ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Β΄ Ομάδα Ασκήσεων "Λογικού Προγραμματισμού" Ακαδημαϊκού Έτους 2021-22

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), μέσω της βιβλιοθήκης **`ic`**, η οποία τεκμηριώνεται στα κεφάλαια 3 και 4 του "Constraint Library Manual". 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, μπορείτε να χρησιμοποιήσετε είτε την βιβλιοθήκη **`gfd`**, που τεκμηριώνεται στο κεφάλαιο 7 του "Constraint Library Manual", και αποτελεί τη διεπαφή της **ECL^{i}PS^e** με τον επιλυτή περιορισμών Gecode, είτε την παλαιότερη βιβλιοθήκη **`fd`**, που τεκμηριώνεται στο κεφάλαιο 2 του "Obsolete Libraries Manual" (http://www.di.uoa.gr/~takis/obsman.pdf).

Άλλα 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(2022), maxclq(15, 60, Clique, Size).
Clique = [1, 3, 5, 10, 14, 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 **ένα πηγαίο αρχείο Prolog** με όνομα **maxclq.pl**, μέσα στο οποίο **δεν** θα πρέπει να περιέχεται ο ορισμός του κατηγορήματος `create_graph/3`.

Άσκηση 5

Υπάρχει μία παρέα από φίλους, κάποιοι από τους οποίους λένε πάντα ψέματα, ενώ οι υπόλοιποι λένε πάντα την αλήθεια. Όλοι τους γνωρίζουν για τον καθένα άλλο, αν είναι ψεύτης ή όχι. Κάποια ημέρα, για να περάσουν την ώρα τους, παίζουν το εξής παιγνίδι. Το κάθε άτομο κάνει μία δήλωση της μορφής "*στην παρέα υπάρχουν τουλάχιστον Κ ψεύτες*". Για παράδειγμα, έστω ότι υπάρχουν 5 άτομα και οι δηλώσεις τους είναι οι εξής:

- Ο 1ος δηλώνει: "*Υπάρχουν τουλάχιστον 3 ψεύτες μεταξύ μας*"
- Ο 2ος δηλώνει: "*Υπάρχουν τουλάχιστον 2 ψεύτες μεταξύ μας*"
- Ο 3ος δηλώνει: "*Υπάρχει τουλάχιστον 1 ψεύτης μεταξύ μας*"
- Ο 4ος δηλώνει: "*Υπάρχουν τουλάχιστον 4 ψεύτες μεταξύ μας*"
- Ο 5ος δηλώνει: "*Υπάρχουν τουλάχιστον 2 ψεύτες μεταξύ μας*"

Σχετικά εύκολα μπορούμε να διαπιστώσουμε ότι με βάση τα παραπάνω δεδομένα, στην παρέα πρέπει το 1ο και το 4ο άτομο να είναι ψεύτες και μόνο αυτοί.

Υλοποιήστε ένα κατηγόρημα `liars/2`, το οποίο όταν καλείται με πρώτο όρισμα τη λίστα των αριθμών που δηλώνονται από κάθε άτομο ως ελάχιστο πλήθος ψευτών στην παρέα, να επιστρέφει στο δεύτερο όρισμα μία λίστα που να δείχνει τι είναι το κάθε άτομο της παρέας, ψεύτης ή όχι, μέσω κατάλληλης τιμής, 1 ή 0. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:

```
?- liars([3, 2, 1, 4, 2], Liars).
Liars = [1, 0, 0, 1, 0]

?- liars([9, 1, 7, 1, 8, 3, 8, 9, 1, 3], Liars).
Liars = [1, 0, 1, 0, 1, 0, 1, 1, 0, 0]

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

?- liars([4, 9, 1, 12, 14, 8, 1, 17, 3, 6, 5, 6, 18, 20, 0, 8,
          7, 9, 4, 16], Liars).
```

```
Liars = [0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
         0, 1]

?- liars([13, 2, 2, 14, 24, 7, 25, 19, 10, 14, 16, 3, 24, 12,
          9, 16, 16, 0, 15, 16, 5, 19, 2, 3, 16], Liars).
Liars = [0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
         1, 1, 0, 1, 0, 0, 1]

?- liars([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]

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

?- liars([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([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([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([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]

[3] Σε μηχάνημα Linux του εργαστηρίου του Τμήματος.

