# Ατομική Διπλωματική Εργασία

## ΚΩΔΙΚΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΔΡΑΣΗΣ ΣΕ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΠΕΡΙΟΡΙΣΜΩΝ

*Όνομα: Βουτουρή Παναγιώτης*

ΠΑΝΕΠΙΣΤΗΜΙΟ *ΚΥΠΡΟΥ*

*ΤΜΗΜΑ* ΠΛΗΡΟΦΟΡΙΚΗΣ

*Ιούνιος 2008*

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

# *ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ CHIPC*

*Όνομα: Βουτουρή Παναγιώτης*

Επιβλέπων Καθηγητής: Γιάννης Δημόπουλος

# Ευχαριστίες:

Θέλω να ευχαριστήσω τα πιο κάτω άτομα τα οποία βοήθησαν στην εκπόνηση της διπλωματικής εργασίας μου:

- Τον επιβλέποντα καθηγητή μου κύριο Γιάννη Δημόπουλο ο οποίος με βοήθησε πολύ στα προβλήματα τα οποία αντιμετώπισα σε διάφορες περιπτώσεις της υλοποίησης μου. Επίσης έδειξε μεγάλη κατανόηση σε τομείς καθυστέρησης παράδοσης τμημάτων της εργασίας μου.
- Ακόμα θέλω να ευχαριστήσω την τεχνική υπηρεσία του τμήματός μας οι οποίοι βοήθησαν σε προβλήματα που είχαν να κάνουν σχέση με την εγκατάσταση της chipc.

# Περίληψη:

Σε αυτήν την εργασία θα ασχοληθούμε με την υλοποίηση προβλημάτων λογικού προγραμματισμού. Συγκεκριμένα θα υλοποιήσουμε τρία προγράμματα. Το πρώτο είναι η μετακίνηση κύβων(γνωστό και σαν ο κόσμος των κύβων) , το δεύτερο η μεταφορά φορτίων σε πόλεις όπου μας ενδιαφέρει η τελική κατάσταση του κάθε φορτίου και το τρίτο η μεταφορά φορτίων σε πόλεις όπου δεν μας ενδιαφέρει η τελική κατάσταση του κάθε φορτίου αλλά μόνο ο αριθμός των φορτίων που πρέπει να έχει η κάθε πόλη. Η υλοποίηση αυτών των προγραμμάτων θα γίνει στην γλώσσα CHIPC η οποία είναι μια γλώσσα λογικού προγραμματισμού βασισμένη σε C. Ένα πρόβλημα λογικού προγραμματισμού είναι πολύ δύσκολο να υλοποιηθεί χωρίς την χρησιμοποίηση κάποιας γλώσσας λογικού προγραμματισμού αφού μιλάμε για μεταβλητές οι οποίες δεν παίρνουν κάποια συγκεκριμένη τιμή για παράδειγμα ένα ακέραιο αλλά παίρνουν τιμές από ένα πεδίο συγκεκριμένων τιμών από τις οποίες τελικά θα ‘διαλέξει’ κάποια που θα πληρεί ορισμένους περιορισμούς. Έτσι με την chipc μπορούμε να υλοποιήσουμε τα ποιο πάνω με ορισμένες συναρτήσεις οι οποίες παρέχονται από την ίδια την γλώσσα και μπορούμε ‘αυτόματα’ να ψάξουμε όλους τους συνδυασμούς που μπορεί να υπάρξουν και να βρούμε τελικά μια

# Περιεχόμενα:

# Κεφάλαιο 1 Εισαγωγή:

### 1.1 Προγραμματισμός δράσης:

Είναι ένα είδος προγραμματισμού ο οποίος χρησιμοποιεί μια σειρά από ενέργειες για να αναπαράγει δραστηριότητες και να πετύχει συγκεκριμένους στόχους. Τα προβλήματα αυτά διακρίνονται από καταστάσεις(states), στόχους(goals) και ενέργειες(actions).

Το πώς γίνεται η αναπαράσταση των καταστάσεων(states) είναι να μεταφράσουμε το πρόβλημά μας σε λογικές συνθήκες και να τις αναπαραστήσουμε σε καταστάσεις της μορφής state(variable1,value1) ^ state(variable2,value2) …. Για παράδειγμα At(plane1,Melbourne) ^ At(plane2,Sydney).

Η αναπαράσταση των στόχων (goals) είναι ακριβώς η ίδια με την αναπαράσταση των καταστάσεων αφού ένας στόχος είναι και αυτός μια αναπαράσταση στην οποία μεταφερόμαστε μετά από κάποιες ενέργειες.

Η ενέργειες μπορούν να αναπαρασταθούν με τον τύπο Action = Precond + Effect. Έστω ότι έχουμε σαν ενέργεια το πέταγμα ενός αεροπλάνου από μια πόλη σε άλλη. Έτσι η ενέργεια μας θα είναι Fly(p from to) και θα έχουμε σαν Precond: At(p,from) ^ Plane(p) ^ Airport(from) ^ Airport(to). Με αυτό εννοούμε ότι το ‘p’ που είναι αεροπλάνο θέλουμε να μεταφερθεί από το ‘from’ στο ‘to’ όπου είναι αεροδρόμια. Effect: ¬AT(p,from) ^ At(p,to)). Το αποτέλεσμα θα είναι να φύγει το αεροπλάνο ‘p’ από το αεροδρόμιο ‘from’ και να πάει στο αεροδρόμιο ‘to’. [6]

Έχοντας καθορίσει το πώς αναπαριστούμε συντακτικά προβλήματα δράσης θα δούμε μερικές σημαντικές παρατηρήσεις. Το πιο σημαντικό είναι να δούμε πως μια ενέργεια μπορεί να επηρεάσει μια κατάσταση. Πρώτα ορίζουμε ότι μια ενέργεια μπορεί να υλοποιηθεί σε όποια κατάσταση της οποίας όλα τα preconditions της ισχύουν. Αλλιώς η ενέργεια αυτή δεν θα έχει κανένα αποτέλεσμα. Για παράδειγμα ας υποθέσουμε ότι βρισκόμαστε στην πιο κάτω κατάσταση.

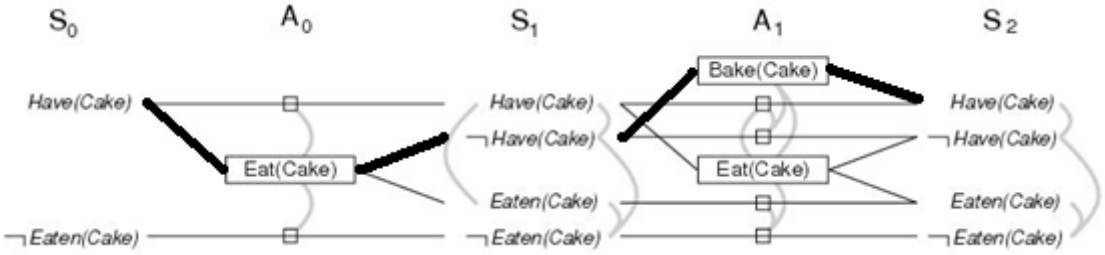
At (P1; JFK) ^ At (P2; SFO) ^ Plane(P1) ^ Plane(P2)
^ Airport(JFK) ^ Airport(SFO) .
Αυτή η κατάσταση όπως φαίνεται ικανοποιεί τους πιο κάτω περιορισμούς.
At (p; from) ^ Plane(p) ^ Airport(from) ^ Airport(to)
Αυτό σημαίνει ότι θα εκτελεστή στην κατάσταση αυτή η ενέργεια
Fly(P1; JFK; SFO)
Έτσι μεταβαίνουμε στην επόμενή μας κατάσταση όπου θα είναι
At (P1; SFO) ^ At (P2; SFO) ^ Plane(P1) ^ Plane(P2)
^ Airport(JFK) ^ Airport(SFO) .
Επιπλέον αν ένα αποτέλεσμα έχει είδη εφαρμοστή από κάποια άλλη ενέργεια δεν το προσθέτουμε ξανά στην κατάστασή μας.
Τέλος μπορούμε να καθορίσουμε τον στόχο του προβλήματός μας ο οποίος είναι και αυτός μια κατάσταση στην οποία θέλουμε να μεταβούμε.

Μια λύση μπορεί να αναπαρασταθεί με γράφους δράσης (planning graphs) οι οποίοι περιλαμβάνουν μια ακολουθία από επίπεδα που αντιπροσωπεύουν τα χρονικά βήματα στο πρόβλημά μας. Το επίπεδο 0 είναι η αρχική μας κατάσταση και κάθε επίπεδο περιέχει μια σειρά από ενέργειες.

Πιο κάτω φαίνεται ένα παράδειγμα μεταβάσεις από πρόγραμμα δράσης σε γράφο δράσης. Σε αυτό το πρόβλημα ξεκινούμε από μια αρχική κατάσταση στην οποία έχουμε ένα cake. Σκοπός μας είναι να φάμε το cake και να έχουμε ένα cake. Αφού όμως φάμε το cake τότε θα έχει σαν αποτέλεσμα να μην έχουμε cake. Έτσι μπορούμε να εκτελέσουμε την ενέργεια ψήσε cake όπου θα έχει με την σειρά της αποτέλεσμα να έχουμε cake.

Init(Have(Cake))
Goal(Have(Cake) ^ Eaten(Cake))
Action(Eat(Cake), PRECOND: Have(Cake)
EFFECT: ¬Have(Cake) ^ Eaten(Cake))
Action(Bake(Cake), PRECOND: ¬ Have(Cake)
EFFECT: Have(Cake)

Πιο κάτω φαίνεται ο γράφος που παράγεται από το πιο πάνω πρόγραμμα. Αν ακολουθήσουμε την έντονη γραμμή βλέπουμε πως φτάνουμε στην λύση μας. Το S αναπαριστά καταστάσεις και το A αναπαριστά ενέργειες. [4]




Σε ένα πρόβλημα πραγματικού χρόνου δεν αναφερόμαστε πλέον σε μια μόνο κατάσταση αλλά σε ένα πλήθος καταστάσεων που πρέπει να ολοκληρωθούν μα μια συγκεκριμένη σειρά. Εδώ είναι που πρέπει να αναλύσουμε τον όρο Προβλήματα δράσης σε Προτασιακή λογική.

Με τον πιο πάνω όρο εννοούμε τον διαχωρισμό τον καταστάσεων ενός προβλήματος δράσης σε καταστάσεις οι οποίες έχουν κάποια σημασία ανάλογα με την σειρά εκτέλεσης τους. Έτσι πρέπει να βρούμε ένα τρόπο να ξεχωρίζουμε τις μεταβάσεις μας. Αυτά γίνεται απλά με το να τις αριθμήσουμε. Η αρχική μας κατάσταση θα συμβολίζεται με τον αριθμό μηδέν. Έτσι για να μεταβούμε από την κατάσταση μηδέν στην κατάσταση ένα πρέπει να προσθέσουμε τα preconditions που πρέπει να ισχύουν στην κατάσταση μηδέν και ούτω καθεξής. Πιο κάτω φαίνεται ένα παράδειγμα αυτής της μετάβασης. Ο στόχος μας είναι να μεταφερθεί το P1 στο JFK. Οι συνθήκες που υπάρχουν μετά το βελάκι είναι τα preconditions μας.

$$\text{At}(P1,SFO)^0 \wedge \text{At}(P2,JFK)^0$$

$$\text{At}(P1,JFK)^1 \rightarrow (\text{At}(P1,JFK)^0 \wedge \neg(\text{Fly}(P1, JFK, SFO)^0 \wedge \text{At}(P1, JFK)^0)) \text{ OR}$$

$$(\text{Fly}(P1, SFO, JFK)^0 \wedge \text{At}(P1, SFO)^0)$$

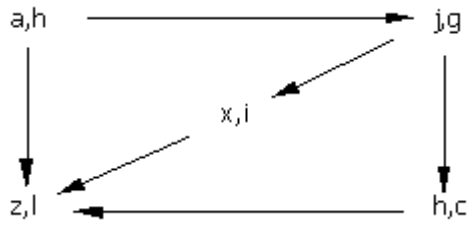
Έτσι έχουμε δηλώσει ότι το P1 θα βρίσκεται στην κατάσταση ένα στο JFK αν ήταν στην κατάσταση μηδέν στο JFK και δεν έχει μετακινηθεί ή αν ήταν στην κατάσταση μηδέν στο SFO και έχει μεταφερθεί στο JFK. Επίσης για να μεταφερθούμε από ένα αεροδρόμιο στο άλλο υπάρχουν επιπλέων περιορισμοί που πρέπει να ισχύουν.

Για να είναι το πρόγραμμα μας ολοκληρωμένο θα πρέπει να προστεθούν και άλλοι περιορισμοί όπως ότι ένα αεροπλάνο δεν μπορεί να μεταφερθεί ταυτόχρονα σε δύο αεροδρόμια: $not(At\ (p, x)^t \wedge At\ (p; y)^t)$ .

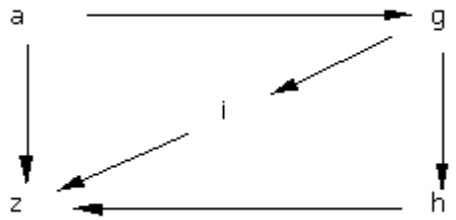
### 1.2 Προγραμματισμός περιορισμών

Προγραμματισμός περιορισμών (CSP) είναι ένας τομέας της τεχνικής νοημοσύνης ο οποίος ασχολείται με την ικανοποίηση περιορισμών. Ουσιαστικά μας δίνεται ένα πρόβλημα και εξάγουμε τους περιορισμούς που πρέπει να υπάρχουν έτσι ώστε το πρόγραμμα μας να φτάσει σε λύση. Ένα κλασσικό παράδειγμα προγραμματισμού περιορισμών είναι το χρονοδιάγραμμα τάξεων στο οποίο πρέπει να τοποθετήσουμε σε τάξεις όσο πιο πολλούς φοιτητές μπορούμε αναλόγως με τα μαθήματά τους. Οι περιορισμοί που ισχύουν είναι να βρούμε ποιες τάξεις είναι διαθέσιμες, ποιοι φοιτητές έχουν συγκρουόμενες τάξεις και ποιοι καθηγητές είναι διαθέσιμοι να διδάξουν το συγκεκριμένο μάθημα.

Για το λόγο ότι το χρονοδιάγραμμα τάξεων είναι ένα πολύπλοκο πρόβλημα προγραμματισμού περιορισμών ας δούμε ένα ποιο εύκολα παράδειγμα. Έστω ότι υπάρχει η πιο κάτω αναπαράσταση και θέλουμε να επιλέξουμε γράμματα τα οποία να είναι λεξικογραφικά το ένα πριν το άλλο. Τα βέλη δείχνουν από ποια κατεύθυνση ισχύει ο πιο πάνω περιορισμός.




Με βάση το πιο πάνω σχήμα μπορούμε να βρούμε ότι υπάρχει μόνο μια λύση στην οποία ικανοποιούνται όλοι οι περιορισμοί. Η λύση αναπαριστάται από την πιο κάτω αναπαράσταση. Αν και σε αυτό το πρόβλημα υπάρχει μόνο μια λύση στα προβλήματα λογικού προγραμματισμού μπορεί να υπάρξουν πιο πολλές ή και καμία λύση.




Φυσικά ένας υπολογιστής για να βρει λύση πρέπει να μεταφραστούν οι περιορισμοί σε μια γλώσσα περιορισμού. Εδώ είναι που έρχεται η γλώσσα chipc στην οποία μπορούμε να θέσουμε τους περιορισμούς μας και να μας βρει λύσεις για διάφορα προβλήματα. [5]

### 1.3 Η γλώσσα chipc:

Η γλώσσα αυτή χρησιμοποιείτε κυρίως για προγραμματισμό με περιορισμούς (constraint logic programming). Δηλαδή για την επίλυση προβλημάτων με περιορισμούς(constraint satisfaction problems) στα οποία γίνεται ανάθεση τιμών σε μεταβλητές με τρόπο ώστε να πληρούν συγκεκριμένους περιορισμούς.

Η γλώσσα chip ήταν μια από της πρώτες γλώσσες προγραμματισμού περιορισμών. Αναπτύχθηκε από το European Computer-Industry Research center στο Μόναχο γύρω στο 1985-1990. Τώρα η CHIP αναπτύσσετε από την εταιρία COSYTEC. [2]

### 1.4 Κωδικοποίηση προβλημάτων δράσης σε προβλήματα περιορισμού.

Αφού έχουμε κατανοήσει σε γενικές γραμμές τι είναι τα προβλήματα δράσης θα προσπαθήσουμε να βρούμε ένα γενικό αλγόριθμο κωδικοποίησης αυτών των προβλημάτων σε προβλήματα περιορισμών.

Τα προβλήματα δράσεων έχουν μια αρχική κατάσταση, μια τελική κατάσταση και ενέργειες με τις οποίες θα μας μεταφέρουν από την αρχική στην τελική μας κατάσταση. Αυτό για να το κωδικοποιήσουμε σε προγραμματισμό περιορισμών θα χρειαστεί να δημιουργήσουμε ένα δυσδιάστατο πίνακα.

| Sydney | Melbourne | Cyprus |
|---|---|---|
| | | |
| | | |
| Cyprus | Sydney | Melbourne |

Σε κάθε μας πρόγραμμα θα υπάρχει και ένας παρόμοιος πίνακας με τον πιο πάνω πίνακα. Η πρώτη μας γραμμή θα είναι η αρχική κατάσταση του προβλήματος και η τελευταία γραμμή θα είναι ο στόχος μας δηλαδή η τελική κατάσταση που θα πρέπει να φτάσουμε.

Ο αριθμός των στηλών θα αντιπροσωπεύει των αριθμό των αντικειμένων που υπάρχουν στο πρόβλημα(στο παράδειγμά μας τρία αεροπλάνα).

Ο αριθμός των γραμμών δηλώνει τον αριθμό των προσπαθειών που διακαιόμαστε μέχρι να φτάσουμε σε λύση.

Κάθε γραμμή του πίνακα αντιπροσωπεύει και μια προσπάθεια και οι τιμές των πεδίων το που βρίσκονται τα αντικείμενα μας στην συγκεκριμένη προσπάθεια.

Μια μετάβαση από μια κατάσταση σε άλλη γίνεται από την στιγμή που όλες οι τιμές του πίνακα στην συγκεκριμένη προσπάθεια πάρουν τιμές. Έτσι μπορούμε να προχωρήσουμε και στην ανάθεση τιμών της επόμενης προσπάθειας και ούτω καθεξής.

Στον προγραμματισμό περιορισμών πρέπει να κάνουμε ένα επιπλέων βήμα και να αναθέσουμε πεδία ορισμού στης μεταβλητές μας (αεροπλάνα). Έτσι πρέπει σε κάθε πρόβλημα να μελετάμε ποιες είναι οι πιθανές τιμές που μπορούν να πάρουν οι μεταβλητές μας. Στο πιο πάνω πρόβλημα είναι εύκολα να καταλάβουμε ότι οι τιμές που μπορούν να πάρουν τα αεροπλάνα είναι όλα τα αεροδρόμια που υπάρχουν στο πρόβλημά μας. Σε άλλα προβλήματα υπάρχει η πιθανότητα να φτάσουμε σε ποιο πολύπλοκη ανάθεση πεδίων τιμών.

Τα preconditions που υπάρχουν στον προγραμματισμό δράσης θα μεταφραστούν σε περιορισμούς. Αυτοί οι περιορισμοί θα μας εξασφαλίσουν ότι η επόμενη μας κατάσταση είναι μια επιτρεπτή κατάσταση σύμφωνα με το πρόβλημά μας. Έτσι αφού η τελευταία μας γραμμή στον πίνακα είναι και ο στόχος που πρέπει να επιτύχουμε και φτάσουμε σε μια προηγούμενη κατάσταση που μπορούμε να μεταβούμε σε αυτήν την κατάσταση σημαίνει ότι έχουμε φτάσει σε λύση. Αλλιώς αν δεν μπορούμε σύμφωνα με τους περιορισμούς μας να μεταβούμε σε αυτήν την κατάσταση σημαίνει ότι το πρόγραμμά μας έχει αποτύχει να βρει λύση.

Συνήθως οι περιορισμοί μας θα μας περιορίζουν τις τιμές μια μετάβασης με βάση της τιμές που θα έχουμε στην προηγούμενή μας κατάσταση. Αυτό είναι εφικτό στον προγραμματισμό με περιορισμούς αφού μπορούμε να αναφερθούμε στις συγκεκριμένες τιμές του πίνακα που μας αφορούν. Ένα πρόβλημα που θα πρέπει να επιλύσουμε είναι να μην ανατίθενται περιορισμοί στην επόμενη μας κατάσταση προτού πάρει όλες της τιμές η προηγούμενη κατάσταση.  Αυτό γίνεται με το να κάνουμε delay τους περιορισμούς μας μέχρι να πάρουν κάποια τιμή οι μεταβλητές μας στην προηγούμενη κατάσταση.

Πάντα στα προβλήματα μας θα πρέπει να έχουμε μια αρχική κατάσταση για να μπορέσει ο αλγόριθμος να προσθέσει με βάση αυτήν τους περιορισμούς για την επόμενη κατάσταση και να προσπαθήσει να βρει ένα επιτρεπτό συνδυασμό τιμών.

Η πολυπλοκότητα του πιο πάνω αλγόριθμου αυξάνεται από την στιγμή που δεν μιλούμε πλέον για ένα πίνακα μεταβλητών αλλά για περισσότερους. Σε αυτήν την περίπτωση θα η κωδικοποίηση γίνεται όπως έχουμε προαναφέρει. Οι περιορισμοί μας όμως αυξάνονται αφού θα υπάρχουν και περιορισμοί που θα συνδυάζουν τις τιμές του ενός πίνακα με τον άλλο.

### 1.5 Από την STRIP στην CHIPC:

Strip (Stanford Research Institute problem solver) είναι η κλασσική γλώσσα για αναπαράσταση προβλημάτων δράσης. Η strip αποτελείται από καταστάσεις(states), στόχους(goals) και ενέργειες(actions). Πιο κάτω εξηγούμε πως αναπαρίσταται η strip και πως από αυτή μπορούμε να μεταφέρουμε το πρόβλημα μας σε αναπαράσταση όπου θα μπορούμε να το υλοποιήσουμε μέσο της chipc.

- **1.5.1 Αναπαράσταση καταστάσεων:**

  Με την δήλωση At(Plane1, Melbourne) ^ At(Plane2,Sydney) αναπαριστούμε την κατάσταση όπου το πρώτο αεροπλάνο βρίσκεται στην Melbourne και το δεύτερο στο Sydney. Για να μπορέσουμε αυτό να το αναπαραστήσουμε σε chipc πρέπει να το μετατρέψουμε πρώτα σε πίνακες αφού αυτός είναι ο τρόπος με τον οποίο διαχειρίζεται η chipc τα δεδομένα της. Έτσι το At(Plane1, Melbourne) ^ At(Plane2,Sydney) γίνεται ένας μονοδιάστατος πίνακας δυο θέσεων. Η πρώτη θέση του πίνακα ισούται με Melbourne και αναπαριστά τη θέση που βρίσκεται το πρώτο αεροπλάνο και η δεύτερη ισούται με Sydney και αναπαριστά την θέση του δεύτερου αεροπλάνου.

  Τώρα δηλαδή

| Melbourne | Sydney |
|---|---|

  έχουμε:

  Το πρώτο βήμα έχει γίνει και το μόνο που μας μένει είναι να γράψουμε αυτό που εξηγήσαμε ποιο πάνω στην γλώσσα chipc:

  Πρώτα ορίζουμε τις τιμές που μπορούν να πάρουν οι μεταβλητές μας (αεροπλάνα) σαν ακέραιους αριθμούς έτσι ώστε να γίνει πιο εύκολος ο τρόπος με τον οποίο χειριζόμαστε τον πίνακά μας.

  ```
  #define MELBURNE 1

  #define SYDNEY 2
  ```

  Μετά δηλώνουμε ένα πίνακα τύπου domain variable μεγέθους 2 και domain το οποίο μπορούν να πάρουν οι μεταβλητές του από 1 μέχρι 2 δηλαδή Melbourne και Sydney.

  ```
  DvarPtr * vars;

  vars = (void**)malloc(sizeof(int)*(2));

  c_create_domain_array(vars, 2, 1, 2, CONSEC);
  ```

Τέλος βάζουμε τον περιορισμό η πρώτη θέση του πίνακα να είναι ίση με Melbourne και η δεύτερη με Sydney.

c_dom_domcst(vars[0], EQUAL,NULL, MELBURNE);

c_dom_domcst(vars[1], EQUAL,NULL, SYDNEY);

- **1.5.2 Αναπαράσταση στόχων:**

  Ένας στόχος είναι και αυτός μια κατάσταση. Η μόνη διαφορά ότι μιλούμε για μια κατάσταση η οποία παράχθηκε μετά από κάποιες ενέργειες. Έτσι ισχύουν ακριβώς τα ίδια πράγματα που αναφέραμε προηγουμένως για την αναπαράσταση σε strip και την μεταφορά τους στην chipc.

