ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Β΄ Ομάδα Ασκήσεων "Λογικού Προγραμματισμού" Ακαδημαϊκού Έτους 2022-23

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

Μία εκδοχή του προβλήματος διαμέρισης αριθμών είναι η εξής. Δεδομένου κάποιου άρτιου θετικού ακεραίου N, να διαμερισθεί το σύνολο $S = \{1, 2, 3, \ldots, N\}$ σε δύο υποσύνολα S_1 και S_2 ($S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = S$) τέτοια ώστε τα S_1 και S_2 να έχουν ίδιο πλήθος στοιχείων ($|S_1| = |S_2|$), το άθροισμα των στοιχείων του S_1 να ισούται με το άθροισμα των στοιχείων του S_2 ($\sum_{i \in S_1} i = \sum_{j \in S_2} j$) και το άθροισμα των τετραγώνων των στοιχείων του S_1 να ισούται με το άθροισμα των τετραγώνων των στοιχείων του S_2 ($\sum_{i \in S_1} i^2 = \sum_{j \in S_2} j^2$). Για παράδειγμα, για $N = 8$, το πρόβλημα έχει μία λύση, την $S_1 = \{1, 4, 6, 7\}$, $S_2 = \{2, 3, 5, 8\}$. Είναι προφανές ότι το πρόβλημα δεν έχει λύση αν το N είναι περιττός. Επίσης, φαίνεται ότι υπάρχουν λύσεις μόνο για N που είναι πολλαπλάσια του 4 και μεγαλύτερα ή ίσα του 8, αλλά κάτι τέτοιο δεν έχει αποδειχθεί μαθηματικά.

Ορίστε στο σύστημα λογικού προγραμματισμού της επιλογής σας ένα κατηγόρημα `numpart/3`, το ποίο όταν καλείται σαν `numpart(N, L1, L2)`, για δεδομένο `N`, να επιστρέφει στα `L1` και `L2`, μία διαμέριση του συνόλου {1, 2, 3, …, N}, σύμφωνα με τον παραπάνω ορισμό, και τελικά, μέσω οπισθοδρόμησης, όλες τις διαφορετικές λύσεις. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:

```
?- numpart(2, L1, L2).
no

?- numpart(4, L1, L2).
no

?- numpart(8, L1, L2).
L1 = [1, 4, 6, 7]
L2 = [2, 3, 5, 8]  --> ;
no

?- numpart(9, L1, L2).
no

?- numpart(10, L1, L2).
no

?- numpart(12, L1, L2).
L1 = [1, 3, 7, 8, 9, 11]
L2 = [2, 4, 5, 6, 10, 12]            --> ;
no

?- numpart(16, L1, L2).
L1 = [1, 4, 5, 8, 10, 11, 14, 15]
L2 = [2, 3, 6, 7, 9, 12, 13, 16]     --> ;
L1 = [1, 5, 6, 7, 8, 11, 14, 16]
L2 = [2, 3, 4, 9, 10, 12, 13, 15]    --> ;
L1 = [1, 4, 6, 7, 9, 12, 14, 15]
L2 = [2, 3, 5, 8, 10, 11, 13, 16]    --> ;
L1 = [1, 3, 6, 9, 10, 11, 12, 16]
L2 = [2, 4, 5, 7, 8, 13, 14, 15]     --> ;
L1 = [1, 3, 6, 8, 10, 12, 13, 15]
L2 = [2, 4, 5, 7, 9, 11, 14, 16]     --> ;
L1 = [1, 4, 6, 7, 10, 11, 13, 16]
L2 = [2, 3, 5, 8, 9, 12, 14, 15]     --> ;
L1 = [1, 2, 7, 8, 11, 12, 13, 14]
L2 = [3, 4, 5, 6, 9, 10, 15, 16]     --> ;
no

?- findall((L1,L2), numpart(20, L1, L2), L), length(L, N).
.............................
N = 24

?- findall((L1,L2), numpart(24, L1, L2), L), length(L, N).
.............................
N = 296

?- findall((L1,L2), numpart(28, L1, L2), L), length(L, N).
.............................
```

```
N = 1443

?- findall((L1,L2), numpart(32, L1, L2), L), length(L, N).
.............................
N = 17444

?- findall((L1,L2), numpart(36, L1, L2), L), length(L, N).[1]
.............................
N = 138905
```

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

<u>Άσκηση 5</u>

Στην προτασιακή λογική, ένας τύπος είναι σε *συζευκτική κανονική μορφή* ή ΣΚΜ (conjunctive normal form - CNF), όταν είναι μία σύζευξη διαζεύξεων, όπου κάθε διάζευξη, που ονομάζεται και *πρόταση* (clause), περιέχει *λεκτικά* (literals), δηλαδή προτασιακά σύμβολα ή αρνήσεις προτασιακών συμβόλων. Για παράδειγμα, ο ακόλουθος τύπος είναι σε ΣΚΜ:

$$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee x_3) \wedge (\neg x_3 \vee x_4)$$

Το πρόβλημα της *μέγιστης ικανοποιησιμότητας* (maximum satisfiability - MAXSAT) έγκειται στην εύρεση κατάλληλων αναθέσεων τιμών (true ή false) στα προτασιακά σύμβολα έτσι ώστε να μεγιστοποιείται το πλήθος των προτάσεων του τύπου που αληθεύουν. Στον παραπάνω τύπο, που αποτελείται από επτά προτάσεις, δεν υπάρχει ανάθεση τιμών στα προτασιακά σύμβολα που να κάνει αληθείς όλες τις προτάσεις, αλλά μπορεί να βρεθεί ανάθεση ώστε να αληθεύουν έξι από αυτές (αναθέτοντας, για παράδειγμα, σε όλα τα προτασιακά σύμβολα την τιμή true).

