ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Β΄ Ομάδα Ασκήσεων "Λογικού Προγραμματισμού" Ακαδημαϊκού Έτους 2017-18

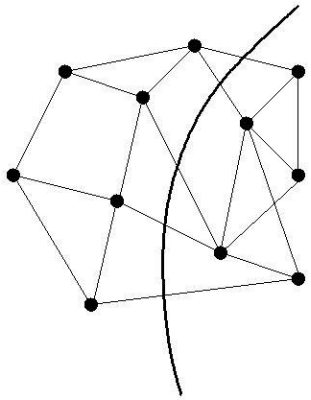
Οι ασκήσεις της ομάδας αυτής πρέπει να αντιμετωπισθούν με τη βοήθεια της τεχνολογίας του **προγραμματισμού με περιορισμούς**. Ένα σύστημα λογικού προγραμματισμού που υποστηρίζει την τεχνολογία αυτή είναι η **ECL^{i}PS^e** (http://www.eclipseclp.org), μέσω της βιβλιοθήκης **`ic`**, η οποία τεκμηριώνεται 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, δεν απαγορεύεται να χρησιμοποιήσετε 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.

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

Μία εκδοχή του προβλήματος της *διαμέρισης γράφου* (graph partitioning) είναι να διασπάσουμε ένα γράφο σε δύο υπογράφους, με πλήθη κόμβων στους δύο υπογράφους ίσα ή να διαφέρουν κατά ένα. Το κόστος μίας διαμέρισης ισούται με το πλήθος των ακμών του αρχικού γράφου που πρέπει να διασπασθούν για να γίνει η διαμέριση. Για παράδειγμα, το κόστος της διαμέρισης που φαίνεται στο διπλανό σχήμα ισούται με 5. Το ζητούμενο είναι να βρεθεί διαμέριση με ελάχιστο κόστος.



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 `grpart/5`, το οποίο όταν καλείται σαν `grpart(N, D, P1, P2, Cost)`, αφού δημιουργήσει ένα γράφο `N` κόμβων και πυκνότητας `D`, καλώντας το κατηγόρημα `create_graph/3`, να βρίσκει μία διαμέριση του γράφου με ελάχιστο κόστος, επιστρέφοντας στα `P1` και `P2` τις λίστες των κόμβων των δύο υπογράφων μετά τη διαμέριση και στο `Cost` το κόστος της διαμέρισης. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:[2]

```
?- seed(1), grpart(10, 100, P1, P2, Cost).
P1 = [1, 2, 3, 4, 5]
P2 = [6, 7, 8, 9, 10]
Cost = 25

?- seed(2018), grpart(15, 80, P1, P2, Cost).
P1 = [1, 2, 5, 8, 9, 11, 13, 14]
P2 = [3, 4, 6, 7, 10, 12, 15]
Cost = 38

?- seed(100), grpart(20, 70, P1, P2, Cost).
P1 = [1, 2, 6, 7, 8, 10, 12, 13, 15, 17]
P2 = [3, 4, 5, 9, 11, 14, 16, 18, 19, 20]
Cost = 58

?- seed(1234), grpart(25, 50, P1, P2, Cost).
P1 = [1, 3, 4, 8, 10, 13, 15, 17, 18, 21, 22, 24, 25]
P2 = [2, 5, 6, 7, 9, 11, 12, 14, 16, 19, 20, 23]
Cost = 55

?- seed(1000), grpart(30, 15, P1, P2, Cost).
P1 = [1, 4, 6, 7, 8, 9, 10, 13, 14, 15, 18, 22, 23, 25, 29]
P2 = [2, 3, 5, 11, 12, 16, 17, 19, 20, 21, 24, 26, 27, 28, 30]
Cost = 18

?- seed(987), grpart(35, 10, P1, P2, Cost).
P1 = [1, 2, 3, 4, 6, 10, 11, 14, 15, 19, 25, 26, 27, 28, 29,
      31, 32, 33]
P2 = [5, 7, 8, 9, 12, 13, 16, 17, 18, 20, 21, 22, 23, 24, 30,
      34, 35]
Cost = 8
```

