ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Β΄ Ομάδα Ασκήσεων "Λογικού Προγραμματισμού" Ακαδημαϊκού Έτους 2016-17

Οι ασκήσεις της ομάδας αυτής πρέπει να αντιμετωπισθούν με τη βοήθεια της τεχνολογίας του **προγραμματισμού με περιορισμούς**. Ένα σύστημα λογικού προγραμματισμού που υποστηρίζει την τεχνολογία αυτή είναι η **ECL^{i}PS^e** (http://www.eclipseclp.org). Μπορείτε να χρησιμοποιήσετε είτε την παλιότερη βιβλιοθήκη **`fd`** είτε την νεώτερη **`ic`**. Η βιβλιοθήκη **`fd`** τεκμηριώνεται στο κεφάλαιο 2 του "Obsolete Libraries Manual" και η **`ic`** στα κεφάλαια 3 και 4 του "Constraint Library Manual". Αν χρησιμοποιήσετε την **`ic`**, θα σας χρειαστεί και η βιβλιοθήκη **`branch_and_bound`**, ιδιαιτέρως το κατηγόρημα `bb_min/3`, το οποίο τεκμηριώνεται, όπως και όλα τα κατηγορήματα που παρέχει η **ECL^{i}PS^e**, στο "Alphabetical Predicate Index".

Εναλλακτικά συστήματα λογικού προγραμματισμού με περιορισμούς που μπορείτε να χρησιμοποιήσετε για τις ασκήσεις αυτής της ομάδας είναι η **GNU Prolog** (http://www.gprolog.org) και η **SWI-Prolog** (http://www.swi-prolog.org), αλλά δεν προτείνονται, διότι οι βιβλιοθήκες περιορισμών τους είναι περιορισμένης λειτουργικότητας και όχι ιδιαίτερα αποδοτικές.

Σε κάθε περίπτωση, στα αρχεία που θα παραδώσετε, θα πρέπει να αναφέρεται στην αρχή τους, σαν σχόλιο, για ποιο σύστημα Prolog έχουν υλοποιηθεί τα αντίστοιχα προγράμματα, εάν αυτό είναι διαφορετικό από την **ECL^{i}PS^e**.

Άσκηση 4

Το *πρόβλημα της μέγιστης κλίκας* (maximum clique problem) σε γράφο συνίσταται στην εύρεση ενός συνόλου κόμβων του γράφου που ανά δύο να συνδέονται μεταξύ τους και το πλήθος τους να είναι το μέγιστο δυνατό. Για παράδειγμα, μπορούμε να βρούμε ένα μέγιστο σύνολο μελών του Facebook που όλοι να είναι φίλοι μεταξύ τους ανά δύο;

Για την αντιμετώπιση του προβλήματος αυτού, πρέπει να χρησιμοποιήσετε γράφους που κατασκευάζονται μέσω του κατηγορήματος `create_graph(N, D, G)`, όπως αυτό ορίζεται στο αρχείο `http://www.di.uoa.gr/~takis/graph.pl`. Κατά την κλήση του κατηγορήματος δίνεται το πλήθος `N` των κόμβων του γράφου και η πυκνότητά του `D` (σαν ποσοστό των ακμών που υπάρχουν στον γράφο σε σχέση με όλες τις δυνατές ακμές) και επιστρέφεται ο γράφος `G` σαν μία λίστα από ακμές της μορφής `N1-N2`, όπου `N1` και `N2` είναι οι δύο κόμβοι της ακμής (οι κόμβοι του γράφου

παριστάνονται σαν ακέραιοι από το `1` έως το `N`). Ένα παράδειγμα εκτέλεσης του κατηγορήματος αυτού είναι το εξής:[1]

```
?- seed(1), create_graph(9, 30, G).

G = [1 - 5, 2 - 4, 2 - 6, 3 - 4, 3 - 6, 3 - 9, 4 - 7, 5 - 7,
     6 - 7, 6 - 8, 6 - 9]
```

Για την άσκηση αυτή, θα πρέπει να ορίσετε ένα κατηγόρημα `maxclq/4`, το οποίο όταν καλείται σαν `maxclq(N, D, Clique, Size)`, αφού δημιουργήσει ένα γράφο `N` κόμβων και πυκνότητας `D`, καλώντας το κατηγόρημα `create_graph/3`, να βρίσκει μία μέγιστη κλίκα του γράφου, επιστρέφοντας στο `Clique` τη λίστα με τους κόμβους της κλίκας και στο `Size` το μέγεθός της. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:[2]

```
?- seed(1), maxclq(8, 80, Clique, Size).
Clique = [2, 4, 5, 6, 7]
Size = 5

?- seed(2017), maxclq(15, 60, Clique, Size).
Clique = [2, 3, 5, 8, 9, 15]
Size = 6

?- seed(100), maxclq(32, 50, Clique, Size).
Clique = [1, 2, 3, 8, 13, 19]
Size = 6

?- seed(12345), maxclq(80, 50, Clique, Size).
Clique = [1, 3, 7, 42, 44, 51, 72, 79]
Size = 8

?- seed(8231), maxclq(120, 40, Clique, Size).
Clique = [12, 31, 54, 68, 73, 75, 92, 111]
Size = 8

?- seed(1010101), maxclq(165, 30, Clique, Size).
Clique = [12, 54, 81, 95, 108, 109, 130]
Size = 7

?- seed(654321), maxclq(220, 20, Clique, Size).
Clique = [11, 14, 135, 141, 153, 176]
Size = 6

?- seed(1111), maxclq(300, 15, Clique, Size).
Clique = [1, 4, 69, 138, 232]
Size = 5
```