```
?- seed(100), genrand(100, C), liars(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(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(C, Liars).
no

?- seed(100000), genrand(100000, C), liars(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(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 **ένα πηγαίο αρχείο Prolog** με όνομα **liars.pl**, μέσα στο οποίο **δεν** θα πρέπει να περιέχεται ο ορισμός του κατηγορήματος `genrand/2`.

Άσκηση 6

Στις αυτοκινητοβιομηχανίες, τα αυτοκίνητα κατασκευάζονται σε μία γραμμή συναρμολόγησης (assembly line). Σ' ένα δεδομένο χρονικό διάστημα, π.χ. μία ημέρα, πρέπει να κατασκευασθεί ένας συγκεκριμένος αριθμός αυτοκινήτων από κάποιο μοντέλο που παράγει η εταιρεία, τα οποία δεν είναι κατ' ανάγκη απολύτως όμοια στον εξοπλισμό τους. Κάποια αυτοκίνητα μπορεί να διαθέτουν κλιματισμό, κάποια άλλα όχι. Κάποια μπορεί να έχουν ηλιοροφή, άλλα όχι. Γενικώς, υπάρχει ένα σύνολο από επιλογές (κλιματισμός, ηλιοροφή, ηχοσύστημα, δερμάτινα καθίσματα, ζάντες αλουμινίου, κλπ.), που, με βάση τις παραγγελίες προς την εταιρεία, άλλα αυτοκίνητα πρέπει να διαθέτουν κάποια επιλογή, ενώ άλλα όχι. Δύο αυτοκίνητα που απαιτούν τις ίδιες ακριβώς επιλογές λέμε ότι ανήκουν στην ίδια σύνθεση (configuration).

Για κάθε πιθανή επιλογή στον εξοπλισμό του μοντέλου, υπάρχει μία ομάδα εργασίας που δουλεύει σε κάποια περιοχή της γραμμής συναρμολόγησης. Όμως, για να μπορεί η κάθε ομάδα εργασίας να φέρει σε πέρας το έργο της, πρέπει, όσον αφορά την επιλογή με την οποία ασχολείται, να μην υπάρχουν ποτέ σε κάθε M συνεχόμενα αυτοκίνητα στη γραμμή συναρμολόγησης περισσότερα από K που να απαιτούν την

επιλογή. Τα M και K είναι συγκεκριμένα για κάθε δυνατή επιλογή. Η απαίτηση αυτή ονομάζεται περιορισμός χωρητικότητας (capacity constraint).

Το πρόβλημα της δρομολόγησης παραγωγής αυτοκινήτων (car sequencing) συνίσταται στον καθορισμό της σειράς με την οποία πρέπει να παραχθούν N αυτοκίνητα σε μία γραμμή συναρμολόγησης, έχοντας επίσης δεδομένες τις δυνατές επιλογές, τους περιορισμούς χωρητικότητάς τους και τα πλήθη των αυτοκινήτων που αντιστοιχούν σε επιθυμητές συνθέσεις (συνδυασμοί επιλογών).

