ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΙΕΡΑΡΧΙΚΟΙ ΠΙΝΑΚΕΣ ΚΑΙ Η ΕΦΑΡΜΟΓΗ ΤΟΥΣ ΣΤΗΝ ΠΡΟΣΟΜΟΙΩΣΗ ΠΥΚΝΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΜΟΙΒΑΙΩΝ ΕΠΑΓΩΓΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΤΡΙΓΓΛΗ ΚΩΝΣΤΑΝΤΙΝΟΥ

Βόλος Σεπτέμβριος 2015

Αυτή η σελίδα είναι σκόπιμα κενή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ

ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΙΕΡΑΡΧΙΚΟΙ ΠΙΝΑΚΕΣ ΚΑΙ Η ΕΦΑΡΜΟΓΗ ΤΟΥΣ ΣΤΗΝ ΠΡΟΣΟΜΟΙΩΣΗ ΠΥΚΝΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΜΟΙΒΑΙΩΝ ΕΠΑΓΩΓΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΤΡΙΓΓΛΗ ΚΩΝΣΤΑΝΤΙΝΟΥ

Επιβλέπων: Νέστορας Ευμορφόπουλος

Λέκτορας Καθηγητής Η.Μ.Μ.Υ

(Υπογραφή)

………………..

Νέστορας Ευμορφόπουλος

Λέκτορας Καθηγητής Η.Μ.Μ.Υ.

(Υπογραφή)

………………..

Γεώργιος Σταμούλης

Καθηγητής Η.Μ.Μ.Υ.

(Υπογραφή)

………………..

ΣΤΡΙΓΓΛΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

© 2015 – All rights reserved

Περιεχόμενα

Κεφάλαιο 1 .... **5**

Εισαγωγή στην Προσομοίωση Κυκλωμάτων **5**

1.1 Ανάλυση κυκλωμάτων **5**

1.2 Μεταβατική Ανάλυση (transient analysis) **6**

Κεφάλαιο 2 .... **8**

Μέθοδοι Επίλυσης Γραμμικών Συστημάτων **8**

2.1 Μέθοδοι επίλυσης συστημάτων **8**

2.2 Conjugate Gradients (CG) **8**

2.3 Προβλήματα των μεθόδων επίλυσης **10**

Κεφάλαιο 3 .... **11**

Κατασκευή και Βασική Ορολογία **11**

3.1 Ιεραρχικοί Πίνακες **11**

3.2 Διαδικασία Κατασκευής Ιεραρχικού Πίνακα **11**

3.3 Admissibility Condition **14**

3.4 Singular Value Decomposition **15**

Κεφάλαιο 4 .... **16**

Ανάπτυξη και Λειτουργία του Προγράμματος **16**

4.1 Αρχεία Κώδικα **16**

4.2 Δομές του προγράμματος **16**

4.2.1 Fullmatrix **17**

4.2.2 Rkmatrix **17**

4.2.3 Stuct List **18**

4.3 Παράδειγμα Τρεξίματος **19**

4.4 Πολλαπλασιασμός με διάνυσμα **25**

Κεφάλαιο 5 .... **27**

Κώδικας και Πηγές **27**

5.1 Κώδικας **27**

5.2 Πηγές **34**

Κεφάλαιο 1

Εισαγωγή στην Προσομοίωση Κυκλωμάτων

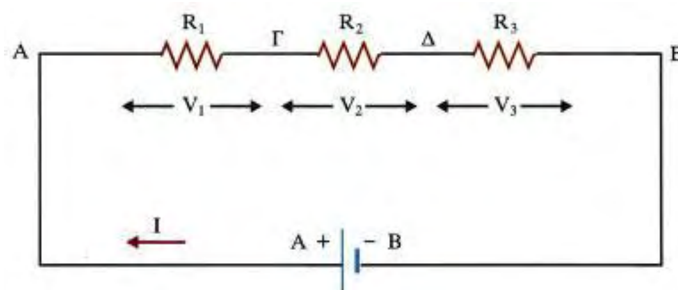
1.1 Ανάλυση κυκλωμάτων

Η διείσδυση των ηλεκτρονικών διατάξεων και οργάνων, σε πολλά επίπεδα της καθημερινής ζωής, είναι μια από τις συνέπειες της ηλεκτρονικής επανάστασης που χαρακτηρίζει το δεύτερο μισό του $20^{ου}$ αιώνα, και συνεχίζεται ακόμα και στις αρχές του $21^{ου}$. Η κατασκευή όλων αυτών των διατάξεων, βασίζεται στην ανάπτυξη κυκλωμάτων διάφορων μορφών που εξυπηρετούν τους σκοπούς δημιουργίας των εκάστοτε συσκευών. Στη σημερινή εποχή, η ανάπτυξη κυκλωμάτων για οποιαδήποτε χρήση, έχει 2 κεντρικούς στόχους:

1. Την επίτευξη της μεγίστης δυνατής απόδοσης
2. Την ελαχιστοποίηση της κατανάλωσης ενέργειας

Για να σχεδιαστεί και να αναλυθεί ένα κύκλωμα και εν τέλει να βελτιστοποιηθεί η απόδοσή του, απαιτείται η μεταφορά του στον κόσμο των μαθηματικών, όπου είναι πιο εύκολη η ανάλυση των διάφορων ιδιοτήτων του. Τα "εργαλεία" που πραγματοποιούν αυτή τη σύνδεση ενός φυσικού κυκλώματος με τα μαθηματικά, είναι οι μέθοδοι ανάλυσης κυκλωμάτων, όπως π.χ. η μέθοδος κόμβων και η μέθοδος βρόχων που βασίζονται στον κανόνα του Kirchhoff.

Οι μέθοδοι αυτές αντιστοιχίζουν το προς ανάλυση κύκλωμα, σε ένα πίνακα, το μέγεθος του οποίου εξαρτάται από το πλήθος των κόμβων του, και οι ιδιότητές του από τη συνδεσμολογία των στοιχείων του και το είδος τους. Ένα κύκλωμα όπως αυτό της Εικόνας 1 για παράδειγμα, θα αντιστοιχηθεί σε ένα πίνακα μεγέθους 3Χ3 , αφού έχει 3 κόμβους.



Εικόνα 1

Από έναν τέτοιο πίνακα, προκύπτει ένα σύστημα της μορφής Ax=b, από το οποίο μπορούν να εξαχθούν μεγέθη όπως η τάση και το ρεύμα που διατρέχει κάθε κόμβο. Για να λυθεί αυτό το σύστημα, απαιτείται να βρεθεί ο αντίστροφος του A, δηλαδή ο A^{-1}. Η διαδικασία εύρεσης του αντίστροφου ενός πίνακα, είναι αρκετά επίπονη, χρονοβόρα και με πολλές πράξεις. Για το παράδειγμα της Εικόνας 1 είναι αρκετά εύκολο και γρήγορο να βρούμε τον A^{-1} γιατί είναι αρκετά μικρός.

Η πρόοδος όμως της τεχνολογίας που ακόμα και σήμερα συνεχίζει να επιβεβαιώνει το νόμο του Moore, έχει οδηγήσει σε πολύ υψηλής πολυπλοκότητας κυκλώματα όπως αυτό της Εικόνας 2. Εδώ γίνεται εύκολα αντιληπτό ότι η εύρεση του A^{-1} δεν είναι καθόλου απλή υπόθεση.

Εικόνα 2:

ΜΗΤΡΙΚΗ ASROCK X99 WS-E RETAIL

1.2 Μεταβατική Ανάλυση (transient analysis)

Η ανάλυση κυκλωμάτων που περιέχουν αμοιβαίες επαγωγές (RLC), είναι πιο σύνθετη γιατί εισάγεται και η παράμετρος του χρόνου ώστε να μελετηθεί η απόκριση του συστήματος σε μια διέγερση. Στη μεταβατική ανάλυση, το σύστημα MNA ενός γραμμικού κυκλώματος RLC που περιέχει και ανεξάρτητες πηγές τάσης και ρεύματος, είναι το εξής (θεωρώντας ότι δεν υπάρχει αντίσταση στην ομάδα G_2):

$$\underbrace{\begin{bmatrix} A_1 G A_1^T & A_2 \\ A_2^T & 0 \end{bmatrix}}_{\tilde{G}} \underbrace{\begin{bmatrix} \underline{v}(t) \\ \underline{i}(t) \end{bmatrix}}_{\underline{x}(t)} + \underbrace{\begin{bmatrix} A_1 C A_1^T & 0 \\ 0 & -L \end{bmatrix}}_{\tilde{C}} \underbrace{\begin{bmatrix} \frac{d\underline{v}(t)}{dt} \\ \frac{d\underline{i}(t)}{dt} \end{bmatrix}}_{\frac{d\underline{x}(t)}{dt}} = \underbrace{\begin{bmatrix} -A_1 S_1(t) \\ S_2(t) \end{bmatrix}}_{\underline{e}(t)}$$