Αντικείμενο της άσκησης αυτής είναι να επιλυθεί το πρόβλημα της μέγιστης ικανοποιησιμότητας ως πρόβλημα βελτιστοποίησης μέσω της τεχνικής του λογικού προγραμματισμού με περιορισμούς και της μεθόδου «διακλάδωσε-και-φράξε». Ως προς την αναπαράσταση σε Prolog ενός τύπου σε ΣΚΜ, αυτή θα γίνει μέσω μίας λίστας από λίστες, όπου κάθε εσωτερική λίστα κωδικοποιεί μία πρόταση περιλαμβάνοντας ως στοιχεία τους δείκτες των εμπλεκόμενων προτασιακών συμβόλων, με θετικό πρόσημο, αν είναι χωρίς άρνηση στην πρόταση και με αρνητικό, αν έχουν άρνηση. Δηλαδή, ο παραπάνω τύπος θα μπορούσε να αναπαρασταθεί ως:

```
[[1,-2,4],[-1,2],[-1,-2,3],[-2,-4],[2,-3],[1,3],[-3,4]]
```

Για την αντιμετώπιση του προβλήματος αυτού, θα χρησιμοποιήσετε τύπους που κατασκευάζονται μέσω του κατηγορήματος `create_formula(NV, NC, D, F)`, όπως αυτό ορίζεται στο αρχείο http://www.di.uoa.gr/~takis/randms.pl. Κατά την κλήση του κατηγορήματος, δίνεται το πλήθος `NV` των μεταβλητών του τύπου, το πλήθος `NC` των προτάσεών του και η πυκνότητά του `D` (σαν ποσοστό των μεταβλητών που περιέχονται σε μία πρόταση, κατά μέσο όρο, σε σχέση με το συνολικό πλήθος

[1] Ενδεχομένως να απαιτείται αρκετός χρόνος για να απαντηθεί αυτό το ερώτημα.

των μεταβλητών του τύπου) και επιστρέφεται ο τύπος `F` στη μορφή που περιγράφηκε προηγουμένως. Ένα παράδειγμα εκτέλεσης του κατηγορήματος αυτού είναι το εξής:[2]

```
?- seed(1), create_formula(5, 7, 40, F).
F = [[-4, -5], [-3, -4], [2, -3, 4], [-2, -3, -4, 5], [1],
     [2, -3, 4], [-1, 2, 4]]
```

Για την άσκηση αυτή, θα πρέπει να ορίσετε ένα κατηγόρημα `maxsat/6`, το οποίο όταν καλείται σαν `maxsat(NV, NC, D, F, S, M)`, αφού δημιουργήσει έναν τύπο `F` με `NV` μεταβλητές, `NC` προτάσεις και πυκνότητας προτάσεων `D`, καλώντας το κατηγόρημα `create_formula/4`, να βρίσκει μία βέλτιστη λύση του προβλήματος της μέγιστης ικανοποιησιμότητας του τύπου, επιστρέφοντας στη μεταβλητή `S` τη λύση, σαν μία λίστα από `1` και `0` (`1` = true, `0` = false), και στη μεταβλητή `M` το (βέλτιστο) πλήθος προτάσεων του τύπου που είναι αληθείς. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:[3]

```
?- seed(1), maxsat(5, 20, 30, F, S, M).[4]
F = [[-4], [-4, -5], [3, -4], [-1], [-2, 5], [], [1, 4],
     [3, 5], [], [4], [1, 2, -4, -5], [-3], [-4], [-3], [],
     [1], [-5], [1, -5], [], [1, -4]]
S = [1, 0, 0, 0, 0]
M = 13

?- seed(10), maxsat(10, 50, 25, F, S, M).
F = [[-2, -4, 8], [4], [3], [5, 8], [2, -5, 6], ...]
S = [1, 0, 1, 1, 0, 0, 0, 1, 0, 1]
M = 48

?- seed(123), maxsat(20, 80, 20, F, S, M).
F = [[-2, 9, -13, 14], [3, -11, -14, -15, 16, -20], ...]
S = [0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, ...]
M = 76

?- seed(7777), maxsat(30, 100, 15, F, S, M).
F = [[-16, -18, 20, -29], [8, -11, -14, 25], ...]
S = [0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, ...]
M = 100

?- seed(1000), maxsat(40, 120, 10, F, S, M).
F = [[5, -6, -16, -38], [3, 6, -19, -24, 30], ...]
S = [1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, ...]
```

[2] 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.

