

**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ  
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ  
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ  
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**Α' Ομάδα Ασκήσεων "Λογικού Προγραμματισμού"**  
**Ακαδημαϊκού Έτους 2019-20**

**Άσκηση 1**

Ορίστε σε Prolog ένα κατηγορημα `cart_prod/2`, το οποίο όταν καλείται με ένα σύνολο συνόλων σαν πρώτο όρισμα (στη μορφή μίας λίστας λιστών), να επιστρέφει στο δεύτερο όρισμα το καρτεσιανό γινόμενο των συνόλων, σαν ένα σύνολο (λίστα) πλειάδων, όπου κάθε πλειάδα είναι μία λίστα στοιχείων. Ένα παράδειγμα εκτέλεσης είναι το εξής:

```
?- cart_prod([[a,b,c],[1,2],[x,y,z,w]], CP).
CP = [[a,1,x],[a,1,y],[a,1,z],[a,1,w],[a,2,x],[a,2,y],
      [a,2,z],[a,2,w],[b,1,x],[b,1,y],[b,1,z],[b,1,w],
      [b,2,x],[b,2,y],[b,2,z],[b,2,w],[c,1,x],[c,1,y],
      [c,1,z],[c,1,w],[c,2,x],[c,2,y],[c,2,z],[c,2,w]]
```

Θεωρώντας ότι ένας πίνακας μπορεί να αναπαρασταθεί στην Prolog σαν μία λίστα από τις γραμμές του, όπου κάθε γραμμή είναι μία λίστα από τα στοιχεία της, ορίστε το κατηγορημα `matr_transp/2` έτσι ώστε το `matr_transp(M1,M2)` να επιστρέφει στο `M2` τον ανάστροφο πίνακα (δηλαδή αυτόν που προκύπτει με εναλλαγή γραμμών και στηλών) του `M1`. Για παράδειγμα:

```
?- matr_transp([[5,8,9,7,2],[3,6,1,1,4],[2,4,2,8,0]],M).
M = [[5,3,2],[8,6,4],[9,1,2],[7,1,8],[2,4,0]]
```

Ορίστε και το κατηγορημα `matr_mult/3` έτσι ώστε το `matr_mult(M1,M2,M3)` να επιστρέφει στο `M3` το γινόμενο των πινάκων `M1` και `M2`. Για παράδειγμα:

```
?- matr_mult([[1,2,3],[4,5,6],[6,5,4],[3,2,1]],
              [[3,4],[5,6],[7,8]],M).
M = [[34,40],[79,94],[71,86],[26,32]]
```

Τέλος, ορίστε και το κατηγορημα `matr_det/2` έτσι ώστε το `matr_det(M,D)` να επιστρέφει στο `D` την ορίζουσα του πίνακα `M`. Για παράδειγμα:

```
?- matr_det([[4,3,2,1],[3,2,1,2],[1,4,1,3],[2,1,3,2]],D).
D = 51
```

Παραδοτέο για την άσκηση είναι ένα πηγαίο αρχείο **Prolog** με όνομα **listsoflists.pl**, μέσα στο οποίο θα πρέπει να είναι ορισμένα και τα τέσσερα κατηγορήματα που ζητούνται. Αν χρειάζεστε τα συνήθη ενσωματωμένα κατηγορήματα της Prolog για επεξεργασία λιστών, όπως, για παράδειγμα, το

append/3, μπορείτε να τα χρησιμοποιήσετε χωρίς να τα ορίσετε στο πρόγραμμά σας.

## Άσκηση 2<sup>1</sup>

Μία εκδοχή του λεγόμενου job-shop προβλήματος είναι η εξής: Έχουμε να εκτελέσουμε μία σειρά από έργα  $j_1, j_2, j_3 \dots$ , ανεξάρτητα το ένα από το άλλο, καθένα από τα οποία αποτελείται από έναν αριθμό από εργασίες  $t_{11}, t_{12} \dots, t_{21}, t_{22}, \dots, t_{31}, t_{32}, \dots$  οι οποίες πρέπει να εκτελεστούν με τη δεδομένη σειρά για κάθε έργο. Κάθε εργασία μπορεί να εκτελεσθεί σε συγκεκριμένο τύπο μηχανής και χρειάζεται δεδομένο χρόνο για να έλθει σε πέρας. Επίσης είναι γνωστό το πλήθος των μηχανών που έχουμε διαθέσιμες από κάθε τύπο. Αυτό που θέλουμε να βρούμε είναι πότε και σε ποιες μηχανές πρέπει να εκτελέσουμε τις εργασίες έτσι ώστε όλα τα έργα να έχουν τελειώσει πριν από δεδομένη προθεσμία. Ένα συγκεκριμένο στιγμιότυπο αυτού του προβλήματος δίνεται με τα παρακάτω γεγονότα Prolog:

```
job(j1, [t11,t12]).
job(j2, [t21,t22,t23]).
job(j3, [t31]).
job(j4, [t41,t42]).
```

```
task(t11,m1,2).
task(t12,m2,6).
task(t21,m2,5).
task(t22,m1,3).
task(t23,m2,3).
task(t31,m2,4).
task(t41,m1,5).
task(t42,m2,2).
```

```
machine(m1,1).
machine(m2,2).
```

```
deadline(14).
```

