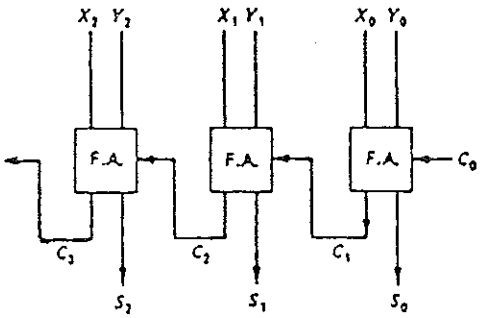
μερικός αθροιστής

X_i	Y_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	1	1	0
0	1	0	0	1
1	1	0	1	0
1	1	1	1	1
1	0	1	1	0
1	0	0	0	1



πλήρης αθροιστής

B. Με τη βοήθεια του πλήρη αθροιστή ορίστε έναν αθροιστή για τριψήφιους δυαδικούς αριθμούς.



Αθροιστής 3 bits

/* **and(X,Y,Z)** : Η πύλη **and** με εσόδους **X** και **Y** δίνει έξοδο **Z**. */

```
and(0,0,0).
and(0,1,0).
and(1,0,0).
and(1,1,1).
```

/* **or(X,Y,Z)** : Η πύλη **or** με εσόδους **X** και **Y** δίνει έξοδο **Z**. */

```
or(0,0,0).
or(0,1,1).
or(1,0,1).
or(1,1,1).
```

/* **not(X,Y)** : Η πύλη **not** με εσόδο **X** δίνει έξοδο **Y**. */

```
not(0,1).
not(1,0).
```

/* **nand(X,Y,Z)** : Η πύλη **nand** με εσόδους **X** και **Y** δίνει έξοδο **Z**. */

```
nand(0,0,1).
nand(0,1,1).
nand(1,0,1).
nand(1,1,0).
```

/* **nor(X,Y,Z)** : Η πύλη **nor** με εσόδους **X** και **Y** δίνει έξοδο **Z**. */

```
nor(0,0,1).
nor(0,1,0).
nor(1,0,0).
nor(1,1,0).
```

/* **xor(X,Y,Z)** : Η πύλη **xor** με εσόδους **X** και **Y** δίνει έξοδο **Z**. */

```
xor(0,0,0).
xor(0,1,1).
xor(1,0,1).
xor(1,1,0).

/*  half_adder(X,Y,S,C) : Ο μερικός αθροιστής half_adder          */
/*  με εισόδους τα ψηφία X και Y παράγει                          */
/*  σαν εξόδουςτο ψηφίο S και το κρατούμενο C.                    */

half_adder(X,Y,S,C) :-
                xor(X,Y,S),
                and(X,Y,C).

/*  full_adder(Co,X,Y,S,Cn) : Ο πλήρης αθροιστής full_adder       */
/*  με εισόδους το κρατούμενο Co και τα ψηφία X και Y             */
/*  παράγει σαν εξόδους το ψηφίο S και το νέο κρατούμενο Cn.      */

full_adder(Co,X,Y,S,Cn) :-
                half_adder(X,Y,Z,T),
                half_adder(Co,Z,S,L),
                or(T,L,Cn).

/*   adder_3_bits(X2,X1,X0,Y2,Y1,Y0,Co,S2,S1,S0,Cn) :             */
/*  Ο αθροιστής adder_3_bits με εισόδους τους τριψήφιους         */
/*  δυαδικούς αριθμούς X2,X1,X0 και Y2,Y1,Y0 και το κρατούμενο   */
/*  Co παράγει σαν αποτέλεσμα τον αριθμό S2,S1,S0 και νέο        */
/*  κρατούμενο το Cn          */

adder_3_bits(X2,X1,X0,Y2,Y1,Y0,Co,S2,S1,S0,Cn) :-
                full_adder(Co,X0,Y0,S0,C1),
                full_adder(C1,X1,Y1,S1,C2),
                full_adder(C2,X2,Y2,S2,Cn).
```