[1] Το κατηγόρημα `seed/1` αρχικοποιεί τη γεννήτρια τυχαίων αριθμών. Η συγκεκριμένη εκτέλεση, όπως και όσες ακολουθούν στην άσκηση αυτή, έγινε σε μηχάνημα Linux του εργαστηρίου του Τμήματος.

[2] Θα πρέπει να σημειωθεί ότι για δεδομένο γράφο μπορεί να υπάρχουν περισσότερες από μία κλίκα με μέγιστο πλήθος κόμβων. Αυτό σημαίνει ότι το δικό σας πρόγραμμα ενδεχομένως να μην βρίσκει την ίδια κλίκα με αυτή που φαίνεται στις ενδεικτικές εκτελέσεις, αλλά θα πρέπει σίγουρα να έχει το ίδιο μέγεθος.

```
?- seed(1821), maxclq(500, 10, Clique, Size).
Clique = [1, 38, 156, 176, 335]
Size = 5

?- seed(1), maxclq(800, 2, Clique, Size).
Clique = [1, 107, 272]
Size = 3
```

Παραδοτέο για την άσκηση είναι **<u>ένα</u> πηγαίο αρχείο Prolog** με όνομα **maxclq.pl**.

<u>Άσκηση 5</u>

Αντιμετωπίστε τη 2η άσκηση της Α' ομάδας ασκήσεων μέσω της τεχνικής του λογικού προγραματισμού με περιορισμούς. Συγκεκριμένα, ορίστε ένα κατηγόρημα ο `liars_csp/2`, με τις ίδιες ακριβώς προδιαγραφές με το `liars/2` της 2ης άσκησης. Κάποια παραδείγματα εκτέλεσης (τα περισσότερα από την εκφώνηση της 2ης άσκησης) είναι τα εξής:

```
?- liars_csp([12, 3, 9, 15, 8, 9, 0, 15, 9, 6, 14, 6, 3, 3,
              9], Liars).
Liars = [1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1]

?- liars_csp([2, 3, 3, 4], Liars).
no

?- liars_csp([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], Liars).
Liars = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

?- liars_csp([15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
              15, 15], Liars).
Liars = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

?- liars_csp([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
              15, 16, 17, 18, 19, 20], Liars).
Liars = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1]

?- liars_csp([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
              5, 5, 5, 5, 5, 5, 5, 5, 5], Liars).
no

?- liars_csp([11, 15, 29, 17, 20, 30, 25, 15, 14, 24, 26, 21,
              8, 21, 28, 8, 5, 28, 9, 6, 28, 8, 20, 18, 10,
              29, 28, 16, 0, 5], Liars).
Liars = [0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
         0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0]
```

Για να δοκιμάσετε το πρόγραμμά σας με μεγαλύτερες εισόδους, ορίστε όπως φαίνεται στη συνέχεια το κατηγόρημα `genrand/2`, το οποίο παράγει τυχαίες εισόδους και χρησιμοποιήστε το για να δοκιμάσετε το πρόγραμμά σας και με μεγαλύτερες εισόδους.

```
genrand(N, List) :-
   length(List, N),
   make_list(N, List).

make_list(_, []).
make_list(N, [X|List]) :-
   random(R),
   X is R mod (N+1),
   make_list(N, List).
```

Κάποια παραδείγματα εκτέλεσης:[3]

```
?- seed(100), genrand(100, C), liars_csp(C, Liars).
C = [31, 74, 62, 82, 60, 86, 19, 22, 62, 35, 53, 32, 15, 90,
     76, 18, 69, 50, 82, ...]
Liars = [0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
         1, ...]

?- seed(1000), genrand(1000, C), liars_csp(C, Liars).
C = [535, 190, 953, 345, 772, 587, 51, 641, 365, 208, 255,
     805, 790, 916, 725, 47, 941, 404, 507, ...]
Liars = [1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0,
         1, ...]

?- seed(10000), genrand(10000, C), liars_csp(C, Liars).
no

?- seed(100000), genrand(100000, C), liars_csp(C, Liars).
C = [31019, 50353, 294, 1630, 8049, 99941, 51677, 56986,
     18096, 98591, 75276, 81816, 93653, 40550, 92271, 11532,
     82321, 35014, 11802, ...]
Liars = [0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0,
         0, ...]

?- seed(300000), genrand(300000, C), liars_csp(C, Liars).
C = [67329, 299743, 125525, 264060, 241426, 242319, 77779,
     49603, 169517, 189144, 201341, 121792, 61687, 243330,
     211703, 159149, 27684, 225457, 38927, ...]
Liars = [0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1,
         0, ...]
```

Παραδοτέο για την άσκηση είναι **<u>ένα</u> πηγαίο αρχείο Prolog** με όνομα **liars_csp.pl**.