[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 διαμερίσεις με ίδιο ελάχιστο κόστος. Αυτό σημαίνει ότι το δικό σας πρόγραμμα ενδεχομένως να μην βρίσκει την ίδια διαμέριση με αυτή που φαίνεται στις ενδεικτικές εκτελέσεις, αλλά θα πρέπει σίγουρα να έχει το ίδιο κόστος.

```
?- seed(547785888), grpart(40, 5, P1, P2, Cost).
P1 = [1, 2, 3, 4, 5, 6, 7, 10, 12, 13, 14, 18, 20, 22, 23, 24,
      27, 30, 34, 38]
P2 = [8, 9, 11, 15, 16, 17, 19, 21, 25, 26, 28, 29, 31, 32,
      33, 35, 36, 37, 39, 40]
Cost = 4

?- seed(111111), grpart(50, 3, P1, P2, Cost).
P1 = [1, 2, 3, 4, 6, 7, 10, 11, 12, 13, 17, 19, 22, 23, 25,
      26, 28, 31, 32, 34, 35, 37, 38, 43, 45]
P2 = [5, 8, 9, 14, 15, 16, 18, 20, 21, 24, 27, 29, 30, 33, 36,
      39, 40, 41, 42, 44, 46, 47, 48, 49, 50]
Cost = 1

?- seed(5423657), grpart(70, 1, P1, P2, Cost).
P1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19,
      24, 25, 26, 32, 35, 37, 39, 40, 41, 42, 43, 47, 49, 53,
      55, 56, 63, 65, 70]
P2 = [16, 17, 18, 20, 21, 22, 23, 27, 28, 29, 30, 31, 33, 34,
      36, 38, 44, 45, 46, 48, 50, 51, 52, 54, 57, 58, 59, 60,
      61, 62, 64, 66, 67, 68, 69]
Cost = 0
```

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

<u>Άσκηση 5</u>

Μία εταιρεία επιθυμεί να κατασκευάσει έναν αριθμό αποθηκών, από τις οποίες θα μπορεί να τροφοδοτεί τους πελάτες της με τα προϊόντα που παράγει. Από κάποια αρχική μελέτη που έγινε, προέκυψαν διάφορες υποψήφιες θέσεις κατασκευής αποθηκών, έστω πλήθους N. Το κόστος κατασκευής αποθήκης σε κάποια από τις υποψήφιες θέσεις είναι δεδομένο και ισούται με F_i, για κάθε i από 1 μέχρι N. Έστω ότι η εταιρεία προμηθεύει M πελάτες. Το κόστος εξυπηρέτησης του πελάτη j από την αποθήκη i, εφ' όσον αυτή κατασκευασθεί τελικά, είναι δεδομένο και ισούται με C_{ij}, για κάθε i από 1 μέχρι N και κάθε j από 1 μέχρι M.

Το ζητούμενο του προβλήματος είναι να αποφασισθεί σε ποιες από τις υποψήφιες θέσεις πρέπει η εταιρεία να κατασκευάσει αποθήκες και από ποια αποθήκη θα εξυπηρετείται κάθε πελάτης, έτσι ώστε το συνολικό κόστος κατασκευής των αποθηκών και εξυπηρέτησης των πελατών να είναι το ελάχιστο δυνατό για την εταιρεία.

Δεδομένα για το πρόβλημα, με N = 20 υποψήφιες θέσεις για αποθήκες και M = 40 πελάτες, βρίσκονται στο http://www.di.uoa.gr/~takis/warehouses_data.pl. Στο αρχείο αυτό, με το κατηγόρημα `fixedcosts/1` δίνονται, στη μορφή λίστας, τα κόστη κατασκευής των αποθηκών στις διάφορες υποψήφιες θέσεις και με το κατηγόρημα `varcosts/1` δίνονται τα κόστη εξυπηρέτησης των πελατών, σαν μία λίστα από λίστες. Κάθε εσωτερική λίστα αντιστοιχεί σε έναν πελάτη, τα στοιχεία της οποίας είναι τα κόστη εξυπηρέτησης του πελάτη από όλες τις υποψήφιες θέσεις αποθηκών.