Όπου

- G ο πίνακας αγωγιμοτήτων των κλάδων
- C ο πίνακας χωρητικοτήτων των κλάδων (διαγώνιος)
- L ο πίνακας αυτεπαγωγών των κλάδων (στη διαγώνιο) και αμοιβαίων επαγωγών μεταξύ κλάδων (εκτός διαγωνίου)

Το παραπάνω είναι ένα **σύστημα γραμμικών διαφορικών εξισώσεων 1ης τάξης με σταθερούς συντελεστές**, της μορφής:

$$\tilde{G}\underline{x}(t) + \tilde{C}\,\frac{d\underline{x}(t)}{dt} = \underline{e}(t) \qquad (1)$$

Αν καθοριστούν οι τιμές του $\underline{x}(t)$σε μια αρχική χρονική στιγμή t_0 ως $\underline{x}(t_0) = \underline{x}_0$ (αρχική συνθήκη), τότε το πρόβλημα

$$\begin{cases} \tilde{G}\underline{x}(t) + \tilde{C}\,\dfrac{d\underline{x}(t)}{dt} = \underline{e}(t) \\ \underline{x}(t_0) = \underline{x}_0 \end{cases}$$

ονομάζεται **πρόβλημα αρχικών τιμών (Initial Value Problem ή IVP)** και υπό ορισμένες προϋποθέσεις έχει μοναδική λύση $\underline{x}(t)$ σε ένα διάστημα $[t_0 , t_f]$.

Η λύση ενός IVP υπολογίζεται αριθμητικά χωρίζοντας το διάστημα $[t_0 , t_f]$ σε έναν αριθμό διακριτών χρόνων και υπολογίζοντας μια προσέγγιση $\underline{x}(t_k)$, σε κάθε χρονική στιγμή t_k, ξεκινώντας από την αρχική συνθήκη $\underline{x}(t_0) = \underline{x}_0$ και οδεύοντας διαδοχικά προς μεγαλύτερους χρόνους. Η τιμή $h_k = t_k - t_{k-1}$ ονομάζεται χρονικό βήμα ή βήμα δειγματοληψίας. Αν τα χρονικά σημεία επιλεγούν ισαπέχοντα τότε το χρονικό βήμα είναι σταθερό.

Ο υπολογισμός της προσέγγισης $x(t_k)$ για κάθε χρονική στιγμή, γίνεται μέσω της προσέγγισης **Backward Euler**

$$\frac{d\underline{x}(t_k)}{dt} \approx \frac{1}{h}\,[\underline{x}(t_k) - \underline{x}(t_{k-1})]$$

Το διαφορικό σύστημα (1) τότε μετατρέπεται στο ακόλουθο αλγεβρικό γραμμικό σύστημα:

$$\left(\tilde{G} + \frac{1}{h}\tilde{C}\right)\underline{x}(t_k) = \underline{e}(t_k) + \frac{1}{h}\tilde{C}\,\underline{x}(t_{k-1}) \;, k = 1,2, \dots , m$$

όπου e είναι το διάνυσμα διέγερσης των πηγών.

Κεφάλαιο 2

Μέθοδοι Επίλυσης Γραμμικών Συστημάτων

2.1 Μέθοδοι επίλυσης συστημάτων

Για τον υπολογισμό του αντιστρόφου και γενικότερα τη λύση του συστήματος Ax=b, έχουν αναπτυχθεί διάφορες μέθοδοι, όπως είναι η παραγοντοποίηση Cholesky και η ανάλυση LU. Η μεν ανάλυση LU είναι η κλασική μέθοδος επίλυσης και είναι brute force μέθοδος που δεν εκμεταλλεύεται τις διάφορες ιδιότητες του πίνακα, ενώ η Cholesky είναι μια καλή μέθοδος από άποψη απόδοσης όταν πρόκειται για αραιούς πίνακες.