[3] Θα πρέπει να σημειωθεί ότι για δεδομένο τύπο μπορεί να υπάρχουν περισσότερες από μία λύσεις με το ίδιο μέγιστο πλήθος `M` προτάσεων που αληθεύουν. Αυτό σημαίνει ότι το δικό σας πρόγραμμα ενδεχομένως να μην βρίσκει την ίδια λύση `S` με αυτή που φαίνεται στις ενδεικτικές εκτελέσεις, αλλά θα πρέπει σίγουρα να έχει το ίδιο `M`.

[4] Σημειώστε ότι το κατηγόρημα `create_formula/4` επιστρέφει τυχαίους τύπους, οπότε δεν αποκλείεται οι τύποι αυτοί να περιέχουν την ίδια πρόταση παραπάνω από μία φορά. Αυτό δεν είναι πρόβλημα. Ακόμα και επαναλαμβανόμενες προτάσεις στον τύπο θεωρούνται ως διακριτές μεταξύ τους για το πρόβλημα της μέγιστης ικανοποιησιμότητας. Επίσης, ο τύπος μπορεί να περιέχει και κενή πρόταση (ή κενές προτάσεις). Ούτε αυτό είναι πρόβλημα. Στη λογική, η κενή πρόταση είναι ψευδής.

```
M = 119

?- seed(567), maxsat(50, 250, 9, F, S, M).
F = [[-12, -15, 27, 32, -35, 36], [-7, 10, 13, 14], ...]
S = [1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, ...]
M = 245

?- seed(2023), maxsat(100, 260, 5, F, S, M).
F = [[-92], [4, 48, -51, 54, -97], ...]
S = [1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, ...]
M = 257
```

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

<u>Άσκηση 6</u>

Στην άσκηση αυτή καλείστε να επιλύσετε μέσω προγραμματισμού με περιορισμούς διάφορα στιγμιότυπα, κυμαινόμενης δυσκολίας, του γρίφου των ουρανοξυστών (https://www.puzzle-skyscrapers.com/). Στον γρίφο αυτό, έχουμε ένα τετράγωνο πλέγμα διάστασης *N*, σε κάθε κελί του οποίου πρέπει να τοποθετηθεί ένας ουρανοξύστης ακέραιου ύψους από 1 έως *N*, έτσι ώστε σε κάθε γραμμή και κάθε στήλη του πλέγματος να μην υπάρχουν δύο ουρανοξύστες ίδιου ύψους. Επίσης, στα άκρα των γραμμών και των στηλών του πλέγματος (σε όλα ή μερικά μόνο από αυτά) δίνονται και κάποιοι ακέραιοι αριθμοί που περιορίζουν το πλήθος των ουρανοξυστών που είναι ορατοί από το σημείο αυτό σε όλο το μήκος της γραμμής ή της στήλης μέχρι το τέλος της. Θεωρείται ότι ένας ουρανοξύστης αποκρύπτει όλους εκείνους που είναι πίσω από αυτόν και έχουν μικρότερο ύψος. Τέλος, είναι πιθανόν κάποια από τα κελιά του πλέγματος να είναι ήδη συμπληρωμένα εξ αρχής.

Ας θεωρήσουμε τον γρίφο-πλέγμα που φαίνεται στο παρακάτω σχήμα αριστερά. Πρέπει να συμπληρωθούν τα κελιά του πλέγματος (πλην αυτού στην $3^{η}$ γραμμή και $2^{η}$ στήλη που είναι ήδη συμπληρωμένο με το 5) με ακεραίους από το 1 έως το 5, που παριστάνουν τα ύψη των ουρανοξυστών στο κάθε κελί, έτσι ώστε σε κάθε γραμμή και κάθε στήλη να υπάρχουν διαφορετικοί αριθμοί και κάθε αριθμός που βρίσκεται έξω από τις πλευρές να είναι συμβατός με το πλήθος των ουρανοξυστών που είναι ορατοί από το σημείο αυτό μέχρι την άλλη πλευρά της γραμμής ή της στήλης. Για παράδειγμα, το 2 που είναι έξω από τη $2^{η}$ γραμμή αριστερά σημαίνει ότι από το σημείο αυτό, θα πρέπει να είναι ορατοί ακριβώς 2 ουρανοξύστες μέχρι το τέλος της $2^{ης}$ γραμμής δεξιά. Μία λύση του γρίφου αυτού φαίνεται στο παρακάτω σχήμα δεξιά.

						4
2						
		5				2
2						
4						
		3		2		

	5	4	3	1	2	4
2	2	1	5	4	3	
	3	5	1	2	4	2
2	4	3	2	5	1	
4	1	2	4	3	5	
	3		2			

Ο προηγούμενος γρίφος θα μπορούσε να αναπαρασταθεί μέσω ενός γεγονότος Prolog της μορφής:

```
puzzle(demo, 5,
       [0,2,0,2,4], [4,0,2,0,0],
       [0,0,0,0,0], [0,3,0,2,0],
       [[_,_,_,_,_],
        [_,_,_,_,_],
        [_,5,_,_,_],
        [_,_,_,_,_],
        [_,_,_,_,_]]).
```