Γράψτε ένα κατηγόρημα `warehouses/5`, το οποίο να καλείται σαν `warehouses(N1, M1, YesNoLocs, CustServs, Cost)`, και λαμβάνοντας υπόψη μόνο τις `N1` πρώτες υποψήφιες θέσεις και τους `M1` πρώτους πελάτες από τα δεδομένα του αρχείου που δόθηκε, να βρίσκει **λύση ελαχίστου κόστους** του προβλήματος, επιστρέφοντας στην μεταβλητή `YesNoLocs` μία λίστα (μήκους `N1`) από `1` και `0`, όπου το `1` σημαίνει ότι η αντίστοιχη αποθήκη θα κατασκευασθεί, ενώ `0` ότι δεν θα κατασκευασθεί, στην μεταβλητή `CustServs` μία λίστα (μήκους `M1`) από τους αριθμούς αποθηκών από τις οποίες εξυπηρετούνται οι πελάτες και στην μεταβλητή `Cost` το συνολικό κόστος της λύσης. Αν κάποιο από τα `N1` ή/και `M1` δοθεί ίσο με 0, τότε να ληφθούν υπόψη όλες οι υποψήφιες θέσεις ή/και όλοι οι πελάτες, αντίστοιχα, που δίνονται στο αρχείο. Κάποιες ενδεικτικές εκτελέσεις είναι οι εξής:

```
?- warehouses(5, 10, YesNoLocs, CustServs, Cost).
YesNoLocs = [0, 1, 1, 0, 1]
CustServs = [5, 5, 5, 3, 5, 5, 2, 3, 5, 5]
Cost = 105452

?- warehouses(5, 15, YesNoLocs, CustServs, Cost).
YesNoLocs = [0, 1, 1, 1, 1]
CustServs = [4, 4, 5, 3, 4, 5, 2, 3, 4, 4, 4, 5, 5, 5, 5]
Cost = 156604

?- warehouses(10, 15, YesNoLocs, CustServs, Cost).
YesNoLocs = [0, 1, 0, 1, 1, 1, 1, 0, 0, 0]
CustServs = [7, 7, 5, 6, 4, 5, 2, 6, 4, 4, 4, 5, 6, 5, 7]
Cost = 148880

?- warehouses(10, 20, YesNoLocs, CustServs, Cost).
YesNoLocs = [0, 1, 0, 1, 1, 1, 1, 1, 1, 0]
CustServs = [8, 7, 5, 6, 8, 5, 2, 6, 8, 8, 4, 5, 6, 5, 7, 8,
             4, 9, 4, 7]
Cost = 192820

?- warehouses(15, 20, YesNoLocs, CustServs, Cost).
YesNoLocs = [0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
CustServs = [8, 7, 5, 6, 8, 5, 2, 6, 8, 8, 4, 5, 6, 5, 7, 8,
             4, 9, 4, 7]
Cost = 192820

?- warehouses(20, 30, YesNoLocs, CustServs, Cost).
YesNoLocs = [1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
             0, 0, 0, 0]
CustServs = [12, 12, 1, 6, 12, 1, 2, 6, 12, 12, 4, 5, 6, 1, 7,
             12, 4, 9, 4, 7, 4, 7, 5, 1, 12, 5, 13, 5, 5, 1]
Cost = 385634

?- warehouses(0, 0, YesNoLocs, CustServs, Cost).
YesNoLocs = [1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
             1, 1, 0, 0]
CustServs = [8, 12, 1, 18, 8, 1, 2, 6, 8, 8, 4, 5, 6, 1, 12,
             8, 4, 9, 4, 12, 4, 16, 5, 1, 12, 5, 13, 5, 5,
             1, 16, 5, 16, 17, 12, 12, 18, 6, 8, 18]
Cost = 643420
```

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

<u>Άσκηση 6</u>

Στο *συμμετρικό πρόβλημα του πλανόδιου πωλητή* (symmetric traveling salesman problem), είναι δεδομένο ένα σύνολο από πόλεις και, για κάθε πιθανό ζευγάρι πόλεων C_1 και C_2, είναι γνωστό το κόστος μετάβασης από τη μία προς την άλλη, που είναι το ίδιο για οποιαδήποτε από τις δύο πιθανές κατευθύνσεις. Αυτό το κόστος μπορεί να είναι, για παράδειγμα, η απόσταση των δύο πόλεων. Το ζητούμενο του προβλήματος είναι να βρεθεί με ποια σειρά πρέπει να επισκεφθεί ο πωλητής όλες τις πόλεις, αρχίζοντας από κάποια και καταλήγοντας σ' αυτήν από την οποία ξεκίνησε, ώστε το συνολικό κόστος των μεταβάσεών του να είναι το ελάχιστο δυνατό.