Ορίστε ένα κατηγόρημα `carseq/1`, το οποίο όταν καλείται σαν `carseq(S)`, να επιστρέφει στο `S` τη σειρά με την οποία πρέπει να τοποθετηθούν αυτοκίνητα των διαφόρων πιθανών συνθέσεων σε μία γραμμή συναρμολόγησης. Το `S` που επιστρέφεται πρέπει να είναι μία λίστα, κάθε στοιχείο της οποίας είναι ο αύξων αριθμός της σύνθεσης στην οποία ανήκει το αντίστοιχο αυτοκίνητο στη γραμμή συναρμολόγησης. Τα δεδομένα του προβλήματος δίνονται μέσω των κατηγορημάτων `classes/1` και `options/1`. Ένα παράδειγμα δεδομένων είναι το εξής:

```
classes([1,1,2,2,2,2]).
options([2/1/[1,0,0,0,1,1],
         3/2/[0,0,1,1,0,1],
         3/1/[1,0,0,0,1,0],
         5/2/[1,1,0,1,0,0],
         5/1/[0,0,1,0,0,0]]).
```

Τα παραπάνω δεδομένα περιγράφουν την εξής κατάσταση: Υπάρχουν 6 πιθανές συνθέσεις, όσα είναι τα στοιχεία της λίστας που είναι όρισμα στο κατηγόρημα `classes/1`. Η πρώτη σύνθεση περιλαμβάνει 1 αυτοκίνητο, η δεύτερη επίσης 1, η τρίτη 2 αυτοκίνητα, κοκ. Συνολικά πρέπει να κατασκευασθούν 10 αυτοκίνητα, όσο είναι το άθροισμα των στοιχείων της λίστας που δίνεται μέσω του `classes/1`. Υπάρχουν 5 πιθανές επιλογές, όσα είναι τα στοιχεία της λίστας που είναι όρισμα στο κατηγόρημα `options/1`. Κάθε επιλογή ορίζεται μέσω μίας τριάδας `M/K/O`, όπου τα `M` και `K` εκφράζουν τον περιορισμό χωρητικότητας της επιλογής (το πολύ `K` αυτοκίνητα σε κάθε `M` συνεχόμενα στη γραμμή συναρμολόγησης πρέπει να απαιτούν την επιλογή) και το `O` είναι μία λίστα από `1` και `0`, μήκους όσο το πλήθος των πιθανών συνθέσεων, που εκφράζει το αν η εν λόγω επιλογή περιλαμβάνεται στην αντίστοιχη σύνθεση (τιμή `1`) ή όχι (τιμή `0`). Για παράδειγμα, το πρώτο στοιχείο της λίστας που δίνεται μέσω του `options/1` αντιστοιχεί σε μία επιλογή (π.χ. κλιματισμός) και εκφράζει ότι πρέπει να υπάρχει το πολύ 1 αυτοκίνητο σε κάθε 2 συνεχόμενα στη γραμμή συναρμολόγησης που να απαιτεί την επιλογή και ότι η εν λόγω επιλογή περιλαμβάνεται στην $1^η$, την $5^η$ και την $6^η$ σύνθεση. Ομοίως και για τις άλλες επιλογές.

Τα γεγονότα `classes/1` και `options/1` που δίνονται παραπάνω μπορείτε να τα βρείτε στο αρχείο http://www.di.uoa.gr/~takis/carseq_data1.pl. Τα αποτελέσματα εκτέλεσης για τα δεδομένα αυτά είναι:

```
?- carseq(S).
S = [1, 2, 6, 3, 5, 4, 4, 5, 3, 6]        --> ;
S = [1, 3, 6, 2, 5, 4, 3, 5, 4, 6]        --> ;
S = [1, 3, 6, 2, 6, 4, 5, 3, 4, 5]        --> ;
```

```
S = [5, 4, 3, 5, 4, 6, 2, 6, 3, 1]          --> ;
S = [6, 3, 5, 4, 4, 5, 3, 6, 2, 1]          --> ;
S = [6, 4, 5, 3, 4, 5, 2, 6, 3, 1]          --> ;
no
```

Το πρώτο από τα αποτελέσματα (`[1, 2, 6, 3, 5, 4, 4, 5, 3, 6]`) εκφράζει ότι η αλυσίδα συναρμολόγησης πρέπει να αποτελείται κατά σειρά από ένα αυτοκίνητο της $1^{ης}$ σύνθεσης, ένα της $2^{ης}$, ένα της $6^{ης}$, ένα της $3^{ης}$, κοκ. Όπως παρατηρείτε, για τα δεδομένα αυτά, υπάρχουν έξι δυνατές λύσεις.