8.4. Οι Πύργοι του Hanoi

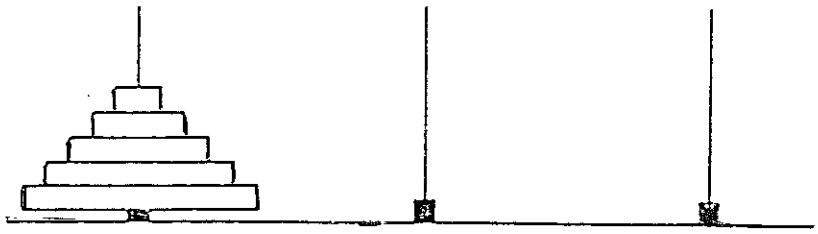
Το πρόβλημα των πύργων του Hanoi, αποτελεί ένα χαρακτηριστικό παράδειγμα, ενός προβλήματος που ενώ φαίνεται πολύπλοκο, η λύση του μπορεί να περιγραφεί εύκολα με τη χρήση ενός αναδρομικού αλγόριθμου. Δίνονται τρείς κατακόρυφοι άξονες και Ν ξύλινοι δίσκοι με διαφορετικές διαστάσεις. Οι δίσκοι τοποθετούνται αρχικά στον αριστερό άξονα, ώστε να σχηματίσουν ένα πύργο (σε φθίνουσα τάξη ώς προς τη διαμετρό τους). Το πρόβλημα συνίσταται στη μετακίνηση των Ν δίσκων από τον αριστερό άξονα στο δεξιό με βάση τους παρακάτω κανόνες:

(1) μόνο ένας δίσκος μπορεί να μετακινηθεί κάθε φορά

(2) ένας δίσκος δε μπορεί να τοποθετηθεί πάνω σε άλλο μικρότερης διαμέτρου

(3) ο μεσαίος άξονας μπορεί να χρησιμοποιηθεί σα βοηθητικός

(4) σε οποιαδήποτε χρονική στιγμή ο κάθε δίσκος πρέπει να βρίσκεται σε έναν από τους τρεις άξονες.



Οι πύργοι του Hanoi (για N = 5)

Μπορούμε να θεωρήσουμε ότι ένας πύργος που αποτελείται από **N** δίσκους μπορεί να αποσυντεθεί σε δύο πύργους, ένα πύργο που αποτελείται από ένα μόνο δίσκο και έναν πύργο που αποτελείται απο τους υπόλοιπους **N-1** δίσκους. Με βάση τη θεώρηση αυτή, το πρόβλημα της μετακίνησης **N** δίσκων (**N>1**) μπορεί να αναχθεί αναδρομικά στη μετακίνηση **N-1** δίσκων ώς εξής (βλέπε procedure **move**):

(α) μετακίνησε τους επάνω N-1 δίσκους από τον αριστερό στο μεσαίο άξονα χρησιμοποιώντας σαν βοηθητικό το δεξιό άξονα

(β) μετακίνησε το δίσκο που έμεινε από τον αριστερό στο δεξιό άξονα

(γ) μετακίνησε τους Ν-1 δίσκους από το μεσαίο άξονα στο δεξιό άξονα χρησιμοποιώντας σα βοηθητικό τον αριστερό άξονα

Η μετακίνηση των **Ν-1** δίσκων στα βήματα **(α)** και **(γ)** μπορεί να αναχθεί στη μετακίνηση των **Ν-2** δίσκων κ.ο.κ.

```
/* H1 */    hanoi(N):-
                move(N,left,center,right).

/* M1 */    move(0,_,_,_).
/* M2 */    move(N,Pole1,Pole2,Pole3) :-
                M is N-1,
                move(M,Pole1,Pole3,Pole2),
                move_one(Pole1,Pole3),
                move(M,Pole2,Pole1,Pole3).

/* I1 */    move_one(X,Y) :-
                write('move a disk from'),
                write(X), write(' to '),
                write(Y), nl.
```

```
/* M1 */    move(0,X,_,Y) :-
                write('move a disk from'),
                write(X), write(' to '),
                write(Y), nl.

/* M2 */    move(N,Pole1,Pole2,Pole3) :-
                M is N-1,
                move(M,Pole1,Pole3,Pole2),
                move(1,Pole1,Pole3),
                move(M,Pole2,Pole1,_,Pole3).
```

8.5. Διάταξη των Στοιχείων μιας Λίστας.