Το πρώτο όρισμα του κατηγορήματος `puzzle/7` είναι ένα αναγνωριστικό για το συγκεκριμένο στιγμιότυπου του γρίφου. Το 2ο όρισμα είναι η διάσταση του πλέγματος. Το 3ο και το 4ο όρισμα είναι τα πλήθη των ορατών ουρανοξυστών σε κάθε γραμμή από αριστερά προς τα δεξιά και από δεξιά προς τα αριστερά, αντίστοιχα (αν δεν υπάρχει περιορισμός πλήθους, η αντίστοιχη τιμή είναι 0). Το 5ο και το 6ο όρισμα είναι τα πλήθη των ορατών ουρανοξυστών σε κάθε στήλη από επάνω προς τα κάτω και από κάτω προς τα επάνω, αντίστοιχα (ομοίως, αν δεν υπάρχει περιορισμός πλήθους, η αντίστοιχη τιμή είναι 0). Τέλος, το 7ο όρισμα είναι ένα πρότυπο (template) της λύσης, ως μία λίστα από λίστες για την αναπαράσταση του δισδιάστατου πλέγματος κατά γραμμές, με προ-συμπληρωμένα ενδεχομένως κάποια κελιά. Στο αρχείο http://www.di.uoa.gr/~takis/skyscr_data.pl μπορείτε να βρείτε μία σειρά από γεγονότα `puzzle/7` για διάφορα στιγμιότυπα του γρίφου.

Ορίστε στο σύστημα λογικού προγραμματισμού της επιλογής σας ένα κατηγόρημα `skyscr/2`, το οποίο όταν καλείται σαν `skyscr(PuzzleId, Solution)`, να επιστρέφει στο `Solution` τη λύση του γρίφου `PuzzleId`. Παραδείγματα εκτέλεσης:

```
?- skyscr(demo, Solution).
Solution = [[5, 4, 3, 1, 2],
            [2, 1, 5, 4, 3],
            [3, 5, 1, 2, 4],
            [4, 3, 2, 5, 1],
            [1, 2, 4, 3, 5]]

?- skyscr(easy4x4, Solution).
Solution = [[2, 1, 3, 4],
            [3, 4, 2, 1],
            [4, 3, 1, 2],
            [1, 2, 4, 3]]

?- skyscr(normal4x4, Solution).
Solution = [[4, 2, 3, 1],
            [1, 3, 2, 4],
            [3, 4, 1, 2],
            [2, 1, 4, 3]]

?- skyscr(hard4x4, Solution).
Solution = [[1, 3, 2, 4],
            [3, 4, 1, 2],
```

```
            [2, 1, 4, 3],
            [4, 2, 3, 1]]

?- skyscr(easy5x5, Solution).
Solution = [[1, 2, 3, 4, 5],
            [3, 5, 4, 2, 1],
            [4, 1, 2, 5, 3],
            [5, 4, 1, 3, 2],
            [2, 3, 5, 1, 4]]

?- skyscr(normal5x5, Solution).
Solution = [[2, 3, 4, 5, 1],
            [1, 2, 5, 4, 3],
            [4, 1, 2, 3, 5],
            [3, 5, 1, 2, 4],
            [5, 4, 3, 1, 2]]

?- skyscr(hard5x5, Solution).
Solution = [[1, 5, 3, 2, 4],
            [4, 1, 5, 3, 2],
            [2, 3, 1, 4, 5],
            [5, 4, 2, 1, 3],
            [3, 2, 4, 5, 1]]

?- skyscr(easy6x6, Solution).
Solution = [[5, 3, 2, 1, 4, 6],
            [4, 2, 1, 5, 6, 3],
            [1, 4, 6, 2, 3, 5],
            [6, 5, 3, 4, 1, 2],
            [2, 6, 4, 3, 5, 1],
            [3, 1, 5, 6, 2, 4]]

?- skyscr(normal6x6, Solution).
Solution = [[3, 6, 4, 2, 5, 1],
            [2, 3, 1, 5, 6, 4],
            [5, 2, 6, 1, 4, 3],
            [1, 4, 3, 6, 2, 5],
            [6, 1, 5, 4, 3, 2],
            [4, 5, 2, 3, 1, 6]]

?- skyscr(hard6x6, Solution).
Solution = [[1, 3, 5, 4, 2, 6],
            [5, 1, 2, 6, 4, 3],
            [2, 6, 4, 1, 3, 5],
            [6, 4, 3, 5, 1, 2],
            [4, 2, 6, 3, 5, 1],
            [3, 5, 1, 2, 6, 4]]

?- skyscr(hard7x7, Solution).
Solution = [[4, 1, 3, 2, 7, 5, 6],
            [5, 3, 1, 6, 4, 7, 2],
            [1, 5, 6, 4, 2, 3, 7],
            [7, 6, 2, 1, 5, 4, 3],
            [3, 2, 5, 7, 6, 1, 4],
            [6, 4, 7, 3, 1, 2, 5],
            [2, 7, 4, 5, 3, 6, 1]]
```

```
?- skyscr(hard8x8, Solution).
Solution = [[1, 7, 5, 2, 4, 3, 6, 8],
            [8, 4, 3, 1, 6, 7, 2, 5],
            [2, 3, 6, 4, 5, 8, 7, 1],
            [7, 2, 1, 5, 8, 6, 3, 4],
            [4, 8, 7, 6, 3, 5, 1, 2],
            [3, 1, 4, 8, 7, 2, 5, 6],
            [5, 6, 8, 7, 2, 1, 4, 3],
            [6, 5, 2, 3, 1, 4, 8, 7]]

?- skyscr(hard9x9, Solution).
Solution = [[8, 3, 1, 9, 4, 5, 7, 2, 6],
            [2, 1, 9, 5, 6, 7, 3, 8, 4],
            [3, 8, 4, 6, 1, 2, 9, 7, 5],
            [7, 5, 8, 4, 2, 6, 1, 9, 3],
            [9, 2, 6, 8, 5, 3, 4, 1, 7],
            [1, 6, 7, 2, 3, 8, 5, 4, 9],
            [6, 7, 5, 1, 9, 4, 2, 3, 8],
            [4, 9, 3, 7, 8, 1, 6, 5, 2],
            [5, 4, 2, 3, 7, 9, 8, 6, 1]]
```