Δεδομένα για το πρόβλημα, που αντιστοιχούν σε ένα δίκτυο 13 πόλεων, φαίνονται στη συνέχεια. Τα δεδομένα αυτά μπορείτε να τα πάρετε από το αρχείο http://www.di.uoa.gr/~takis/tsp_data.pl.

```
costs([[24,11,37,9,22,11,5,41,17,24,8,34],
       [31,12,7,2,17,19,25,21,30,10,32],
       [13,11,8,29,23,14,17,4,21,11],
       [9,14,28,25,11,32,14,10,8],
       [22,11,15,8,13,7,23,25],
       [18,19,3,29,5,18,34],
       [14,30,9,22,17,11],
       [22,11,10,8,19],
       [8,24,42,33],
       [17,6,32],
       [15,28],
       [22]]).
```

Η λίστα που δίνεται σαν όρισμα στο κατηγόρημα `costs/1` περιλαμβάνει στοιχεία λίστες, όπου η πρώτη περιέχει τα κόστη μετάβασης από την $1^η$ πόλη προς όλες τις επόμενές της ($2^η$, $3^η$, κλπ.), η δεύτερη λίστα περιλαμβάνει τα κόστη μετάβασης από τη $2^η$ πόλη προς όλες τις επόμενές της ($3^η$, $4^η$, κλπ.), κοκ. Το τελευταίο στοιχείο (`22`) είναι το κόστος μετάβασης από την $12^η$ πόλη στη $13^η$ (ή αντίστροφα).

Ορίστε ένα κατηγόρημα `tsp/1`, το οποίο όταν καλείται σαν `tsp(N, R, C)`, να επιλύει το συμμετρικό πρόβλημα του πλανόδιου πωλητή, λαμβάνοντας ως είσοδο τις `N` **<u>τελευταίες</u>** πόλεις των δεδομένων που έχουν οριστεί μέσω του κατηγορήματος `costs/1` (δηλαδή τα `N-1` τελευταία στοιχεία του ορίσματός του). Το κατηγόρημα να επιστρέφει στο `R` τη βέλτιστη σειρά επίσκεψης των `N` πόλεων και στο `C` το βέλτιστο συνολικό κόστος. Παραδείγματα εκτέλεσης είναι τα εξής:

```
?- tsp(1, R, C).
R = [1]
C = 0

?- tsp(2, R, C).
```

```
R = [1, 2]
C = 44

?- tsp(3, R, C).
R = [1, 3, 2]
C = 65

?- tsp(4, R, C).
R = [1, 3, 4, 2]
C = 73

?- tsp(5, R, C).
R = [1, 3, 5, 4, 2]
C = 88

?- tsp(6, R, C).
R = [1, 6, 5, 3, 2, 4]
C = 89

?- tsp(7, R, C).
R = [1, 7, 6, 2, 5, 3, 4]
C = 92

?- tsp(8, R, C).
R = [1, 6, 3, 7, 8, 2, 5, 4]
C = 76

?- tsp(9, R, C).
R = [1, 7, 2, 5, 6, 8, 4, 9, 3]
C = 78

?- tsp(10, R, C).
R = [1, 10, 4, 7, 9, 5, 8, 3, 6, 2]
C = 77

?- tsp(11, R, C).
R = [1, 9, 6, 10, 8, 5, 11, 2, 3, 7, 4]
C = 84

?- tsp(12, R, C).
R = [1, 5, 8, 3, 12, 2, 10, 7, 11, 9, 6, 4]
C = 90

?- tsp(13, R, C).
R = [1, 8, 12, 10, 7, 13, 4, 9, 6, 2, 5, 11, 3]
C = 92
```

Ποια είναι η απόδοση του προγράμματός σας για τα δεδομένα του αρχείου http://www.di.uoa.gr/~takis/tsp_big_data.pl, αλλά και του http://www.di.uoa.gr/~takis/tsp_very_big_data.pl;

Παραδοτέο για την άσκηση είναι **ένα πηγαίο αρχείο Prolog** με όνομα **tsp.pl**.

Άσκηση 7

Το πρόβλημα της πρόβας ορχήστρας συνίσταται στην εύρεση της σειράς με την οποία πρέπει να γίνει η πρόβα των διαφόρων μερών ενός κονσέρτου που πρόκειται να δώσει μία ορχήστρα, έτσι ώστε να ελαχιστοποιηθεί ο συνολικός χρόνος αναμονής των μουσικών της ορχήστρας. Κάθε μουσικός συμμετέχει σε συγκεκριμένα μέρη του κονσέρτου και αφού αποφασισθεί η σειρά με την οποία θα γίνει η πρόβα των μερών, ο μουσικός οφείλει να είναι παρών όταν αρχίζει η πρόβα του πρώτου μέρους στο οποίο συμμετέχει και αποδεσμεύεται όταν τελειώσει η πρόβα του τελευταίου μέρους στο οποίο συμμετέχει. Οι μουσικοί αμείβονται ανάλογα με τον χρόνο που τους ζητείται να είναι παρόντες στην πρόβα, πράγμα που σημαίνει ότι ενδιαφερόμαστε να ελαχιστοποιήσουμε, όσο είναι δυνατόν, τα χρονικά διαστήματα που οι μουσικοί είναι παρόντες, αλλά γίνονται πρόβες μερών στα οποία δεν συμμετέχουν. Επίσης, για κάθε μέρος του κονσέρτου, είναι γνωστή η διάρκειά του. Δείτε ένα παράδειγμα στον παρακάτω πίνακα:

Μέρος	1	2	3	4	5	6	7	8	9
Μουσικός 1	1	1	0	1	0	1	1	0	1
Μουσικός 2	1	1	0	1	1	1	0	1	0
Μουσικός 3	1	1	0	0	0	0	1	1	0
Μουσικός 4	1	0	0	0	1	1	0	0	1
Μουσικός 5	0	0	1	0	1	1	1	1	0
Διάρκεια	2	4	1	3	3	2	5	7	6

Στο παράδειγμα αυτό, το κονσέρτο αποτελείται από 9 μέρη και συμμετέχουν σ' αυτό 5 μουσικοί. Η διάρκεια κάθε μέρους φαίνεται στην τελευταία γραμμή του πίνακα. Τα 1 και 0 στον πίνακα δείχνουν αν ο κάθε μουσικός συμμετέχει ή όχι στο αντίστοιχο μέρος του κονσέρτου. Αν οι πρόβες των μερών γινόντουσαν με τη σειρά του πίνακα, τότε οι χρόνοι αναμονής των μουσικών θα ήταν 11 (=1+3+7), 6 (=1+5), 9 (=1+3+3+2), 20 (=4+1+3+5+7) και 3, αντίστοιχα, δίνοντας ένα συνολικό χρόνο αναμονής για τους μουσικούς ίσο με 49 (=11+6+9+20+3). Υπάρχουν όμως και καλύτερες λύσεις απ' αυτήν για τη σειρά των μερών στην πρόβα. Ορίστε ένα κατηγόρημα `rehearsal/3`, το οποίο όταν καλείται σαν `rehearsal(Sequence, WaitTime, TimeOut)` να επιστρέφει στο `Sequence` τη βέλτιστη σειρά πρόβας των μερών ενός κονσέρτου, περιορίζοντας την αναζήτηση σε χρόνο CPU `TimeOut` δευτερολέπτων, και στο `WaitTime` τον συνολικό χρόνο αναμονής των μουσικών γι' αυτή τη σειρά. Αν το `TimeOut` δοθεί ίσο με `0`, τότε να μην υπάρχει περιορισμός χρόνου, αλλά να βρεθεί πραγματικά βέλτιστη λύση. Τα δεδομένα του προβλήματος δίνονται, όπως φαίνεται στο http://www.di.uoa.gr/~takis/rehearsal_data.pl, με τους ορισμούς των γεγονότων `musicians/1` και `durations/1`. Μία ενδεικτική εκτέλεση του ζητούμενου κατηγορήματος είναι η εξής:

```
?- rehearsal(Sequence, WaitTime, 0).
Sequence = [3, 8, 2, 7, 1, 6, 5, 4, 9]
WaitTime = 17
```

Δοκιμάστε και το αρχείο http://www.di.uoa.gr/~takis/rehearsal_data1.pl.

```
?- rehearsal(Sequence, WaitTime, 60).
Sequence = [4, 1, 6, 5, 8, 7, 3, 2, 9, 10, 11]
```

```
WaitTime = 12

?- rehearsal(Sequence, WaitTime, 0).
Sequence = [4, 1, 6, 5, 8, 7, 3, 2, 9, 10, 11]
WaitTime = 12
```

Όπως και το αρχείο http://www.di.uoa.gr/~takis/rehearsal_data2.pl.

```
?- rehearsal(Sequence, WaitTime, 120).
Sequence = [11, 1, 8, 10, 9, 6, 7, 5, 4, 3, 2, 12]
WaitTime = 12

?- rehearsal(Sequence, WaitTime, 0).
Sequence = [1, 3, 2, 5, 4, 12, 7, 6, 9, 10, 8, 11]
WaitTime = 7
```

Παραδοτέο για την άσκηση είναι **ένα πηγαίο αρχείο Prolog** με όνομα **rehearsal.pl**.