- **1.5.3 Αναπαράσταση ενεργειών:**

  Μια ενέργεια αναπαρίσταται με action(fly(plane1, Melbourne, Sydney)). Αυτός μας δηλώνει ότι το αεροπλάνο1 πηγαίνει από την Melbourne στο Sydney. Κάθε ενέργεια καθορίζεται από προαπαιτήσεις(preconditions) και αποτέλεσμα(effect). Η προηγούμενη δηλαδή ενέργεια έχει σαν προαπαίτηση at(plane1, Melbourne). Έτσι για να μπορεί ένα αεροπλάνο να πάει από την Melbourne στο Sydney πρέπει πρώτα να βρίσκεται στην Melbourne. Αυτές οι προαπαιτήσεις στην strip είναι οι γνωστοί μας περιορισμοί(constraints) στην chipc. Αποτέλεσμα τώρα αυτής της ενέργειας είναι να φύγει το αεροπλάνο από την Melbourne και να πάει στο Sydney. Αυτό αναπαρίσταται με at(plane1, Melbourne)^at(plane1, Sydney). Στην chipc για να το καταφέρουμε αυτό όπως και στις προηγούμενες αναπαραστάσεις πρέπει να χρησιμοποιήσουμε πίνακες. Έτσι για να δείξουμε την πορεία ενός αεροπλάνου χρησιμοποιούμε ένα δυσδιάστατο πίνακα με αριθμό γραμμών όσες και οι διαδρομές που θα κάνει το αεροπλάνο μας. Για το προηγούμενο παράδειγμα δηλαδή θα είχαμε

| Melbourne |
|---|
| Sydney |

  Έτσι η πρώτη στήλη(στην περίπτωση μας έχουμε μόνο μια αφού μιλούμε για την διαδρομή ενός μόνο αεροπλάνου) αντιστοιχεί στην διαδρομή που

ακολουθά κάθε αεροπλάνο. Όπως φαίνεται και ποιο πάνω το αεροπλάνο1(πρώτη στήλη) πήγε από την  Melbourne στο Sydney.

Αυτό στη chipc γράφεται με τον εξής τρόπο:

Πρώτα ορίζουμε τις τιμές που μπορούν να πάρουν οι μεταβλητές μας (αεροπλάνα) σαν ακέραιους αριθμούς έτσι ώστε να γίνει πιο εύκολος ο τρόπος με τον οποίο χειριζόμαστε τον πίνακά μας. Δηλώνουμε ακόμα και τον αριθμό των μεταβλητών μας.

```
#define MELBURNE 1

#define SYDNEY 2

#define NUM_PL 2
```

Μετά δηλώνουμε ένα πίνακα τύπου domain variable μεγέθους 2 και domain το οποίο μπορούν να πάρουν οι μεταβλητές του από 1 μέχρι 2 δηλαδή Melbourne και Sydney.

```
DvarPtr * vars;

vars = (void**)malloc(sizeof(int)*(2*NUM_PL));

c_create_domain_array(vars, 2, 1, 2, CONSEC);
```

Τέλος βάζουμε τον περιορισμό η πρώτη θέση του πίνακα να είναι ίση με Melbourne και η δεύτερη με Sydney.

```
c_dom_domcst(vars[0], EQUAL,NULL, MELBURNE);

c_dom_domcst(vars[1], EQUAL,NULL, SYDNEY);
```

Αν και αυτός ο κώδικας φαίνεται πολύ παρόμοιος με τον προηγούμενο κώδικα τον οποίο παρουσιάσαμε(αναπαράσταση καταστάσεων) σημασιολογικά είναι πολύ διαφορετικός αφού μας δηλώνει την πορεία που θα ακολουθήσει κάθε αεροπλάνο. Η ομοιότητα οφείλεται στο ότι μιλούμε για ένα μόνο αεροπλάνο το οποίο εκτελεί μια μόνο διαδρομή.

### 1.6 Εκτέλεση της chipc:

Για να τρέξουμε ένα πρόγραμμα σε chipc πρέπει να ακολουθήσουμε τα πιο κάτω βήματα:

- Πρώτα πρέπει να τρέξουμε το πρόγραμμα μας στην μηχανή cs840.in.cs.ucy.ac.cy αφού είναι η μόνη μηχανή στο πανεπιστήμιο η οποία έχει εγκατεστημένη chipc. Για να το κάνουμε αυτό μπορούμε να πάμε απευθείας στην μηχανή αυτή ή να ενωθούμε σε αυτή την μηχανή με telnet χρησιμοποιώντας ένα πρόγραμμα όπως το putty.
- Μετά πρέπει να ορίσουμε το αρχείο όπου βρίσκετε εγκατεστημένη η chipc. Αυτό γίνεται με την εντολή: setenv CHIPCDIR /usr/local/chipc5.7
- Μετά τρέχουμε την εντολή η οποία κάνει compile τον κώδικά μας και μας δημιουργά ένα εκτελέσιμο αρχείο: /usr/bin/gcc -o diplomatiki2 -I/usr/local/chipc5.7/INCLUDE/ -L/usr/local/chipc5.7/LIB/ -lchipc -lm diplomatiki2.c
-
- Τέλος τρέχουμε το εκτελέσιμο αρχείο το οποίο έχει δημιουργηθεί: ./diplomatiki2

### 1.7 Συναρτήσεις που θα χρησιμοποιήσουμε :

- **1.7.1 int c_chip_init(void);**

Είναι η συνάρτηση η οποία πρέπει να καλείτε σε αρχή κάθε προγράμματος που γράφεται σε chipc. Με αυτή την συνάρτηση δεσμεύεται χώρος και γίνονται αρχικοποίηση διάφορες μεταβλητές περιορισμού που θα χρειαστούν για την σωστή λειτουργία του προγράμματός μας.

- **1.7.2 int c_chip_end(void);**

Όταν τελειώσουμε το πρόγραμμα μας καλούμε αυτή την συνάρτηση η οποία αποδεσμεύει τις μεταβλητές που δέσμευσε η c_chip_init. Έτσι μας ελευθερώνει μνήμη που μπορεί να την χρησιμοποιήσουν άλλες διεργασίες.

- **1.7.3 c_domain_array_print()**

void c_domain_array_print(
FILE * *file,*
DvarPtr *array[]*,
int *size*
);

**Παραμέτροι:**

*file* – ένας δείκτης τύπου FILE. Αν θέλουμε τα αποτελέσματα να τυπωθούν στην οθόνη το ορίζουμε σαν stdout.
*array[]* – ένας πίνακας από domain μεταβλητές.
*size* – ένας φυσικός αριθμός που δηλώνει το μέγεθος του πίνακα.

**Περιγραφή:**

Η συνάρτηση αυτή τυπώνει το πεδίο ορισμού για κάθε μια μεταβλητή του πίνακα που παίρνει σαν παράμετρο. Το τύπωμα γίνεται με τον εξής τρόπο. Τυπώνει μια μια τις μεταβλητές του πίνακα δίνοντας τους όνομα D_i = με το πεδίο των τιμών σε αγκύλες {X,…,Y}. Αν παίρνουν μόνο μια τιμή τότε δίνει μόνο αυτή την τιμή χωρίς αγκύλες.

- **1.7.4 c_atleast()**

Pred_result c_atleast(
int low,
DvarPtr vars[],
int nvar,
int val
);

**Μεταβλητές**

low - ένας φυσικός αριθμός
vars[] – ένας πίνακας από domain variables.
nvar – ένας φυσικός αριθμός που δηλώνει το μέγεθος του πίνακα nvar.
val - ένας φυσικός αριθμός

**Περιγραφή**

Αυτή η συνάρτηση χρησιμοποιείτε στην περίπτωση που χρειαζόμαστε να καθορίσουμε έναν ελάχιστο αριθμό παρουσίασης μιας μεταβλητής σε ένα σύνολο από domain variables. Ο περιορισμός που θα ισχύει είναι να υπάρχει τουλάχιστο low φορές στον πίνακα vars με μέγεθος nvar η τιμή val.

- **1.7.5 c_domain_bind_to_value()**

#include “chipc.h”
Pred_result c_domain_bind_to_value(
DvarPtr *x*,
int *val*
);

**Μεταβλητές:**

*x* – μια domain variable.
*val* – ένας φυσικός αριθμός.

**Περιγραφή:**

Αυτή η συνάρτηση προσπαθεί να αναθέσει στην μεταβλητή χ μια δεδομένη τιμή. Αν η τιμή αυτή βρίσκεται στο πεδίο τιμών τότε την αναθέτει και αν όχι τότε η συνάρτηση τερματίζει επιστρέφοντας μας 0.

- **1.7.6 c_domain_value()**

#include “chipc.h”
int c_domain_value(
DvarPtr *x*,
int * *val*
)

**Μεταβλητές:**

*x* - μια domain variable.

*val* – ένας δείκτης σε integer.

**Περιγραφή:**

Αν η domain variable μας έχει ανατεθεί σε μια συγκεκριμένη τιμή τότε επιστρέφει την μεταβλητή αυτή στο val. Αν η μεταβλητέ δεν έχει αρχικοποιηθεί τότε επιστρέφει FAIL.

- **1.7.7 c_demon()**

#include “chipc.h”

Pred_result c_demon(

DvarPtr *x,*

Pred_result (* *callback*)(),

void * *arg*,

Ctr_event *event*

);

**Μεταβλητές:**

*x* – μια domain variable.

*callback* – ένας δείκτης σε μια συνάρτηση τύπου Pred_result.

*arg* – ένας δείκτης σε *void* όπου είναι η παράμετρος που θα πάρει η συνάρτηση callback().

*Event* - παίρνει τις ακόλουθες τιμές: GROUND,MIN, MAX, MINMAX or ALL.

**Περιγραφή:**

Με αυτήν την συνάρτηση μπορούμε να δημιουργήσουμε ένα trigger για μια domain variable το οποίο θα ενεργοποιηθεί με κάποιο συγκεκριμένο γεγονός εμφανιστεί σε αυτή την συνάρτηση. Έτσι με αυτό τον τρόπο μπορούμε να δημιουργήσουμε περιορισμούς οι οποίοι ενεργοποιούνται μόνο για ορισμένα γεγονότα. Είναι σε ένα πιο αφηρημένο επίπεδο σαν ένα if statement για περιορισμούς. Σε περίπτωση που το trigger αυτό ενεργοποιηθεί τότε καλείται η συνάρτηση *callback().* Τα γεγονότα που μπορούν να ενεργοποιήσουν αυτή την συνάρτηση είναι:

GROUND όπου η domain variable πρέπει να πάρει μια συγκεκριμένη τιμή.

MIN όπου η domain variable πρέπει να πάρει την πιο χαμηλή τιμή από το πεδίο τιμών.

MAX όπου η domain variable πρέπει να πάρει την πιο ψηλή τιμή από το πεδίο τιμών.

MINMAX όπου η domain variable πρέπει να πάρει την πιο χαμηλή ή την πιο ψηλή τιμή από το πεδίο τιμών.
ALL όταν γίνει κάποια αλλαγή στο πεδίο τιμών της domain variable.

- **1.7.8 c_dom_domcst()**

#include “chipc.h”
Pred_result c_dom_domcst(
DvarPtr *x,*
Ctr_linear_type *op*,
DvarPtr *y,*
int *c*
);

**Μεταβλητές:**
*x* - μια domain variable.
*y* - μια domain variable ή την τιμή NULL
*op* – μια από τις τιμές: GREATER,GREATER_EQUAL, EQUAL, LESS_EQUAL, LESS,DIFFERENT.
*c* – ένας φυσικός αριθμός.
**Περιγραφή:**
Αυτή η συνάρτηση είναι η κλασσική συνάρτηση της chipc για εισαγωγή περιορισμών. Με αυτή την συνάρτηση μπορούμε να αναγκάσουμε ισότητα, ανισότητα κ.α μεταξύ μεταβλητών και μιας σταθεράς. Αν η τιμή που προκύπτει ανήκει στο πεδίο τιμών τότε η συνάρτηση επιτυγχάνει και προστεθεί τον περιορισμό ανάλογα με τον συντελεστή σύγκρισης. Αν η τιμή αυτή δεν υπάρχει τότε η συνάρτηση επιστρέφει FAIL. Ο έλεγχος γίνεται κάθε φορά που αλλάζει το πεδίο τιμών και εκτελεί την πράξη : $x\ op\ y + c$.

- **1.7.9 c_if()**

Pred_result c_if(
DvarPtr *x*,
Ctr_linear_type *op,*

DvarPtr *y*,
int *c*,
int *(*ifthen)()*,
void **argifthen*
int *(*ifelse)(),*
void **argifelse*
);

**Μεταβλητές:**

*x* - μια domain variable.
*op* – μια από τις τιμές: GREATER,GREATER_EQUAL, EQUAL, LESS_EQUAL, LESS,DIFFERENT.
*y* - μια domain variable ή την τιμή NULL
*c* – ένας φυσικός αριθμός.
*ifthen() – ένας δείκτης σε μια συνάρτηση τύπου integer.
argifthen - ένας δείκτης σε void όπου αντιστοιχεί στην παράμετρο της ifthen συνάρτησης.
*ifelse() – ένας δείκτης σε μια συνάρτηση τύπου integer.
argiflse - ένας δείκτης σε void όπου αντιστοιχεί στην παράμετρο της ifelse συνάρτησης.

**Περιγραφή:**

Με αυτή την συνάρτηση μπορούμε να δημιουργήσουμε προγράμματα σε chipc που να υπακούν στην συνθήκη if-then-else που υπάρχει σε όλες σχεδόν τις γλώσσες. Με τον χειριστή op μπορούμε να δημιουργήσουμε μια συνθήκη τύπου: x (op) y + c. Με το που η συνθήκη έχει τα απαραίτητα δεδομένα για να πάρει την απόφαση αν είναι αληθής ή όχι τότε εκτελείτε και η συγκεκριμένη συνάρτηση. Οι τρεις πιθανές περιπτώσεις είναι:

1. *Για όλες της μεταβλητές στο πεδίο της domain variable x , y η συνθήκη είναι πάντα αληθής η συνάρτηση ifthen() καλείτε με παράμετρο argifthen.*
2. *Για όλες της μεταβλητές στο πεδίο της domain variable x , y η συνθήκη είναι πάντα ψευδής η συνάρτηση ifthen() καλείτε με παράμετρο argifthen*

3. *Η συνάρτηση δεν έχει της απαραίτητες πληροφορίες για να αποφασίσει αν η συνθήκη είναι αληθής ή ψευδής τότε κάνει delay (συνεχίζει με την εκτέλεση του υπόλοιπου κώδικα) μέχρι να αποκτήσει περισσότερες πληροφορίες. Στην δική μας περίπτωση θα κάνει delay μέχρι να φτάσει στην labeling όπου οι μεταβλητές θα αρχίσουν να παίρνουν τιμές.*

- **1.7.10 c_if_in()**

Pred_result c_if_in(
DvarPtr *x*,
int * *array*,
int *size*,
int *(*ifthen)()*,
void **argifthen*
int *(*ifelse)(),*
void **argifelse*
);

**Μεταβλητές:**

*x* - μια domain variable.
*array* – ένας πίνακας από φυσικούς αριθμούς
*size – ένας φυσικός αριθμός που δηλώνει το μέγεθος του πίνακα.*
*ifthen() – ένας δείκτης σε μια συνάρτηση τύπου integer.
argifthen - ένας δείκτης σε void όπου αντιστοιχεί στην παράμετρο της ifthen συνάρτησης.
*ifelse() – ένας δείκτης σε μια συνάρτηση τύπου integer.
argiflse - ένας δείκτης σε void όπου αντιστοιχεί στην παράμετρο της ifelse συνάρτησης.

**Περιγραφή:**

Με αυτή την συνάρτηση μπορούμε να δημιουργήσουμε προγράμματα σε chipc που να υπακούν στην συνθήκη (if (χ==array[V1] || χ==array[V2] || χ==array[V3] ….) όπου οι μεταβλητές μας είναι οι τιμές που υπάρχουν στον πίνακα array. Με το που η

συνθήκη έχει τα απαραίτητα δεδομένα για να πάρει την απόφαση αν είναι αληθής ή όχι τότε εκτελείτε και η συγκεκριμένη συνάρτηση. Οι τρεις πιθανές περιπτώσεις είναι:

1. *Για όλες της μεταβλητές στο πεδίο της domain variable x , y η συνθήκη είναι πάντα αληθής η συνάρτηση ifthen() καλείτε με παράμετρο argifthen.*
2. *Για όλες της μεταβλητές στο πεδίο της domain variable x , y η συνθήκη είναι πάντα ψευδής η συνάρτηση ifthen() καλείτε με παράμετρο argifthen*
3. *Η συνάρτηση δεν έχει της απαραίτητες πληροφορίες για να αποφασίσει αν η συνθήκη είναι αληθής ή ψευδής τότε κάνει delay (συνεχίζει με την εκτέλεση του υπόλοιπου κώδικα) μέχρι να αποκτήσει περισσότερες πληροφορίες. Στην δική μας περίπτωση θα κάνει delay μέχρι να φτάσει στην labeling όπου οι μεταβλητές θα αρχίσουν να παίρνουν τιμές.*

- **1.7.11 c_labeling()**

#include “chipc.h”
int c_labeling(
DvarPtr *array[],*
int *size,*
Strategy *strategy,*
LabelingPtr *info*
);

**Μεταβλητές:**

*array[]* – πίνακας από domain variables.
*size* – φυσικός αριθμός που δηλώνει το μέγεθος του πίνακα.
*Strategy* – μια από τις τιμές που δηλώνουν με ποια σειρά θα πάρουν τιμές οι μεταβλητές μας:
METHOD_ACTIVE, METHOD_FIRST_FAIL,
METHOD_INPUT_ORDER, METHOD_LARGEST,
METHOD_MAX_OF_MIN, METHOD_MIN_OF_MAX,
METHOD_MAX_REGRET, METHOD_MOST_CONSTRAINED,
METHOD_OCCUR, METHOD_SIZE or

METHOD_SMALLEST.

*info* – ένας δείκτης σε NULL που δείχνει στο πιο κάτω structure:

typedef struct {

Labeling_type type,

Labeling_indomain indomain,

Labeling_solution solution,

int middle,

int (*first_value)(),

int (*next_value)(),

int *member,

int size_member,

int (*continue_var)(),

int (*continue_all)(),

Labeling_search_type search_type,

Labeling_partial_search partial_search,

int remove_failed_value

} Labeling

Όπου:

Το type πάιρνει τιμές:

LABELING_INDOMAIN, LABELING_USER_DEFINED or

LABELING_MEMBER

Το indomain παίρνει τιμές:

LABELING_INDOMAIN_MIN,

LABELING_INDOMAIN_MAX,

LABELING_INDOMAIN_MIDDLE, or

LABELING_INDOMAIN_VALUE

Το solution παίρνει:

LABELING_FIRST_SOL or LABELING_ALL_SOL

Το search_type παίρνει τις τιμές:

CHIP_DEFAULT_SEARCH, CHIP_LFS, CHIP_BARRIER

or CHIP_CREDIT

**Περιγραφή:**

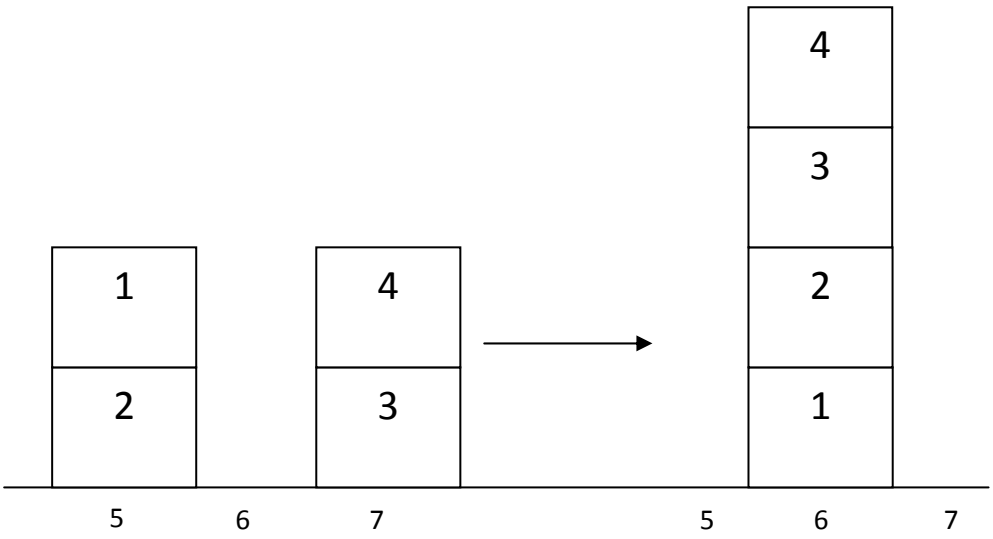
Αυτή η συνάρτηση είναι η κλασική συνάρτηση της chip για εύρεση λύσεων. Με βάση του πίνακα που παίρνει σαν παράμετρο βρίσκει λύσεις οι οποίες ικανοποιούν τους περιορισμούς που δοθήκαν από άλλες συναρτήσεις. Βασικά αυτή η συνάρτηση δημιουργεί ένα δέντρο με όλες τις πιθανές τιμές μέσα από το πεδίο τιμών. Μετά χρησιμοποιώντας την μέθοδο depth first διασχίζει όλο το δέντρο δοκιμάζοντας μια μια τιμή. Αν αυτή η τιμή ικανοποιεί τους περιορισμούς μας την δίνει στην μεταβλητή. Αν όχι τότε την βγάζει από το πεδίο τιμών της μεταβλητής. Αν φτάσει σε κάποιο σημείο όπου δεν μπορεί να θέσει κάποια τιμή στην μεταβλητή τότε κάνει backtrack στο δέντρο προσπαθώντας να δώσει στην προηγουμένη μεταβλητή κάποια άλλη τιμή. Τέλος αν ο αλγόριθμος δοκιμάσει όλους τους πιθανούς συνδυασμούς και δεν καταφέρει να βρει λύση αποτυγχάνει επιστρέφοντας FAIL. [3]

# Κεφάλαιο 2 Ο κόσμος των κύβων:

Υπάρχουν διάφορα προβλήματα που η επίλυση τους γίνεται με γλώσσες λογικού προγραμματισμού όπως για παράδειγμα ο χρωματισμός χάρτη, χρονικός προγραμματισμός εργασιών και ο κόσμος των κύβων το οποίο θα υλοποιήσουμε.

### 2.1 Το πρόβλημα:

Σε αυτό το σενάριο είμαστε σε ένα μικρόκοσμο ο οποίος αποτελείται από κύβους οι οποίοι είναι τοποθετημένοι στο πάτωμα με μια αρχική κατάσταση(βλ. Σχήμα). Ξέρουμε επίσης και την τελική κατάσταση στην οποία πρέπει να μεταβούμε(βλ. Σχήμα). Σκοπός μας είναι να μεταβούμε από την αρχική κατάσταση στην τελική μέσο κάποιων επιτρεπτών κινήσεων. Μια κίνηση είναι επιτρεπτή αν ο κύβος πουν θα μετακινήσουμε είναι ελεύθερος(δεν έχει κάποιο κύβο από πάνω του) και αν ο χώρος ή ο κύβος που θα τον μετακινήσουμε είναι ελεύθερος. Πιο καλή λύση είναι αυτή με τις λιγότερες κινήσεις.



**2.2 Ο αλγόριθμος:**

Ξέροντας την αρχική και τελική κατάσταση μπορούμε να δημιουργήσουμε ένα δυσδιάστατο πίνακα. Σε αυτόν υπάρχουν στήλες όσες και οι κύβοι και γραμμές όσες είναι οι κινήσεις-2 που δικαιούμαστε να εκτελέσουμε μέχρι την επίλυση.

Σε αυτόν τον πίνακα η πρώτη γραμμή είναι η αρχική μας κατάσταση και η τελευταία η τελική μας κατάσταση. Ενδιάμεσα ο πίνακας μπορεί να συμπληρωθεί με τιμές οι οποίες ικανοποιούν τις προϋποθέσεις του προγράμματός μας. Πρέπει δηλαδή κάθε γραμμή του πίνακα να πάρει τιμές(θέσεις, κύβους που θα μεταβούν) φτάνει αυτές οι τιμές να ήταν ελεύθερες(clear) στην προηγούμενη κατάσταση(ι-1).

Κάθε θέση του πίνακα αντιπροσωπεύει πάνω σε τι βρίσκετε ο κύβος με όνομα την συγκεκριμένη θέση. Δηλαδή η πρώτη θέση του παραδείγματος μας δείχνει ότι ο πρώτος κύβος είναι πάνω στον δεύτερο, η δεύτερη θέση ότι ο δεύτερος κύβος βρίσκεται στην πρώτη θέση του πατώματος (αριθμός 5) κ.ο.κ.

| | | | |
|---|---|---|---|
| 2 | 5 | 7 | 3 |
| | | | |
| | | | |
| 6 | 1 | 2 | 3 |

Σε αυτό τον αλγόριθμο αυτό που κάνουμε είναι να γεμίσουμε με τιμές τις οποίες δίνει ο χρήστης την πρώτη και την τελευταία γραμμή του πίνακα για την αρχική και τελική θέση των κιβωτίων. Αμέσως μετά τρέχει ο αλγόριθμος γεμίζοντας

τις υπόλοιπες θέσεις του πίνακα. Οι περιορισμοί που θέτουμε είναι αυτοί που καθορίζουν τις ενδιάμεσες τιμές που θα πάρουν οι θέσεις του πίνακα.

Το πώς δουλεύουν οι περιορισμοί είναι να πάρουν πρώτα μια τιμή μέσα από το πεδίο τιμών των κύβων. Αμέσως βλέπουν αν ο κύβος αυτός δεν έχει κάποια θέση στον πίνακα που να έχει την ίδια τιμή με του κύβου που βρισκόμαστε δηλ. αν ο κύβος μας είναι ελεύθερος να μετακινηθεί. Τέλος βλέπει την τιμή την οποία παίρνει ο κύβος η οποία πρέπει να μην έχει κάποια τιμή του πίνακα στην προηγουμένη θέση με την ίδια τιμή δηλ. η θέση στην οποία θα μεταβούμε να είναι ελεύθερη. Οι τιμές τις οποίες παίρνει ένας κύβος είναι οι διαθέσιμες θέσεις του πατώματος και οι τιμές των άλλων κύβων δηλ. ένας κύβος μπορεί να βρίσκετε είτε στο πάτωμα είτε πάνω σε κάποιο άλλο κύβο. Αυτός ο αλγόριθμος συνεχίζεται μέχρι να συμπληρωθούν όλες οι θέσεις του πίνακα. Υπάρχει η πιθανότητα όμως ο αλγόριθμος να μην καταφέρει να ολοκληρωθεί από το πρώτο πέρασμα. Αυτό μπορεί να γίνει αν οι προσπάθειες που έχουμε για να φτάσουμε στην τελική κατάσταση δεν είναι αρκετές για να μπορέσουμε να βρεθούμε στην τελική κατάσταση χωρίς να παραβιάζουμε τους περιορισμούς. Τότε ο αλγόριθμος θα κάνει οπισθοδρόμηση και θέση νέα αρχική τιμή για την πρώτη μετάβαση του πρώτου κύβου πράγμα που μπορεί να οδηγήσει σε διαφορετικές μεταβάσεις των κύβων μας και στην συνέχεια. Αν και αυτό αποτύχει θα δόση διαφορετική τιμή και για την πρώτη μετάβαση του δεύτερου κύβου. Αυτό θα συνεχιστεί δοκιμάζοντας όλους τους πιθανούς συνδυασμούς που υπάρχουν. Τέλος αν ο αλγόριθμος δεν καταφέρει, αφού δοκιμάσει όλους τους συνδυασμούς, τότε τερματίζει βγάζοντας το αποτέλεσμα ότι δεν υπάρχει πιθανή λύση.