8.5.1. Απλή Διάταξη

Το κατηγόρημα **insert(E,L,NL)**, τοποθετεί το στοιχείο **E** στη λίστα **L**, με την προϋπόθεση ότι αυτή είναι διατεταγμένη σε αύξουσα τάξη και δημιουργεί την επίσης διατεταγμένη λίστα **NL**.

Το κατηγόρημα **sort(L,S)**, το οποίο μετατρέπει μία οποιαδήποτε λίστα **L** στη λίστα **S** της οποίας τα στοιχεία είναι διατεταγμένα σε αύξουσα τάξη.

```
/* S1 */    sort([X|Y],Z) :- sort(Y,T), insert(X,T,Z).
/* S2 */    sort([ ],[ ]).
```

8.5.2. Γρήγορη διάταξη

Το κατηγόρημα **partition(P,L,A,D)**, δοθέντος ενός στοιχείου **P (στοιχείο οδηγός)**, διασπά την λίστα **L** σε δύο λίστες **A** και **D** τέτοιες ώστε η λίστα **A** να περιέχει τα στοιχεία της **L** που είναι μικρότερα του **P** και η λίστα **D** τα στοιχεία της **L** που είναι μεγαλύτερα του **P**.

Το κατηγόρημα **quicksort(L,S)**, μετατρέπει μία οποιαδήποτε λίστα **L** στη λίστα **S** της οποίας τα στοιχεία είναι διατεταγμένα σε αύξουσα τάξη, με βάση τον αλγόριθμο της γρήγορης διάταξης (με οδηγό το πρώτο στοιχείο της **L**).

```
/* Q1 */    quicksort([ ],[ ]).

/* Q2 */    quicksort([Head|Tail],Sorted) :-
                partition(Head,Tail,Left,Right),
                quicksort(Left,Lsorted),
                quicksort(Right,Rsorted),
                append(Lsorted,[Head|Rsorted],Sorted).
```

```
/* P1 */    partition(H,[A|X],[A|Y],Z)  :-
                A =< H, partition(H,X,Y,Z).
/* P2 */    partition(H,[A|X],Y,[A|Z])  :-
                A > H, partition(H,X,Y,Z).
/* P2 */    partition(_,[ ],[ ],[ ]).
```

