Λογικός Προγραμματισμός με Περιορισμούς 2

Καπόν Μύριαμ - dai17093
dai17093@uom.edu.gr

1. No 4s available

```
1   split_check(Amount, Check1, Check2):-
2                       Check1 #:: 0..Amount,
3                       Check2 #:: 0..Amount,
4                       Amount #= Check1 + Check2,
5                       labeling([Check1,Check2]),
6                       not_contain(Check1,4),
7                       not_contain(Check2,4).
8
9   not_contain(0,_):- !.
10
11  not_contain(N,X):-
12              N2 is N mod 10,
13              N2 #\= X,
14              N1 is N div 10,
15              not_contain(N1,X).
```

Τα Check1, Check2 έχουν πεδία τιμών [0,Amount] για την περίπτωση που το Amount δεν περιέχει τον αριθμό 4 και τα checks διαμορφώνονται ως Check1 = Amount, Check2 = 0. Για να ελεγχθεί αν τα checks περιέχουν τον αριθμό 4 χρησιμοποιείται το κατηγόρημα not_contain/2. Ο τρόπος που λειτουργεί είναι ο εξής:

Έστω για παράδειγμα ο αριθμός **1253**.

1. Υπολογίζεται το N2 ως 1253mod10 = 3
2. Ελέγχεται αν το N2 διαφορετικό του 4 (3 \= 4) ✅
3. Υπολογίζεται το N1 ως 1253div10 = 125

1. 125mod10 = 5
2. 5 \= 4 ✅
3. 125div10 = 12

1. 12mod10 = 2
2. 2 \= 4 ✅
3. 12div10 = 1

1. 1mod10 = 1
2. 1 \= 4 ✅
3. 1div10 = 0

Όταν ο αριθμός γίνει 0, η αναδρομή κάνει fail και σταματάει, αλλιώς θα έπεφτε σε infinite loop. Στην ουσία, εφαρμόζονται διαδοχικά mod 10 στον αριθμό ώστε να ελεχθεί το τελευταίο ψηφίο του και div ώστε να περάσει στην επόμενη δεκάδα, αφού το 1253 μπορεί να γραφεί και ως άθροισμα δυνάμεων του 10:

$10^3 * 1 + 10^2 * 2 + 10^1 * 5 + 10^0 * 3$

Σημείωση: Ο έλεγχος του not_contain γίνεται μετά το labeling, γιατί αλλιώς το πρόγραμμα δε φτάνει ποτέ μέχρι το labeling και βγάζει No. Δεν ήξερα πώς να βάλω τον έλεγχο να γίνεται αναδρομικά πριν το labeling, αφου χρειάζεται άλλο κατηγόρημα, και δεν είμαι σίγουρη αν χρειάζεται.

Παραδείγματα εκτέλεσης

```
?- split_check(50, Check1, Check2).
Check1 = 0
Check2 = 50
Yes (0.00s cpu, solution 1, maybe more)

?- split_check(50, Check1, Check2).
Check1 = 0
Check2 = 50
Yes (0.00s cpu, solution 1, maybe more)

?- split_check(4328957, Check1, Check2).
Check1 = 328958
Check2 = 3999999
Yes (7.66s cpu, solution 1, maybe more)
```

2. BackUp my Servers

```
1   schedule_backups(DbStarts, WebStarts, MakeSpan):-
2           findall(DbServer,backup(db,DbServer,_,_,_),DbServers),
3           findall(DbRelease,backup(db,_,DbRelease,_,_),DbReleases),
4           findall(DbDuration,backup(db,_,_,DbDuration,_),DbDurations),
5           findall(DbCost,backup(db,_,_,_,DbCost),DbCosts),
6
7           findall(WebServer,backup(web,WebServer,_,_,_),WebServers),
8           findall(WebRelease,backup(web,_,WebRelease,_,_),WebReleases),
9           findall(WebDuration,backup(web,_,_,WebDuration,_),WebDurations),
10          findall(WebCost,backup(web,_,_,_,WebCost),WebCosts),
11
12          length(WebServers,WebN),
13          length(WebStarts,WebN),
14          length(DbServers,DbN),
15          length(DbStarts,DbN),
16
17          DbStarts #:: 0..inf,
18          WebStarts #:: 0..inf,
19
20          state_crossing_times(WebStarts, WebReleases, WebDurations,WebEnds),
21          state_crossing_times(DbStarts, DbReleases, DbDurations,DbEnds),
22          disjunctive(DbStarts, DbDurations),
23          disjunctive(WebStarts, WebDurations),
24
25          append(WebStarts,DbStarts,Starts),
26          append(DbDurations,WebDurations,Durations),
27          append(DbCosts,WebCosts,Costs),
28          append(DbEnds,WebEnds,Ends),
29          cumulative(Starts, Durations, Costs, 25),
30
31          ic:maxlist(Ends,MakeSpan),
32          bb_min(labeling(Starts),MakeSpan,
33          bb_options{strategy:restart}).
34
35  state_crossing_times([],[],[],[]).
36  state_crossing_times([S|Starts],[R|Releases],[D|Durations],[E|Ends]):-
37          S #>= R,
38          S + D #= E,
39          state_crossing_times(Starts,Releases,Durations,Ends).
```