### 2.3 Μοντελοποίηση

Στο πιο κάτω παράδειγμα ας υποθέσουμε ότι τα κουτιά μας είναι τέσσερα, οι θέσεις πατώματος είναι τρεις και οι προσπάθειες είναι δύο. Τότε ο πίνακας που θα δημιουργηθεί θα έχει τέσσερεις γραμμές το οποίο αντιστοιχεί σε προσπάθειες συν δύο, και τέσσερεις στήλες που αντιστοιχούν στον αριθμό των κύβων. Κάθε γραμμή αναπαριστά μια προσπάθεια, κάθε στήλη αναπαριστά το κουτί στο οποίο βρισκόμαστε και η τιμή αυτού του πεδίου αναπαριστά την θέση του κουτιού στην συγκεκριμένη προσπάθεια. Κάθε θέση του πίνακα έχει πεδίο τιμών από ένα μέχρι εφτά που αντιστοιχεί στον αριθμό των κύβων συν τον αριθμό των θέσεων του πατώματος. Αρχικά θα συμπληρωθεί η πρώτη και τελευταία γραμμή του πίνακα

(αρχική και τελική θέση των κύβων) και οι υπόλοιπες θέσεις θα πάρουν τιμές σύμφωνα με τους περιορισμούς μας.

| | | | |
|---|---|---|---|
| 2 | 5 | 7 | 3 |
| 1-7 | 1-7 | 1-7 | 1-7 |
| 1-7 | 1-7 | 1-7 | 1-7 |
| 6 | 1 | 2 | 3 |

### 2.4 Περιγραφή επίλυσης στην chip:

Το πρόγραμμά μας ξεκινά από την main.

Αμέσως παίρνουμε από τον χρήστη τις απαραίτητες τιμές που αφορούν το πρόγραμμα όπως αριθμό κύβων, διαθεσίμων θέσεων και προσπαθειών που πρέπει να γίνουν για να βρεθεί λύση.

```
printf("Kalwsirthate sto programa twn kibwn\n");
printf("Dwse arithmo twn blocks:");
scanf("%d",&blocks);
printf("Dwse arithmo twn prospathiwn pou tha ginoun mexri na brethei lisi:");
scanf("%d",&tries);
printf("Dwse arithmo diathesimwn thesew pou iparxoun sto patwma:");
scanf("%d",&floor);
```

Αμέσως μετά δεσμεύουμε χώρο για τον πίνακα vars ο οποίος είναι ένας πίνακας DvarPtr ο οποίος είναι αυτός που θα δουλέψουμε στην συνέχεια για να μας βρει την διαδρομή που θα ακολουθήσουν οι κύβοι έτσι ώστε να μεταφερθούν με επιτυχία στον προορισμό τους. Αυτός ο πίνακας έχει μέγεθος ίσο με τον αριθμό των κύβων επί τις προσπάθειες που πρέπει να γίνουν. Αυτό το κάνουμε με την λογική ότι οι τιμές που πρέπει να βρούμε λύση για τιμές ίσες με τον πιο πάνω αριθμό.

```
vars = (void**)malloc(sizeof(int)* (blocks*tries));
```

Μετά δίνουμε το πεδίο τιμών που θα έχει κάθε μια από αυτές τις μεταβλητές δηλ. τις τιμές που μπορεί να έχει κάθε κύβος. Αφού κάθε κύβος μπορεί να βρίσκεται σε κάποια από της ελεύθερες θέσεις που υπάρχουν ή σε κάποιο άλλο κύβο οι τιμές αυτές θα ξεκινούν από ένα και θα φτάνουν μέχρι και τον αριθμό των κύβων επί τον αριθμό των ελεύθερων θέσεων του κάθε προγράμματος. Έτσι αν ένα κουτί πάρει κάποια τιμή μεγαλύτερη από των αριθμό των κύβων σημαίνει ότι βρίσκεται σε κάποια διαθέσιμη θέση του πατώματος και με τον υπολογισμό τις διαφοράς κιβωτίων με θέση που βρισκόμαστε μπορούμε να βρούμε ποια θέση του πατώματος είναι αυτή.

```
c_create_domain_array(&vars[0], blocks*tries ,1,blocks+floor,CONSEC);
```

Εδώ δεσμεύουμε χώρο για δυο πίνακας. Ο δυο αυτοί πίνακες αφορούν τις αρχικές και τελικές θέσεις των κύβων αφού θα τις χρειαστούμε μετέπειτα στους περιορισμούς.

```
on= (int *)malloc(sizeof(int)*blocks);

on_f= (int *)malloc(sizeof(int)*blocks);
```

Εδώ απλά εισάγουμε τιμές στους πίνακες που μας δείχνουν την αρχικές θέσεις των φορτηγών και τις αρχικές και τελικές θέσεις των κιβωτίων. Αυτοί οι πίνακες έχουν δημιουργηθεί προηγούμενος.

```
printf("dwse to simeio pou brisketai to kathe block (an einai sto patoma tote dwse %d-%d)\n",blocks+1,blocks+floor);

for(i=0; i<blocks; i++){

    printf("block %d:",i+1);

    scanf("%d",&on[i]);
```

```
            while (on[i]<=0 || on[i]>(blocks+floor)){
                printf("Den iparxei tetoia thesi, ksanadwse arithmo:");
                scanf("on:%d",&on[i]);
            }
        }
printf("dwse to simeio pou tha brisketai to kathe mplock stin teliki katastasi(an einai sto patoma tote dwse %d-%d)\n",blocks+1,blocks+floor);
        for(i=0; i<blocks; i++){
            printf("block %d:",i+1);
            scanf("%d",&on_f[i]);
            while (on[i]<=0 || on[i]>(blocks+floor)){
                printf("Den iparxei tetoia thesi, ksanadwse arithmo:");
                scanf("on:%d",&on[i]);
            }
        }
```

Εδώ καλούμε την συνάρτηση cons όπου είναι η συνάρτηση που μας προσθέτει τους περιορισμούς στις μεταβλητές μας.

```
cons(on,on_f,blocks,tries,vars);
```

Τώρα ας δούμε βήμα βήμα πως θα λειτουργήσει αυτή η συνάρτηση.
Αρχικά βάζουμε των περιορισμό ότι η πρώτη γραμμή του πίνακα που vars πρέπει να είναι ίση με τον πίνακα τις αρχικής κατάστασης των κύβων. Έτσι επιτυγχάνουμε οι κύβοι μας να ξεκινούν την διαδρομή τους από την αρχική θέση.

```
for(i=0;i<blocks;i++){
    c_dom_domcst(vars[i], EQUAL,NULL, on[i]);
```

```
        }
```

Το ίδιο ακριβώς με την αρχική κατάσταση γίνεται και με την τελική κατάσταση των κύβων

```
for(i=0;i<blocks;i++){
                c_dom_domcst(vars[blocks*(tries-1)+i], EQUAL,NULL, on_f[i]);
        }
```

Πιο κάτω προσθέτουμε τον περιορισμό ο οποίος μας διασφαλίζει ότι ο κύβος που θα μετακινηθεί είναι ελεύθερος. Αυτό γίνεται με το να δώσουμε τον περιορισμό ότι κάθε τιμή που θα πάρει ο κύβος πρέπει να είναι διαφορετική με όλες τις τιμές των κύβων στην προηγουμένη κατάσταση. Έτσι αν κανένας κύβος δεν έχει την τιμή του κύβου μας σημαίνει ο κύβος αυτός δεν έχει κανένα κύβο να βρίσκεται πάνω του. Έτσι είναι και ελεύθερος να μετακινηθεί.

```
//dimiourga periorismou wste na einai to block pou tha pame elefthero
        for(i=blocks;i<blocks*tries;i++){   //for gia grammes ektos apo tin prwti
                for(j=1; j<blocks; j++){   //for gia stiles

                                        c_dom_domcst(vars[i], DIFFERENT,vars[i-
j],0);
                }
        }
}
```

Μετά θέτουμε ένα trigger το οποίο ενεργοποιείτε για κάθε μεταβλητή μόλις πάρει μια συγκεκριμένη τιμή. Το αποτέλεσμα αυτού είναι να καλεστεί η συνάρτηση callback.

```
for(i=0;i<(blocks*tries)-blocks; i++){              //gia ola ta blocks
                        c_demon(vars[i], callback, arg, GROUND);          //otan parei
timi i thesi pou ginetai afipnisi tou demon
```

```
        }
```

Με την συνάρτηση callback προσθέτουμε τον περιορισμό ώστε αν υπάρχει κάποιος κύβος ο οποίος έχει τιμή κάποιου άλλου κύβου τότε σημαίνει ότι δεν είναι ελεύθερος να μετακινηθεί αφού υπάρχει από πάνω του κάποιος άλλος κύβος.

```
//dimoirgia periorismou wste to block pou tha metakinisoukme na einai elefthero
        c_domain_value(vars[count],&val);
        if(val <= blocks){
        c_dom_domcst(vars[count], EQUAL,vars[count+blocks] ,0);
        }
```

Μετά από τα πιο πάνω επιστρέφουμε στην main και προσπαθούμε να βρούμε αν υπάρχει πιθανή λύση στο πρόβλημά μας χρησιμοποιώντας την συνάρτηση c_labeling.

```
   options.type     = LABELING_INDOMAIN;//Use domain to find solution

   options.indomain = LABELING_INDOMAIN_MIDDLE;//Start from middle of
domain

   options.solution = LABELING_FIRST_SOL; //Find first possible solution and stop

   options.continue_var = 0; //Check is assigment was successful

   options.continue_all = show_result;//Use everything in domain and when done
show solution


            //Find solution if there is one

   if (c_labeling(vars,blocks*tries, METHOD_FIRST_FAIL, &options)==0) //If
FIRST FAIL is obtained then stop

        {

           printf("\nNo solution found!");

        }
```

```
c_chip_end();

return 0;

}
```

### 2.5 Παράδειγμα εκτέλεσης:

#### 2.5.1 Παράδειγμα πρώτο

Kalwsirthate sto programa twn kibwn
Dwse arithmo twn blocks:4
Dwse arithmo twn prospathiwn pou tha ginoun mexri na brethei lisi:2
Dwse arithmo diathesimwn thesew pou iparxoun sto patwma:3
dwse to simeio pou brisketai to kathe block (an einai sto patoma tote dwse 5-7)
block 1:2
block 2:5
block 3:7
block 4:3
to patoma xaraktirizete me ton arithmo 5-7
to block 1 brisketai pano sto 2
to block 2 brisketai pano sto 5
to block 3 brisketai pano sto 7
to block 4 brisketai pano sto 3
dwse to simeio pou tha brisketai to kathe mplock stin teliki katastasi(an einai sto patoma tote dwse 5-7)
block 1:6

block 2:1
block 3:2
block 4:3

Allagi prospatheias

To block 1 stin prospatheia 1 brisketai stin thesi 2
To block 2 stin prospatheia 1 brisketai stin thesi 5
To block 3 stin prospatheia 1 brisketai stin thesi 7
To block 4 stin prospatheia 1 brisketai stin thesi 3

Allagi prospatheias

To block 1 stin prospatheia 2 pigainei stin thesi 6
To block 2 stin prospatheia 2 pigainei stin thesi 1
To block 3 stin prospatheia 2 menei stin thesi 7
To block 4 stin prospatheia 2 pigainei stin thesi 2

Allagi prospatheias

To block 1 stin prospatheia 3 menei stin thesi 6
To block 2 stin prospatheia 3 menei stin thesi 1
To block 3 stin prospatheia 3 pigainei stin thesi 2
To block 4 stin prospatheia 3 pigainei stin thesi 3

Total time : 10 ms

Στο πιο πάνω παράδειγμα έχουμε δυο προσπάθειες για να φτάσουμε σε λύση. Τα κουτιά μας θέλουμε να μεταφερθούν όλα σε μια τελική θέση 6 του πατώματος το ένα πάνω στο άλλο ξεκινώντας από το ένα και φτάνοντας στο τέσσερα. Στην αρχική θέση το κουτί ένα βρίσκεται πάνω στο κουτί δύο και δεν έχει κάποιο άλλο κουτί από πάνω του. Οπόταν μπορεί και να μεταφερθεί σε κάποια θέση. Έτσι επιλέγει να πάει στην

θέση έξι του πατώματος όπου και είναι η τελική του θέση. Έτσι ελευθερώνεται ρο κουτί δύο όπου και αυτό μπορεί να μεταφερθεί κάπου αλλού επιλέγοντας το κουτί ένα που είναι και η τελική του θέση. Το κουτί τρία αφού έχει από πάνω του το κουτί τέσσερα δεν μπορεί να μεταφερθεί σε αυτήν την προσπάθεια. Τέλος το κουτί τέσσερα μπορεί να μεταφερθεί αφού είναι ελεύθερο και θα πάει σε κάποια άλλη ελεύθερη θέση όπου είναι η δύο που έχει βρει πρώτη.
Έτσι πηγαίνουμε στην επόμενή μας προσπάθεια όπου τα δυο πρώτα κουτιά μένουν στις θέσεις που ήταν και το κουτί τρία αφού είναι ελεύθερο τώρα μπορεί να μεταφερθεί στην τελική του κατάσταση που είναι στο κουτί δύο. Τέλος το κουτί τέσσερα μπορεί να πάει και αυτό στην τελική του κατάσταση που είναι το κουτί τρία.

### 2.5.2 Παράδειγμα δεύτερο

Kalwsirthate sto programa twn kibwn
Dwse arithmo twn blocks:4
Dwse arithmo twn prospathiwn pou tha ginoun mexri na brethei lisi:3
Dwse arithmo diathesimwn thesew pou iparxoun sto patwma:4
dwse to simeio pou brisketai to kathe block (an einai sto patoma tote dwse 5-8)
block 1:5
block 2:1
block 3:2
block 4:3
to patoma xaraktirizete me ton arithmo 5-8
to block 1 brisketai pano sto 5
to block 2 brisketai pano sto 1
to block 3 brisketai pano sto 2
to block 4 brisketai pano sto 3
dwse to simeio pou tha brisketai to kathe mplock stin teliki katastasi(an einai sto patoma tote dwse 5-8)
block 1:5
block 2:8

block 3:7
block 4:6

Allagi prospatheias

To block 1 stin prospatheia 1 brisketai stin thesi 5
To block 2 stin prospatheia 1 brisketai stin thesi 1
To block 3 stin prospatheia 1 brisketai stin thesi 2
To block 4 stin prospatheia 1 brisketai stin thesi 3

Allagi prospatheias

To block 1 stin prospatheia 2 menei stin thesi 5
To block 2 stin prospatheia 2 menei stin thesi 1
To block 3 stin prospatheia 2 menei stin thesi 2
To block 4 stin prospatheia 2 pigainei stin thesi 6

Allagi prospatheias

To block 1 stin prospatheia 3 menei stin thesi 5
To block 2 stin prospatheia 3 menei stin thesi 1
To block 3 stin prospatheia 3 pigainei stin thesi 7
To block 4 stin prospatheia 3 menei stin thesi 6

Allagi prospatheias

To block 1 stin prospatheia 4 menei stin thesi 5
To block 2 stin prospatheia 4 pigainei stin thesi 8
To block 3 stin prospatheia 4 menei stin thesi 7
To block 4 stin prospatheia 4 menei stin thesi 6
Total time : 10 ms

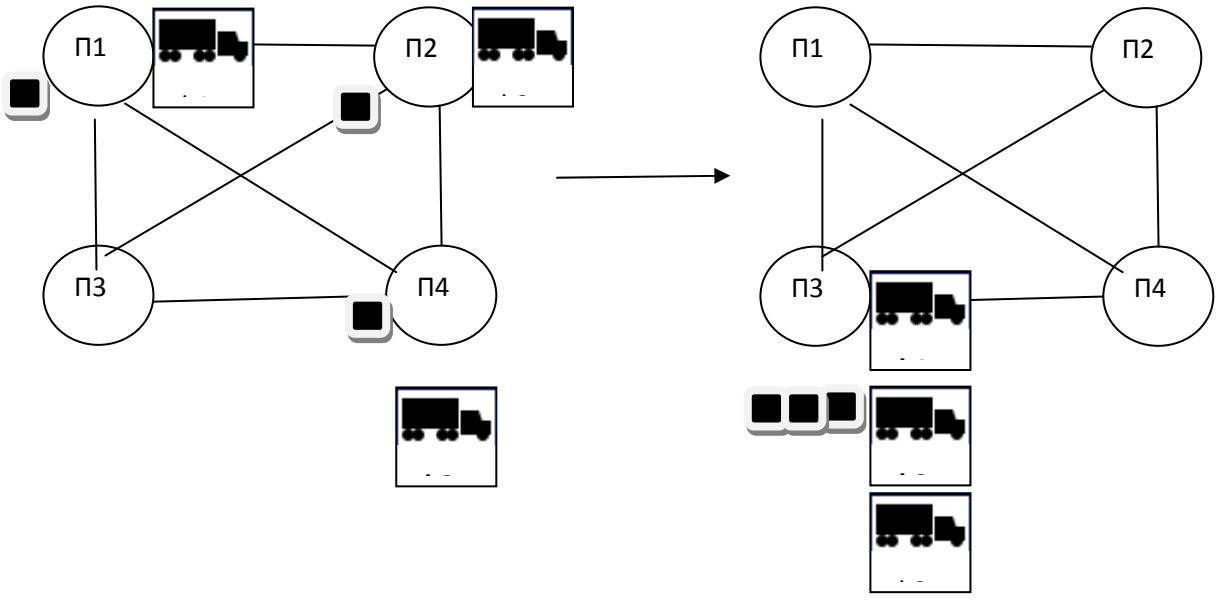
Στο πιο πάνω παράδειγμα έχουμε τρεις προσπάθειες να φτάσουμε σε λύση. Στην αρχική τους θέση τα κουτιά βρίσκονται στην θέση 5 το ένα πάνω στο άλλο σε αύξουσα σειρά ξεκινώντας από το κουτί ένα. Στην πρώτη προσπάθεια τα κουτιά από ένα μέχρι και τρία δεν μπορούν να μετακινηθούν. Μόνο το κουτί τέσσερα είναι ελεύθερο και θα επιλέξει να πάει στην τελική του θέση όπου είναι η θέση 6.
Έτσι στην επόμενη προσπάθεια το κουτί τρία θα ελευθερωθεί και θα πάει στην τελική του θέση που είναι η θέση 7.
Τέλος στην επόμενη προσπάθεια το κουτί δύο θα ελευθερωθεί και θα πάει στην τελική του θέση που είναι η θέση 8. Αφού το κουτί ένα βρίσκεται ήδη στην τελική του θέση έχουμε φτάσει λύση.

# Κεφάλαιο 3 Πρόβλημα μεταφοράς αγαθών με συγκεκριμένο κουτί στην τελική θέση:

Το επόμενο πρόβλημα το οποίο θα επιλύσουμε είναι η μεταφορά εμπορευμάτων από συγκεκριμένες πόλεις σε κάποιες άλλες.

### 3.1 Το πρόβλημα:

Σε αυτό το πρόβλημα βρισκόμαστε σε ένα μικρόκοσμο ο οποίος αποτελείται από πόλεις και φορτηγά τα οποία μεταφέρουν εμπορεύματα. Τα φορτηγά μπορεί να βρίσκονται αρχικά σε οποιαδήποτε από τις πόλεις και όχι απαραίτητα να είναι στην ίδια πόλη. Καθώς επίσης και το φορτία μπορεί να βρίσκονται αρχικά σε οποιαδήποτε από τις πόλεις . Επίσης όλες οι πόλεις επικοινωνούν απαραίτητα μεταξύ τους(βλ. Σχήμα). Σκοπός μας είναι να μεταφέρουμε τα φορτία από μια πόλη σε κάποια άλλη με κάποιες επιτρεπτές κινήσεις. Μια επιτρεπτή κίνηση θεωρείτε αυτή που περνά μόνο από πόλης οι οποίες είναι συνδεδεμένες μεταξύ τους(υπάρχει δρόμος για να μεταβούμε στην επόμενη). Πιο καλή λύση είναι αυτή με της λιγότερες κινήσεις.



### 3.2 Ο αλγόριθμος:

Ξέροντας την αρχική κατάσταση των φορτηγών μπορούμε να δημιουργήσουμε ένα δυσδιάστατο πίνακα από domain variables. Σε αυτόν υπάρχουν στήλες όσα και τα φορτηγά και γραμμές όσες είναι οι κινήσεις-2 που δικαιούμαστε να εκτελέσουμε μέχρι την επίλυση.

Σε αυτόν τον πίνακα η πρώτη γραμμή είναι η αρχική μας κατάσταση των φορτηγών. Ενδιάμεσα ο πίνακας μπορεί να συμπληρωθεί με τιμές οι οποίες ικανοποιούν τις προϋποθέσεις του προγράμματός μας. Πρέπει δηλαδή κάθε γραμμή του πίνακα να πάρει τιμές(πόλεις που θα μεταβούν) εκτός από την περίπτωση που το φορτηγό θα φορτώσει κάποιο κουτί οπόταν δεν δικαιούται να μεταβεί σε άλλη πόλη μέχρι να επιτευχθεί το φόρτωμα.

Κάθε θέση του πίνακα αντιπροσωπεύει σε ποια πόλη βρίσκεται το φορτηγό με όνομα την συγκεκριμένη θέση. Δηλαδή η πρώτη θέση του παραδείγματος μας δείχνει ότι το πρώτο φορτηγό βρίσκεται στην πρώτη πόλη, το δεύτερο στην δεύτερη πόλη και το τρίτο στην τέταρτη πόλη.

| 1 | 2 | 4 |
|---|---|---|
| | | |
| | | |
| | | |

Ξέροντας την αρχική και τελική κατάσταση των κουτιών μπορούμε να δημιουργήσουμε ένα δυσδιάστατο πίνακα από domain variables. Σε αυτόν υπάρχουν στήλες όσα και τα κουτιά και γραμμές όσες είναι οι κινήσεις-2 που δικαιούμαστε να εκτελέσουμε μέχρι την επίλυση.

Σε αυτόν τον πίνακα η πρώτη γραμμή είναι η αρχική κατάσταση και η τελευταία με την τελική κατάσταση των κουτιών. Ενδιάμεσα ο πίνακας μπορεί να συμπληρωθεί με τιμές οι οποίες ικανοποιούν τις προϋποθέσεις του προγράμματός μας. Πρέπει δηλαδή κάθε κουτί να μεταφερθεί σε κάποια πόλη μόνο μέσου ενός φορτηγού, για να φορτωθεί στο φορτηγό πρέπει να βρίσκονται στην ίδια πόλη και για να ξεφορτωθεί σε κάποια πόλη πρέπει να είναι η πόλη η οποία βρίσκεται το φορτηγό στο οποίο είναι φορτωμένο το κουτί μας .

Κάθε θέση του πίνακα αντιπροσωπεύει σε ποια πόλη βρίσκεται το φορτηγό με όνομα την συγκεκριμένη θέση ή σε ποιο φορτηγό είναι φορτωμένο με τιμή τον αριθμό του φορτηγού + πόλεις. Δηλαδή η πρώτη θέση του παραδείγματος μας δείχνει ότι το πρώτο κουτί βρίσκεται στην πρώτη πόλη, το δεύτερο στην δεύτερη πόλη και το τρίτο στην τέταρτη πόλη. Η τελευταία θέση του πίνακα μας δείχνει ότι τα κουτιά στην τελική τους κατάσταση πρέπει να βρίσκονται και τα τρία στην πόλη τρία.

| | | |
|---|---|---|
| 1 | 2 | 4 |
| | | |
| | | |
| 3 | 3 | 3 |

Τέλος έχουμε ακόμη τρείς μονοδιάστατους πίνακες. Ο πρώτος μας δείχνει που βρίσκονται τα κουτιά στην αρχική θέση, ο δεύτερος που βρίσκονται τα κουτιά στην τελική μας κατάσταση και ο τρίτος που βρίσκονται τα φορτηγά στην αρχική τους κατάσταση. Έτσι αν το φορτηγό βρει ότι σε κάποια πόλη υπάρχουν περισσότερα κουτιά από ότι θα έπρεπε θα πάρει τα κουτιά και θα προσπαθήσει να τα μεταφέρει στην πόλη η οποία τα έχει ανάγκη. Έτσι σύμφωνα με το παράδειγμα μας οι πίνακες θα ήταν οι ακόλουθοι:

| | | |
|---|---|---|
| 1 | 2 | 4 |

| | | |
|---|---|---|
| 3 | 3 | 3 |

| | | |
|---|---|---|
| 1 | 2 | 4 |

Το πώς τρέχει ο αλγόριθμος είναι να δημιουργήσουμε ένα loop το οποίο θα επαναλαμβάνει κάθε φορά την ανάθεση περιορισμών ξεκινώντας από προσπάθειες ίσες με 2 αυξάνοντας τις προσπάθειες μέχρι να φτάσει στον μέγιστο αριθμό προσπαθειών. Τώρα μέσα σε αυτό το loop δημιουργούμε τους πίνακες των domain variables όπως ακριβώς έχουμε εξηγήσει πιο πάνω. Πάνω σε αυτούς τους πίνακες μπαίνουν οι πιο κάτω περιορισμοί:

- Περιορισμός για φόρτωμα

Για να επιτευχθεί το φόρτωμα πρέπει να διασφαλίσουμε ότι τα κουτιά μας μπορούν να πάρουν την τιμή ενός φορτηγού μόνο αν αυτό το φορτηγό στην συγκεκριμένη προσπάθεια βρίσκεται στην ίδια πόλη.