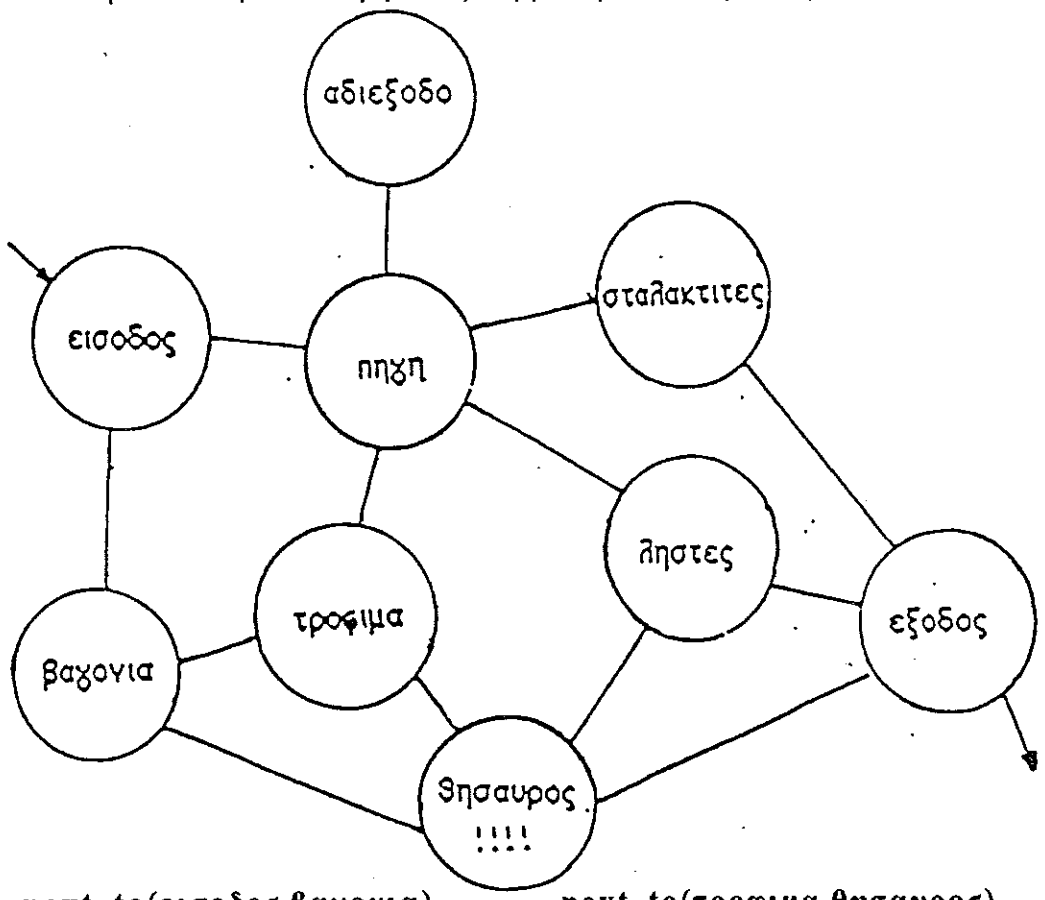
```
B1: < ( ?- sort([2,1],S). ),
          (sort([X|Y],Z) :- sort(Y,T), insert(X,T,Z)),
          { X=2, Y=[1], S=Z } >

B2: < ( ?- sort([1],T), insert(2,T,Z). ),
          ( sort([X1|Y1],Z1):- sort(Y1,T1), insert(X1,T1,Z1) ),
          { X1=1, Y1=[], T=Z1 } >

B3: < ( ?- sort([],T1), insert(1,T1,Z1), insert(2,Z1,Z). ),
          sort([],[]),
          { T1=[] } >

B4: < ( ?- insert(1,[],Z1), insert(2,Z1,Z). ),
      insert(X2,[],[X2]),
          { Z1=[X2], X2=1 } >

B5: < ?- insert(2,[1],Z). ,
    ( insert(X3,[Y3|Z3],[Y3|T3]):- X3>Y3, insert(X3,Z3,T3) ),
    { X3=2, Y3=1, Z3=[], Z=[1|T3] } >

B6: < ( ?- 2>1, insert(2,[],T3). ),
      2>1,
      { } >

B7: < ( ?- insert(2,[],T3). ),
      insert(X4,[],X4),
      { X4=2, T3=[2] } >

B8: < ?- true, -, { } >
```

Σχήμα : Ο υπολογισμός της ερώτησης ?- sort([2,1],S).

8.6. Το κυνήγι του Θησαυρού!!!

Να βρεθεί μία διαδρομή στο γράφημα (σπηλιά) του παρακάτω σχήματος τέτοια ώστε:

(α) να αρχίζει από τον κόμβο **είσοδο** και να τελειώνει στον κόμβο **έξοδο**
(β) να περνάει υποχρεωτικά από τον κόμβο **θησαυρός**
(γ) να μην περνάει από τους κόμβους **ληστές** και **αδιέξο**
(δ) δεν επιτρέπεται η επίσκεψη ενός κόμβου για δεύτερη φορά.