Επίσης έχουν αναπτυχθεί επαναληπτικές μέθοδοι, όπως Gauss-Seidel, Conjugate Gradients(CG) και Jacobi, οι οποίες δεν υπολογίζουν ακριβώς το αποτέλεσμα, αλλά το προσεγγίζουν (με διαφορετική εστίαση η κάθε μια) με πολύ ικανοποιητική ακρίβεια και αρκετά μικρό σφάλμα, στα όρια του αμελητέου. Η φιλοσοφία των μεθόδων αυτών είναι να δημιουργούν ολοένα και καλύτερες προσεγγίσεις της λύσης, μέχρι την ικανοποίηση ενός κριτηρίου σύγκλισης. Συνήθως για να δουλέψει μια τέτοια μέθοδος, απαιτεί ο πίνακας A να είναι συμμετρικός και θετικά ορισμένος

2.2 Conjugate Gradients (CG)

Από τη μεταβατική ανάλυση πηγάζουν και οι επαναληπτικές μέθοδοι επίλυσης γραμμικών συστημάτων που έχουν αναπτυχθεί και οι οποίες στοχεύουν στην εκμετάλλευση κάποιων ειδικών ιδιοτήτων των πινάκων που προκύπτουν από την ανάλυση, ώστε να επιταχύνουν τη διαδικασία επίλυσης.

Για να γίνει πιο κατανοητό το πόσο κρίσιμη είναι η εύκολη πρόσβαση στα στοιχεία του πίνακα εισόδου, κρίνεται σκόπιμο να αναφερθεί ενδεικτικά και εν συντομία, η μέθοδος CG.

Η μέθοδος αυτή, όπως έχει ήδη αναφερθεί, είναι επαναληπτική και προϋποθέτει ο πίνακας να είναι συμμετρικός και θετικά ορισμένος. Σκοπός της μέσα από διαδοχικές επαναλήψεις, είναι να ελαχιστοποιήσει την εξίσωση:

$$f(\underline{x}) = \frac{1}{2}\underline{x}^T A\underline{x} - \underline{b}^T \underline{x}$$

Ξεκινώντας από μια αρχική προσέγγιση της λύσης $x^{(0)}$ (συνήθως αυθαίρετη), η μέθοδος αυτή, δημιουργεί διαδοχικές προσεγγίσεις $x^{(m)}$ της λύσης, χρησιμοποιώντας σε κάθε επανάληψη την προσέγγιση που βρέθηκε στην προηγούμενη επανάληψη. Ο αλγόριθμος ολοκληρώνεται όταν επιτευχθεί σύγκλιση καλύτερη από ένα όριο itol που έχει οριστεί, ή όταν ολοκληρωθούν n επαναλήψεις, όπου n η τάξη του πίνακα. Η σύγκλιση όμως στις

περισσότερες των περιπτώσεων επιτυγχάνεται αρκετά γρήγορα και δεν απαιτούνται όλες οι επαναλήψεις.

Κάθε επανάληψη έχει κόστος $O(n^2)$ το οποίο είναι αρκετά υψηλό αλλά λόγω της φύσης του αλγορίθμου δεν μπορεί να μειωθεί. Στο χρόνο εκτέλεσης αυτό όμως προστίθεται και ένα πολύ υψηλό κόστος αναζήτησης στοιχείων στο δίσκο λόγω page faults. Έτσι ο χρόνος εκτέλεσης πολύ εύκολα μπορεί να ξεφύγει από τα αποδεκτά όρια. Είναι επομένως φανερή πλέον η επιτακτική ανάγκη για εύκολη πρόσβαση στα στοιχεία του πίνακα και ένας εύκολος και γρήγορος τρόπος πολλαπλασιασμού πίνακα με διάνυσμα και ο περιορισμός των page faults καθώς είναι ο μόνος τρόπος βελτιστοποίησης του χρόνου εκτέλεσης.

Παρακάτω δίνεται ο κώδικας της διαδικασίας σε MATLAB

```
function [x,numIter] = conjGrad(func,x,b,epsilon)
% Solves Ax = b by conjugate gradient method.
% USAGE: [x,numIter] = conjGrad(func,x,b,epsilon)
% INPUT:
% func    = handle of function that returns the vector A*v
% x       = starting solution vector
% b       = constant vector in A*x = b
% epsilon = error tolerance (default = 1.0e-9)
% OUTPUT:
% x       = solution vector
% numIter = number of iterations carried out

if nargin == 3; epsilon = 1.0e-9; end
n = length(b);
r = b - feval(func,x); s = r;
for numIter = 1:n
    u = feval(func,s);
    alpha = dot(s,r)/dot(s,u);
    x = x + alpha*s;
    r = b - feval(func,x);
    if sqrt(dot(r,r)) < epsilon
        return
    else
        beta = -dot(r,u)/dot(s,u);
        s = r + beta*s;
    end
end
error('Too many iterations')
```