- Περιορισμός για ξεφόρτωμα

Για να επιτευχθεί το ξεφόρτωμα πρέπει να διασφαλίσουμε ότι τα κουτιά μας μπορούν να πάρουν την τιμή μιας πόλης μόνο αν το φορτηγό στο οποίο είναι φορτωμένο βρίσκεται σε αυτήν την πόλη

- Περιορισμοί για να μην μετακινηθούν τα κουτιά αν δεν έχουν πρώτα φορτωθεί.

Για να διασφαλίσουμε ότι τα κουτιά μας δεν θα μεταφερθούν από μια πόλη σε κάποια άλλη πρέπει να ελέγχουμε ότι αν δεν έχει επιτευχθεί φόρτωμα και το κουτί βρίσκεται σε κάποια πόλη στην επόμενη προσπάθεια να μείνει στην συγκεκριμένη πόλη.

- Βελτιστοποιήσεις

- Σε περίπτωση που δεν γίνει φόρτωμα τότε το φορτηγό που δεν θα έχει φορτωμένο κανένα κουτί στην επόμενή του προσπάθεια θα πάει σε κάποια πόλη στην οποία θα υπάρχει κουτί. Έτσι αποφεύγουμε τις περιττές μετακινήσεις των φορτηγών.
- Όταν τελειώσει ο καθορισμός των περιορισμών προσθέτουμε ένα επιπλέον ένα περιορισμό για τα φορτηγά ο οποίος μας καθορίζει ότι πρέπει τα φορτηγά να περάσουν τουλάχιστο μια φορά από τις πόλης που βρίσκονται στην τελική τους κατάσταση τα κουτιά.

### 3.3 Μοντελοποίηση

Ας υποθέσουμε ότι ο αριθμός των πόλεων είναι ίσος με τέσσερα, ο αριθμός των φορτίων είναι ίσος με τρία και οι μέγιστες προσπάθειες που διακαιόμαστε είναι δύο.

Αρχικά το πρόγραμμά μας θα δοκιμάσει να βρει λύση σε μία προσπάθεια. Έτσι οι πίνακες που θα δημιουργηθούν θα είναι:

Ο πίνακας που αναπαριστά τα φορτία θα έχει δυο γραμμές που αντιστοιχούν σε μια προσπάθεια και τρείς στήλες που αντιστοιχούν στον αριθμό των φορτίων. Τα πεδία τιμών κάθε θέσης του πίνακα είναι από ένα μέχρι και εφτά που είναι ο αριθμός των πόλεων συν τα φορτηγά.

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 5 | 6 | 7 |

Ο πίνακας που αναπαριστά τα φορτηγά θα έχει δυο γραμμές που αντιστοιχούν σε μια προσπάθεια και τρείς στήλες που αντιστοιχούν στον αριθμό των φορτηγών. Τα πεδία τιμών κάθε θέσης του πίνακα είναι από ένα μέχρι και τέσσερα που είναι ο αριθμός των πόλεων.

Το πρόγραμμα θα προσπαθήσει να βρει λύση για μία προσπάθεια όπου και θα αποτύχει αφού δεν θα μπορέσει να μεταφέρει τα φορτία στην τελική τους θέση

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 2 | 4 |

Στην επόμενη προσπάθεια έχοντας δυο προσπάθειες στην διάθεση μας ο πίνακας των φορτίων και των φορτηγών θα έχουν τρείς γραμμές. Έτσι το πρόγραμμά μας θα μπορέσει να βρει λύση.

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 2 | 4 |
| 3 | 3 | 3 |

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 5 | 6 | 7 |
| 3 | 3 | 3 |

### 3.4 Περιγραφή επίλυσης στην chip:

Το πρόγραμμά μας ξεκινά την εκτέλεσή του από την main.

Αμέσως παίρνουμε από τον χρήστη τις απαραίτητες τιμές που αφορούν το πρόγραμμα όπως αριθμό πόλεων, φορτηγών και προσπαθειών που πρέπει να γίνουν για να βρεθεί λύση.

```
printf("Kalosorisate sto programa metaforas fortiwn\n");
printf("dwse ton arithmo twn polewn pou tha iparxoun:");
scanf("%d",&cities);
printf("dwse ton arithmo twn fortigwn pou iparxoun:");
scanf("%d",&trucks);
printf("dwse ton arithmo twn fortiwn pou iparxoun:");
scanf("%d",&box);
printf("dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:");
scanf("%d",&maxtries);
```

Μετά αχικοποιούμε των αρχικό αριθμό των προσπαθειών σε 2 αφού για να γίνει μια προσπάθεια τότε πρέπει να υπάρχουν 2 θεσεις στον πίνακα. Επίσης αρχικοποιούμε τον μέγιστο αριθμό προσπαθειών με τον αριθμό που πήραμε από τον χρήστη +2.

```
tries=2;
maxtries=maxtries+2;
```

Μετά δεσμέυουμε χώρο για τους πίνακες if_in και final_state. Ο πίνακας if_in είναι ένας πίνακας μεγέθους ίσου με τα φορτιγά όπου θα χρεισιμοποιηθεί αργότερα στην συνάρτηση c_if_in(). Ο πίνακας final_state είναι ένας πίνακας μεγέθους ίσου με τον αριθμό των κουτιών όπου θα μας χρεισημέυση για να δούμε αν υπάρχει λύση.

```
if_in=(int *)malloc(sizeof(int)*trucks);
final_state=(int *)malloc(sizeof(int)*box);
```

Εδώ δεσμεύουμε χώρο για τρεις πίνακας. Ο πρώτος αφορά τις αρχικές θέσεις των φορτηγών αφού θα τις χρειαστούμε μετέπειτα στους περιορισμούς και οι άλλοι δυο

αφορούν τις αρχικές και τελικές θέσεις των φορτίων αφού θα τους χρειαστούμε για τον ίδιο ακριβώς λόγω.

```
on= (int *)malloc(sizeof(int)*trucks);
boxes= (int *)malloc(sizeof(int)*box);
boxes_f= (int *)malloc(sizeof(int)*box);
```

Πιο κάτω βάζουμε τιμές στον πίνακα που θα χρησιμοποιήσουμε πιο κάτω στην συνάρτηση c_if_in. Έτσι σε αυτό τον πίνακα μπαίνουν όλες οι πόλεις του προγράμματος μας.

```
for(i=0;i<trucks;i++){
        if_in[i]=i+1;
}
```

Εδώ απλά εισάγουμε τιμές στους πίνακες που μας δείχνουν την αρχικές θέσεις των φορτηγών και τις αρχικές και τελικές θέσεις των κιβωτίων. Αυτοί οι πίνακες έχουν δημιουργηθεί προηγούμενος.

```
        printf("dwse tin poli pou brisketai to kathe fortigo\n");
                for(i=0; i<trucks; i++){
                        printf("truck %d:",i+1);
                        scanf("%d",&on[i]);

                        while (on[i]<=0 || on[i]>cities){
                                printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                                scanf("%d",&on[i]);
                        }
                }
//////////////////////

        printf("dwse tin poli pou brisketai to kathe fortio\n");
                for(i=0; i<box; i++){
                        printf("fortio %d:",i+1);
```

```
            scanf("%d",&boxes[i]);

            while (on[i]<=0 || on[i]>cities){
                printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                scanf("%d",&boxes[i]);
            }
        }
        print_boxes(box,boxes);
        printf("dwse to simeio pou tha brisketai to kathe fortio liki
        katastasi\n");
        for(i=0; i<box; i++){
            printf("fortio %d:",i+1);
            scanf("%d",&boxes_f[i]);

            while (on[i]<=0 || on[i]>cities){
                printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                scanf("%d",&boxes_f[i]);
            }
        }
```

Μετά στον κώδικα μας υπάρχει το for στο οποίο γίνεται η αλλαγή των προσπαθειών. Αυτό το loop εκτελείτε από προσπάθειες ίσες με 2 και προχωρά μέχρι να βρεθεί λύση ή μέχρι να φτάσει των μέγιστο αριθμό προσπαθειών που πήραμε από τον χρήστη.
for(tries=2; tries<maxtries; tries++, pos=0, pos2=0,k=0, k2=0,pos3=0)

Αμέσως μετά δεσμεύουμε χώρο για τον πίνακα vars ο οποίος είναι ένας πίνακας DvarPtr ο οποίος είναι αυτός που θα δουλέψουμε στην συνέχεια για να μας βρει την διαδρομή που θα ακολουθήσουν τα φορτηγά έτσι ώστε τα κιβώτια μας να μεταφερθούν με επιτυχία στον προορισμό τους. Αυτός ο πίνακας έχει μέγεθος ίσο με τον αριθμό των φορτηγών επί τις προσπάθειες που πρέπει να γίνουν. Αυτό το κάνουμε με την λογική ότι οι τιμές που πρέπει να βρούμε λύση για τιμές ίσες με τον πιο πάνω αριθμό.

```
vars =  (void**)realloc(vars,sizeof(int)*(trucks*tries));
```

Αμέσως μετά δεσμεύουμε χώρο για τον πίνακα vars2 ο οποίος είναι ένας πίνακας DvarPtr ο οποίος είναι αυτός που θα δουλέψουμε στην συνέχεια για να μας βρει την διαδρομή που θα ακολουθήσουν τα κουτιά έτσι ώστε να σιγουρευτούμε μεταφερθούν με επιτυχία στον προορισμό τους. Αυτός ο πίνακας έχει μέγεθος ίσο με τον αριθμό των κουτιών επί τις προσπάθειες που πρέπει να γίνουν. Αυτό το κάνουμε με την λογική ότι οι τιμές που πρέπει να βρούμε λύση για τιμές ίσες με τον πιο πάνω αριθμό.

```
vars2 = (void**)realloc(vars2,sizeof(int)*(box*tries));
```

Μετά δεσμεύουμε χώρο για το struct param. Αυτό το struct μας χρησιμεύει στο να φυλάγουμε τις παραμέτρους που θα χρειαστούν μετά οι συναρτήσεις  load, unload και stay.

Σε αυτό το struct υπάρχουν 2 πεδία. Το πρώτο είναι το πεδίο curr_box και το δεύτερο curr_truck. Το πρώτο πεδίο χρησιμοποιείται για να φυλάγουμε το αριθμό του πίνακα των κουτιών  που θα προστεθούν οι περιορισμοί μας στις αντίστοιχες συναρτήσεις και το δεύτερο πεδίο χρησιμοποιείται για να φυλάγουμε το αριθμό του πίνακα των φορτηγών  που θα προστεθούν οι περιορισμοί μας στις αντίστοιχες συναρτήσεις

```
param =(ARGUM *)realloc(param,sizeof(ARGUM));
param->curr_box= (int *) realloc(param->
curr_box,sizeof(int)*(box*tries*trucks));
param->curr_truck=(int*) realloc(param->
curr_truck,sizeof(int)*(box*tries*trucks));
```

Μετά δίνουμε το πεδίο τιμών που θα έχει κάθε μια από τις μεταβλητές του πίνακα vars δηλ. τις τιμές που μπορεί να έχει κάθε φορτηγό. Αφού κάθε φορτηγό μπορεί να βρίσκεται σε κάποια από της πόλεις μας οι τιμές αυτές θα ξεκινούν από ένα και θα φτάνουν μέχρι και τον αριθμό των πόλεων του κάθε προγράμματος.

```
c_create_domain_array(vars, trucks*tries+1, 1, cities, CONSEC);
```

Μετά δίνουμε το πεδίο τιμών που θα έχει κάθε μια από τις μεταβλητές του πίνακα vars2 δηλ. τις τιμές που μπορεί να έχει κάθε κουτί. Αφού κάθε κουτί μπορεί να βρίσκεται σε κάποια από της πόλεις ή σε κάποιο φορτηγό οι τιμές αυτές θα ξεκινούν

από ένα και θα φτάνουν μέχρι και τον αριθμό των πόλεων + τον αριθμό των φορτηγών του προγράμματος.

```
c_create_domain_array(vars2, box*tries, 1, cities+trucks, CONSEC);
```

Εδώ καλούμε την συνάρτηση cons όπου είναι η συνάρτηση που μας προσθέτει τους περιορισμούς στις μεταβλητές μας.

```
cons(grafos,on,boxes,boxes_f,cities,trucks,tries,box,vars,vars2);
```

Τώρα ας δούμε βήμα βήμα πως θα λειτουργήσει αυτή η συνάρτηση. Αρχικά βάζουμε των περιορισμό ότι η πρώτη γραμμή του πίνακα που vars πρέπει να είναι ίση με τον πίνακα τις αρχικής κατάστασης των φορτηγών. Έτσι επιτυγχάνουμε τα φορτηγά μας να ξεκινούν την διαδρομή τους από την αρχική θέση.

```
for(i=0;i<trucks;i++){
        c_dom_domcst(vars[i], EQUAL,NULL, on[i]);
}
```

Μετά βάζουμε των περιορισμό ότι η πρώτη γραμμή του πίνακα που vars2 πρέπει να είναι ίση με τον πίνακα τις αρχικής κατάστασης των κουτιών. Έτσι επιτυγχάνουμε τα κουτιά μας να βρίσκονται στην αρχική θέση.

```
for(i=0;i<box;i++){
        c_dom_domcst(vars2[i], EQUAL,NULL, boxes[i]);
}
```

Εδώ επιτυγχάνουμε την εκτέλεση της λειτουργίας του φορτώματος. Αυτό γίνεται με το να ελέγξουμε για κάθε φορτίο αν υπάρχει φορτηγό στην ίδια προσπάθεια ( γραμμή του πίνακα ) που να μπορεί να το φορτώσει. Αυτό ακριβώς κάνουν γενικά οι πιο κάτω γραμμές.

Ξεκινούμε δηλαδή από ένα loop το οποίο αντιστοιχεί σε κάθε κουτί. Μετά έχουμε ακόμα ένα loop για να ελέγξουμε για κάθε κουτί αν υπάρχει ένα φορτηγό που μπορεί να το φορτώσει (βρίσκεται στην ίδια πόλη). Αυτό επιτυγχάνεται με την συνάρτηση c_if όπου αν για κάποιο κουτί υπάρχει φορτηγό στην ίδια πόλη τότε καλή στην συνάρτηση load διαφορετικά θα εκτελεστή η συνάρτηση fin.

```
for(i=0; i<(box*tries)-box; i++){
        if(i%(box)==0 && i!=0){
                k++;
        }
        for(j=0;j<trucks;j++,pos++){
                param->curr_box[pos]=i;
                param->curr_truck[pos]=j+trucks*k;
                c_if(vars2[i], EQUAL,vars[j+trucks*k] ,0, load, pos , fin,pos );
        }
}
```

Ας δούμε τώρα πως λειτουργεί η συνάρτηση load. Βασικά σε αυτή την συνάρτηση υπάρχουν οι πιο κάτω 2 περιορισμοί. Ο πρώτος προσθέτει τον περιορισμό ότι η τιμή που θα πάρει το κουτί στην επόμενη προσπάθεια να είναι ίση με την τιμή του φορτηγού + πόλεις. Έτσι αν ένα κουτί έχει τιμή μεγαλύτερη από τον αριθμό των πόλεων που υπάρχουν τότε σημαίνει ότι είναι φορτωμένο σε κάποιο φορτηγό. Έτσι κάνοντας την αφαίρεση μπορούμε να δούμε σε πιο φορτηγό είναι φορτωμένο.

```
c_dom_domcst(vars2[param->curr_box[pos]+box], EQUAL,NULL, (param->curr_truck[pos]%(trucks)+1+cities));
```

Ο δεύτερος περιορισμός μας λέει ότι στην επόμενη προσπάθεια του το φορτηγό δεν θα αλλάξει θέση έτσι ώστε να μπορεί να επιτευχτεί το φόρτωμα.

```
c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars[param->curr_truck[pos]],0);
```

Αν δεν εκτελεστή η συνάρτηση load τότε θα εκτελεστή η συνάρτηση fin. Με αυτή την συνάρτηση το μόνο που γίνετε είναι μια βελτιστοποίηση του χρόνου εκτέλεσης του προγράμματος. Δηλαδή αν τα φορτηγά δεν φορτώσουν κάποιο κουτί στην επόμενη τους προσπάθεια πάνε σε κάποια πόλη που σίγουρα θα έχει κάποιο κουτί. Έτσι αντί να οπισθοδρομεί άσκοπα το πρόγραμμα μας τα φορτηγά που δεν έχουν φορτωμένο κάποιο κουτί πηγαίνουν κάπου που θα είναι χρήσιμα.

c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars2[param->curr_box[pos]], 0 );

Εδώ επιτυγχάνουμε την εκτέλεση της λειτουργίας του ξεφορτώματος. Αυτό γίνεται με το να ελέγξουμε αν κάποιο κουτί είναι φορτωμένο σε ένα φορτηγό τότε αυτό το κουτί μπορεί να ξεφορτωθεί στην πόλη που έχει πάει το φορτηγό. Αυτό ακριβώς κάνουν γενικά οι πιο κάτω γραμμές.
Ξεκινούμε δηλαδή από ένα loop το οποίο αντιστοιχεί σε κάθε κουτί. Μετά έχουμε ακόμα ένα loop για να ελέγξουμε για κάθε κουτί αν υπάρχει ένα φορτηγό που μπορεί να το ξεφορτώσει. Αυτό επιτυγχάνεται με την συνάρτηση c_if όπου αν για κάποιο κουτί υπάρχει φορτηγό στην ίδια πόλη τότε καλή στην συνάρτηση unload διαφορετικά θα εκτελεστή η συνάρτηση fin2.

```
for(i=0; i<(box*tries)-box; i++){
        if(i%(box)==0 && i!=0){
                        k2++;
                }
        for(j=0;j<trucks;j++,pos2++){

                param->curr_box[pos2]=i;
                param->curr_truck[pos2]=j+trucks*k2;
                c_if(vars2[i], EQUAL,vars[j+trucks*k2] ,cities, unload, pos2 ,
        fin2,pos2 );
        }
}
```

Ας δούμε τώρα πως λειτουργεί η συνάρτηση unload. Βασικά σε αυτή την συνάρτηση υπάρχει ο πιο κάτω περιορισμός. Με αυτό τον περιορισμό επιτυγχάνετε το ξεφόρτωμα του κουτιού (στην επόμενη προσπάθεια) στην πόλη όπου βρίσκεται το φορτηγό.

c_dom_domcst(vars2[param->curr_box[pos2]+box], EQUAL,vars[param->curr_truck[pos2]+trucks], 0);

Με τον πιο κάτω κώδικα εξασφαλίζουμε ότι αν κάποιο κουτί δεν είναι φορτωμένο σε κάποιο φορτηγό τότε να μείνει στην θέση που ήταν στην επόμενη προσπάθεια (αφού ένα κουτί μπορεί να μετακινηθεί μόνο από ένα φορτηγό).

```
for(i=0; i<(box*tries)-box; i++){
                pos3=i;
                c_if_in(vars2[i],if_in,trucks, stay, pos3 , fin3,pos3);
}
```

Στην συνάρτηση stay υπάρχει ο πιο κάτω περιορισμός ο οποίος μας δηλώνει ότι το κιβώτιο στην επόμενη προσπάθεια θα μείνει στην θέση που ήταν.
c_dom_domcst(vars2[pos3], EQUAL,vars2[pos3+box], 0 );

Συνεχίζοντας στην συνάρτηση cons υπάρχει ο περιορισμός ότι τα κουτιά στην τελευταία θέση πρέπει να βρίσκονται στις πόλεις τις οποίες έχει εισάγει ο χρήστης σαν είσοδο. Αν φτάσουμε σε αυτό το σημείο και ο πιο κάτω περιορισμός κάνει FAILL τότε ξέρουμε ότι δεν υπάρχει λύση.

```
for(i=0;i<box;i++){
c_dom_domcst(vars2[box*(tries-1)+i], EQUAL,NULL, boxes_f[i]);
}
```

Τέλος υπάρχει ο πιο κάτω περιορισμός που είναι απλά μια βελτίωση του χρόνου εκτέλεσης του προγράμματος όπου τα φορτηγά προσπαθούν να περάσουν στην τελευταία τους κατάσταση από πόλεις στις οποίες θα βρίσκονται τα κουτιά στην τελική τους κατάσταση.

```
for(i=0;i<box;i++){
        c_dom_domcst(vars[trucks*tries-box+i], EQUAL,vars2[box*tries-
        box+i],0);
}
```

Μετά από τα πιο πάνω επιστρέφουμε στην main και προσπαθούμε να βρούμε αν υπάρχει πιθανή λύση στο πρόβλημά μας χρησιμοποιώντας την συνάρτηση c_labeling.

```
options.type     = LABELING_INDOMAIN;
options.indomain = LABELING_INDOMAIN_VALUE;
options.solution = LABELING_FIRST_SOL;
options.continue_var = 0;
options.continue_all = 0;



if (!c_labeling(vars, trucks*tries, METHOD_INPUT_ORDER,
&options))
{
        printf("\nDen iparxei lisi\n");
}

if (!c_labeling(vars2, box*tries, METHOD_INPUT_ORDER,
&options))
{
        printf("\nDen iparxei lisi\n");
}



c_domain_array_print(stdout, vars,  trucks*tries);
printf("\n");
c_domain_array_print(stdout, vars2, box*tries);
printf("\n");
```

Τέλος τυπώνουμε τα αποτελέσματά μας χρησιμοποιώντας την συνάρτηση print_result.

```
printf("\nPrint boxes results\n");
print_result(vars2,box*tries);
```

```
printf("\nPrint trucks results\n");
print_trucks(vars);
```

Ας δούμε τώρα πως δουλεύει η συνάρτηση print_result. Σε γενικές γραμμές αυτή η συνάρτηση τυπώνει το μονοπάτι που έχουν ακολουθήσει τα κουτιά δηλ. από ποιες πόλεις έχουν περάσει και από ποια φορτηγά έχουν φορτωθεί, και τέλος ελέγχει αν υπάρχει ή όχι λύση σε αυτή την προσπάθεια.

```
int print_result(DvarPtr *vars2, int size){
int *preval;
```

Στον πιο κάτω κώδικα δεσμεύουμε χώρο για τον πίνακα preval ο οποίος κρατά την θέση των κουτιών στην προηγούμενη προσπάθεια. Αυτό το κάνουμε για να ξέρουμε αν ένα κουτί έχει μεταφερθεί από φορτηγό σε πόλη ή και αντίστροφα για να μπορούμε να τυπώσουμε αν έχει φορτωθεί ή ξεφορτωθεί.

```
preval= (int *)malloc(sizeof(int)*box);
```

Σε αυτή την συνάρτηση υπάρχει ένα for που παίρνει για κάθε κουτί την πόλη στην οποία βρίσκεται με την βοήθεια της συνάρτησης c_domain_value.

```
for(i=0; i<box*tries; i++){
        if(i%(box)==0 && i!=0){
                printf("\n Allagi prospatheias\n");
        }
        c_domain_value(vars2[i],&val);
```

Έχοντας ήδη πάρει την τιμή του κιβωτίου στη μεταβλητή val μπορούμε με ένα if να δούμε αν το κουτί μας βρίσκεται σε κάποια πόλη και αν ναι να δούμε αν στην προηγούμενη του προσπάθεια ήταν φορτωμένο από κάποιο φορτηγό οπόταν σημαίνει ότι στην πόλη στην οποία βρίσκεται έχει ξεφορτωθεί από τον φορτηγό που βρίσκεται στον πίνακα preval.

```
        if(val<cities+1){
                if(preval[(i)%box]<cities){
                        printf("to kouti %d brisketai stin poli
                %d\n",((i)%box)+1,val);
                }
                else{
```

```
                    printf("to kouti %d ksefortwnetai stin poli
            %d\n",((i)%box)+1,val);
                    }
        }
```

Αλλιώς αν το φορτηγό δεν βρίσκεται σε κάποια πόλη πάει να πει ότι βρίσκεται (φορτώνεται) σε κάποιο φορτηγό.

```
        else{
                printf("to kouti %d fortwnetai apo to fortigo
        %d\n",((i)%box)+1,(val-cities));
        }
        preval[i%box]=val;

}
```

Πιο κάτω γίνεται ο έλεγχος αν υπάρχει λύση σε αυτή την προσπάθεια. Αυτό γίνεται με το να ελέγξουμε την τελευταία γραμμή του πίνακα των κουτιών. Αν αυτή συμφωνεί με την θέση που πήραμε σαν είσοδο από τον χρήστη για την τελική κατάσταση των κουτιών τότε το πρόγραμμα μας έχει φτάσει σε λύση και σταματά την εκτέλεση του. Αν όχι τότε θα συνεχίσει αυξάνοντας τις προσπάθειές του (αν μπορεί).

```
for(i=0;i<box;i++){
        c_domain_value(vars2[box*(tries-1)+i],&val);
        final_state[i]=val;

}
if(equals(final_state,boxes_f,box)==1){
        printf("SUCCEEEEEEEED\n");
        exit(-1);
}
else{
        printf("No solution for this try\n");

}
```

Τέλος επιστρέφουμε πίσω στην main() όπου και κάνουμε reset την chipc έτσι ώστε να μπορέσουν να ανατεθούν νέοι περιορισμοί την επόμενη φορά που θα τρέξει το πρόγραμμά μας με διαφορετικό αριθμό προσπαθειών.

```
            if (c_chip_reset() != 1)
            {
                    printf("\nInitialize FAILEDDD\n");
            }
    }   //end for gia prospatheies
```

**3.5 Παράδειγματα  εκτέλεσης:**

**3.5.1 Παράδειγμα πρώτο:**

Kalosorisate sto programa metaforas fortiwn

dwse ton arithmo twn polewn pou tha iparxoun:4

dwse ton arithmo twn fortigwn pou iparxoun:3

dwse ton arithmo twn fortiwn pou iparxoun:3

dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:2

dwse tin poli pou brisketai to kathe fortigo

truck 1:1

truck 2:2

truck 3:4

to fotigo 1 brisketai stin poli 1

to fotigo 2 brisketai stin poli 2

to fotigo 3 brisketai stin poli 4

dwse tin poli pou brisketai to kathe fortio

fortio 1:1

fortio 2:2

fortio 3:4

to fortio 1 brisketai stin poli 1

to fortio 2 brisketai stin poli 2

to fortio 3 brisketai stin poli 4

dwse to simeio pou tha brisketai to kathe fortio stin teliki katastasi

fortio 1:3

fortio 2:3

fortio 3:3

to fortio 1 tha brisketai stin poli 3

to fortio 2 tha brisketai stin poli 3

to fortio 3 tha brisketai stin poli 3

Setting Cons

[D_1=1, D_2=2, D_3=4, D_4=1, D_5=2, D_6=4]

[D_7=1, D_8=2, D_9=4, D_10=5, D_11=6, D_12=7]

Print boxes results

to kouti 1 brisketai stin poli 1

to kouti 2 brisketai stin poli 2

to kouti 3 brisketai stin poli 4

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 1

to kouti 2 fortwnetai apo to fortigo 2

to kouti 3 fortwnetai apo to fortigo 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 4

Allagi prospatheias

to fortigo 1 menei stin poli 1

to fortigo 2 menei stin poli 2

to fortigo 3 menei stin poli 4

Setting Cons

[D_37=1, D_38=2, D_39=4, D_40=1, D_41=2, D_42=4, D_43=3, D_44=3, D_45=3]

[D_46=1, D_47=2, D_48=4, D_49=5, D_50=6, D_51=7, D_52=3, D_53=3, D_54=3]

Print boxes results

to kouti 1 brisketai stin poli 1

to kouti 2 brisketai stin poli 2

to kouti 3 brisketai stin poli 4

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 1

to kouti 2 fortwnetai apo to fortigo 2

to kouti 3 fortwnetai apo to fortigo 3

Allagi prospatheias

to kouti 1 ksefortwnetai stin poli 3

to kouti 2 ksefortwnetai stin poli 3

to kouti 3 ksefortwnetai stin poli 3

SUCCEEEEEEEED

Στο πιο πάνω παράδειγμα θέλουμε 2 προσπάθειες για να φτάσουμε σε λύση αφού τα κουτιά μας βρίσκονται αρχικά στην ίδια θέση που βρίσκονται και τα φορτηγά μας αντίστοιχα. Έτσι θα χρειαστεί μια προσπάθεια για να φορτωθούν τα κουτιά από τα

φορτηγά και ακόμη μία για να μεταφερθεί κάθε κουτί στον τελικό του προορισμό. Όπως φένεται και πιο πάνω το πρόγραμμα δοκιμάζει να βρει λύση με μία προσπάθεια όπου και αποτυγχάνει αν και το φόρτωμα το κουτιών γίνετε. Όταν οι προσπάθειες αυξηθούν σε δύο τότε στην πρώτη γίνετε το φόρτωμα και στην δεύτερη το ξεφόρτωμα με αποτέλεσμα ο κώδικας μας να βρει λύση.