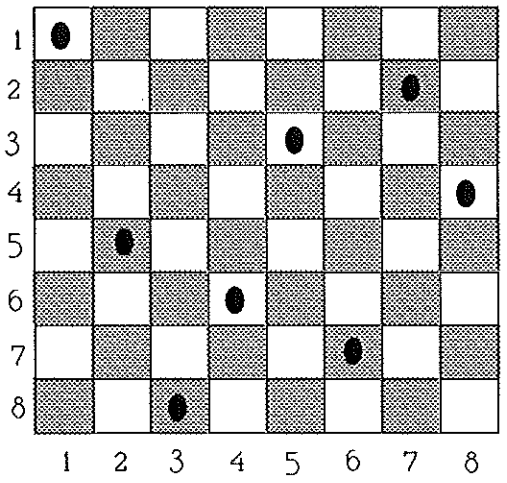



```
next_to(εισοδος,βαγονια).
next_to(εισοδος,πηγη).
next_to(πηγη,αδιεξοδο).
next_to(πηγη,ληστες).
next_to(πηγη,τροφιμα).
next_to(πηγη,σταλακτιτες).
next_to(ληστες,εξοδος).
next_to(τροφιμα,θησαυρος).
next_to(βαγονια,τροφιμα).
next_to(βαγονια,θησαυρος).
next_to(σταλακτιτες,εξοδος).
next_to(θησαυρος,εξοδος).
next_to(ληστες,θησαυρος).
```

```
avoid([αδιεξοδο,ληστες]).

travesrse(Place1,Place2) :-
        avoid(Dangers),
        path(Place1,Place2,Dangers,[Place1]).

path(Place1,Place2,Dangers,Memory) :-
        member(Place2,Memory),
        member(θησαυρος,Memory),
        reverse_write(Memory).

path(Place1,Place2,Dangers,Memory) :-
        joint_with(Place1,NewPlace),
        not member(NewPlace,Dangers),
        not member(NewPlace,Memory),
        path(NewPlace,Place2,Dangers,[NewPlace|Memory]).

joint_with(Place1,Place2) :-
        next_to(Place1,Place2).

joint_with(Place1,Place2) :-
        next_to(Place2,Place1).

reverse_write([ ]).
reverse_write([Head|Tail]) :-
        reverse_write(Tail),
        nl,
        write(Head).

    member(X,[X|Y]).
    member(X,[Y|Z]) :- member(X,Z).
```

8.7. Το πρόβλημα των 8 βασιλισσών

Να τοποθετηθούν οκτώ βασίλισσες σε μία σκακιέρα, έτσι ώστε καμμία από αυτές να μην απειλείται από τις υπόλοιπες επτά.



Μία λύση στο πρόβλημα των 8 βασιλισσών

```
solve(X) :-
        template(X),
        solution(X).

    template([[1,_],[2,_],[3,_],[4,_],[5,_],[6,_],[7,_],[8,_]]).

    solution([ ]).
    solution([[X,Y] | Rest]) :-
        solution(Rest),
        member(Y,[1,2,3,4,5,6,7,8]),
        noattack([X,Y],Rest).

    noattack(_,[ ]).
    noattack([X,Y],[[X1,Y1]|Rest]) :-
            Y =\= Y1,
```

```
        Y1-Y  =\=  X1-X,
        Y1-Y  =\=  X-X1,
        noattack([X,Y],Rest).

member(X,[X|_]).
member(X,[_|Y])  :-  member(X,Y).
```

8.8. Ενα πρόγραμμα που μαθαίνει !!!

Να γραφεί ένα πρόγραμμα που να υλοποιεί ένα διάλογο με το χρήστη, σαν αυτόν που δίνεται παρακάτω. Παρατηρήστε οτι το πρόγραμμα έχει τη δυνατότητα να μαθαίνει πρωτεύουσες κρατών. Την καινούργια αυτή γνώση πρέπει στο τέλος της εκτέλεσης του να την αποθηκεύει με μόνιμο τρόπο. (Η πληροφορία που πληκτρολογείται από το χρήστη είναι με έντονα γράμματα.)

```
?- consult(learning).

Country? greece.
The capital of greece is athens

Country? france.
The capital of france is paris

Country? england.
Sorry! I do not know the capital of england
Please tell me
capital? london.
Thank you

Country? italy.
The capital of italy is rome

Country? england.
The capital of england is london

Country? byebye.
Saving the knowledge base . . .
Done

?
```

```
capital(greece,athens).   /* Οι φράσεις αυτές πρέπει          */
capital(france,paris).    /* να τοποθετηθούν αρχικά           */
capital(italy,rome).      /* στο αρχείο kn_base               */

dialog :- write('Country? '),
          read(Country),
          find_capital(Country).

find_capital(byebye) :-
          write('Saving the knowledge base. . .'),
          my_save(kn_base),
          nl, write('Done').

find_capital(Country) :-
          capital(Country,Capital),
          write('The capital of '),
          write(Country), write(' is '),
          write(Capital), nl,nl,
          dialog.

find_capital(Country) :-
          write('Sorry! I don' t know the capital of '),
          write(Country), nl,
          learn_capital(Country),
          dialog.

learn_capital(Country) :-
          write('Please tell me'), nl,
          write('Capital? '),
          read(Capital),
          assert(capital(Country,Capital)),
          nl, nl.

my_save(F) :- tell(F), listing(capital), told.

start :-  reconsult(kn_base), dialog.

:- start.
```