2.3 *Προβλήματα των μεθόδων επίλυσης*

Η ανάλυση Cholesky όπως αναφέρθηκε ήδη, είναι καλή μέθοδος για αραιά συστήματα. Αραιά συστήματα συνήθως προκύπτουν όταν ένα κύκλωμα αποτελείται από συμβατικά κυκλωματικά στοιχεία, όπως αντιστάσεις, ιδανικές πηγές κτλ. Αν στο κύκλωμα όμως υπάρχουν και αμοιβαίες επαγωγές, τότε αυτόματα το σύστημα γίνεται πυκνό και η μέθοδος αυτή παύει να είναι αποδοτική.

Επίσης, λόγω της φύσης της ανάλυσης, οι πίνακες είναι συνήθως συμμετρικοί και θετικά ορισμένοι, όπως απαιτούν οι επαναληπτικές μέθοδοι. Τα μεγέθη όμως και η πολυπλοκότητα των σύγχρονων κυκλωμάτων, έχουν οδηγήσει ακόμα και αυτές τις μεθόδους στα όρια της αποδοτικότητας τους καθώς όσο πιο πολύπλοκο είναι ένα κύκλωμα τόσο μεγαλύτερος προκύπτει ο πινάκας A. Έτσι τίθεται σημαντικός περιορισμός όσον αφορά το μέγεθος του προβλήματος που μπορούν να λύσουν και έχει άμεση εξάρτηση από τους διαθέσιμους πόρους (μνήμη κατά κύριο λόγο). Ενδεικτικά να αναφερθεί ότι το δίκτυο τροφοδοσίας μόνο, ενός τυπικού μικροεπεξεργαστή στα 0.18 micron, είναι της τάξης έως και των 100.000.000 κόμβων. Με την πολυπλοκότητα ακόμα και των πιο ευφυών μεθόδων να είναι $O(n^2)$ή $O(n^3)$, γίνεται εύκολα κατανοητό το πόσο απαγορευτικοί γίνονται οι υπολογισμοί.

Οι πίνακες που προκύπτουν από προβλήματα τέτοιου μεγέθους, όταν αποθηκεύονται σε πλήρη μορφή, είναι πολύ μεγάλοι για να χωρέσουν στη RAM. Τα σύγχρονα λειτουργικά για να λύσουν αυτό το πρόβλημα όταν δεν τους επαρκεί η φυσική μνήμη, την επεκτείνουν αυτόματα χρησιμοποιώντας εικονική μνήμη από το δίσκο, που είναι όμως αρκετές τάξεις μεγέθους πιο αργός σε σχέση με τη RAM. Μια επαναληπτική μέθοδος, απαιτεί την πολλαπλή πρόσβαση σε διάφορα κομμάτια του πίνακα τα οποία μπορεί να βρίσκονται σε απομακρυσμένα σημεία της εικονικής μνήμης στο δίσκο αλλά μεταξύ τους στον πίνακα να είναι "γειτονικά", καθώς και πολλαπλές διατρέξεις του ίδιου πίνακα. Αυτή είναι σε αδρές γραμμές η αρχή της τοπικότητας (όταν γίνεται αίτηση για ένα κομμάτι της μνήμης τότε είναι πολύ πιθανό να γίνει αίτηση σύντομα και για γειτονικά του κομμάτια), η οποία ισχύει και στους αλγορίθμους επίλυσης που αναφέρθηκαν παραπάνω.

Κεφάλαιο 3

Κατασκευή και Βασική Ορολογία

3.1 Ιεραρχικοί Πίνακες

Οι ιεραρχικοί πίνακες πρόεκυψαν από την ανάγκη να λυθεί το παραπάνω πρόβλημα με ένα ενιαίο τρόπο ώστε να εκμεταλλευτούμε την αρχή της τοπικότητας. Η βασική ιδέα είναι να σπάσει ο αρχικός πίνακας σε μικρότερους υποπίνακες, οι οποίοι μπορούν να αποθηκευτούν είτε σε πλήρη μορφή (fullmatrices), είτε σε παραγοντοποιημένη μορφή (rkmatrices). Ο στόχος της διάσπασης είναι η εύρεση όσο το δυνατόν περισσότερων κομματιών του πίνακα που μπορούν να αναπαρασταθούν σε παραγοντοποιημένη μορφή, η με απλά λόγια, η δημιουργία όσο το δυνατόν περισσότερων rkmatrices. Η ιεραρχία (εξ ου και το όνομα) των υποπινάκων που προκύπτει κατά τη διάσπαση, αναπαριστάται γραφικά με ένα δέντρο, οι ιδιότητες και το βάθος του οποίου καθορίζονται από τη φύση και τις παραμέτρους του προβλήματος.