<u>Άσκηση 6</u>

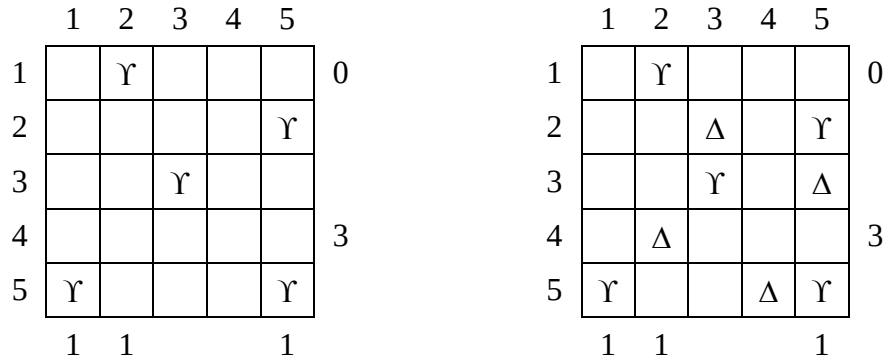
Θεωρούμε έναν ορθογώνιο αγρό δεδομένων διαστάσεων, έστω NxM. Σε συγκεκριμένες θέσεις (i,j) του αγρού, με το i από 1 έως N και το j από 1 έως M, υπάρχουν K δέντρα. Θέλουμε να τοποθετήσουμε στον αγρό ένα πλήθος από τέντες σε θέσεις που πρέπει να βρεθούν έτσι ώστε:

- Σε τουλάχιστον μία από τις γειτονικές θέσεις κάθε δέντρου, οριζόντια, κάθετα ή διαγώνια, να υπάρχει τέντα.

[3] Σε μηχάνημα Linux του εργαστηρίου του Τμήματος.

- Δύο τέντες δεν πρέπει να βρίσκονται σε γειτονικές θέσεις, ούτε οριζόντια, ούτε κάθετα, ούτε διαγώνια.
- Δεν μπορεί να υπάρχει τέντα σε θέση που υπάρχει δέντρο.
- Για κάποιες, όχι απαραίτητα όλες, από τις γραμμές και τις στήλες του αγρού δίνονται μέγιστοι αριθμοί τεντών που μπορεί να υπάρχουν στη γραμμή ή τη στήλη, αντίστοιχα.
- Οι τέντες που θα τοποθετηθούν να είναι οι ελάχιστες δυνατές.

Παρακάτω, στο σχήμα αριστερά, φαίνεται ένας αγρός με $N = 5$ και $M = 5$, στον οποίο βρίσκονται $K = 5$ δέντρα, που σημειώνονται με το σύμβολο ϒ, στις θέσεις (1,2), (2,5), (3,3), (5,1) και (5,5). Επίσης, δίνονται οι περιορισμοί ότι στην 1η και στην 4η γραμμή πρέπει να υπάρχουν έως 0 και έως 3 τέντες, αντίστοιχα, και ότι σε καθεμία από την 1η, την 2η και την 5η στήλη πρέπει να υπάρχει έως 1 τέντα. Στο σχήμα δεξιά φαίνεται μία λύση του προβλήματος, με τον ελάχιστο δυνατό αριθμό τεντών, οι θέσεις των οποίων σημειώνονται με το σύμβολο Δ.