### 3.5.2 Παράδειγμα δεύτερο:

Kalosorisate sto programa metaforas fortiwn

dwse ton arithmo twn polewn pou tha iparxoun:4

dwse ton arithmo twn fortigwn pou iparxoun:3

dwse ton arithmo twn fortiwn pou iparxoun:3

dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:2

dwse tin poli pou brisketai to kathe fortigo

truck 1:1

truck 2:2

truck 3:1

to fotigo 1 brisketai stin poli 1

to fotigo 2 brisketai stin poli 2

to fotigo 3 brisketai stin poli 1

dwse tin poli pou brisketai to kathe fortio

fortio 1:3

fortio 2:3

fortio 3:3

to fortio 1 brisketai stin poli 3

to fortio 2 brisketai stin poli 3

to fortio 3 brisketai stin poli 3

dwse to simeio pou tha brisketai to kathe fortio stin teliki katastasi

fortio 1:2

fortio 2:2

fortio 3:2

to fortio 1 tha brisketai stin poli 2

to fortio 2 tha brisketai stin poli 2

to fortio 3 tha brisketai stin poli 2

Setting Cons

[D_1=1, D_2=2, D_3=1, D_4=3, D_5=3, D_6=3]

[D_7=3, D_8=3, D_9=3, D_10=3, D_11=3, D_12=3]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Setting Cons

[D_37=1, D_38=2, D_39=1, D_40=3, D_41=3, D_42=3, D_43=3, D_44=3, D_45=3]

[D_46=3, D_47=3, D_48=3, D_49=3, D_50=3, D_51=3, D_52=7, D_53=7, D_54=7]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 3

to kouti 2 fortwnetai apo to fortigo 3

to kouti 3 fortwnetai apo to fortigo 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Allagi prospatheias

to fortigo 1 menei stin poli 3

to fortigo 2 menei stin poli 3

to fortigo 3 menei stin poli 3

Total time : 3 ms

Στο πιο πάνω παράδειγμα δεν μπορούμε να φτάσουμε σε λύση με αριθμό προσπαθειών ίσο με 2. αφού κανένα φορτηγό δεν μπορεί να φορτώσει τα κουτιά μας που βρίσκονται όλα στην πόλη 3. έτσι θα χρειαστούμε μία προσπάθεια για να μεταφερθεί κάποιο φορτηγό σε αυτήν την πόλη , ακόμη μια προσπάθεια για να φορτώσει το φορτηγό μας και τα κουτιά και μια τελευταία προσπάθεια για να τα μεταφέρει στην πόλη 2. έτσι θέλουμε συνολικά 3 προσπάθειες και όχι 2 που είναι ο μέγιστος αριθμός προσπαθειών που δικαιούμαστε.

### 3.5.3 Παράδειγμα τρίτο:

Kalosorisate sto programa metaforas fortiwn

dwse ton arithmo twn polewn pou tha iparxoun:4

dwse ton arithmo twn fortigwn pou iparxoun:3

dwse ton arithmo twn fortiwn pou iparxoun:3

dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:3

dwse tin poli pou brisketai to kathe fortigo

truck 1:1

truck 2:2

truck 3:1

to fotigo 1 brisketai stin poli 1

to fotigo 2 brisketai stin poli 2

to fotigo 3 brisketai stin poli 1

dwse tin poli pou brisketai to kathe fortio

fortio 1:3

fortio 2:3

fortio 3:3

to fortio 1 brisketai stin poli 3

to fortio 2 brisketai stin poli 3

to fortio 3 brisketai stin poli 3

dwse to simeio pou tha brisketai to kathe fortio stin teliki katastasi

fortio 1:2

fortio 2:2

fortio 3:2

to fortio 1 tha brisketai stin poli 2

to fortio 2 tha brisketai stin poli 2

to fortio 3 tha brisketai stin poli 2

Setting Cons

[D_1=1, D_2=2, D_3=1, D_4=3, D_5=3, D_6=3]

[D_7=3, D_8=3, D_9=3, D_10=3, D_11=3, D_12=3]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Setting Cons

[D_37=1, D_38=2, D_39=1, D_40=3, D_41=3, D_42=3, D_43=3, D_44=3, D_45=3]

[D_46=3, D_47=3, D_48=3, D_49=3, D_50=3, D_51=3, D_52=7, D_53=7, D_54=7]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 3

to kouti 2 fortwnetai apo to fortigo 3

to kouti 3 fortwnetai apo to fortigo 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Allagi prospatheias

to fortigo 1 menei stin poli 3

to fortigo 2 menei stin poli 3

to fortigo 3 menei stin poli 3

Setting Cons

[D_103=1, D_104=2, D_105=1, D_106=3, D_107=3, D_108=3, D_109=3, D_110=3, D_111=3, D_112=2, D_113=2, D_114=2]

[D_115=3, D_116=3, D_117=3, D_118=3, D_119=3, D_120=3, D_121=7, D_122=7, D_123=7, D_124=2, D_125=2, D_126=2]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 3

to kouti 2 fortwnetai apo to fortigo 3

to kouti 3 fortwnetai apo to fortigo 3

Allagi prospatheias

to kouti 1 ksefortwnetai stin poli 2

to kouti 2 ksefortwnetai stin poli 2

to kouti 3 ksefortwnetai stin poli 2

SUCCEEEEEEEED

Στο πιο πάνω παράδειγμα τα φορτηγά και τα κουτιά μας είναι τοποθετημένα ακριβώς με τον ίδιο τρόπο όπως και στο προηγούμενο μας παράδειγμα με την μόνη διαφορά ότι έχουμε τρεις προσπάθειες για να βρούμε λύση. Έτσι τα φορτηγάμας πηγαίνουν στην πόλη που βρίσκονται τα κουτιά στην πρώτη προσπάθεια. Στην δεύτερη προσπάθεια το τρίτο φορτηγό φορτώνει τα κουτιά και στην Τρίτη προσπάθεια τα παίρνει στην πόλη 2 όπου είναι και η τελική τους θέση.

### 3.5.4 Παράδειγμα τέταρτο:

Kalosorisate sto programa metaforas fortiwn

dwse ton arithmo twn polewn pou tha iparxoun:4

dwse ton arithmo twn fortigwn pou iparxoun:3

dwse ton arithmo twn fortiwn pou iparxoun:3

dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:19

dwse tin poli pou brisketai to kathe fortigo

truck 1:1

truck 2:2

truck 3:1

to fotigo 1 brisketai stin poli 1

to fotigo 2 brisketai stin poli 2

to fotigo 3 brisketai stin poli 1

dwse tin poli pou brisketai to kathe fortio

fortio 1:3

fortio 2:3

fortio 3:3

to fortio 1 brisketai stin poli 3

to fortio 2 brisketai stin poli 3

to fortio 3 brisketai stin poli 3

dwse to simeio pou tha brisketai to kathe fortio stin teliki katastasi

fortio 1:2

fortio 2:2

fortio 3:2

to fortio 1 tha brisketai stin poli 2

to fortio 2 tha brisketai stin poli 2

to fortio 3 tha brisketai stin poli 2

Setting Cons

[D_1=1, D_2=2, D_3=1, D_4=3, D_5=3, D_6=3]

[D_7=3, D_8=3, D_9=3, D_10=3, D_11=3, D_12=3]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Setting Cons

[D_37=1, D_38=2, D_39=1, D_40=3, D_41=3, D_42=3, D_43=3, D_44=3, D_45=3]

[D_46=3, D_47=3, D_48=3, D_49=3, D_50=3, D_51=3, D_52=7, D_53=7, D_54=7]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 3

to kouti 2 fortwnetai apo to fortigo 3

to kouti 3 fortwnetai apo to fortigo 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Allagi prospatheias

to fortigo 1 menei stin poli 3

to fortigo 2 menei stin poli 3

to fortigo 3 menei stin poli 3

Setting Cons

[D_103=1, D_104=2, D_105=1, D_106=3, D_107=3, D_108=3, D_109=3, D_110=3, D_111=3, D_112=2, D_113=2, D_114=2]

[D_115=3, D_116=3, D_117=3, D_118=3, D_119=3, D_120=3, D_121=7, D_122=7, D_123=7, D_124=2, D_125=2, D_126=2]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 3

to kouti 2 fortwnetai apo to fortigo 3

to kouti 3 fortwnetai apo to fortigo 3

Allagi prospatheias

to kouti 1 ksefortwnetai stin poli 2

to kouti 2 ksefortwnetai stin poli 2

to kouti 3 ksefortwnetai stin poli 2

SUCCEEEEEEEED

Στο πιο πάνω παράδειγμα θέλουμε απλός να δείξουμε ότι αν ο αριθμός των προσπαθειών που δικαιούμαστε είναι πιο μεγάλος από τον αριθμό στον οποίο μπορούμε να βρούμε λύση το πρόγραμμά μας σταματά με το που θα βρει λύση. Έτσι χρησιμοποιήσαμε το προηγούμενο παράδειγμα με μέγιστο αριθμό προσπάθεια 19 και το πρόγραμμα μας έχει σταματήσει στην προσπάθεια 3 όπου και έχει βρει λύση.

**3.5.5 Παράδειγμα πέμπτο:**

Kalosorisate sto programa metaforas fortiwn

dwse ton arithmo twn polewn pou tha iparxoun:4

dwse ton arithmo twn fortigwn pou iparxoun:3

dwse ton arithmo twn fortiwn pou iparxoun:3

dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:3

dwse tin poli pou brisketai to kathe fortigo

truck 1:1

truck 2:2

truck 3:1

to fotigo 1 brisketai stin poli 1

to fotigo 2 brisketai stin poli 2

to fotigo 3 brisketai stin poli 1

dwse tin poli pou brisketai to kathe fortio

fortio 1:3

fortio 2:3

fortio 3:3

to fortio 1 brisketai stin poli 3

to fortio 2 brisketai stin poli 3

to fortio 3 brisketai stin poli 3

dwse to simeio pou tha brisketai to kathe fortio stin teliki katastasi

fortio 1:1

fortio 2:2

fortio 3:4

to fortio 1 tha brisketai stin poli 1

to fortio 2 tha brisketai stin poli 2

to fortio 3 tha brisketai stin poli 4

Setting Cons

[D_1=1, D_2=2, D_3=1, D_4=3, D_5=3, D_6=3]

[D_7=3, D_8=3, D_9=3, D_10=3, D_11=3, D_12=3]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Setting Cons

[D_37=1, D_38=2, D_39=1, D_40=3, D_41=3, D_42=3, D_43=3, D_44=3, D_45=3]

[D_46=3, D_47=3, D_48=3, D_49=3, D_50=3, D_51=3, D_52=5, D_53=6, D_54=7]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 1

to kouti 2 fortwnetai apo to fortigo 2

to kouti 3 fortwnetai apo to fortigo 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias

to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Allagi prospatheias

to fortigo 1 menei stin poli 3

to fortigo 2 menei stin poli 3

to fortigo 3 menei stin poli 3

Setting Cons

[D_103=1, D_104=2, D_105=1, D_106=3, D_107=3, D_108=3, D_109=3, D_110=3, D_111=3, D_112=1, D_113=2, D_114=4]

[D_115=3, D_116=3, D_117=3, D_118=3, D_119=3, D_120=3, D_121=5, D_122=6, D_123=7, D_124=1, D_125=2, D_126=4]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 fortwnetai apo to fortigo 1

to kouti 2 fortwnetai apo to fortigo 2

to kouti 3 fortwnetai apo to fortigo 3

Allagi prospatheias

to kouti 1 ksefortwnetai stin poli 1

to kouti 2 ksefortwnetai stin poli 2

to kouti 3 ksefortwnetai stin poli 4

SUCCEEEEEEEED

Total time : 5 ms

Στο πιο πάνω παράδειγμα τα κουτιά μας βρίσκονται σε κάποια πόλη στην οποία δεν υπάρχει φορτηγό (όπως και στο προηγούμενο παράδειγμα) με την μόνη διαφορά ότι στην τελική κατάσταση κάθε κουτί πρέπει να πάει σε διαφορετική πόλη. Έτσι πρέπει

να πάνε και τα τρία φορτηγά στην πόλη που βρίσκονται τα κουτιά στην πρώτη προσπάθεια και να φορτώσει κάθε φορτηγό και από ένα κουτί στην δεύτερη προσπάθεια. Έτσι στην Τρίτη προσπάθεια μπορούμε να βρούμε λύση αφού τα φορτηγά θα μεταφέρουν τα κουτιά μας στην τελική τους κατάσταση.

# Κεφάλαιο 4 Πρόβλημα μεταφοράς αγαθών με αριθμό κουτιών στην τελική θέση:

### 4.1 Το πρόβλημα:

Το πρόβλημα αυτό είναι το ίδιο με το προηγούμενο πρόβλημα το οποίο μελετήσαμε με την μόνη διαφορά ότι δεν μας ενδιαφέρει το που βρίσκεται ένα συγκεκριμένο φορτίο φτάνει μόνο οι πόλεις να έχουν από ένα συγκεκριμένο αριθμό φορτίων στην τελική τους κατάσταση.

### 4.2 Ο αλγόριθμος:

Και ο αλγόριθμος είναι επίσης ο ίδιος με τον προηγούμενο μας αλγόριθμο με την διαφορά ότι οι πίνακες που δημιουργούμε για την αρχική και τελική θέση των κιβωτίων είναι μεγέθους όσο και οι πόλεις μας με τιμές που μας δίνουν τον αριθμό φορτίων που έχει η κάθε πόλη.

Έτσι σύμφωνα με το παράδειγμα μας αυτή οι πίνακες θα έχουν την πιο κάτω μορφή:

| | | | |
|---|---|---|---|
| 1 | 1 | 0 | 1 |

| | | | |
|---|---|---|---|
| 0 | 0 | 3 | 0 |

Αυτό που μας δείχνει είναι ότι η πρώτη πόλη έχει στην αρχική της κατάσταση ένα κιβώτιο ενώ στην τελική κατάσταση δεν έχει κανένα κιβώτιο. Με τον ίδιο τρόπο μπορούμε να πάρουμε πληροφορίες για όλες τις πόλεις.

### 4.3Μοντελοποίηση

Ας υποθέσουμε ότι ο αριθμός των πόλεων είναι ίσος με τέσσερα, ο αριθμός των φορτίων είναι ίσος με τρία και οι μέγιστες προσπάθειες που διακαιόμαστε είναι δύο.

Αρχικά το πρόγραμμά μας θα δοκιμάσει να βρει λύση σε μία προσπάθεια. Έτσι οι πίνακες που θα δημιουργηθούν θα είναι:

Ο πίνακας που αναπαριστά τα φορτία θα έχει δυο γραμμές που αντιστοιχούν σε μια προσπάθεια και τρείς στήλες που αντιστοιχούν στον αριθμό των φορτίων. Τα πεδία τιμών κάθε θέσης του πίνακα είναι από ένα μέχρι και εφτά που είναι ο αριθμός των πόλεων συν τα φορτηγά.

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 3 | 3 | 3 |

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 3 | 3 | 3 |

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 3 | 3 | 3 |

Ο πίνακας που αναπαριστά τα φορτηγά θα έχει δυο γραμμές που αντιστοιχούν σε μια προσπάθεια και τρείς στήλες που αντιστοιχούν στον αριθμό των φορτηγών. Τα πεδία

τιμών κάθε θέσης του πίνακα είναι από ένα μέχρι και τέσσερα που είναι ο αριθμός των πόλεων.

Το πρόγραμμα θα προσπαθήσει να βρει λύση για μία προσπάθεια όπου και θα αποτύχει αφού δεν θα μπορέσει να μεταφέρει τα φορτία στην τελική τους θέση

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 2 | 4 |

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 2 | 4 |

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 2 | 4 |

Στην επόμενη προσπάθεια έχοντας δυο προσπάθειες στην διάθεση μας ο πίνακας των φορτίων και των φορτηγών θα έχουν τρείς γραμμές. Έτσι το πρόγραμμά μας θα μπορέσει να βρει λύση.

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 2 | 4 |
| 3 | 3 | 3 |

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 5 | 6 | 7 |
| 3 | 3 | 3 |

### 4.4 Περιγραφή επίλυσης στην chip:

Ακόμα και ο κώδικας είναι πολύ παρόμοιος. Οι διαφορές του κώδικα βρίσκονται στις πιο κάτω γραμμές:

Ο κώδικας που αφορά την δέσμευση χώρου έχει αλλάξει αφού τώρα δεσμεύουμε χώρο ίσο με τον αριθμό των πόλεων.

```
Cities_box= (int *)malloc(sizeof(int)*cities);
Cities_box_f= (int *)malloc(sizeof(int)*cities);
```

Η κύρια διαφορά του αλγόριθμου μας είναι ότι στον πίνακα των domain variables οι περιορισμοί μας για αρχικοποίηση είναι να υπάρχει τουλάχιστο χ φορές, στην συγκεκριμένη μας προσπάθεια, η θέση του πίνακα (πόλη) που περιέχει τον αριθμό των κουτιών που υπάρχουν στης πόλεις. Όπου χ είναι η τιμή αυτού του πίνακα. Με την προϋπόθεση ότι αυτές οι τιμές έχουν δοθεί σωστά από τον χρήστη αυτός ο περιορισμός δεν θα κάνει FAIL. Το πώς το εξασφαλίζουμε αυτό είναι να θέσουμε ένα περιορισμό c_atleast στον πίνακα boxes όπου σε αυτό το πρόγραμμα είναι domain variables και όχι ένας πίνακας από integers. Τέλος ο πίνακας vars2 που φυλάγονται οι διαδρομές των κουτιών έχει περιορισμό να είναι ίσος η πρώτη του προσπάθεια με τον πίνακα boxes για τον οποίο αν δεν βρούμε λύση με τον συγκεκριμένο συνδυασμό τιμών το πρόγραμμά μας θα οπισθοδρομήσει δοκιμάζοντας άλλον συνδυασμό. Αυτό μας εξασφαλίζει την ύπαρξη του αριθμού των κουτιών, σε κάθε προσπάθεια, στις αντίστοιχες πόλεις.

```
for(i=0; i<cities; i++){
        c_atleast(cities_box_f[i],boxes_f,box,i+1);
}

for(i=0;i<box;i++){
        if(c_domain_value(boxes[i],&val)==FAIL)
                printf("get value faillll\n");
```

```
        c_dom_domcst(vars2[i], EQUAL,NULL, val);
}
```

Στον πιο πάνω κώδικα αντί να προσθέτουμε το περιορισμό c_dom_domcst() τον αντικαταστήσαμε με τον περιορισμό c_atleast() αφού δεν μας ενδιαφέρει ακριβώς για το σε ποια πόλη είναι το κουτί μας σε κάθε προσπάθεια φτάνει μόνο να υπάρχει σε αυτήν την πόλη. Έτσι το τι ακριβώς κάνουμε είναι να μετασχηματίσουμε τον πίνακα που μας δείχνει πόσα κουτιά έχει κάθε πόλη στο που μπορεί να βρίσκονται αυτά τα κουτιά στον πίνακα με τις domain variables. Αν στην πρώτη ανάθεση των τιμών δεν βρεθεί λύση τότε ο αλγόριθμος μας θα οπισθοδρομήσει δοκιμάζοντας ένα άλλον συνδυασμό ανάθεσης κουτιών στις πόλεις.

Το ίδιο συμβαίνει και με τις τελικές τιμές του πίνακα domain variables όπου σύμφωνα με τον πιο κάτω περιορισμό εξασφαλίζουμε την ύπαρξη του αριθμού των κουτιών, σε κάθε προσπάθεια,  στις αντίστοιχες πόλεις.

```
for(i=0; i<cities; i++){

        c_atleast(cities_box[i],boxes,box,i+1);
}
for(i=0;i<box;i++){
        c_domain_value(boxes_f[i],&val);
        c_dom_domcst(vars2[box*(tries-1)+i], EQUAL,NULL, val);
}
```

Ας δούμε τώρα με λεπτομέρεια την εκτέλεση του κώδικά μας.