Τα γεγονότα `job/2` περιγράφουν τα έργα που πρέπει να εκτελέσουμε και τις εργασίες από τις οποίες αποτελούνται, κάθε γεγονός `task/3` δίνει, για μία εργασία, σε τι τύπου μηχανή πρέπει να εκτελεσθεί και πόσες μονάδες χρόνου απαιτούνται για την εκτέλεσή της, τα γεγονότα `machine/2` μας πληροφορούν πόσες μηχανές έχουμε διαθέσιμες από κάθε τύπο και τέλος το κατηγορήμα `deadline/1` δίνει την προθεσμία (σε μονάδες χρόνου) μέσα στην οποία πρέπει να έχουν εκτελεσθεί όλα τα έργα. Ορίστε σε Prolog το κατηγορήμα `jobshop/1`, το οποίο να επιστρέφει το πρόγραμμα εκτέλεσης των εργασιών στις μηχανές. Παράδειγμα:

```
?- jobshop(S).
S = [execs(m1,[t(t11,0,2),t(t41,2,7),t(t22,7,10)]),
     execs(m2,[t(t12,2,8),t(t42,8,10),t(t23,10,13)]),
     execs(m2,[t(t21,0,5),t(t31,5,9)])] -> ;
```

---

<sup>1</sup> Για την άσκηση αυτή πρέπει να χρησιμοποιήσετε απλή Prolog και όχι επεκτάσεις της που υποστηρίζουν προγραμματισμό με περιορισμούς.

```

S = .....
.....

?- findall(S,jobshop(S),L), length(L,N).
.....
N = 9136

```

Η παραπάνω πρώτη λύση του προβλήματος σημαίνει ότι στη μηχανή τύπου  $m_1$  θα εκτελεστούν από τη χρονική στιγμή 0 έως τη 2 η εργασία  $t_{11}$ , από τη χρονική στιγμή 2 έως την 7 η εργασία  $t_{41}$  και από τη χρονική στιγμή 7 έως τη 10 η εργασία  $t_{22}$ . Μετά ακολουθούν τα προγράμματα εργασίας για τις δύο μηχανές τύπου  $m_2$ .

Μία δεύτερη εκδοχή του job-shop προβλήματος είναι αυτή στην οποία για κάθε εργασία έχουμε και ένα δεδομένο πλήθος ατόμων που πρέπει να εργασθούν στη μηχανή που θα τη φέρει σε πέρας. Δηλαδή, τώρα έχουμε το κατηγορήμα `task/4` (αντί για `task/3`), όπου το τελευταίο επιπλέον όρισμα δίνει και το πλήθος αυτό των ατόμων. Τα γεγονότα τώρα είναι:

```

task(t11,m1,2,3).
task(t12,m2,6,2).
task(t21,m2,5,2).
task(t22,m1,3,3).
task(t23,m2,3,2).
task(t31,m2,4,2).
task(t41,m1,5,4).
task(t42,m2,2,1).

```

Επίσης έχουμε δεδομένο και το προσωπικό που υπάρχει διαθέσιμο ανά πάσα χρονική στιγμή, το οποίο ορίζεται με το γεγονός `staff/1`, δηλαδή:

```
staff(6).
```

Υποτίθεται ότι κάθε εργαζόμενος μπορεί να δουλέψει σε οποιαδήποτε μηχανή για οποιοδήποτε εργασία.

Ορίστε σε Prolog το κατηγορήμα `jobshop_with_manpower/1`, το οποίο να λύνει και αυτήν την εκδοχή του job-shop προβλήματος. Παράδειγμα εκτέλεσης:

```

?- jobshop_with_manpower(S).
S = [execs(m1,[t(t11,0,2),t(t41,2,7),t(t22,7,10)]),
      execs(m2,[t(t21,0,5),t(t12,5,11),t(t23,11,14)]),
      execs(m2,[t(t42,7,9),t(t31,10,14)])] -> ;

S = .....
.....

?- findall(S,jobshop_with_manpower(S),L), length(L,N).
.....
N = 8

```

Τα δεδομένα του προβλήματος, στη μορφή των γεγονότων Prolog που δίνονται στην εκφώνηση, βρίσκονται στο [http://www.di.uoa.gr/~takis/jobshop\\_data.pl](http://www.di.uoa.gr/~takis/jobshop_data.pl).

Παραδοτέο για την άσκηση είναι ένα **πηγαίο αρχείο Prolog** με όνομα **jobshop.pl**, μέσα στο οποίο **δεν** θα πρέπει να περιέχονται γεγονότα που ορίζουν τα δεδομένα του προβλήματος.