Γράψτε ένα κατηγόρημα `tents/4`, το οποίο να καλείται σαν `tents(RowTents, ColumnTents, Trees, Tents)`, όπου `RowTents` είναι μία λίστα με τα επιθυμητά συνολικά μέγιστα πλήθη τεντών ανά γραμμή, `ColumnTents` είναι μία λίστα με τα επιθυμητά συνολικά μέγιστα πλήθη τεντών ανά στήλη και `Trees` είναι μία λίστα από συντεταγμένες της μορφής `Row-Column`, στις οποίες βρίσκονται τα δέντρα. Το πλήθος των γραμμών N του αγρού ισούται με το μήκος της λίστας `RowTents`, το πλήθος των στηλών M ισούται με το μήκος της λίστας `ColumnTents` και το πλήθος των δέντρων K ισούται με το μήκος της λίστας `Trees`. Αν δεν θέλουμε να δώσουμε περιορισμό μεγίστου πλήθους για τις τέντες σε κάποια γραμμή ή κάποια στήλη, αρκεί στο αντίστοιχο στοιχείο της λίστας `RowTents` ή `ColumnTents`, αντίστοιχα, να βάλουμε κάποιο αρνητικό αριθμό. Το κατηγόρημα που θα γράψετε να επιστρέφει στη μεταβλητή `Tents` μία λίστα με τις συντεταγμένες των θέσεων στις οποίες πρέπει να τοποθετηθούν οι τέντες, ώστε να ισχύουν οι περιορισμοί που έχουν τεθεί. **Αν το πρόβλημα έχει περισσότερες από μία λύση με το ελάχιστο δυνατό πλήθος τεντών, να βρίσκονται όλες μέσω οπισθοδρόμησης.** Κάποιες ενδεικτικές εκτελέσεις είναι οι εξής:

```
?- tents([0, -1, -1, 3, -1], [1, 1, -1, -1, 1],
         [1-2, 2-5, 3-3, 5-1, 5-5], Tents).
Tents = [2 - 3, 3 - 5, 5 - 2, 5 - 4]  --> ;
Tents = [2 - 3, 3 - 5, 4 - 2, 5 - 4]  --> ;
Tents = [2 - 3, 3 - 5, 4 - 1, 5 - 4]  --> ;
Tents = [2 - 2, 3 - 5, 4 - 1, 5 - 4]  --> ;
```

```
.......................

?- findall(Tents, tents([0, -1, -1, 3, -1], [1, 1, -1, -1, 1],
           [1-2, 2-5, 3-3, 5-1, 5-5], Tents), AllTents).
AllTents = [[2 - 3, 3 - 5, 5 - 2, 5 - 4],
            [2 - 3, 3 - 5, 4 - 2, 5 - 4],
            [2 - 3, 3 - 5, 4 - 1, 5 - 4],
            [2 - 2, 3 - 5, 4 - 1, 5 - 4],
            [2 - 2, 3 - 4, 4 - 1, 5 - 4],
            [2 - 2, 2 - 4, 4 - 1, 5 - 4],
            [2 - 2, 2 - 4, 4 - 1, 4 - 5],
            [2 - 2, 2 - 4, 4 - 1, 4 - 4],
            [2 - 1, 3 - 5, 4 - 2, 5 - 4],
            [2 - 1, 3 - 4, 5 - 2, 5 - 4],
            [2 - 1, 3 - 4, 4 - 2, 5 - 4],
            [2 - 1, 2 - 4, 5 - 2, 5 - 4],
            [2 - 1, 2 - 4, 4 - 5, 5 - 2],
            [2 - 1, 2 - 4, 4 - 4, 5 - 2],
            [2 - 1, 2 - 4, 4 - 2, 5 - 4],
            [2 - 1, 2 - 4, 4 - 2, 4 - 5],
            [2 - 1, 2 - 4, 4 - 2, 4 - 4]]

?- tents([-1, -1, -1, 2, -1, -1, 2, 1],
         [2, 1, -1, 1, 1, -1, 1, -1, -1, 1, 2, -1],
         [1-4, 1-9, 1-12, 2-1, 2-5, 2-8, 3-1, 3-6, 3-8, 3-12,
          4-5, 4-7, 4-11, 5-3, 5-9, 6-1, 6-7, 6-11, 7-5,
          8-10], Tents).
Tents = [2 - 4, 2 - 9, 2 - 12, 3 - 2, 4 - 6, 5 - 10, 6 - 3,
         7 - 1, 7 - 6, 8 - 11]  --> ;
Tents = [2 - 4, 2 - 9, 2 - 12, 3 - 2, 4 - 6, 5 - 10, 6 - 3,
         7 - 1, 7 - 6, 8 - 9]  --> ;
Tents = [2 - 4, 2 - 9, 2 - 12, 3 - 2, 4 - 6, 5 - 10, 6 - 3,
         6 - 6, 7 - 1, 8 - 11]  --> ;
.......................

?- findall(Tents, tents([-1, -1, -1, 2, -1, -1, 2, 1],
           [2, 1, -1, 1, 1, -1, 1, -1, -1, 1, 2, -1],
           [1-4, 1-9, 1-12, 2-1, 2-5, 2-8, 3-1, 3-6, 3-8,
            3-12, 4-5, 4-7, 4-11, 5-3, 5-9, 6-1, 6-7, 6-11,
            7-5, 8-10], Tents), AllTents),
   length(AllTents, N).
AllTents = .......................
N = 1262

?- tents([1, -1, -1, -1, -1, -1, 3, 1, -1, 1, 2, 2, 1],
         [2, 1, -1, 1, 4, 1, 3, -1, 2, 1, 1, 0],
         [2-3, 1-5, 5-4, 4-5, 7-7, 10-6, 2-2, 4-8, 8-5, 9-9,
          1-8, 9-2, 3-3, 1-1, 9-8, 8-7, 10-10, 2-7, 8-6, 4-4,
          9-1], Tents).
Tents = [2 - 1, 2 - 4, 2 - 9, 3 - 7, 5 - 5, 8 - 8, 9 - 6,
         10 - 2, 11 - 11]  --> ;
.......................

[4]?- tents([1, -1, -1, -1, -1, -1, 3, 1, -1, 1, 2, 2, -1],
```

[4] Η ερώτηση αυτή είναι πιθανό να αργεί πάρα πολύ, ίσως και μία ώρα.

```
          [2, 1, -1, 1, 4, -1, 3, -1, 2, 1, 1, -1],
          [2-3, 1-5, 5-4, 4-5, 7-7, 10-6, 2-2, 4-8, 8-5, 9-9,
           1-8, 9-2, 3-3, 1-1, 9-8, 8-7, 10-10, 2-7, 8-6,
           4-4, 9-1, 11-4, 11-8, 12-5, 12-9, 12-12, 6-11,
           9-11, 6-12], Tents).
Tents = [2 - 1, 2 - 4, 2 - 9, 3 - 7, 5 - 5, 7 - 12, 8 - 8,
        9 - 6, 9 - 12, 10 - 2, 11 - 5, 11 - 11, 12 - 8]  --> ;
.......................
```

Παραδοτέο για την άσκηση είναι **<u>ένα</u> πηγαίο αρχείο Prolog** με όνομα **tents.pl**.