Το πρόγραμμά μας ξεκινά την εκτέλεσή του από την main.
Αμέσως παίρνουμε από τον χρήστη τις απαραίτητες τιμές που αφορούν το πρόγραμμα όπως αριθμό πόλεων, φορτηγών και προσπαθειών που πρέπει να γίνουν για να βρεθεί λύση.

```
printf("Kalosorisate sto programa metaforas fortiwn\n");
printf("dwse ton arithmo twn polewn pou tha iparxoun:");
scanf("%d",&cities);
printf("dwse ton arithmo twn fortigwn pou iparxoun:");
scanf("%d",&trucks);
printf("dwse ton arithmo twn fortiwn pou iparxoun:");
scanf("%d",&box);
printf("dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na
brethei lisi:");
scanf("%d",&maxtries);
```

Μετά αχικοποιούμε των αρχικό αριθμό των προσπαθειών σε 2 αφού για να γίνει μια προσπάθεια τότε πρέπει να υπάρχουν 2 θεσεις στον πίνακα. Επίσης αρχικοποιούμε τον μέγιστο αριθμό προσπαθειών με τον αριθμό που πήραμε από τον χρήστη +2.

```
tries=2;
maxtries=maxtries+2;
```

Μετά δεσμέυουμε χώρο για τους πίνακες if_in και final_state. Ο πίνακας if_in είναι ένας πίνακας μεγέθους ίσου με τα φορτιγά όπου θα χρεισιμοποιηθεί αργότερα στην συνάρτηση c_if_in(). Ο πίνακας final_state είναι ένας πίνακας μεγέθους ίσου με τον αριθμό των κουτιών όπου θα μας χρεισημέυση για να δούμε αν υπάρχει λύση.

```
if_in=(int *)malloc(sizeof(int)*trucks);
final_state=(int *)malloc(sizeof(int)*box);
```

Εδώ δεσμεύουμε χώρο για πέντε πίνακας. Ο πρώτος αφορά τις αρχικές θέσεις των φορτηγών αφού θα τις χρειαστούμε μετέπειτα στους περιορισμούς και οι άλλοι τέσσερις αφορούν τις αρχικές και τελικές θέσεις των φορτίων αφού θα τους χρειαστούμε για τον ίδιο ακριβώς λόγω.

```
on= (int *)malloc(sizeof(int)*trucks);
boxes= (void **)malloc(sizeof(int)*box);
```

```
boxes_f= (void **)malloc(sizeof(int)*box);
cities_box= (int *)malloc(sizeof(int)*cities);
cities_box_f= (int *)malloc(sizeof(int)*cities);
```

Πιο κάτω βάζουμε τιμές στον πίνακα που θα χρησιμοποιήσουμε πιο κάτω στην συνάρτηση c_if_in. Έτσι σε αυτό τον πίνακα μπαίνουν όλες οι πόλεις του προγράμματος μας.

```
for(i=0;i<trucks;i++){
        if_in[i]=i+1;
}
```

Εδώ απλά εισάγουμε τιμές στους πίνακες που μας δείχνουν την αρχικές θέσεις των φορτηγών και τις αρχικές και τελικές θέσεις των κιβωτίων. Αυτοί οι πίνακες έχουν δημιουργηθεί προηγούμενος.

```
printf("dwse tin poli pou brisketai to kathe fortigo\n");
        for(i=0; i<trucks; i++){
                printf("truck %d:",i+1);
                scanf("%d",&on[i]);

                while (on[i]<=0 || on[i]>cities){
                        printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                        scanf("%d",&on[i]);
                }
        }
        //user input gia pinaka cities_box
        printf("dwse ton arithmo twn koutiwn pou tha exei i kathe poli\n");
        for(i=0; i<cities; i++){
                printf("poli %d:",i+1);
                scanf("%d",&cities_box[i]);

                while (cities_box[i]<0 || cities_box[i]>box){
                        printf("Den iparxoun tosa koutia, ksanadwse arithmo:");
```

```
                scanf("%d",&cities_box[i]);
            }
        }

        //user input gia pinaka cities_box_f
        printf("dwse ton arithmo twn fortiwn pou tha exei i kathe poli stin teliki katastasi\n");
        for(i=0; i<cities; i++){
            printf("fortio %d:",i+1);
            scanf("%d",&cities_box_f[i]);

            while (cities_box_f[i]<0 || cities_box_f[i]>box){
                printf("Den iparxoun tosa koutia, ksanadwse arithmo:");
                scanf("%d",&cities_box_f[i]);
            }
        }
```

Μετά στον κώδικα μας υπάρχει το for στο οποίο γίνεται η αλλαγή των προσπαθειών. Αυτό το loop εκτελείτε από προσπάθειες ίσες με 2 και προχωρά μέχρι να βρεθεί λύση ή μέχρι να φτάσει των μέγιστο αριθμό προσπαθειών που πήραμε από τον χρήστη.
for(tries=2; tries<maxtries; tries++, pos=0, pos2=0,k=0, k2=0,pos3=0)

Αμέσως μετά δεσμεύουμε χώρο για τον πίνακα vars ο οποίος είναι ένας πίνακας DvarPtr ο οποίος είναι αυτός που θα δουλέψουμε στην συνέχεια για να μας βρει την διαδρομή που θα ακολουθήσουν τα φορτηγά έτσι ώστε τα κιβώτια μας να μεταφερθούν με επιτυχία στον προορισμό τους. Αυτός ο πίνακας έχει μέγεθος ίσο με τον αριθμό των φορτηγών επί τις προσπάθειες που πρέπει να γίνουν. Αυτό το κάνουμε με την λογική ότι οι τιμές που πρέπει να βρούμε λύση για τιμές ίσες με τον πιο πάνω αριθμό.

```
vars =  (void**)realloc(vars,sizeof(int)*(trucks*tries));
```

Αμέσως μετά δεσμεύουμε χώρο για τον πίνακα vars2 ο οποίος είναι ένας πίνακας DvarPtr ο οποίος είναι αυτός που θα δουλέψουμε στην συνέχεια για να μας βρει την διαδρομή που θα ακολουθήσουν τα κουτιά έτσι ώστε να σιγουρευτούμε μεταφερθούν με επιτυχία στον προορισμό τους. Αυτός ο πίνακας έχει μέγεθος ίσο με τον αριθμό των κουτιών επί τις προσπάθειες που πρέπει να γίνουν. Αυτό το κάνουμε με την λογική ότι οι τιμές που πρέπει να βρούμε λύση για τιμές ίσες με τον πιο πάνω αριθμό.

```
vars2 = (void**)realloc(vars2,sizeof(int)*(box*tries));
```

Μετά δεσμεύουμε χώρο για το struct param. Αυτό το struct μας χρησιμεύει στο να φυλάγουμε τις παραμέτρους που θα χρειαστούν μετά οι συναρτήσεις load, unload και stay.

Σε αυτό το struct υπάρχουν 2 πεδία. Το πρώτο είναι το πεδίο curr_box και το δεύτερο curr_truck. Το πρώτο πεδίο χρησιμοποιείται για να φυλάγουμε το αριθμό του πίνακα των κουτιών που θα προστεθούν οι περιορισμοί μας στις αντίστοιχες συναρτήσεις και το δεύτερο πεδίο χρησιμοποιείται για να φυλάγουμε το αριθμό του πίνακα των φορτηγών που θα προστεθούν οι περιορισμοί μας στις αντίστοιχες συναρτήσεις

```
param =(ARGUM *)realloc(param,sizeof(ARGUM));
param->curr_box= (int *) realloc(param->
curr_box,sizeof(int)*(box*tries*trucks));
param->curr_truck=(int*) realloc(param->
curr_truck,sizeof(int)*(box*tries*trucks));
```

Μετά δίνουμε το πεδίο τιμών που θα έχει κάθε μια από τις μεταβλητές του πίνακα vars δηλ. τις τιμές που μπορεί να έχει κάθε φορτηγό. Αφού κάθε φορτηγό μπορεί να βρίσκεται σε κάποια από της πόλεις μας οι τιμές αυτές θα ξεκινούν από ένα και θα φτάνουν μέχρι και τον αριθμό των πόλεων του κάθε προγράμματος.

```
c_create_domain_array(vars, trucks*tries+1, 1, cities, CONSEC);
```

Μετά δίνουμε το πεδίο τιμών που θα έχει κάθε μια από τις μεταβλητές του πίνακα vars2 δηλ. τις τιμές που μπορεί να έχει κάθε κουτί. Αφού κάθε κουτί μπορεί να βρίσκεται σε κάποια από της πόλεις ή σε κάποιο φορτηγό οι τιμές αυτές θα ξεκινούν από ένα και θα φτάνουν μέχρι και τον αριθμό των πόλεων + τον αριθμό των φορτηγών του προγράμματος.

```
c_create_domain_array(vars2, box*tries, 1, cities+trucks, CONSEC);
```

Μετά δίνουμε το πεδίο τιμών που θα έχει κάθε μια από τις μεταβλητές του πίνακα boxes δηλ. τις τιμές που μπορεί να έχει κάθε κουτί. Αφού κάθε κουτί μπορεί να βρίσκεται σε κάποια από της πόλεις ή σε κάποιο φορτηγό οι τιμές αυτές θα ξεκινούν από ένα και θα φτάνουν μέχρι και τον αριθμό των πόλεων + τον αριθμό των φορτηγών του προγράμματος.

```
c_create_domain_array(boxes_f, box, 1, cities+trucks, CONSEC);
c_create_domain_array(boxes, box, 1, cities+trucks, CONSEC);
```

Αμέσως μετά καλούμε την συνάρτηση όπου θα μας δημιουργήσει τους περιορισμούς στους πίνακες boxes και boxes_f

```
boxes_cons(boxes,boxes_f);
```

εξασφαλίζουμε την ύπαρξη του αριθμού των κουτιών, σε κάθε προσπάθεια, στις αντίστοιχες πόλεις.

```
for(i=0; i<cities; i++){
        c_atleast(cities_box_f[i],boxes_f,box,i+1);
}
for(i=0; i<cities; i++){

        c_atleast(cities_box[i],boxes,box,i+1);
}
}
```

Τέλος βρίσκουμε λύση για την συγκεκριμένη ανάθεση τιμών.

```
options.type     = LABELING_INDOMAIN;
options.indomain = LABELING_INDOMAIN_VALUE;
options.solution = LABELING_FIRST_SOL;
```

```
options.continue_var = 0;
options.continue_all = 0;


if (!c_labeling(boxes, box, METHOD_INPUT_ORDER, &options))
        {
                printf("\nDen iparxei lisi\n");
        }
        if (!c_labeling(boxes_f, box, METHOD_INPUT_ORDER, &options))
        {
                printf("\nDen iparxei lisi\n");
        }
```

Εδώ καλούμε την συνάρτηση cons όπου είναι η συνάρτηση που μας προσθέτει τους περιορισμούς στις μεταβλητές μας.

```
cons(grafos,on,boxes,boxes_f,cities,trucks,tries,box,vars,vars2);
```

Τώρα ας δούμε βήμα βήμα πως θα λειτουργήσει αυτή η συνάρτηση. Αρχικά βάζουμε των περιορισμό ότι η πρώτη γραμμή του πίνακα που vars πρέπει να είναι ίση με τον πίνακα τις αρχικής κατάστασης των φορτηγών. Έτσι επιτυγχάνουμε τα φορτηγά μας να ξεκινούν την διαδρομή τους από την αρχική θέση.

```
for(i=0;i<trucks;i++){
        c_dom_domcst(vars[i], EQUAL,NULL, on[i]);
}
```

Μετά βάζουμε των περιορισμό ότι η πρώτη γραμμή του πίνακα που vars2 πρέπει να είναι ίση με τον πίνακα τις αρχικής κατάστασης των κουτιών. Έτσι επιτυγχάνουμε τα κουτιά μας να βρίσκονται στην αρχική θέση.

```
for(i=0;i<box;i++){
        if(c_domain_value(boxes[i],&val)==FAIL)
                printf("get value faillll\n");
        c_dom_domcst(vars2[i], EQUAL,NULL, val);
}
```

Εδώ επιτυγχάνουμε την εκτέλεση της λειτουργίας του φορτώματος. Αυτό γίνεται με το να ελέγξουμε για κάθε φορτίο αν υπάρχει φορτηγό στην ίδια προσπάθεια ( γραμμή του πίνακα ) που να μπορεί να το φορτώσει. Αυτό ακριβώς κάνουν γενικά οι πιο κάτω γραμμές.
Ξεκινούμε δηλαδή από ένα loop το οποίο αντιστοιχεί σε κάθε κουτί. Μετά έχουμε ακόμα ένα loop για να ελέγξουμε για κάθε κουτί αν υπάρχει ένα φορτηγό που μπορεί να το φορτώσει (βρίσκεται στην ίδια πόλη). Αυτό επιτυγχάνεται με την συνάρτηση c_if όπου αν για κάποιο κουτί υπάρχει φορτηγό στην ίδια πόλη τότε καλή στην συνάρτηση load διαφορετικά θα εκτελεστή η συνάρτηση fin.

```
for(i=0; i<(box*tries)-box; i++){
        if(i%(box)==0 && i!=0){
                        k++;
                }
        for(j=0;j<trucks;j++,pos++){
                param->curr_box[pos]=i;
                param->curr_truck[pos]=j+trucks*k;
                c_if(vars2[i], EQUAL,vars[j+trucks*k] ,0, load, pos , fin,pos );
        }
}
```

Ας δούμε τώρα πως λειτουργεί η συνάρτηση load. Βασικά σε αυτή την συνάρτηση υπάρχουν οι πιο κάτω 2 περιορισμοί. Ο πρώτος προσθέτει τον περιορισμό ότι η τιμή που θα πάρει το κουτί στην επόμενη προσπάθεια να είναι ίση με την τιμή του φορτηγού + πόλεις. Έτσι αν ένα κουτί έχει τιμή μεγαλύτερη από τον αριθμό των πόλεων που υπάρχουν τότε σημαίνει ότι είναι φορτωμένο σε κάποιο φορτηγό. Έτσι κάνοντας την αφαίρεση μπορούμε να δούμε σε πιο φορτηγό είναι φορτωμένο.

```
c_dom_domcst(vars2[param->curr_box[pos]+box], EQUAL,NULL, (param->curr_truck[pos]%(trucks)+1+cities));
```

Ο δεύτερος περιορισμός μας λέει ότι στην επόμενη προσπάθεια του το φορτηγό δεν θα αλλάξει θέση έτσι ώστε να μπορεί να επιτευχτεί το φόρτωμα.

c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars[param->curr_truck[pos]],0);

Αν δεν εκτελεστή η συνάρτηση load τότε θα εκτελεστή η συνάρτηση fin. Με αυτή την συνάρτηση το μόνο που γίνετε είναι μια βελτιστοποίηση του χρόνου εκτέλεσης του προγράμματος. Δηλαδή αν τα φορτηγά δεν φορτώσουν κάποιο κουτί στην επόμενη τους προσπάθεια πάνε σε κάποια πόλη που σίγουρα θα έχει κάποιο κουτί. Έτσι αντί να οπισθοδρομεί άσκοπα το πρόγραμμα μας τα φορτηγά που δεν έχουν φορτωμένο κάποιο κουτί πηγαίνουν κάπου που θα είναι χρήσιμα.

c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars2[param->curr_box[pos]], 0 );

Εδώ επιτυγχάνουμε την εκτέλεση της λειτουργίας του ξεφορτώματος. Αυτό γίνεται με το να ελέγξουμε αν κάποιο κουτί είναι φορτωμένο σε ένα φορτηγό τότε αυτό το κουτί μπορεί να ξεφορτωθεί στην πόλη που έχει πάει το φορτηγό. Αυτό ακριβώς κάνουν γενικά οι πιο κάτω γραμμές.
Ξεκινούμε δηλαδή από ένα loop το οποίο αντιστοιχεί σε κάθε κουτί. Μετά έχουμε ακόμα ένα loop για να ελέγξουμε για κάθε κουτί αν υπάρχει ένα φορτηγό που μπορεί να το ξεφορτώσει. Αυτό επιτυγχάνεται με την συνάρτηση c_if όπου αν για κάποιο κουτί υπάρχει φορτηγό στην ίδια πόλη τότε καλή στην συνάρτηση unload διαφορετικά θα εκτελεστή η συνάρτηση fin2.

```
for(i=0; i<(box*tries)-box; i++){
        if(i%(box)==0 && i!=0){
                        k2++;
                }
        for(j=0;j<trucks;j++,pos2++){

                param->curr_box[pos2]=i;
                param->curr_truck[pos2]=j+trucks*k2;
```

```
                c_if(vars2[i], EQUAL,vars[j+trucks*k2] ,cities, unload, pos2 ,
        fin2,pos2 );
        }
}
```

Ας δούμε τώρα πως λειτουργεί η συνάρτηση load. Βασικά σε αυτή την συνάρτηση υπάρχει ο πιο κάτω περιορισμός. Με αυτό τον περιορισμό επιτυγχάνετε το ξεφόρτωμα του κουτιού (στην επόμενη προσπάθεια) στην πόλη όπου βρίσκεται το φορτηγό.

```
c_dom_domcst(vars2[param->curr_box[pos2]+box], EQUAL,vars[param-
>curr_truck[pos2]+trucks], 0);
```

Με τον πιο κάτω κώδικα εξασφαλίζουμε ότι αν κάποιο κουτί δεν είναι φορτωμένο σε κάποιο φορτηγό τότε να μείνει στην θέση που ήταν στην επόμενη προσπάθεια (αφού ένα κουτί μπορεί να μετακινηθεί μόνο από ένα φορτηγό).

```
for(i=0; i<(box*tries)-box; i++){
                pos3=i;
                c_if_in(vars2[i],if_in,trucks, stay, pos3 , fin3,pos3);
}
```

Στην συνάρτηση stay υπάρχει ο πιο κάτω περιορισμός ο οποίος μας δηλώνει ότι το κιβώτιο στην επόμενη προσπάθεια θα μείνει στην θέση που ήταν.

```
c_dom_domcst(vars2[pos3], EQUAL,vars2[pos3+box], 0 );
```

Συνεχίζοντας στην συνάρτηση cons υπάρχει ο περιορισμός ότι τα κουτιά στην τελευταία θέση πρέπει να βρίσκονται στις πόλεις τις οποίες έχει εισάγει ο χρήστης σαν είσοδο. Αν φτάσουμε σε αυτό το σημείο και ο πιο κάτω περιορισμός κάνει FAILL τότε ξέρουμε ότι δεν υπάρχει λύση.

```
for(i=0;i<box;i++){
        c_domain_value(boxes_f[i],&val);
        c_dom_domcst(vars2[box*(tries-1)+i], EQUAL,NULL, val);
```

```
}
```

Τέλος υπάρχει ο πιο κάτω περιορισμός που είναι απλά μια βελτίωση του χρόνου εκτέλεσης του προγράμματος όπου τα φορτηγά προσπαθούν να περάσουν στην τελευταία τους κατάσταση από πόλεις στις οποίες θα βρίσκονται τα κουτιά στην τελική τους κατάσταση.

```
for(i=0;i<box;i++){
        c_dom_domcst(vars[trucks*tries-box+i], EQUAL,vars2[box*tries-
        box+i],0);
}
```

Μετά από τα πιο πάνω επιστρέφουμε στην main και προσπαθούμε να βρούμε αν υπάρχει πιθανή λύση στο πρόβλημά μας χρησιμοποιώντας την συνάρτηση c_labeling.

```
if (!c_labeling(vars, trucks*tries, METHOD_INPUT_ORDER,
&options))
{
        printf("\nDen iparxei lisi\n");
}

if (!c_labeling(vars2, box*tries, METHOD_INPUT_ORDER,
&options))
{
        printf("\nDen iparxei lisi\n");
}



c_domain_array_print(stdout, vars,  trucks*tries);
printf("\n");
c_domain_array_print(stdout, vars2, box*tries);
printf("\n");
```

Τέλος τυπώνουμε τα αποτελέσματά μας χρησιμοποιώντας την συνάρτηση print_result.

```
printf("\nPrint boxes results\n");
print_result(vars2,box*tries);
printf("\nPrint trucks results\n");
print_trucks(vars);
```

Ας δούμε τώρα πως δουλεύει η συνάρτηση print_result. Σε γενικές γραμμές αυτή η συνάρτηση τυπώνει το μονοπάτι που έχουν ακολουθήσει τα κουτιά δηλ. από ποιες πόλεις έχουν περάσει και από ποια φορτηγά έχουν φορτωθεί, και τέλος ελέγχει αν υπάρχει ή όχι λύση σε αυτή την προσπάθεια.

```
int print_result(DvarPtr *vars2, int size){
int *preval;
```

Στον πιο κάτω κώδικα δεσμεύουμε χώρο για τον πίνακα preval ο οποίος κρατά την θέση των κουτιών στην προηγούμενη προσπάθεια. Αυτό το κάνουμε για να ξέρουμε αν ένα κουτί έχει μεταφερθεί από φορτηγό σε πόλη ή και αντίστροφα για να μπορούμε να τυπώσουμε αν έχει φορτωθεί ή ξεφορτωθεί.

```
preval= (int *)malloc(sizeof(int)*box);
```

Σε αυτή την συνάρτηση υπάρχει ένα for που παίρνει για κάθε κουτί την πόλη στην οποία βρίσκεται με την βοήθεια της συνάρτησης c_domain_value.

```
for(i=0; i<box*tries; i++){
        if(i%(box)==0 && i!=0){
                printf("\n Allagi prospatheias\n");
        }
        c_domain_value(vars2[i],&val);
```

Έχοντας ήδη πάρει την τιμή του κιβωτίου στη μεταβλητή val μπορούμε με ένα if να δούμε αν το κουτί μας βρίσκεται σε κάποια πόλη και αν ναι να δούμε αν στην προηγούμενη του προσπάθεια ήταν φορτωμένο από κάποιο φορτηγό οπόταν σημαίνει ότι στην πόλη στην οποία βρίσκεται έχει ξεφορτωθεί από τον φορτηγό που βρίσκεται στον πίνακα preval.

```
        if(val<cities+1){
                if(preval[(i)%box]<cities){
                        printf("to kouti %d brisketai stin poli
                %d\n",((i)%box)+1,val);
```

```
                }
                else{
                        printf("to kouti %d ksefortwnetai stin poli
                %d\n",((i)%box)+1,val);
                }
        }
```

Αλλιώς αν το φορτηγό δεν βρίσκεται σε κάποια πόλη πάει να πει ότι βρίσκεται (φορτώνεται) σε κάποιο φορτηγό.

```
        else{
                printf("to kouti %d fortwnetai apo to fortigo
        %d\n",((i)%box)+1,(val-cities));
        }
        preval[i%box]=val;

}
```

Πιο κάτω γίνεται ο έλεγχος αν υπάρχει λύση σε αυτή την προσπάθεια. Αυτό γίνεται με το να ελέγξουμε την τελευταία γραμμή του πίνακα των κουτιών. Αν αυτή συμφωνεί με την θέση που πήραμε σαν είσοδο από τον χρήστη για την τελική κατάσταση των κουτιών τότε το πρόγραμμα μας έχει φτάσει σε λύση και σταματά την εκτέλεση του. Αν όχι τότε θα συνεχίσει αυξάνοντας τις προσπάθειές του (αν μπορεί).

```
for(i=0;i<box;i++){
        c_domain_value(vars2[box*(tries-1)+i],&val);
        final_state[i]=val;

}
if(equals(final_state,boxes_f,box)==1){
        printf("SUCCEEEEEEEED\n");
        exit(-1);
}
else{
        printf("No solution for this try\n");
```

```
}
```

Τέλος επιστρέφουμε πίσω στην main() όπου και κάνουμε reset την chipc έτσι ώστε να μπορέσουν να ανατεθούν νέοι περιορισμοί την επόμενη φορά που θα τρέξει το πρόγραμμά μας με διαφορετικό αριθμό προσπαθειών.

```
        if (c_chip_reset() != 1)
        {
                printf("\nInitialize FAILEDDD\n");
        }
}   //end for gia prospatheies
```

### 4.5 Παράδειγματα εκτέλεσης:

#### 4.5.1 Παράδειγμα πρώτο

Library CHIP ANSI C Version 5.7.0.0
Copyright (c) 1992-2005 COSYTEC SA

Created Oct 14 2005 10:15:39

Kalosorisate sto programa metaforas fortiwn
dwse ton arithmo twn polewn pou tha iparxoun:4
dwse ton arithmo twn fortigwn pou iparxoun:3
dwse ton arithmo twn fortiwn pou iparxoun:3
dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:2
dwse tin poli pou brisketai to kathe fortigo
truck 1:1
truck 2:2
truck 3:4
to fotigo 1 brisketai stin poli 1
to fotigo 2 brisketai stin poli 2
to fotigo 3 brisketai stin poli 4
dwse ton arithmo twn koutiwn pou tha exei i kathe poli
poli 1:1
poli 2:1
poli 3:0
poli 4:1
dwse ton arithmo twn fortiwn pou tha exei i kathe poli stin teliki katastasi
fortio 1:0
fortio 2:0
fortio 3:3
fortio 4:0

Setting Cons
[D_1=1, D_2=2, D_3=4, D_4=1, D_5=1, D_6=1]
[D_7=1, D_8=2, D_9=4, D_10=5, D_11=6, D_12=7]

Print boxes results
to kouti 1 brisketai stin poli 1
to kouti 2 brisketai stin poli 2
to kouti 3 brisketai stin poli 4

Allagi prospatheias
to kouti 1 fortwnetai apo to fortigo 1
to kouti 2 fortwnetai apo to fortigo 2
to kouti 3 fortwnetai apo to fortigo 3
No solution for this try

Print trucks results
to fortigo 1 brisketai stin poli 1
to fortigo 2 brisketai stin poli 2
to fortigo 3 brisketai stin poli 4

Allagi prospatheias
to fortigo 1 menei stin poli 1
to fortigo 2 pigainei stin poli 1
to fortigo 3 pigainei stin poli 1
Setting Cons
[D_47=1, D_48=2, D_49=4, D_50=1, D_51=1, D_52=1, D_53=3, D_54=3, D_55=3]
[D_56=1, D_57=2, D_58=4, D_59=5, D_60=6, D_61=7, D_62=3, D_63=3, D_64=3]

Print boxes results
to kouti 1 brisketai stin poli 1
to kouti 2 brisketai stin poli 2
to kouti 3 brisketai stin poli 4

Allagi prospatheias
to kouti 1 fortwnetai apo to fortigo 1
to kouti 2 fortwnetai apo to fortigo 2
to kouti 3 fortwnetai apo to fortigo 3

Allagi prospatheias
to kouti 1 ksefortwnetai stin poli 3
to kouti 2 ksefortwnetai stin poli 3
to kouti 3 ksefortwnetai stin poli 3
SUCCEEEEEEEED

Στο πιο πάνω παράδειγμα θέλουμε 2 προσπάθειες για να φτάσουμε σε λύση αφού τα κουτιά μας βρίσκονται αρχικά στην ίδια θέση που βρίσκονται και τα φορτηγά μας αντίστοιχα. Έτσι θα χρειαστεί μια προσπάθεια για να φορτωθούν τα κουτιά από τα φορτηγά και ακόμη μία για να μεταφερθεί κάθε κουτί στον τελικό του προορισμό. Όπως φαίνεται και πιο πάνω το πρόγραμμα δοκιμάζει να βρει λύση με μία προσπάθεια όπου και αποτυγχάνει αν και το φόρτωμα το κουτιών γίνετε. Όταν οι προσπάθειες αυξηθούν σε δύο τότε στην πρώτη γίνετε το φόρτωμα και στην δεύτερη το ξεφόρτωμα με αποτέλεσμα ο κώδικας μας να βρει λύση.