Ο παραπάνω κώδικας δε λειτουργεί, αλλά το σκεπτικό είναι το εξής:
Από τη στιγμή που η άσκηση ζητάει δύο τύπους σέρβερ, με findall βρίσκονται δύο φορές οι λίστες με τα στοιχεία που χρειάζονται, μία για db και μία για web. Γενικά, όλος ο κώδικας βασίζεται στο παράδειγμα με τη γέφυρα της διάλεξης 11, απλά διπλός (ξεχωριστά για κάθε τύπο). Από τις υποθέσεις της εκφώνησης φαίνεται ότι:

- **disjunctive**: οι εργασίες αντιγράφων ασφαλείας για εξυπηρετητές (servers) του ιδίου τύπου δεν μπορεί να γίνει ταυτόχρονα.
- **cumulative**: το συνολικό εύρος δικτύου (bandwidth) είναι 25, το οποίο σημαίνει ότι δεν είναι δυνατό να γίνουν ταυτόχρονα δύο οποιεσδήποτε εργασίες με άθροισμα σε απαιτήσεις πάνω από 25.

Για να βγει το cumulative, τα Starts των db και web (DbStarts + WebStarts) συνδυάζονται σε μια λίστα Start. Ίσως αντί ο κώδικας να επαναλαμβανόταν δύο φορές, να μπορούσε να γίνει κάτι με το element.

<u>Παράδειγμα Εκτέλεσης</u>
?- schedule_backups(DbStarts, WebStarts, M).
DbStarts = [0, 5, 20]
WebStarts = [0, 12]
M = 24
Yes (0.00s cpu)

3. Jumps in a List

```
1   arrange_list(Len,List):-
2           Len #>= 7,
3           length(List,Len),
4           List #:: 1..Len,
5           ic_global: alldifferent(List),
6           labeling(List),
7           element(1, List, Start),
8           abs(1 - Start) #> 2,
9           jump(List, Start, [Start]).
10
11  jump(List, 1, Visited):-
12          length(List, N),
13          length(Visited,N).
14
15  jump(List, From, Visited):-
16          element(From, List, To),
17          abs(From - To) #> 2,
18          not(member(To, Visited)),
19          jump(List, To, [To|Visited]).
```

Το πρόβλημα λύνεται με τη χρήση του κατηγορήματος jump/2, το οποίο πετυχαίνει όταν List είναι μια λίστα τέτοια ώστε να πληροί την περιγραφή της εκφώνησης.
Για να το πετύχει αυτό χρειάζεται το κατηγόρημα element(Ind, List, Val), όπου Ind ο Index και Val το value στο οποίο δείχνει στη λίστα List. Η λογική είναι η εξής:

“Ξεκινώντας από το πρώτο στοιχείο, πάρε το ως Index και βρες την τιμή στην οποία δείχνει (To), την οποία μετά στείλε ως Index (From), μέχρι το στοιχείο από το οποίο προσπαθείς να κάνεις jump να είναι το 1. “

Το 1, επειδή αυτό θα είναι πάντα το τελευταίο στοιχείο που θα επισκεφτεί στη διαδρομή του το jump, αφού πρέπει να επιστρέφει στην αρχή. Για να βεβαιωθούμε ότι το 1 είναι πράγματι το τελευταίο στοιχείο, καθώς και ότι δεν υπάρχουν κύκλοι (για αυτό δεν μπήκε counter), χρησιμοποιείται μια λίστα Visited. Η λογική αυτή μοιάζει με την αναζήτηση σε γράφους και η διαδρομή που ακολουθείται είναι της μορφής List[List[List[1]]].

Σημαντικό είναι να γίνεται ο έλεγχος abs(From - To) #> 2, που ελέγχει ότι το jump είναι μεγαλύτερο από 2 στοιχεία, και με την αρχικοποίηση του πρώτου στοιχείου έξω από το jump. Επίσης, έπρεπε το element να μπει μετά το labeling, αλλιώς έβγαζε **instantiation fault.**

Παραδείγματα εκτέλεσης

```
?- arrange_list(8, List).
List = [4, 5, 6, 7, 8, 1, 2, 3]
Yes (0.91s cpu, solution 1, maybe more)

?- arrange_list(10, List).
List = [4, 5, 6, 7, 8, 9, 10, 1, 2, 3]
Yes (66.47s cpu, solution 1, maybe more)
```

Σημείωση: Δε θεωρώ ότι η λύση μου είναι ιδιαίτερα "έξυπνη", έχω ένα προαίσθημα ότι μπορούν να βγουν περιορισμοί με περισσότερα element. Θα ήθελα αν γίνεται να μου στείλετε την πιο σωστή λύση.