Παραδοτέο για την άσκηση είναι **ένα πηγαίο αρχείο Prolog** με όνομα **`skyscr.pl`**, μέσα στο οποίο **δεν** θα πρέπει να περιέχονται γεγονότα `puzzle/7`.

Άσκηση 7

Επιλύστε το πρόβλημα της άσκησης 2 της Α' ομάδας μέσω της τεχνικής του προγραμματισμού με περιορισμούς, υλοποιώντας ένα κατηγόρημα `assignment_csp/4` με την ίδια λειτουργικότητα με το `assignment/4` της άσκησης 2. Δηλαδή, θεωρώντας πάλι δεδομένο ένα σύνολο γεγονότων `activity/2` που κωδικοποιούν δραστηριότητες, καλώντας το `assignment_csp(NP, MT, ASP, ASA)`, όπου `NP` είναι το πλήθος των διαθέσιμων ατόμων και `MT` είναι ο μέγιστος συνολικός χρόνος των δραστηριοτήτων που μπορεί να αναλάβει ένα άτομο, να ανατίθενται οι δοθείσες δραστηριότητες στα δοθέντα άτομα με εφικτό τρόπο, επιστρέφοντας στις μεταβλητές `ASP` και `ASA` τις αναπαραστάσεις της ανάθεσης που πραγματοποιήθηκε, όπως αυτές περιγράφονται στην εκφώνηση της άσκησης 2. Επιβεβαιώστε ότι τα αποτελέσματα που παίρνετε με το νέο πρόγραμμα είναι ισοδύναμα με αυτά του προγράμματος που παραδώσατε για την άσκηση 2.[5]

Στη συνέχεια, δοκιμάστε και τα ακόλουθα ερωτήματα με βάση τα γεγονότα `activity/2` του αρχείου http://www.di.uoa.gr/~takis/activity_big.pl και ελέγξτε αν το πρόγραμμα που παραδώσατε για την άσκηση 2 είναι σε θέση να απαντήσει σε εύλογο χρόνο τα ερωτήματα αυτά.

```
?- assignment_csp(13, 50, ASP, ASA).

ASP = [1 - [a001, a015, a018, a029, a049, a056, a075, a077,
            a098] - 40,
```

[5] Με βάση τα δεδομένα του αρχείου http://www.di.uoa.gr/~takis/activity.pl.

```
        2 - [a002, a003, a017, a051, a052, a067, a082] - 43,
        3 - [a004, a036, a044, a057, a083, a086, a087, a088,
             a091] - 44,
        4 - [a005, a028, a032, a071, a076, a080, a085, a100]
             - 45,
        5 - [a006, a039, a059, a068, a073, a084, a089, a099]
             - 43,
        6 - [a007, a025, a033, a045, a070] - 41,
        7 - [a008, a022, a023, a040, a042, a053, a063, a065,
             a090] - 42,
        8 - [a009, a021, a031, a035, a079, a092, a094] - 41,
        9 - [a010, a012, a019, a024, a026, a034, a041, a060,
             ...] - 45,
       10 - [a011, a046, a061, a081, a093, a096, a097] - 43,
       11 - [a013, a016, a020, a047, a054, a058, ...] - 46,
       12 - [a014, a030, a037, a050, a072, ...] - 41,
       13 - [a027, a038, a043, a048, ...] - 41]
ASA = [a001 - 1, a002 - 2, a003 - 2, a004 - 3, a005 - 4,
       a006 - 5, a007 - 6, a008 - 7, a009 - 8, a010 - 9,
       a011 - 10, a012 - 9, a013 - 11, a014 - 12, a015 - 1,
       a016 - 11, a017 - 2, a018 - 1, ... - ..., ...]    --> ;
.......................

?- assignment_csp(12, 52, ASP, ASA).
ASP = [1 - [a001, a011, a029, a044, a046, a057, a077, a086,
            a088, a098] - 47,
       2 - [a002, a003, a022, a033, a039, a045, a067, a083,
            a084, a095] - 47,
       3 - [a004, a021, a036, a048, a052, a082, a097] - 47,
       4 - [a005, a017, a024, a028, a050, a055, a056, a072]
            - 45,
       5 - [a006, a049, a068, a075, a081, a085, a091, a099]
            - 43,
       6 - [a007, a020, a025, a027, a058, a074] - 49,
       7 - [a008, a010, a019, a040, a042, a069, a070, a100]
            - 48,
       8 - [a009, a026, a032, a051, a054, a064, a080, a092]
            - 45,
       9 - [a012, a037, a047, a065, a066, a073, a078, a090]
            - 46,
      10 - [a013, a015, a016, a018, a030, a041, a053, ...]
            - 44,
      11 - [a014, a031, a038, a059, a076, a079, ...] - 46,
      12 - [a023, a034, a035, a043, a061, ...] - 48]
ASA = [a001 - 1, a002 - 2, a003 - 2, a004 - 3, a005 - 4,
       a006 - 5, a007 - 6, a008 - 7, a009 - 8, a010 - 7,
       a011 - 1, a012 - 9, a013 - 10, a014 - 11, a015 - 10,
       a016 - 10, a017 - 4, a018 - 10, ... - ..., ...]   --> ;
.......................
```