### 4.5.2 Παράδειγμα δεύτερο

Kalosorisate sto programa metaforas fortiwn
dwse ton arithmo twn polewn pou tha iparxoun:4
dwse ton arithmo twn fortigwn pou iparxoun:3
dwse ton arithmo twn fortiwn pou iparxoun:3
dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:2
dwse tin poli pou brisketai to kathe fortigo
truck 1:1

truck 2:2

truck 3:1

to fotigo 1 brisketai stin poli 1

to fotigo 2 brisketai stin poli 2

to fotigo 3 brisketai stin poli 1

dwse ton arithmo twn koutiwn pou tha exei i kathe poli

poli 1:0

poli 2:0

poli 3:3

poli 4:0

dwse ton arithmo twn fortiwn pou tha exei i kathe poli stin teliki katastasi

fortio 1:0

fortio 2:3

fortio 3:0

fortio 4:0

Setting Cons

[D_1=1, D_2=2, D_3=1, D_4=3, D_5=3, D_6=3]

[D_7=3, D_8=3, D_9=3, D_10=3, D_11=3, D_12=3]

Print boxes results

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

Allagi prospatheias

to kouti 1 brisketai stin poli 3

to kouti 2 brisketai stin poli 3

to kouti 3 brisketai stin poli 3

No solution for this try

Print trucks results

to fortigo 1 brisketai stin poli 1

to fortigo 2 brisketai stin poli 2

to fortigo 3 brisketai stin poli 1

Allagi prospatheias
to fortigo 1 pigainei stin poli 3
to fortigo 2 pigainei stin poli 3
to fortigo 3 pigainei stin poli 3
Setting Cons
[D_47=1, D_48=2, D_49=1, D_50=3, D_51=3, D_52=3, D_53=3, D_54=3, D_55=3]
[D_56=3, D_57=3, D_58=3, D_59=3, D_60=3, D_61=3, D_62=7, D_63=7, D_64=7]

Print boxes results
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 fortwnetai apo to fortigo 3
to kouti 2 fortwnetai apo to fortigo 3
to kouti 3 fortwnetai apo to fortigo 3
No solution for this try

Print trucks results
to fortigo 1 brisketai stin poli 1
to fortigo 2 brisketai stin poli 2
to fortigo 3 brisketai stin poli 1

Allagi prospatheias
to fortigo 1 pigainei stin poli 3

to fortigo 2 pigainei stin poli 3
to fortigo 3 pigainei stin poli 3

Allagi prospatheias
to fortigo 1 menei stin poli 3
to fortigo 2 menei stin poli 3
to fortigo 3 menei stin poli 3

Total time : 3 ms

Στο πιο πάνω παράδειγμα δεν μπορούμε να φτάσουμε σε λύση με αριθμό προσπαθειών ίσο με 2. αφού κανένα φορτηγό δεν μπορεί να φορτώσει τα κουτιά μας που βρίσκονται όλα στην πόλη 3. έτσι θα χρειαστούμε μία προσπάθεια για να μεταφερθεί κάποιο φορτηγό σε αυτήν την πόλη , ακόμη μια προσπάθεια για να φορτώσει το φορτηγό μας και τα κουτιά και μια τελευταία προσπάθεια για να τα μεταφέρει στην πόλη 2. έτσι θέλουμε συνολικά 3 προσπάθειες και όχι 2 που είναι ο μέγιστος αριθμός προσπαθειών που δικαιούμαστε.

### 4.5.3 Παράδειγμα τρίτο

Kalosorisate sto programa metaforas fortiwn
dwse ton arithmo twn polewn pou tha iparxoun:4
dwse ton arithmo twn fortigwn pou iparxoun:3
dwse ton arithmo twn fortiwn pou iparxoun:3
dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na brethei lisi:3
dwse tin poli pou brisketai to kathe fortigo
truck 1:1
truck 2:2

truck 3:1
to fotigo 1 brisketai stin poli 1
to fotigo 2 brisketai stin poli 2
to fotigo 3 brisketai stin poli 1
dwse ton arithmo twn koutiwn pou tha exei i kathe poli
poli 1:0
poli 2:0
poli 3:3
poli 4:0
dwse ton arithmo twn fortiwn pou tha exei i kathe poli stin teliki katastasi
fortio 1:0
fortio 2:3
fortio 3:0
fortio 4:0
Setting Cons
[D_1=1, D_2=2, D_3=1, D_4=3, D_5=3, D_6=3]
[D_7=3, D_8=3, D_9=3, D_10=3, D_11=3, D_12=3]

Print boxes results
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3
No solution for this try

Print trucks results
to fortigo 1 brisketai stin poli 1
to fortigo 2 brisketai stin poli 2
to fortigo 3 brisketai stin poli 1

Allagi prospatheias
to fortigo 1 pigainei stin poli 3
to fortigo 2 pigainei stin poli 3
to fortigo 3 pigainei stin poli 3
Setting Cons
[D_47=1, D_48=2, D_49=1, D_50=3, D_51=3, D_52=3, D_53=3, D_54=3, D_55=3]
[D_56=3, D_57=3, D_58=3, D_59=3, D_60=3, D_61=3, D_62=7, D_63=7, D_64=7]

Print boxes results
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 fortwnetai apo to fortigo 3
to kouti 2 fortwnetai apo to fortigo 3
to kouti 3 fortwnetai apo to fortigo 3
No solution for this try

Print trucks results
to fortigo 1 brisketai stin poli 1
to fortigo 2 brisketai stin poli 2
to fortigo 3 brisketai stin poli 1

Allagi prospatheias
to fortigo 1 pigainei stin poli 3
to fortigo 2 pigainei stin poli 3

to fortigo 3 pigainei stin poli 3

Allagi prospatheias
to fortigo 1 menei stin poli 3
to fortigo 2 menei stin poli 3
to fortigo 3 menei stin poli 3
Setting Cons
[D_123=1, D_124=2, D_125=1, D_126=3, D_127=3, D_128=3, D_129=3, D_130=3, D_131=3, D_132=2, D_133=2, D_134=2]
[D_135=3, D_136=3, D_137=3, D_138=3, D_139=3, D_140=3, D_141=7, D_142=7, D_143=7, D_144=2, D_145=2, D_146=2]

Print boxes results
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 brisketai stin poli 3
to kouti 2 brisketai stin poli 3
to kouti 3 brisketai stin poli 3

Allagi prospatheias
to kouti 1 fortwnetai apo to fortigo 3
to kouti 2 fortwnetai apo to fortigo 3
to kouti 3 fortwnetai apo to fortigo 3

Allagi prospatheias
to kouti 1 ksefortwnetai stin poli 2
to kouti 2 ksefortwnetai stin poli 2
to kouti 3 ksefortwnetai stin poli 2
SUCCEEEEEEEED

Στο πιο πάνω παράδειγμα τα φορτηγά και τα κουτιά μας είναι τοποθετημένα ακριβώς με τον ίδιο τρόπο όπως και στο προηγούμενο μας παράδειγμα με την μόνη διαφορά ότι έχουμε τρεις προσπάθειες για να βρούμε λύση. Έτσι τα φορτηγάμας πηγαίνουν στην πόλη που βρίσκονται τα κουτιά στην πρώτη προσπάθεια. Στην δεύτερη προσπάθεια το τρίτο φορτηγό φορτώνει τα κουτιά και στην Τρίτη προσπάθεια τα παίρνει στην πόλη 2 όπου είναι και η τελική τους θέση.

# Κεφάλαιο 5 Συμπεράσματα:

Η chipc είναι μια γλώσσα προγραμματισμού η οποία μπορεί να λύσει προβλήματα τα οποία δεν μπορεί να λυθούν με κλασσικές γλώσσες προγραμματισμού. Παρόλο που η chipc είναι μια γλώσσα βασισμένη σε C και με μια πρώτη ματιά μπορεί να φαίνεται πολύ παρόμοια στην τελική ανάλυση προσφέρει ένα εντολές ξεχωριστό τρόπο σκέψεις και υλοποιήσεις προγραμμάτων. Προσφέρει ακόμα αρκετές συναρτήσεις οι οποίες σχετίζονται με περιορισμούς και ευρέσεων λύσης δοκιμάζοντας ένα φάσμα μεταβλητών. Παρόλα αυτά η chipc έχει και τα αρνητικά της αφού σε μερικές περιπτώσεις δεν σου δίνει πλήρες ελευθερία στον προγραμματισμό. Ένα άλλο αρνητικό είναι ότι η chip δεν είναι μια ευρέως διαδεδομένη γλώσσα πράγμα που σε δυσκολεύει πολύ στο να βρεις βοήθεια μέσω βιβλίων ή μέσω του διαδικτύου.

Ένα από τα σημαντικά προβλήματα που παρουσιάστηκαν κατά την επίλυση των προβλημάτων είναι ότι η chipc δεν μπορεί να επεξεργαστεί δυσδιάστατους πίνακες τύπου DvarPtr με αποτέλεσμα να μετατρέψουμε αυτούς τους πίνακες σε μονοδιάστατους. Πράγμα που δυσκόλεψε τον προγραμματισμό του προβλήματος.

# Βιβλιογραφία:


1. http://en.wikipedia.org/wiki/Logic_programming
2. COSYTEC: “Application development with the chip system”
3. “Chipc reference manual”
4. Generalizing GraphPlan by Formulating Planning as a CSP
5. http://en.wikipedia.org/wiki/Constraint_satisfaction_problem
6. http://switch.vub.ac.be/~tlenaert/documents/teach/ai1/planning.pdf


# Παράρτημα Α

```
//blocks
```

```
#include <stdio.h> //Standard I/O library
#include <stdlib.h> //Standard functions library
#include "chipc.h"  //CHIPC library

//global variables

int blocks,val;
int count=0;
DvarPtr *vars;

void print_on(int blocks,int *on,int floor){
        int i;
        printf("to patoma xaraktirizete me ton arithmo %d-
%d\n",blocks+1,blocks+floor);
        for (i=0; i<blocks; i++){
        printf("to block %d brisketai pano sto %d\n",i+1,on[i]);
        }
}


Pred_result callback(void * arg) {
        //dimoirgia periorismou wste to block pou tha metakinisoukme na einai
        elefthero
        c_domain_value(vars[count],&val);
        if(val <= blocks){
        c_dom_domcst(vars[count], EQUAL,vars[count+blocks] ,0);
        }

        count++;
        return SUCCEED;
}
int show_result(DvarPtr vars[], int nvars)
{
```

```
        int val,i;
        int preval;
        int block;

        for (i=0; i<nvars; i++){
                block=(i)%blocks;
        if (i >= blocks){
                if(c_domain_value(vars[i-blocks],&preval)==FAIL)
                        printf("Get prevalue FAIL\n");

        }
        if(c_domain_value(vars[i],&val)==FAIL)
                printf("Get value FAIL\n");
        if (block==0)
                printf("\nAllagi prospatheias \n \n");
        if(i < blocks)
                printf("To block %d stin prospatheia %d brisketai stin thesi
%d\n",block+1,i/blocks+1,val);
        else if(preval==val)
                printf("To block %d stin prospatheia %d menei stin thesi
%d\n",block+1,i/blocks+1,val);
        else
                printf("To block %d stin prospatheia %d pigainei stin thesi
%d\n",block+1,i/blocks+1,val);
        }

  return 1;
}




void cons (int *on, int *on_f, int blocks,int tries, DvarPtr *vars){
```

```
        int i,j,k;
        void *arg;
        arg=(void*)vars;
        //i prwti grammi einai isi me to *on
        for(i=0;i<blocks;i++){
                c_dom_domcst(vars[i], EQUAL,NULL, on[i]);
        }

        //i teleftaia grammi einai isi me to *on_f
        for(i=0;i<blocks;i++){
                c_dom_domcst(vars[blocks*(tries-1)+i], EQUAL,NULL, on_f[i]);
        }

//dimiourga periorismou wste na einai to block pou tha pame elefthero
        for(i=blocks;i<blocks*tries;i++){   //for gia grammes ektos apo tin prwti
                for(j=1; j<blocks; j++){   //for gia stiles

                                        c_dom_domcst(vars[i], DIFFERENT,vars[i-
j],0);
                        }
                }

//dimoirgia periorismou wste to block pou tha metakinisoukme na einai elefthero
        for(i=0;i<(blocks*tries)-blocks; i++){              //gia ola ta blocks
                        c_demon(vars[i], callback, arg, GROUND);          //otan parei
timi i thesi pou ginetai afipnisi tou demon
        }


}


int main() {
```

```
int i,tries,floor,nvars;
int *on,*on_f,start;

Labeling options;
start = c_cputime();
if (c_chip_init() != 1)
   {
        return 0;
   }

printf("Kalwsirthate sto programa twn kibwn\n");
printf("Dwse arithmo twn blocks:");
scanf("%d",&blocks);
printf("Dwse arithmo twn prospathiwn pou tha ginoun mexri na brethei lisi:");
scanf("%d",&tries);
printf("Dwse arithmo diathesimwn thesew pou iparxoun sto patwma:");
scanf("%d",&floor);
tries=tries+2;
vars = (void**)malloc(sizeof(int)* (blocks*tries));

//kathorise domain
c_create_domain_array(&vars[0], blocks*tries ,1,blocks+floor,CONSEC);


   on= (int *)malloc(sizeof(int)*blocks);
        on_f= (int *)malloc(sizeof(int)*blocks);

        printf("dwse to simeio pou brisketai to kathe block (an einai sto
patoma tote dwse %d-%d)\n",blocks+1,blocks+floor);
        for(i=0; i<blocks; i++){
                printf("block %d:",i+1);
                scanf("%d",&on[i]);
```

```
                while (on[i]<=0 || on[i]>(blocks+floor)){
                        printf("Den iparxei tetoia thesi, ksanadwse arithmo:");
                        scanf("on:%d",&on[i]);
                }
        }
        print_on(blocks,on,floor);
        printf("dwse to simeio pou tha brisketai to kathe mplock stin teliki katastasi(an einai sto patoma tote dwse %d-%d)\n",blocks+1,blocks+floor);
        for(i=0; i<blocks; i++){
                printf("block %d:",i+1);
                scanf("%d",&on_f[i]);
                while (on[i]<=0 || on_f[i]>(blocks+floor)){
                        printf("Den iparxei tetoia thesi, ksanadwse arithmo:");
                        scanf("on:%d",&on[i]);
                }
        }

        nvars=blocks*tries;
    cons(on,on_f,blocks,tries,vars);


  //Set options for solution finding method
    options.type     = LABELING_INDOMAIN;//Use domain to find solution
    options.indomain = LABELING_INDOMAIN_MIDDLE;//Start from middle of domain
    options.solution = LABELING_FIRST_SOL; //Find first possible solution and stop
    options.continue_var = 0; //Check is assigment was successful
    options.continue_all = show_result;//Use everything in domain and when done show solution

        //Find solution if there is one
```

```
   if (c_labeling(vars,blocks*tries, METHOD_INPUT_ORDER, &options)==FAIL)
//If FIRST FAIL is obtained then stop
        {
          printf("\nNo solution found!");
        }
        printf("Total time : %d ms\n", c_cputime()-start);
c_chip_end();
return 0;
}
```

# Παράρτημα Β

```
//transfer

#include <stdio.h>
#include "chipc.h"
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <ctype.h>

//global variables : the domain variable vars is for trucks and vars2 is for boxes
DvarPtr *vars,*vars2;
int *if_in,*final_state;
int trucks,box,maxtries,cities,temp,tries,i,j,val,**grafos, *boxes,
*boxes_f,start,curr_truck;
int preval=0;
int count=0;
int fail=0;
int k=0;
int k2=0;
```

```
int pos=0;
int pos2=0;
int pos3=0;

//einai ena struct sto opoio filagoume tis parametrous pou tha xreiastoun oi sinartiseis load, unload , stay
typedef struct argum
{
	int *curr_box;
	int *curr_truck;

}ARGUM;

ARGUM *param;

//sinartisi pou arxikopoia tis times tou pinaka preval(proigoumenes times twn koutiwn)
init_preval(int *preval){

	for(i=0; i<trucks*tries; i++){
		preval[i]=0;

	}
}


//sinartisi pou elegxei an dio pinakes einai isoi. An nai tote epestrefei 1 alliws 0
int equals(int *boxes, int *boxes_f,int box)
{
	int value,i;
	value=1;
	for(i=0; i<box; i++){
		if(boxes[i]!=boxes_f[i]){
```

```
                value=0;
            }
        }
        return value;
}

//sinartisi pou ektipwnei tis theseis tou pinaka vars(times twn polewn pou exoun paei ta fortiga)
void print_trucks(DvarPtr *vars){
        int *preval;
         preval= (int *)malloc(sizeof(int)*trucks);
        for(i=0; i<trucks*tries; i++){
            if(i%(trucks)==0 && i!=0){
                printf("\n Allagi prospatheias\n");
            }

            c_domain_value(vars[i],&val);
            if(i<trucks){
                printf("to fortigo %d brisketai stin poli %d\n",((i)%trucks)+1,val);
            }
            else{
                if(preval[(i)%trucks]==val){
                    printf("to fortigo %d menei stin poli %d\n",((i)%trucks)+1,val);
                }
                else{

                    printf("to fortigo %d pigainei stin poli %d\n",((i)%trucks)+1,val);
                }
            }
            preval[i%box]=val;
```

```
        }


        init_preval(preval);
}

//sinartisi pou ektipwnei tis theseis tou pinaka vars2(polewn pou briskontai ta fortia),
//episis elegxei an stin teliki katastasi ta fortia briskontai stis poleis pou prepei, an nai
//tote termatizei to programma, an oxi tote sinexizete i ektelesei afksanontas tis
prospatheies
int print_result(DvarPtr *vars2, int size){
        int *preval;
         preval= (int *)malloc(sizeof(int)*box);
        for(i=0; i<box*tries; i++){
                if(i%(box)==0 && i!=0){
                        printf("\n Allagi prospatheias\n");
                }
                c_domain_value(vars2[i],&val);
                if(val<cities+1){
                        if(preval[(i)%box]<cities){
                                printf("to kouti %d brisketai stin poli
%d\n",((i)%box)+1,val);
                        }
                        else{
                                printf("to kouti %d ksefortwnetai stin poli
%d\n",((i)%box)+1,val);
                        }
                }
                else{
                        printf("to kouti %d fortwnetai apo to fortigo
%d\n",((i)%box)+1,(val-cities));
                }
                preval[i%box]=val;
```

```
		}

		//elegxos oti exei epitixei
		for(i=0;i<box;i++){
			c_domain_value(vars2[box*(tries-1)+i],&val);
			final_state[i]=val;

		}
		if(equals(final_state,boxes_f,box)==1){
			printf("SUCCEEEEEEEED\n");
			exit(-1);
		}
		else{
			printf("No solution for this trie\n");

		}



		init_preval(preval);
		return 1;
}

//sinartisi pou ekteleite an den ginei load kapoiou koutiou apo fortigo
int fin(int pos){
		//an den ginei load tote ta fortiga stin epomeni prospatheia tha pane se thesi
pou iparxei kouti
		//to programma leitourga kai xwris afton ton periorismo alla einai mia
beltistopoiisi wste na min kanoume axreiastes metakiniseis(grigoroteri ektelesi)
		c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars2[param-
>curr_box[pos]], 0 );
```

```
return 1;
}


//sinartisi pou ekteleite an den ginei unload
int fin2(int pos2){


        //DO NOTHING

return 1;
}


//sinartisi pou ekteleite an den ektelesti i sinartisi stay
int fin3(int pos3){

        //DO NOTHING

return 1;
}

//sinartisi stin opoia ta koutia den exoun fwrtwthei apo kanena fortigo ara kai tha
meinoun stin  thesi pou itan
int stay(int pos3){

        //to kouti stin epomeni prospatheia na meinei ekei pou itan
        c_dom_domcst(vars2[pos3], EQUAL,vars2[pos3+box], 0 );

return 1;
}
```

```
//sinartisi pou ksefwrtwnei ta kooutia stin thesi pou brisketai to fortigo pou ta exei fortwsei
int unload(int pos2){

	//to kouti stin epomeni prospatheia na parei tin timi pou tha exei  kai to fortigo stin epomeni prospatheia
	c_dom_domcst(vars2[param->curr_box[pos2]+box], EQUAL,vars[param->curr_truck[pos2]+trucks], 0);


return 1;
}

//sinartisei pou fortwnei ta koutia sta fortiga
int load(int pos){
	//to kouti stin epomeni prospatheia tha parei tin timi pou fortigou+poleis pou dilwnei oti einai fortwmeno
	c_dom_domcst(vars2[param->curr_box[pos]+box], EQUAL,NULL, (param->curr_truck[pos]%(trucks)+1+cities));

	//to fortigo stin epomeni prospatheia tha meinei stin thesi pou itan gia na epitefthei to fortwma
	c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars[param->curr_truck[pos]],0);

return 1;
}


Pred_result callback(void * arg) {
	int k,truck;
	truck=(count)%trucks;
	count=count % (trucks*tries);
```

```
        if (truck==0)
                printf("\nAllagi prospatheias \n \n");
        if(c_domain_value(vars[count],&val)==FAIL)
                printf("Get value FAIL\n");
        if (count >= trucks){
                if(c_domain_value(vars[count-trucks],&preval)==FAIL)
                        printf("Get prevalue FAIL\n");

        }

        if(count < trucks)
                printf("To fortigo %d stin prospatheia %d brisketai stin poli
%d\n",truck+1,count/trucks+1,val);
        else if(preval==val)
                printf("To fortigo %d stin prospatheia %d menei stin poli
%d\n",truck+1,count/trucks+1,val);
        else
                printf("To fortigo %d stin prospatheia %d pigainei stin poli
%d\n",truck+1,count/trucks+1,val);

        if(count<(trucks*tries)-trucks){
        //periorismos etsi wste ta fortiga na pernoun mono apo poleis pou epitrepete
                for(k=0; k<cities; k++){        //gia kathe timi tou grafou
                        if (grafos[val-1][k]==0){   //bres tin grami pou mas endiaferei
kai elekse tin timi tis
                                if(c_dom_domcst(vars[count+trucks], DIFFERENT,
NULL, (k+1))==FAIL){      //an i timi einai 0 prosthese ton periorismo

                                        printf("Periorismos tis polis stin prospatheia %d
apetixe\n",count/trucks+1);
                                        fail=1;
                                }
                }
```

```
        }

        }
                for(j=0; j<box; j++){              //gia kathe timi tou pinaka boxes

                        if(val==boxes[j] && boxes_f[j]!=boxes[j]){         //an i timi
pou tha parei to fortigo einai isi me tin poli tou koutiou
                                //fortwse
                                boxes[j]=(truck+1)+cities;
                                printf("To fortigo %d fortwnei to fortio
%d\n",truck+1,j+1);
                        }
                        //an exw fortwmeno to kouti kai an to thelei i poli pou
briskomai

                        if(boxes[j]==(truck+1)+cities && boxes_f[j]==val){ //an i timi
pou tha parei to fortigo einai isi me tin poli tou koutiou stin teliki
                                //ksefortwse
                                boxes[j]=val;
                                printf("To fortigo %d ksefortwnei to fortio
%d\n",truck+1,j+1);
                        }
                }

                /*if(equals(boxes,boxes_f,box)==1){
                        c_create_domain_array(&vars[trucks*tries], trucks*tries+1, 0,
cities, CONSEC);
                        if(c_dom_domcst(vars[trucks*tries], EQUAL,NULL,
0)==FAIL){
                                printf("flag2 FAILLLLLLL");
                        }
                }*/
```

```
                count++;
                if(count==trucks*tries){
                        c_create_domain_array(&vars[trucks*tries], trucks*tries+1, 0,
cities, CONSEC);
                        c_choice_remember(&trail);

                        if(c_dom_domcst(vars[trucks*tries], EQUAL,NULL,
0)==FAIL){
                                printf("flag2 FAILLLLLLL");
                        }
                        c_create_domain_array(&vars[trucks*tries], trucks*tries+1, 1,
cities, CONSEC);
                        if(c_dom_domcst(vars[trucks*tries], EQUAL,NULL,
0)==FAIL){
                                printf("flag FAILLLLLLL");
                        }
                        c_create_domain_array(&vars[trucks*tries], trucks*tries+1, 0,
cities, CONSEC);
                                if(c_dom_domcst(vars[trucks*tries], EQUAL,NULL,
0)==FAIL){
                                printf("flag2 FAILLLLLLL");
                        }
                        if(equals(boxes,boxes_f,box)==1){
                                c_create_domain_array(&vars[trucks*tries],
trucks*tries+1, 0, cities, CONSEC);
                                if(c_dom_domcst(vars[trucks*tries], EQUAL,NULL,
0)==FAIL){
                                printf("flag2 FAILLLLLLL");
                                }
                                //c_choice_undo(trail);
                                //if (c_domain_bind_to_value(vars[trucks*tries], 0) ==
FAIL) {
                                //      fprintf(stdout, "Binding fails\n");
```

```
                                        //}
                                }

                        }
                return SUCCEED;
}


//tipoma tou pinaka on pou deilwnei pou brisketai kathe fortigo stin arxiki katastasi
void print_on(int trucks,int *on){
        int i;
        for (i=0; i<trucks; i++){
        printf("to fotigo %d brisketai stin poli %d\n",i+1,on[i]);
        }
}

//tipwma tou pinaka on_f pou deilwnei pou tha brisketai to fortigo stin teliki katastasi
void print_on_f(int trucks,int *on){
        int i;
        for (i=0; i<trucks; i++){
        printf("to fotigo %d tha brisketai stin poli %d\n",i+1,on[i]);
        }
}

//tipoma tou pinaka boxes pou deilwnei pou briskontai ta koutia stin arxiki katastasi
void print_boxes(int box,int *boxes){
        int i;
        for (i=0; i<box; i++){
        printf(" to fortio %d brisketai stin poli %d\n",i+1,boxes[i]);
        }
}
```

```
//tipwma tou pinaka boxes_f pou deilwnei pou tha briskontai ta koutia stin teliki
katastasi
void print_boxes_f(int box,int *boxes_f){
        int i;
        for (i=0; i<box; i++){
        printf("to fortio %d tha brisketai stin poli %d\n",i+1,boxes_f[i]);
        }
}

//sinartisi anatheseis periorismwn
void cons(int **grafos,int *on,int *on_f,int *boxes, int *boxes_f,int cities,int trucks,
int tries,int box,
                                DvarPtr *vars,DvarPtr *vars2, int count)
{
        int k=0;
        void *arg,*arg2;
        arg=(void*)vars;
        arg2=(void*)vars2;


        //i prwti grammi tou pinaka vars2 einai isi me to arxiki thesi fortiwn
        for(i=0;i<box;i++){
                c_dom_domcst(vars2[i], EQUAL,NULL, boxes[i]);
        }

        //i prwti grammi tou pinaka vars einai isi me to arxiki thesi fortigwn
        for(i=0;i<trucks;i++){
                c_dom_domcst(vars[i], EQUAL,NULL, on[i]);

        }


        //LOADDDDDDDD
```

```
        for(i=0; i<(box*tries)-box; i++){    //gia kathe kouti
                if(i%(box)==0 && i!=0){
                                k++;
                        }
                for(j=0;j<trucks;j++,pos++){   //kai gia kathe fortigo
                        //curr_truck=j;
                        param->curr_box[pos]=i;    //apothikefsi timwn sto struct
param->curr_box
                        param->curr_truck[pos]=j+trucks*k;   //apothikefsi timwn sto
struct param->curr_truck
                        //printf("FORRR curr_box:%d - curr_truck:%d,
pos:%d\n",i,j+trucks*k,pos);
                        //an gia kapoio kouti pou brisketai stin current grammi iparxei
kapoio fortigo pou mporei na to fwrtwsei
                        //(na briskontai dil stin idia poli) tote kaleitai i sinartisi load...
                        c_if(vars2[i], EQUAL,vars[j+trucks*k] ,0, load, pos , fin,pos );
                }
        }


        //UNLOADDDDDDDDD
        for(i=0; i<(box*tries)-box; i++){    //gia kathe kouti
                if(i%(box)==0 && i!=0){
                                k2++;
                        }
                for(j=0;j<trucks;j++,pos2++){  //kai gia kathe fortigo
                        //curr_truck=j;
                        param->curr_box[pos2]=i;  //apothikefsi timwn sto struct
param->curr_box
                        param->curr_truck[pos2]=j+trucks*k2;     //apothikefsi timwn
sto struct param->curr_truck
                        //printf("FORRR2222222222 curr_box:%d - curr_truck:%d,
pos2:%d\n",i,j+trucks*k2,pos2);
```

```
                        //an kapoio kouti einai fortwmeno se ena fortigo tote to current kouti mporei na ginei unload
                        c_if(vars2[i], EQUAL,vars[j+trucks*k2] ,cities, unload, pos2 , fin2,pos2 );
                }
        }


        //STAY
        for(i=0; i<(box*tries)-box; i++){   //gia ola ta koutia
                        //an o kwdikas proxwrisei kai kapoio kouti den exei ginei oute load oute unload tote prepei na minei stin thesi pou itan
                        //alliws o pio katw periorismos tha kanei FAIL
                        pos3=i;
                        c_if_in(vars2[i],if_in,trucks, stay, pos3 , fin3,pos3);
        }



        //I teleftaia thesi einai isi me tin teliki thesi ton fortiwn
        for(i=0;i<box;i++){
                //an aftos o periorismos kanei fail tote ta koutia den mporoun na metaferthoun stin teliki tous katastasi ara den iparxei lisi
                c_dom_domcst(vars2[box*(tries-1)+i], EQUAL,NULL, boxes_f[i]);
        }


        for(i=0;i<box;i++){
                //beltistopoiisi etsi wste ta fortiga na prospathoun na perasoun apo tis telikes theseis twn koutiwn
```

```
                c_dom_domcst(vars[trucks*tries-box+i], EQUAL,vars2[box*tries-
box+i],0);
        }


}


int main(){
        //dilwsei metablitwn tis main()
        int nvars;
        int i,j,x,y;
        char ans;
        int *on,*on_f;

   Labeling  options;

        //xronos ekteleseis programmatos
        start = c_cputime();

        //arxikopoiisi chipc
        if (c_chip_init() != 1)
        {
                return 0;
        }

        //get user input
        printf("Kalosorisate sto programa metaforas fortiwn\n");
        printf("dwse ton arithmo twn polewn pou tha iparxoun:");
        scanf("%d",&cities);
        printf("dwse ton arithmo twn fortigwn pou iparxoun:");
        scanf("%d",&trucks);
        printf("dwse ton arithmo twn fortiwn pou iparxoun:");
```

```
        scanf("%d",&box);
        printf("dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na
brethei lisi:");
        scanf("%d",&maxtries);

        //arxikopoiisi prospatheiwn se 2
        tries=2;
        //arxikopoiisi megistwn prospatheiwn se +2
        maxtries=maxtries+2;

        //desmefsi xwrou gia pinakes
        if_in=(int *)malloc(sizeof(int)*trucks);
        final_state=(int *)malloc(sizeof(int)*box);
   on= (int *)malloc(sizeof(int)*trucks);
        on_f= (int *)malloc(sizeof(int)*trucks);
   boxes= (int *)malloc(sizeof(int)*box);
        boxes_f= (int *)malloc(sizeof(int)*box);


        //dimiourgia grafou
        grafos = (int **) malloc(sizeof(int*)*cities);
        for(i = 0 ; i < cities; i++)
        {
        grafos[i] = (int *) malloc (sizeof(int)* cities);
        }

        for(i=0;i<trucks;i++){
                if_in[i]=i+1;
        }

        //user input gia pinaka on
        printf("dwse tin poli pou brisketai to kathe fortigo\n");
                for(i=0; i<trucks; i++){
```

```
                printf("truck %d:",i+1);
                scanf("%d",&on[i]);

                while (on[i]<=0 || on[i]>cities){
                        printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                        scanf("%d",&on[i]);
                }
        }
        //tipoma pinaka on
        print_on(trucks,on);

        //user input gia pinaka boxes
        printf("dwse tin poli pou brisketai to kathe fortio\n");
        for(i=0; i<box; i++){
                printf("fortio %d:",i+1);
                scanf("%d",&boxes[i]);

                while (boxes[i]<=0 || boxes[i]>cities){
                        printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                        scanf("%d",&boxes[i]);
                }
        }
        print_boxes(box,boxes);


        //user input gia pinaka boxes_f
        printf("dwse to simeio pou tha brisketai to kathe fortio stin teliki
katastasi\n");
        for(i=0; i<box; i++){
                printf("fortio %d:",i+1);
                scanf("%d",&boxes_f[i]);

                while (boxes_f[i]<=0 || boxes_f[i]>cities){
```

```
                        printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                        scanf("%d",&boxes_f[i]);
                }
        }

        print_boxes_f(box,boxes_f);



    for(tries=2; tries<maxtries; tries++, pos=0, pos2=0,k=0, k2=0,pos3=0){
//for gia ektelesei tou kwdika me diaforetiki timi prospatheiwn

        //desmefsi xwrou gia domain variables
        vars =  (void**)realloc(vars,sizeof(int)*(trucks*tries));
        vars2 = (void**)realloc(vars2,sizeof(int)*(box*tries));

        //desmefsi xwrou gia struct
        param =(ARGUM *)realloc(param,sizeof(ARGUM));
        param->curr_box= (int *) realloc(param-
>curr_box,sizeof(int)*(box*tries*trucks));
        param->curr_truck=(int*) realloc(param-
>curr_truck,sizeof(int)*(box*tries*trucks));

        //dimiourgeia pediou timwn
        c_create_domain_array(vars, trucks*tries, 1, cities, CONSEC);
        c_create_domain_array(vars2, box*tries, 1, cities+trucks, CONSEC);
        for(i=0; i<trucks; i++){
                c_create_domain_array(&vars_merge[i], 1, 1, cities,
CONSEC);
        }
        for(i=trucks; i<trucks*tries; i++){
                c_create_domain_array(&vars_merge[i], 1, 1, cities+trucks,
CONSEC);
```

```
        }



        //kalesma sinartisis gia eisagwgi periorismwn
        printf("Setting Cons\n");

    cons(grafos,on,on_f,boxes,boxes_f,cities,trucks,tries,box,vars,vars2,count);

        nvars=trucks*tries+1;

        //epiloges gia tin anathesi timwn apo tin labelling
        options.type     = LABELING_INDOMAIN;
        options.indomain = LABELING_INDOMAIN_VALUE;
        options.solution = LABELING_FIRST_SOL;
        options.continue_var = 0;
        options.continue_all = 0;




        //labelling gia fortiga
        if (!c_labeling(vars, trucks*tries, METHOD_INPUT_ORDER,
&options))
        {
                printf("\nDen iparxei lisi\n");
        }
        //labelling gia koutia
        if (!c_labeling(vars2, box*tries, METHOD_INPUT_ORDER,
&options))
        {
                printf("\nDen iparxei lisi\n");
        }
```

```
                //ektipwsi apotelesmatwn
                c_domain_array_print(stdout, vars,  trucks*tries);
                printf("\n");
                c_domain_array_print(stdout, vars2, box*tries);
                printf("\n");
                printf("\nPrint boxes results\n");
                print_result(vars2,box*tries);
                printf("\nPrint trucks results\n");
                print_trucks(vars);


                if (c_chip_reset() != 1)
                {
                        printf("\nInitialize FAILEDDD\n");
                }


        }  //end for gia prospatheies

        printf("\nTotal time : %d ms\n", c_cputime()-start);
        c_chip_end();
        return 0;
}
```