<u>Άσκηση 7</u>

Στην άσκηση αυτή, θα αντιμετωπίσετε το *ετερογενές πρόβλημα δρομολόγησης οχημάτων με χωρητικότητες* (heterogeneous capacitated vehicle routing problem). Στο πρόβλημα αυτό, υπάρχει μία εταιρεία, η οποία πρόκειται να διανείμει συγκεκριμένες ποσότητες από το προϊόν που παράγει σε συγκεκριμένους πελάτες. Όλο το προϊόν βρίσκεται αρχικά στην αποθήκη της εταιρείας. Για τη διανομή των παραγγελιών στους πελάτες, θα χρησιμοποιηθεί ένας στόλος από οχήματα, πιθανώς διαφορετικών χωρητικοτήτων το καθένα. Δεδομένα ενός στιγμιοτύπου του προβλήματος για 8 οχήματα και 20 πελάτες δίνονται στη μορφή των γεγονότων Prolog που φαίνονται στη συνέχεια. Τα δεδομένα αυτά μπορείτε να τα πάρετε από το αρχείο http://www.di.uoa.gr/~takis/hcvrp_data.pl.

```
vehicles([35, 40, 55, 15, 45, 25, 85, 55]).

clients([c(15,  77,  97), c(23, -28,  64), c(14,  77, -39),
         c(13,  32,  33), c(18,  32,   8), c(18, -42,  92),
         c(19,  -8,  -3), c(10,   7,  14), c(18,  82, -17),
         c(20, -48, -13), c(15,  53,  82), c(19,  39, -27),
         c(17, -48, -13), c(12,  53,  82), c(11,  39, -27),
         c(15, -48, -13), c(25,  53,  82), c(14, -39,   7),
         c(22,  17,   8), c(23, -38,  -7)]).
```

Η λίστα που δίνεται σαν όρισμα στο κατηγόρημα `vehicles/1` αντιστοιχεί στα φορτηγά της εταιρείας. Κάθε στοιχείο της λίστας είναι η χωρητικότητα του αντίστοιχου φορτηγού, για το προϊόν που παράγει η εταιρεία. Η λίστα στο κατηγόρημα `clients/1` αναπαριστά τα δεδομένα των πελατών της εταιρείας. Τα στοιχεία της λίστας είναι δομές της μορφής `c(D, X, Y)`, που κάθε μία αντιστοιχεί σε έναν πελάτη, όπου `D` είναι η ποσότητα του προϊόντος που έχει παραγγείλει ο πελάτης και τα `X` και `Y` είναι οι συντεταγμένες του.

Το ζητούμενο είναι να διανεμηθεί σε κάθε πελάτη η ποσότητα του προϊόντος που έχει παραγγείλει, με μία αποστολή. Κάθε φορτηγό, εφ' όσον χρησιμοποιηθεί για τη διανομή, θα πρέπει να κάνει ένα μόνο δρομολόγιο, αναλαμβάνοντας να εξυπηρετήσει συγκεκριμένους πελάτες. Θα ξεκινήσει από την αποθήκη, έχοντας φορτώσει ποσότητα του προϊόντος ίση με το σύνολο των παραγγελιών των πελατών που θα εξυπηρετήσει, η οποία δεν πρέπει να υπερβαίνει τη χωρητικότητά του, θα επισκεφθεί τους πελάτες με κάποια σειρά, για να τους παραδώσει τις παραγγελίες τους, και θα επιστρέψει στην αποθήκη. Η αποθήκη βρίσκεται στη θέση (0,0). Η ικανοποίηση των

παραγγελιών πρέπει να γίνει με τον βέλτιστο για την εταιρεία τρόπο, που συνίσταται στην ελαχιστοποίηση της συνολικής απόστασης που θα διανύσουν τα φορτηγά. Η αποθήκη και οι πελάτες συνδέονται ανά δύο μεταξύ τους με δρόμους που είναι ευθείες γραμμές. Δηλαδή, σαν απόσταση μεταξύ δύο πελατών, ή της αποθήκης και ενός πελάτη, θεωρείται η ευκλείδεια απόστασή τους. Για λόγους επαλήθευσης των αποτελεσμάτων που θα δοθούν στη συνέχεια, μπορεί να θεωρηθεί ότι η απόσταση, αφού πολλαπλασιασθεί με το 1000, στρογγυλεύεται στον πλησιέστερο ακέραιο (ουσιαστικά, πρόκειται για στρογγύλευση στο τρίτο δεκαδικό ψηφίο).