Τέλος, θεωρήστε και μία επέκταση του προβλήματος, με την εισαγωγή και μίας συνάρτησης κόστους που ποσοτικοποιεί την ισοκατανομή του χρόνου εργασίας μεταξύ των ατόμων. Ορίζουμε το κόστος μίας ανάθεσης ως εξής:

$$\sum_{i=1}^{NP}(A - W_i)^2$$

όπου NP είναι το πλήθος των ατόμων, W_i είναι ο συνολικός χρόνος εργασίας του ατόμου i, δηλαδή το άθροισμα των διαρκειών των δραστηριοτήτων που του έχουν ανατεθεί, και $A = rnd(\frac{D}{NP})$, με D να είναι η συνολική διάρκεια των δραστηριοτήτων προς ανάθεση και $rnd()$ η συνάρτηση στρογγύλευσης πραγματικού αριθμού στον πλησιέστερο ακέραιο.

Αντιμετωπίστε το πρόβλημα της ανάθεσης δραστηριοτήτων σε άτομα και ως πρόβλημα βελτιστοποίησης, με στόχο την εύρεση λύσης που ελαχιστοποιεί το κόστος, όπως ορίστηκε πριν. Συγκεκριμένα, υλοποιήστε ένα κατηγόρημα `assignment_opt/8`, το οποίο να καλείται ως `assignment_opt(NF, NP, MT, F, T, ASP, ASA, Cost)`, με τα ορίσματα `NP`, `MT`, `ASP` και `ASA` να έχουν την ίδια σημασία με τα αντίστοιχα του `assignment/4` της άσκησης 2. Με την κλήση του κατηγορήματος αυτού θα πρέπει να βρίσκεται βέλτιστη λύση (με τις επιφυλάξεις που θα εξηγηθούν στη συνέχεια) ανάθεσης των δραστηριοτήτων που περιγράφονται από τα πρώτα `NF` γεγονότα `activity/2` που έχουν δοθεί. Αν το `NF` ισούται με `0`, τότε ανατίθενται όλες οι δραστηριότητες. Όταν δίνεται στην παράμετρο `F` τιμή μικρότερη από το `1.0`, η ερμηνεία της είναι ότι σε κάθε νέα επανάληψη της μεθόδου «διακλάδωσε-και-φράξε» αναζητείται λύση καλύτερη από `F·C`, όπου `C` είναι το κόστος της λύσης που βρέθηκε στην προηγούμενη επανάληψη, οπότε η λύση που θα βρεθεί δεν είναι εγγυημένα η βέλτιστη, αλλά σίγουρα το κόστος της βέλτιστης βρίσκεται μέσα στο διάστημα `[F·C,C]`, όπου `C` είναι το κόστος της τελευταίας λύσης που βρέθηκε. Η παράμετρος `T` είναι ο χρόνος (σε CPU secs) που διατίθεται για την εύρεση της βέλτιστης λύσης. Αν παρέλθει ο χρόνος αυτός, επιστρέφεται η καλύτερη λύση που έχει βρεθεί μέχρι αυτή τη στιγμή, οπότε πάλι δεν είναι εγγυημένο ότι βρέθηκε η βέλτιστη λύση. Αν δοθεί για `T` το `0`, τότε δεν υπάρχει περιορισμός χρόνου. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:[6]

```
?- assignment_opt(10, 5, 15, 1.0, 0, ASP, ASA, Cost).
Found a solution with cost 21
Found a solution with cost 17
Found a solution with cost 9
Found a solution with cost 7
Found a solution with cost 5
Found no solution with cost 1.0 .. 4.0

ASP = [1 - [a001, a002, a008] - 11,
       2 - [a004] - 9,
       3 - [a007, a009] - 11,
       4 - [a003, a006] - 11,
       5 - [a005, a010] - 12]
ASA = [a001 - 1, a002 - 1, a003 - 4, a004 - 2, a005 - 5,
       a006 - 4, a007 - 3, a008 - 1, a009 - 3, a010 - 5]
Cost = 5
```