Δεδομένα για άλλο στιγμιότυπο του προβλήματος, μεγαλύτερου μεγέθους, βρίσκονται στο http://www.di.uoa.gr/~takis/carseq_data2.pl. Κάποιες λύσεις για αυτό το στιγμιότυπο είναι οι εξής:

```
?- carseq(S).
S = [1, 4, 7, 15, 3, 16, 2, 10, 8, 15, 1, 13, 2, 15, 8, 9, 2,
     17, 3, 15, 6, 10, 3, 17, 5, 9, 7, 11, 5, 17, 1, 10, 8,
     15, 5, 12, 5, 10, 7, 11, 5, 16, 3, 10, 7, 11, 5, 16, 3,
     10, 7, 11, 5, 16, 3, 15, 7, 11, 1, 17, 3, 15, 7, 9, 5,
     17, 5, 11, 6, 15, 5, 17, 7, 9, 8, 15, 7, 11, 1, 17, 8,
     17, 14, 11, 7, 16, 7, 11, 14, 17, 11, 14, 17, 11, 14, 17,
     11, 14, 17, 18]   --> ;

S = [1, 4, 7, 15, 3, 16, 2, 10, 8, 15, 1, 13, 2, 15, 8, 9, 2,
     17, 3, 15, 6, 10, 3, 17, 5, 9, 7, 11, 5, 17, 1, 10, 8,
     15, 5, 12, 5, 10, 7, 11, 5, 16, 3, 10, 7, 11, 5, 16, 3,
     10, 7, 11, 5, 16, 3, 15, 7, 11, 1, 17, 3, 15, 7, 9, 5,
     17, 5, 11, 6, 15, 5, 17, 7, 9, 8, 15, 7, 11, 1, 17, 8,
     17, 14, 11, 7, 16, 7, 11, 14, 17, 11, 14, 17, 11, 14, 17,
     11, 17, 14, 18]   --> ;

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

Μην αποπειραθείτε να βρείτε όλες τις λύσεις για τα προηγούμενα δεδομένα. Είναι υπερβολικά πολλές.

Για τα δεδομένα στο αρχείο http://www.di.uoa.gr/~takis/carseq_data3.pl, έχουμε:

```
?- carseq(S).
no
```

Αλλά για το http://www.di.uoa.gr/~takis/carseq_data4.pl:

```
?- carseq(S).

S = [1, 1, 2, 2, 1, 1, 4, 2, 1, 1, 2, 2, 2, 4, 5, 3, 2, 5, 5,
     2, 2, 2, 3, 5, 5, 6, 6, 5, 5, 6, 6, 5, 6, 6, 6, 6, 8, 8,
     6, 6, 8, 6, 6, 6, 6, 8, 8, 6, 10, 7, 13, 10, 10, 13, 9,
     11, 15, 9, 11, 13, 12, 13, 15, 13, 15, 13, 15, 13, 15,
     13, 15, 14, 15, 14, 15, 14, 16, 15, 15, 16, 19, 16, 19,
     17, 17, 19, 20, 17, 20, 16, 20, 18, 18, 19, 20, 20, 17,
     21, 20, 22]   --> ;
```

. . . . . .

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

<u>Άσκηση 7</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.
- 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.

	1	2	3	4	5	
1		ϒ				0
2					ϒ	
3			ϒ			
4						3
5	ϒ				ϒ	
	1	1			1	

	1	2	3	4	5	
1		ϒ				0
2			Δ		ϒ	
3			ϒ		Δ	
4		Δ				3
5	ϒ			Δ	ϒ	
	1	1			1	

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],
          [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**.

[4] Η ερώτηση αυτή είναι πιθανό να αργεί αρκετά, ίσως κάποια λεπτά της ώρας.