Ορίστε ένα κατηγόρημα `hcvrp/6`, το οποίο όταν καλείται σαν `hcvrp(NCl, NVe, Timeout, Solution, Cost, Time)`, να επιλύει το πρόβλημα, λαμβάνοντας ως δεδομένα τους πρώτους `NCl` πελάτες από τη λίστα του `clients/1` και τα πρώτα `NVe` οχήματα από τη λίστα του `vehicles/1`. Το κατηγόρημα να επιστρέφει στο `Solution` μία λίστα κάθε στοιχείο της οποίας αντιστοιχεί σε ένα φορτηγό και είναι επίσης μία λίστα με τους αύξοντες αριθμούς των πελατών που θα εξυπηρετήσει το εν λόγω φορτηγό, και μάλιστα με τη σειρά που θα τους επισκεφθεί. Το `Cost` είναι το κόστος της λύσης (συνολική διανυθείσα απόσταση από τα φορτηγά). Ιδανικά, πρέπει να βρίσκεται η βέλτιστη λύση. Όμως αυτό δεν είναι εφικτό για τις μεγαλύτερες εισόδους, οπότε μπορεί να δοθεί κατά την κλήση του κατηγορήματος το `Timeout`, που είναι ο χρόνος CPU (σε δευτερόλεπτα) στον οποίο θα πρέπει να τερματισθεί η αναζήτηση, αν δεν έχει βρεθεί η βέλτιστη λύση, και να επιστραφεί η καλύτερη που έχει βρεθεί μέχρι εκείνη τη στιγμή. Για `Timeout` ίσο με 0, δεν θα πρέπει να διακοπεί η αναζήτηση μέχρι να βρεθεί η βέλτιστη λύση. Στο `Time` να επιστρέφεται ο χρόνος εκτέλεσης (σε CPU seconds). Κάποια παραδείγματα εκτέλεσης είναι τα εξής:[5,6]

```
?- hcvrp(1, 1, 0, Solution, Cost, Time).
Found a solution with cost 247694

Solution = [[1]]
Cost = 247694
Time = 0.0

?- hcvrp(2, 1, 0, Solution, Cost, Time).
Found no solution with cost 0.0 .. 371541.0

?- hcvrp(2, 2, 0, Solution, Cost, Time).
Found a solution with cost 303768
Found no solution with cost 0.0 .. 303767.0

Solution = [[], [2, 1]]
Cost = 303768
Time = 0.0

?- hcvrp(3, 2, 0, Solution, Cost, Time).
Found a solution with cost 485874
```

[5] Στο μηχάνημα linux27 του εργαστηρίου του Τμήματος.