# Παράρτημα Γ

```
//transfer2

#include <stdio.h>
```

```
#include "chipc.h"
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <ctype.h>

//global variables : the domain variable vars is for trucks and vars2 is for boxes
DvarPtr *vars,*vars2,*boxes,*boxes_f;
int *if_in,*final_state;
int trucks,box,maxtries,cities,temp,tries,i,j,val,**grafos,*cities_box,
*cities_box_f ,start,curr_truck;
int preval=0;
int count=0;
int fail=0;
int k=0;
int k2=0;
int pos=0;
int pos2=0;
int pos3=0;

//einai ena struct sto opoio filagoume tis parametrous pou tha xreiastoun oi sinartiseis
load, unload , stay
typedef struct argum
{
        int *curr_box;
        int *curr_truck;

}ARGUM;

ARGUM *param;

//sinartisi pou arxikopoia tis times tou pinaka preval(proigoumenes times twn koutiwn)
init_preval(int *preval){
```

```
	for(i=0; i<trucks*tries; i++){
		preval[i]=0;

	}
}



//sinartisi pou elegxei an dio pinakes einai isoi. An nai tote epestrefei 1 alliws 0
int equals(int *final, DvarPtr *boxes_f,int box)
{
	int value,i;
	value=1;
	for(i=0; i<box; i++){
		if(c_domain_value(boxes_f[i],&val)==0)
			printf("get value FAILLLL\n");
		if(final[i]!=val){
			value=0;
		}
	}
	return value;
}



//sinartisi pou ektipwnei tis theseis tou pinaka vars(times twn polewn pou exoun paei
ta fortiga)
void print_trucks(DvarPtr *vars){
	int *preval;
	 preval= (int *)malloc(sizeof(int)*trucks);
	for(i=0; i<trucks*tries; i++){
		if(i%(trucks)==0 && i!=0){
			printf("\n Allagi prospatheias\n");
```

```
                }

                c_domain_value(vars[i],&val);
                if(i<trucks){
                        printf("to fortigo %d brisketai stin
poli %d\n",((i)%trucks)+1,val);
                }
                else{
                        if(preval[(i)%trucks]==val){
                                printf("to fortigo %d menei stin
poli %d\n",((i)%trucks)+1,val);
                        }
                        else{

                                printf("to fortigo %d pigainei stin
poli %d\n",((i)%trucks)+1,val);
                        }
                }
                preval[i%box]=val;
        }


        init_preval(preval);
}

//sinartisi pou ektipwnei tis theseis tou pinaka vars2(polewn pou briskontai ta fortia),
//episis elegxei an stin teliki katastasi ta fortia briskontai stis poleis pou prepei, an nai
//tote termatizei to programma, an oxi tote sinexizete i ektelesei afksanontas tis
prospatheies
int print_result(DvarPtr *vars2, int size){
        int *preval;
         preval= (int *)malloc(sizeof(int)*box);
        for(i=0; i<box*tries; i++){
```

```
                if(i%(box)==0 && i!=0){
                        printf("\n Allagi prospatheias\n");
                }
                c_domain_value(vars2[i],&val);
                if(val<cities+1){
                        if(preval[(i)%box]<cities){
                                printf("to kouti %d brisketai stin
poli %d\n",((i)%box)+1,val);
                        }
                        else{
                                printf("to kouti %d ksefortwnetai stin
poli %d\n",((i)%box)+1,val);
                        }
                }
                else{
                        printf("to kouti %d fortwnetai apo to
fortigo %d\n",((i)%box)+1,(val-cities));
                }
                preval[i%box]=val;

        }

        //elegxos oti exei epitixei
        for(i=0;i<box;i++){
                c_domain_value(vars2[box*(tries-1)+i],&val);
                final_state[i]=val;


        }
        if(equals(final_state,boxes_f,box)==1){
                printf("SUCCEEEEEEEED\n");
                exit(-1);
        }
```

```
        else{
                printf("No solution for this try\n");

        }

        init_preval(preval);
        return 1;
}

//sinartisi pou ekteleite an den ginei load kapoiou koutiou apo fortigo
int fin(int pos){
        //an den ginei load tote ta fortiga stin epomeni prospatheia tha pane se thesi
pou iparxei kouti
        //to programma leitourga kai xwris afton ton periorismo alla einai mia
beltistopoiisi wste na min kanoume axreiastes metakiniseis(grigoroteri ektelesi)
        c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars2[param-
>curr_box[pos]], 0 );


return 1;
}


//sinartisi pou ekteleite an den ginei unload
int fin2(int pos2){


        //DO NOTHING

return 1;
}
```

```
//sinartisi pou ekteleite an den ektelesti i sinartisi stay
int fin3(int pos3){

	//DO NOTHING

return 1;
}

//sinartisi stin opoia ta koutia den exoun fwrtwthei apo kanena fortigo ara kai tha meinoun stin  thesi pou itan
int stay(int pos3){

	//to kouti stin epomeni prospatheia na meinei ekei pou itan
	c_dom_domcst(vars2[pos3], EQUAL,vars2[pos3+box], 0 );

return 1;
}

//sinartisi pou ksefwrtwnei ta kooutia stin thesi pou brisketai to fortigo pou ta exei fortwsei
int unload(int pos2){

	//to kouti stin epomeni prospatheia na parei tin timi pou tha exei  kai to fortigo stin epomeni prospatheia
	c_dom_domcst(vars2[param->curr_box[pos2]+box], EQUAL,vars[param->curr_truck[pos2]+trucks], 0);


return 1;
}

//sinartisei pou fortwnei ta koutia sta fortiga
int load(int pos){
```

```
        //to kouti stin epomeni prospatheia tha parei tin timi pou fortigou+poleis pou dilwnei oti einai fortwmeno
        c_dom_domcst(vars2[param->curr_box[pos]+box], EQUAL,NULL, (param->curr_truck[pos]%(trucks)+1+cities));

        //to fortigo stin epomeni prospatheia tha meinei stin thesi pou itan gia na epitefthei to fortwma
        c_dom_domcst(vars[param->curr_truck[pos]+trucks], EQUAL,vars[param->curr_truck[pos]],0);

return 1;
}



//tipoma tou pinaka on pou deilwnei pou brisketai kathe fortigo stin arxiki katastasi
void print_on(int trucks,int *on){
        int i;
        for (i=0; i<trucks; i++){
        printf("to fotigo %d brisketai stin poli %d\n",i+1,on[i]);
        }
}

//tipwma tou pinaka on_f pou deilwnei pou tha brisketai to fortigo stin teliki katastasi
void print_on_f(int trucks,int *on){
        int i;
        for (i=0; i<trucks; i++){
        printf("to fotigo %d tha brisketai stin poli %d\n",i+1,on[i]);
        }
}

//tipoma tou pinaka boxes pou deilwnei pou briskontai ta koutia stin arxiki katastasi
```

```
void print_boxes(int box,int *boxes){
        int i;
        for (i=0; i<box; i++){
        printf(" to fortio %d brisketai stin poli %d\n",i+1,boxes[i]);
        }
}

//tipwma tou pinaka boxes_f pou deilwnei pou tha briskontai ta koutia stin teliki
katastasi
void print_boxes_f(int box,int *boxes_f){
        int i;
        for (i=0; i<box; i++){
        printf("to fortio %d tha brisketai stin poli %d\n",i+1,boxes_f[i]);
        }
}

//sinartisi anethesi periorismwn stous pinakes boxes, boxes_f
void boxes_cons(DvarPtr *boxes, DvarPtr *boxes_f){
        int i;
        //na iparxei toulaxisto cities_box_f[i] fores i timi i+1
        for(i=0; i<cities; i++){
                c_atleast(cities_box_f[i],boxes_f,box,i+1);
        }
        for(i=0; i<cities; i++){

                c_atleast(cities_box[i],boxes,box,i+1);
        }
}


//sinartisi anatheseis periorismwn
void cons(int **grafos,int *on,int *on_f,DvarPtr *boxes, DvarPtr *boxes_f,int
cities,int trucks, int tries,int box,
```

```
                                DvarPtr *vars,DvarPtr *vars2, int count)
{
        int k=0;
        void *arg,*arg2;
        arg=(void*)vars;
        arg2=(void*)vars2;


        //i prwti grammi tou pinaka vars2 einai isi me to arxiki thesi fortiwn
        for(i=0;i<box;i++){
                if(c_domain_value(boxes[i],&val)==FAIL)
                        printf("get value faillll\n");
                c_dom_domcst(vars2[i], EQUAL,NULL, val);
        }

        //i prwti grammi tou pinaka vars einai isi me to arxiki thesi fortigwn
        for(i=0;i<trucks;i++){
                c_dom_domcst(vars[i], EQUAL,NULL, on[i]);

        }


        //LOADDDDDDDD
        for(i=0; i<(box*tries)-box; i++){    //gia kathe kouti
                if(i%(box)==0 && i!=0){
                                k++;
                        }
                for(j=0;j<trucks;j++,pos++){   //kai gia kathe fortigo
                        //curr_truck=j;
                        param->curr_box[pos]=i;    //apothikefsi timwn sto struct
param->curr_box
                        param->curr_truck[pos]=j+trucks*k;   //apothikefsi timwn sto
struct param->curr_truck
```

```
                        //printf("FORRR curr_box:%d - curr_truck:%d,
pos:%d\n",i,j+trucks*k,pos);
                        //an gia kapoio kouti pou brisketai stin current grammi iparxei
kapoio fortigo pou mporei na to fwrtwsei
                        //(na briskontai dil stin idia poli) tote kaleitai i sinartisi load...
                        c_if(vars2[i], EQUAL,vars[j+trucks*k] ,0, load, pos , fin,pos );
                }
        }


        //UNLOADDDDDDDDD
        for(i=0; i<(box*tries)-box; i++){    //gia kathe kouti
                if(i%(box)==0 && i!=0){
                                k2++;
                }
                for(j=0;j<trucks;j++,pos2++){  //kai gia kathe fortigo
                        //curr_truck=j;
                        param->curr_box[pos2]=i;  //apothikefsi timwn sto struct
param->curr_box
                        param->curr_truck[pos2]=j+trucks*k2;     //apothikefsi timwn
sto struct param->curr_truck
                        //printf("FORRR2222222222 curr_box:%d - curr_truck:%d,
pos2:%d\n",i,j+trucks*k2,pos2);
                        //an kapoio kouti einai fortwmeno se ena fortigo tote to current
kouti mporei na ginei unload
                        c_if(vars2[i], EQUAL,vars[j+trucks*k2] ,cities, unload, pos2 ,
fin2,pos2 );
                }
        }



        //STAY
```

```
	for(i=0; i<(box*tries)-box; i++){   //gia ola ta koutia
			//an o kwdikas proxwrisei kai kapoio kouti den exei ginei oute load oute unload tote prepei na minei stin thesi pou itan
			//alliws o pio katw periorismos tha kanei FAIL
			pos3=i;
			c_if_in(vars2[i],if_in,trucks, stay, pos3 , fin3,pos3);
	}




	//I teleftaia thesi einai isi me tin teliki thesi ton fortiwn
	for(i=0;i<box;i++){
		c_domain_value(boxes_f[i],&val);
		c_dom_domcst(vars2[box*(tries-1)+i], EQUAL,NULL, val);
	}

	for(i=0;i<box;i++){
		//beltistopoiisi etsi wste ta fortiga na prospathoun na perasoun apo tis telikes theseis twn koutiwn
		c_dom_domcst(vars[trucks*tries-box+i], EQUAL,vars2[box*tries-box+i],0);
	}


}


int main(){
	//dilwsei metablitwn tis main()
	int nvars;
	int i,j,x,y;
	char ans;
```

```
        int *on,*on_f;

    Labeling  options;

        //xronos ekteleseis programmatos
        start = c_cputime();

        //arxikopoiisi chipc
        if (c_chip_init() != 1)
        {
                return 0;
        }

        //get user input
        printf("Kalosorisate sto programa metaforas fortiwn\n");
        printf("dwse ton arithmo twn polewn pou tha iparxoun:");
        scanf("%d",&cities);
        printf("dwse ton arithmo twn fortigwn pou iparxoun:");
        scanf("%d",&trucks);
        printf("dwse ton arithmo twn fortiwn pou iparxoun:");
        scanf("%d",&box);
        printf("dwse ton arithmo twn megistwn prospathiwn pou tha ginoun mexri na
brethei lisi:");
        scanf("%d",&maxtries);

        //arxikopoiisi prospatheiwn se 2
        tries=2;
        //arxikopoiisi megistwn prospatheiwn se +2
        maxtries=maxtries+2;

        //desmefsi xwrou gia pinakes
        if_in=(int *)malloc(sizeof(int)*trucks);
        final_state=(int *)malloc(sizeof(int)*box);
```

```
on= (int *)malloc(sizeof(int)*trucks);
        on_f= (int *)malloc(sizeof(int)*trucks);
boxes= (void **)malloc(sizeof(int)*box);
        boxes_f= (void **)malloc(sizeof(int)*box);
        cities_box= (int *)malloc(sizeof(int)*cities);
        cities_box_f= (int *)malloc(sizeof(int)*cities);


        //dimiourgia grafou
        grafos = (int **) malloc(sizeof(int*)*cities);
        for(i = 0 ; i < cities; i++)
        {
        grafos[i] = (int *) malloc (sizeof(int)* cities);
        }

        for(i=0;i<trucks;i++){
                if_in[i]=i+1;
        }

        //user input gia pinaka on
        printf("dwse tin poli pou brisketai to kathe fortigo\n");
                for(i=0; i<trucks; i++){
                        printf("truck %d:",i+1);
                        scanf("%d",&on[i]);

                        while (on[i]<=0 || on[i]>cities){
                                printf("Den iparxei tetoia poli, ksanadwse arithmo:");
                                scanf("%d",&on[i]);
                        }
                }
                //tipoma pinaka on
                print_on(trucks,on);
```

```
		//user input gia pinaka boxes
		printf("dwse ton arithmo twn koutiwn pou tha exei i kathe poli\n");
		for(i=0; i<cities; i++){
			printf("poli %d:",i+1);
			scanf("%d",&cities_box[i]);

			while (cities_box[i]<0 || cities_box[i]>box){
				printf("Den iparxoun tosa koutia, ksanadwse arithmo:");
				scanf("%d",&cities_box[i]);
			}
		}

		//user input gia pinaka boxes_f
		printf("dwse ton arithmo twn fortiwn pou tha exei i kathe poli stin
teliki katastasi\n");
		for(i=0; i<cities; i++){
			printf("fortio %d:",i+1);
			scanf("%d",&cities_box_f[i]);

			while (cities_box_f[i]<0 || cities_box_f[i]>box){
				printf("Den iparxoun tosa koutia, ksanadwse arithmo:");
				scanf("%d",&cities_box_f[i]);
			}
		}




	for(tries=2; tries<maxtries; tries++, pos=0, pos2=0,k=0,
k2=0,pos3=0){	//for gia ektelesei tou kwdika me diaforetiki timi prospatheiwn

		//desmefsi xwrou gia domain variables
		vars =  (void**)realloc(vars,sizeof(int)*(trucks*tries));
```

```
                vars2 = (void**)realloc(vars2,sizeof(int)*(box*tries));

                //desmefsi xwrou gia struct
                param =(ARGUM *)realloc(param,sizeof(ARGUM));
                param->curr_box= (int *) realloc(param-
>curr_box,sizeof(int)*(box*tries*trucks));
                param->curr_truck=(int*) realloc(param-
>curr_truck,sizeof(int)*(box*tries*trucks));

                //dimiourgeia pediou timwn
                c_create_domain_array(vars, trucks*tries, 1, cities, CONSEC);
                c_create_domain_array(vars2, box*tries, 1, cities+trucks, CONSEC);
                c_create_domain_array(boxes_f, box, 1, cities+trucks, CONSEC);
                c_create_domain_array(boxes, box, 1, cities+trucks, CONSEC);

                //anathesi periorismwn
                boxes_cons(boxes,boxes_f);


                //epiloges gia tin anathesi timwn apo tin labelling
                options.type     = LABELING_INDOMAIN;
                options.indomain = LABELING_INDOMAIN_VALUE;
                options.solution = LABELING_FIRST_SOL;
                options.continue_var = 0;
                options.continue_all = 0;


                if (!c_labeling(boxes, box, METHOD_INPUT_ORDER, &options))
                {
                        printf("\nDen iparxei lisi\n");
                }
                if (!c_labeling(boxes_f, box, METHOD_INPUT_ORDER, &options))
                {
```

```
                printf("\nDen iparxei lisi\n");
        }

        //kalesma sinartisis gia eisagwgi periorismwn
        printf("Setting Cons\n");

    cons(grafos,on,on_f,boxes,boxes_f,cities,trucks,tries,box,vars,vars2,count);

        nvars=trucks*tries+1;



        //labelling gia fortiga
        if (!c_labeling(vars, trucks*tries, METHOD_INPUT_ORDER,
&options))
        {
                printf("\nDen iparxei lisi\n");
        }
        //labelling gia koutia
        if (!c_labeling(vars2, box*tries, METHOD_INPUT_ORDER,
&options))
        {
                printf("\nDen iparxei lisi\n");
        }



        //ektipwsi apotelesmatwn
        //c_domain_array_print(stdout, boxes_f,  box);
        //printf("\n");
        c_domain_array_print(stdout, vars,  trucks*tries);
        printf("\n");
        c_domain_array_print(stdout, vars2, box*tries);
```

```
                printf("\n");
                printf("\nPrint boxes results\n");
                print_result(vars2,box*tries);
                printf("\nPrint trucks results\n");
                print_trucks(vars);


                if (c_chip_reset() != 1)
                {
                        printf("\nInitialize FAILEDDD\n");
                }


        }   //end for gia prospatheies

        printf("\nTotal time : %d ms\n", c_cputime()-start);
        c_chip_end();
        return 0;
}
```