[6] Το παρακάτω ερωτήματα απαντήθηκαν με βάση τα γεγονότα `activity/2` στο αρχείο http://www.di.uoa.gr/~takis/activity_big.pl. Αν επιθυμείτε να δημιουργήσετε και άλλα δεδομένα με τυχαία γεγονότα `activity/2`, μπορείτε να χρησιμοποιήσετε το C πρόγραμμα http://www.di.uoa.gr/~takis/randass.c.

```
?- assignment_opt(20, 5, 25, 1.0, 0, ASP, ASA, Cost).
Found a solution with cost 16
Found a solution with cost 8
Found a solution with cost 6
Found a solution with cost 4
Found a solution with cost 2

ASP = [1 - [a001, a019, a020] - 23,
       2 - [a004, a006, a016] - 22,
       3 - [a010, a011, a013, a014] - 22,
       4 - [a005, a007, a012, a015] - 23,
       5 - [a002, a003, a008, a009, a017, a018] - 23]
ASA = [a001 - 1, a002 - 5, a003 - 5, a004 - 2, a005 - 4,
       a006 - 2, a007 - 4, a008 - 5, a009 - 5, a010 - 3,
       a011 - 3, a012 - 4, a013 - 3, a014 - 3, a015 - 4,
       a016 - 2, a017 - 5, a018 - 5, ... - ..., ...]
Cost = 2

?- assignment_opt(30, 4, 50, 1.0, 0, ASP, ASA, Cost).
Found a solution with cost 19
Found a solution with cost 15
Found a solution with cost 13
Found a solution with cost 11
Found a solution with cost 5
Found a solution with cost 3
Found a solution with cost 1

ASP = [1 - [a001, a005, a007, a011, a012, a019, a024] - 42,
       2 - [a002, a006, a008, a009, a015, a021, a023, a025,
            a027] - 42,
       3 - [a010, a013, a014, a020, a022, a026, a029, a030]
            - 42,
       4 - [a003, a004, a016, a017, a018, a028] - 43]
ASA = [a001 - 1, a002 - 2, a003 - 4, a004 - 4, a005 - 1,
       a006 - 2, a007 - 1, a008 - 2, a009 - 2, a010 - 3,
       a011 - 1, a012 - 1, a013 - 3, a014 - 3, a015 - 2,
       a016 - 4, a017 - 4, a018 - 4, ... - ..., ...]
Cost = 1

?- assignment_opt(50, 7, 60, 1.0, 0, ASP, ASA, Cost).
Found a solution with cost 25
Found a solution with cost 21
Found a solution with cost 19
Found a solution with cost 17
Found a solution with cost 11
Found a solution with cost 9
Found a solution with cost 7
Found a solution with cost 5
Found a solution with cost 3
Found a solution with cost 1

ASP = [1 - [a001, a009, a015, a016, a034, a035, a043, a049]
            - 40,
       2 - [a006, a011, a019, a024, a037, a038, a047, a050]
            - 40,
```

```
       3 - [a005, a014, a021, a029, a030, a036, a048] - 40,
       4 - [a003, a008, a020, a028, a033, a039, a045] - 40,
       5 - [a002, a013, a023, a025, a040, a042, a044] - 39,
       6 - [a007, a010, a012, a017, a026, a032, a041] - 40,
       7 - [a004, a018, a022, a027, a031, a046] - 40]
ASA = [a001 - 1, a002 - 5, a003 - 4, a004 - 7, a005 - 3,
       a006 - 2, a007 - 6, a008 - 4, a009 - 1, a010 - 6,
       a011 - 2, a012 - 6, a013 - 5, a014 - 3, a015 - 1,
       a016 - 1, a017 - 6, a018 - 7, ... - ..., ...]
Cost = 1

?- assignment_opt(70, 10, 80, 1.0, 20, ASP, ASA, Cost).
Found a solution with cost 76
Found a solution with cost 60
Found a solution with cost 50
Found a solution with cost 48
Found a solution with cost 44
Found a solution with cost 42
Found a solution with cost 38
Found a solution with cost 36
Found a solution with cost 32
Found a solution with cost 30
Found a solution with cost 26
Found a solution with cost 22
Found a solution with cost 20
Found a solution with cost 18
Found a solution with cost 16
Found a solution with cost 12
Found a solution with cost 10
Found a solution with cost 8
Found a solution with cost 6
Branch-and-bound timeout while searching for solution
better than 6

ASP = [1 - [a001, a009, a021, a034, a035, a043, a045, a057]
           - 38,
       2 - [a006, a020, a024, a026, a036, a064, a070] - 37,
       3 - [a005, a014, a017, a022, a032, a046] - 38,
       4 - [a028, a031, a033, a055, a066, a067, a068] - 37,
       5 - [a002, a015, a025, a041, a042, a058, a065] - 38,
       6 - [a007, a008, a010, a011, a029, a050, a052, a060]
           - 38,
       7 - [a003, a030, a038, a044, a051, a061, a063, a069]
           - 39,
       8 - [a023, a027, a040, a048, a056] - 37,
       9 - [a004, a016, a018, a039, a047, a053, a054, a059]
           - 37,