[6] Θα πρέπει να σημειωθεί ότι τα κόστη των ενδιάμεσων λύσεων που φαίνονται στις ενδεικτικές εκτελέσεις είναι άμεσα συνυφασμένα με τη συγκεκριμένη υλοποίηση που χρησιμοποιήθηκε. Δεν είναι απαραίτητο στη δική σας υλοποίηση να προκύπτουν τα ίδια ενδιάμεσα κόστη. Όμως, το κόστος της κάθε βέλτιστης λύσης θα πρέπει να είναι το ίδιο με αυτό που φαίνεται εδώ, όχι όμως και η ίδια η βέλτιστη λύση κατ' ανάγκη. Εννοείται ότι το προηγούμενο δεν ισχύει στις ερωτήσεις εκείνες που σταματά η βελτιστοποίηση λόγω εξάντλησης του διαθέσιμου χρόνου.

```
Found a solution with cost 476394
Found no solution with cost 0.0 .. 476393.0

Solution = [[3], [2, 1]]
Cost = 476394
Time = 0.01

?- hcvrp(4, 2, 0, Solution, Cost, Time).
Found a solution with cost 529519
Found a solution with cost 520954
Found no solution with cost 0.0 .. 520953.0

Solution = [[4, 3], [2, 1]]
Cost = 520954
Time = 0.01

?- hcvrp(5, 2, 0, Solution, Cost, Time).
Found no solution with cost 0.0 .. 1718544.0

?- hcvrp(5, 3, 0, Solution, Cost, Time).
Found a solution with cost 606884
Found a solution with cost 572409
Found a solution with cost 569259
Found a solution with cost 552201
Found a solution with cost 541537
Found a solution with cost 526117
Found a solution with cost 523843
Found a solution with cost 506186
Found a solution with cost 488492
Found no solution with cost 0.0 .. 488491.0

Solution = [[], [5, 3], [4, 1, 2]]
Cost = 488492
Time = 0.19

?- hcvrp(6, 3, 0, Solution, Cost, Time).
Found a solution with cost 743868
Found a solution with cost 708939
Found a solution with cost 670958
Found a solution with cost 653900
Found a solution with cost 652408
Found a solution with cost 634714
Found no solution with cost 0.0 .. 634713.0

Solution = [[4, 1], [5, 3], [6, 2]]
Cost = 634714
Time = 0.48

?- hcvrp(7, 3, 0, Solution, Cost, Time).
Found a solution with cost 806024
Found a solution with cost 783895
Found a solution with cost 670944
Found no solution with cost 0.0 .. 670943.0

Solution = [[3, 1], [7, 5], [4, 6, 2]]
Cost = 670944
```

```
Time = 0.96

?- hcvrp(8, 3, 0, Solution, Cost, Time).
Found no solution with cost 0.0 .. 4619061.0

?- hcvrp(8, 4, 0, Solution, Cost, Time).
Found a solution with cost 859115
Found a solution with cost 836986
Found a solution with cost 828848
Found a solution with cost 821892
Found a solution with cost 806634
Found a solution with cost 804930
Found a solution with cost 726824
Found a solution with cost 712619
Found a solution with cost 702248
Found no solution with cost 0.0 .. 702247.0

Solution = [[3, 1], [7, 5], [4, 6, 2], [8]]
Cost = 702248
Time = 8.83

?- hcvrp(9, 4, 0, Solution, Cost, Time).
Found no solution with cost 0.0 .. 6866668.0

?- hcvrp(9, 5, 0, Solution, Cost, Time).
Found a solution with cost 905715
Found a solution with cost 871240
Found a solution with cost 868752
Found a solution with cost 849388
Found a solution with cost 790596
Found a solution with cost 756121
Found a solution with cost 754779
Found a solution with cost 696500
Found a solution with cost 694976
Found a solution with cost 665578
Found no solution with cost 0.0 .. 665577.0

Solution = [[7], [8, 1, 4], [5, 9, 3], [], [6, 2]]
Cost = 665578
Time = 193.5

?- hcvrp(10, 5, 0, Solution, Cost, Time).
Found a solution with cost 1074937
Found a solution with cost 982401
Found a solution with cost 975202
Found a solution with cost 937249
Found a solution with cost 886983
Found a solution with cost 885641
Found a solution with cost 882196
Found a solution with cost 881369
Found a solution with cost 825929
Found a solution with cost 821758
Found a solution with cost 815218
Found a solution with cost 803516
Found a solution with cost 783887
Found a solution with cost 778916
```

```
Found no solution with cost 0.0 .. 778915.0

Solution = [[4, 1], [10, 7], [5, 9, 3], [8], [6, 2]]
Cost = 778916
Time = 415.22

?- hcvrp(11, 5, 0, Solution, Cost, Time).
Found a solution with cost 1264541
Found a solution with cost 1254431
Found a solution with cost 1246860
Found a solution with cost 1233502
Found a solution with cost 1128952
Found a solution with cost 1114301
Found a solution with cost 1111796
Found a solution with cost 1104248
Found a solution with cost 1088194
Found a solution with cost 1083627
Found a solution with cost 1059287
Found a solution with cost 1002961
Found a solution with cost 998790
Found a solution with cost 986016
Found a solution with cost 985820
Found a solution with cost 980274
Found no solution with cost 0.0 .. 980273.0

Solution = [[5, 1], [10, 7], [8, 4, 9, 3], [11], [6, 2]]
Cost = 980274
Time = 1090.0

?- hcvrp(12, 5, 0, Solution, Cost, Time).
Found no solution with cost 0.0 .. 11238035.0

?- hcvrp(12, 6, 900, Solution, Cost, Time).
Found a solution with cost 1197907
Found a solution with cost 1169128
Found a solution with cost 1166623
Found a solution with cost 1159075
Found a solution with cost 1143021
Found a solution with cost 1138474
Found a solution with cost 1114017
Found a solution with cost 1107151
Found a solution with cost 1077065
Found a solution with cost 1060964
Found a solution with cost 1057085
Found a solution with cost 1054551
Found a solution with cost 1040984
Branch-and-bound timeout while searching for solution better
than 1040984

Solution = [[6, 4], [10, 7], [12, 9, 5], [8], [11, 1, 3], [2]]
Cost = 1040984
Time = 899.43

?- hcvrp(13, 6, 900, Solution, Cost, Time).
Branch-and-bound timeout while searching for solution better
than 14547522.0
```

```
?- hcvrp(13, 7, 900, Solution, Cost, Time).
Found a solution with cost 1382116
Found a solution with cost 1122659
Found a solution with cost 1110215
Found a solution with cost 1103349
Found a solution with cost 1091692
Found a solution with cost 1083946