8.9 Υλοποίηση των κατηγορημάτων consult(File) και reconsult(File)

```
my reconsult(File) :-
        see(File),
        make_list(Plist),
        free_db(Plist),
        seen,
        my_consult(File).

make_list(Plist) :-
        read(Clause),
        process(Clause,[],Plist).

process(end_of_file, Plist,Plist) :- !, seen.
process(Clause, Sofar,Plist) :-
        take_head(Clause,Head),
        functor(Head,Pred,N),
        read(New_Clause),
        process(New_Clause,[Pred/N|Sofar],Plist).

take_head((H:-B),H) :- !.
take_head(H,H).

free_db([]).
free_db([P/N | Rest]) :-
        functor(H,P,N),
        retract_all(H),
        free_db(Rest).

retract_all(H) :-
        clause(H,B),
        retract((H:-B)),
        fail.
```

```
retract_all(H) :-
        clause(H,true),
        retract(H),
        fail.

retract_all(H).
```

ΑΣΚΗΣΗ 1

(α) Να γραφεί πρόγραμμα στην PROLOG για το παρακάτω πρόβλημα:

Μία εταιρεία που αποτελείται από 12 τμήματα, πρόκειται να μετακομίσει σε ένα καινούργιο κτίριο με τρείς ορόφους. Κάθε όροφος διαθέτει 40 γραφεία. Στον παρακάτω πίνακα δίνονται οι ανάγκες των τμημάτων της εταιρείας σε γραφεία:

Τμήμα :	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Γραφεία:	9	6	11	13	9	12	11	11	7	10	11	10

Επιπλέον τα παρακάτω ζευγάρια τμημάτων πρέπει να βρίσκονται στον ίδιο όροφο, γιατί οι εργασίες τους σχετίζονται σε μεγάλο βαθμό: (T1 και T3), (T2 και T4), (T8 και T9). Να βρεθεί η κατανομή των γραφείων στα τμήματα.

Υποδείξεις:

α. Η λύση μπορεί να δημιουργηθεί σαν μία λίστα από τριάδες **tr(D, N, F)**, όπου **D** είναι το τμήμα, **N** είναι ο αριθμός των γραφείων που χρειάζεται και **F** είναι ο όροφος που θα δοθεί στο τμήμα. Στο παραπάνω πρόβλημα τα **D** και **N** είναι γνωστά και το **F** πρέπει να υπολογιστεί.

β. Να οριστεί ένα κατηγόρημα **allocate(L,No1,No2,No3)**, όπου **L** είναι μία λίστα από τριάδες που αντιστοιχούν στα τμήματα και **No1**, **No2**, **No3** είναι αντίστοιχα, ο αριθμός των ελεύθερων γραφείων κάθε ορόφου.

Η ερώτηση **?- allocate(L,40,40,40).** λύνει το πρόβλημα επιστρέφοντας στη μεταβλητή **L** τις αντίστοιχες τριάδες.

(β) Δημιουργήστε μία παραλλαγή (γενίκευση) του παραπάνω προβλήματος και δώστε τη λύση

ΑΣΚΗΣΗ 2

(α) Να περιγραφεί με τη βοήθεια της PROLOG, ένα τμήμα του οδικού χάρτη της πόλης Θεσσαλονίκης. Στη συνέχεια να γραφεί πρόγραμμα το οποίο να δίνει τη (τις) διαδρομή (-μες) που πρέπει να ακολουθήσει κανείς (με αυτοκίνητο) για να φθάσει από ένα σημείο της πόλης σε ένα άλλο.

(β) Εάν σε κάθε δρόμο αντιστοιχεί ένας παράγοντας που σχετίζεται με την καθυστέρηση λόγω κυκλοφοριακής συμφόρισης, να μετατραπεί το πρόγραμμα της (α), έτσι ώστε να δίνει τη βέλτιστη διαδρομή.