      10 - [a012, a013, a019, a037, a049, a062] - 37]
ASA = [a001 - 1, a002 - 5, a003 - 7, a004 - 9, a005 - 3,
       a006 - 2, a007 - 6, a008 - 6, a009 - 1, a010 - 6,
       a011 - 6, a012 - 10, a013 - 10, a014 - 3, a015 - 5,
       a016 - 9, a017 - 3, a018 - 9, ... - ..., ...]
Cost = 6

?- assignment_opt(80, 10, 65, 0.8, 0, ASP, ASA, Cost).
Found a solution with cost 57
```

```
Found a solution with cost 45
Found a solution with cost 35
Found a solution with cost 27
Found a solution with cost 21
Found a solution with cost 17
Found a solution with cost 13
Found a solution with cost 9
Found a solution with cost 7
Found a solution with cost 5
Found a solution with cost 3
Found a solution with cost 1

ASP = [1 - [a001, a019, a034, a047, a053, a056, a065, a069,
           a076] - 44,
       2 - [a006, a013, a016, a050, a055, a064, a070, a080]
           - 44,
       3 - [a002, a014, a017, a025, a026, a044, a051] - 44,
       4 - [a004, a021, a028, a033, a038, a039, a052, a072]
           - 44,
       5 - [a005, a022, a040, a041, a042, a048, a062, a066]
           - 44,
       6 - [a003, a007, a009, a011, a029, a060, a071, a074,
           a077] - 44,
       7 - [a023, a024, a030, a035, a058, a061, a063] - 44,
       8 - [a027, a032, a036, a073, a075, a078] - 45,
       9 - [a008, a015, a018, a020, a031, a046, a049, a054,
           ...] - 44,
      10 - [a010, a012, a037, a043, a045, a057, a059, ...]
           - 44]
ASA = [a001 - 1, a002 - 3, a003 - 6, a004 - 4, a005 - 5,
       a006 - 2, a007 - 6, a008 - 9, a009 - 6, a010 - 10,
       a011 - 6, a012 - 10, a013 - 2, a014 - 3, a015 - 9,
       a016 - 2, a017 - 3, a018 - 9, ... - ..., ...]
Cost = 1

?- assignment_opt(0, 12, 100, 0.9, 10, ASP, ASA, Cost).
Found a solution with cost 61
Found a solution with cost 53
Found a solution with cost 45
Found a solution with cost 39
Found a solution with cost 35
Found a solution with cost 31
Branch-and-bound timeout while searching for solution
better than 31

ASP = [1 - [a001, a017, a019, a020, a024, a026, a047, a053,
           a067] - 49,
       2 - [a006, a008, a009, a032, a036, a039, a041, a048,
           a064, a068, a069, a098] - 45,
       3 - [a004, a014, a022, a031, a046, a049, a079, a083]
           - 45,
       4 - [a028, a034, a062, a066, a073, a075, a076] - 47,
       5 - [a002, a025, a029, a033, a040, a042, a052, a077,
           a084, a085] - 46,
       6 - [a005, a051, a060, a071, a087, a088, a090, a093,
           a094] - 48,
```

```
        7 - [a012, a018, a021, a035, a043, a045, a057, a061,
             a063, a100] - 45,
        8 - [a013, a016, a030, a058, a072, a095, a099] - 49,
        9 - [a003, a056, a059, a078, a081, a086, a091] - 46,
       10 - [a007, a027, a038, a044, a055, a074, a080] - 46,
       11 - [a010, a023, a054, a065, a070, a092, ...] - 44,
       12 - [a011, a015, a037, a050, a082, ...] - 45]
ASA = [a001 - 1, a002 - 5, a003 - 9, a004 - 3, a005 - 6,
       a006 - 2, a007 - 10, a008 - 2, a009 - 2, a010 - 11,
       a011 - 12, a012 - 7, a013 - 8, a014 - 3, a015 - 12,
       a016 - 8, a017 - 1, a018 - 7, ... - ..., ...]
Cost = 31
```

Και κάποιες εκτελέσεις με αρκετά μεγαλύτερο σύνολο δεδομένων εισόδου.[7]

```
?- assignment_opt(100, 50, 200, 1.0, 60, ASP, ASA, Cost).
Found a solution with cost 4203
Found a solution with cost 4201
Found a solution with cost 4177
Branch-and-bound timeout while searching for solution
better than 4177

ASP = ...
ASA = ...
Cost = 4177

?- assignment_opt(200, 100, 100, 0.9, 120, ASP, ASA, Cost).
Found a solution with cost 8354
Branch-and-bound timeout while searching for solution
better than 8354

ASP = ....
ASA = ....
Cost = 8354

?- assignment_opt(0, 120, 80, 0.8, 180, ASP, ASA, Cost).
Found a solution with cost 4673
Branch-and-bound timeout while searching for solution
better than 4673

ASP = ....
ASA = ....
Cost = 4673
```

Παραδοτέο για την άσκηση είναι **ένα πηγαίο αρχείο Prolog** με όνομα **`assignment_csp.pl`**, μέσα στο οποίο **δεν** θα πρέπει να περιέχονται γεγονότα `activity/2`.

[7] Από το αρχείο http://www.di.uoa.gr/~takis/activity_very_big.pl.