Found a solution with cost 1052959
Found a solution with cost 1043915
Found a solution with cost 1015882
Found a solution with cost 1005365
Found a solution with cost 998074
Found a solution with cost 976861
Found a solution with cost 973598
Found a solution with cost 950227
Found a solution with cost 945170
Found a solution with cost 943678
Found a solution with cost 929289
Found a solution with cost 928572
Found a solution with cost 915092
Found a solution with cost 895389
Branch-and-bound timeout while searching for solution better
than 895389

Solution = [[6], [7], [5, 4, 2], [8], [13, 10], [],
            [12, 3, 9, 1, 11]]
Cost = 895389
Time = 899.56

?- hcvrp(14, 7, 900, Solution, Cost, Time).
Found a solution with cost 1390779
Found a solution with cost 1319417
Found a solution with cost 1252088
Found a solution with cost 1205882
Found a solution with cost 1200806
Found a solution with cost 1172090
Found a solution with cost 1165185
Found a solution with cost 1146337
Found a solution with cost 1119954
Found a solution with cost 1112663
Found a solution with cost 1104101
Found a solution with cost 1071556
Found a solution with cost 1071220
Found a solution with cost 1013687
Found a solution with cost 1001834
Found a solution with cost 998142
Found a solution with cost 989379
Found a solution with cost 961802
Found a solution with cost 928914
Branch-and-bound timeout while searching for solution better
than 928914

Solution = [[7], [8, 5], [6, 2], [4], [13, 10], [12],
            [11, 14, 1, 9, 3]]
Cost = 928914
```

```
Time = 899.62

?- hcvrp(15, 7, 900, Solution, Cost, Time).
Found a solution with cost 1515714
Found a solution with cost 1390779
Found a solution with cost 1364203
Found a solution with cost 1205882
Found a solution with cost 1165185
Found a solution with cost 1152283
Found a solution with cost 1133549
Found a solution with cost 1133213
Found a solution with cost 1132992
Found a solution with cost 1132656
Found a solution with cost 1075680
Found a solution with cost 1075123
Found a solution with cost 1066950
Branch-and-bound timeout while searching for solution better
than 1066950

Solution = [[7], [8, 5], [6, 2], [4], [11, 14, 1], [9],
            [10, 13, 12, 15, 3]]
Cost = 1066950
Time = 899.66

?- hcvrp(16, 7, 900, Solution, Cost, Time).
Found a solution with cost 1730966
Found a solution with cost 1672113
Found a solution with cost 1667828
Found a solution with cost 1595830
Found a solution with cost 1509417
Found a solution with cost 1507710
Found a solution with cost 1499600
Found a solution with cost 1444113
Found a solution with cost 1438676
Found a solution with cost 1428347
Found a solution with cost 1412792
Found a solution with cost 1398885
Found a solution with cost 1395556
Found a solution with cost 1383217
Found a solution with cost 1363203
Found a solution with cost 1354163
Found a solution with cost 1353197
Found a solution with cost 1342818
Found a solution with cost 1335506
Found a solution with cost 1329987
Found a solution with cost 1307490
Found a solution with cost 1304860
Found a solution with cost 1304751
Branch-and-bound timeout while searching for solution better
than 1304751

Solution = [[8], [9, 5], [7, 16, 6], [11], [10, 2], [13],
            [12, 15, 3, 1, 14, 4]]
Cost = 1304751
Time = 899.65
```

```
?- hcvrp(17, 7, 900, Solution, Cost, Time).
Found a solution with cost 1791150
Found a solution with cost 1772795
Found a solution with cost 1741771
Found a solution with cost 1637514
Found a solution with cost 1578158
Found a solution with cost 1471366
Found a solution with cost 1459582
Found a solution with cost 1450849
Found a solution with cost 1446328
Found a solution with cost 1445135
Found a solution with cost 1423981
Found a solution with cost 1421351
Found a solution with cost 1421114
Branch-and-bound timeout while searching for solution better
than 1421114

Solution = [[8, 15, 3], [9, 5], [7, 16, 6], [11], [10, 2],
            [13], [12, 1, 14, 17, 4]]
Cost = 1421114
Time = 899.73

?- hcvrp(18, 7, 900, Solution, Cost, Time).
Branch-and-bound timeout while searching for solution better
than 23166409.0

?- hcvrp(18, 8, 900, Solution, Cost, Time).
Found a solution with cost 1644977
Found a solution with cost 1496673
Found a solution with cost 1429802
Found a solution with cost 1381760
Found a solution with cost 1380413
Found a solution with cost 1363560
Found a solution with cost 1362213
Found a solution with cost 1359367
Found a solution with cost 1358020
Branch-and-bound timeout while searching for solution better
than 1358020

Solution = [[9, 3], [10], [8, 15, 5], [11], [7, 2], [12],
            [6, 1, 14, 17, 4], [16, 18, 13]]
Cost = 1358020
Time = 899.72

?- hcvrp(19, 8, 900, Solution, Cost, Time).
Found a solution with cost 1921909
Found a solution with cost 1766302
Found a solution with cost 1754399
Found a solution with cost 1679889
Found a solution with cost 1667986
Found a solution with cost 1620511
Found a solution with cost 1610372
Found a solution with cost 1481382
Branch-and-bound timeout while searching for solution better
than 1481382
```

```
Solution = [[9, 3], [10], [8, 19, 5], [11], [7, 15, 4], [12],
            [6, 18, 14, 17, 1], [13, 16, 2]]
Cost = 1481382
Time = 899.7

?- hcvrp(20, 8, 900, Solution, Cost, Time).
Found a solution with cost 1774191
Found a solution with cost 1705144
Branch-and-bound timeout while searching for solution better
than 1705144

Solution = [[10, 1], [11, 14, 4], [9, 18, 2], [8],
            [12, 15, 3], [13], [7, 20, 17, 6], [16, 19, 5]]
Cost = 1705144
Time = 899.74
```

Παραδοτέο για την άσκηση είναι **<u>ένα</u> πηγαίο αρχείο Prolog** με όνομα **hcvrp.pl**.