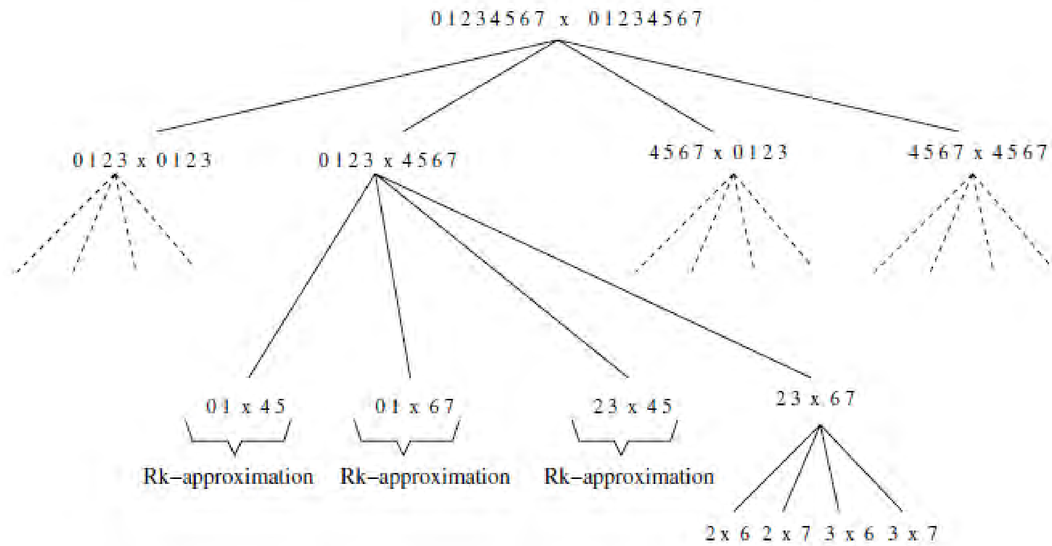
Στην ουσία, στόχος είναι να βοηθήσουμε το λειτουργικό, επιβάλλοντας τον τρόπο με τον οποίο θα χωριστεί ο πίνακας σε κομμάτια, δημιουργώντας μια αποδοτική δομή που θα κάνει εύκολη την αναζήτηση στοιχείων του πίνακα και θα περιορίζει κατά το δυνατόν τα page faults.

ΟΡΙΣΜΟΣ: *ένα block ενός πίνακα ονομάζεται* ***admissible****, αν με βάση κάποιο ορισμένο κριτήριο (admissibility condition), αποφασιστεί ότι μπορεί να αποθηκευτεί σε μορφή rkmatrix. Αλλιώς ονομάζεται* ***inadmissible****.*

Υπάρχουν πολλοί τρόποι και πολλά κριτήρια, από μαθηματικής άποψης με τους οποίους μπορεί ένας πίνακας να σπάσει σε μικρότερους, ανάλογα με τις ιδιότητές του και το ποιοι αλγόριθμοι θα τρέξουν πάνω σ' αυτόν.

3.2 Διαδικασία Κατασκευής Ιεραρχικού Πίνακα

Στα πλαίσια της εργασίας αυτής, αναπτύχθηκε ένα πρόγραμμα, το οποίο δέχεται ως είσοδο ένα μεγάλο πίνακα που προκύπτει από την ανάλυση ενός δικτύου τροφοδοσίας, και δημιουργεί έναν αντίστοιχο ιεραρχικό πίνακα ο οποίος κατόπιν χρησιμοποιείται για να πολλαπλασιαστεί με ένα διάνυσμα κατάλληλης διάστασης. Το πρόγραμμα δημιουργεί μια ιεραρχία αντίστοιχη ενός τετραδικού δέντρου (quad tree), κάθε κόμβος δηλαδή έχει 4 παιδιά, η οποία φαίνεται στην Εικόνα 3



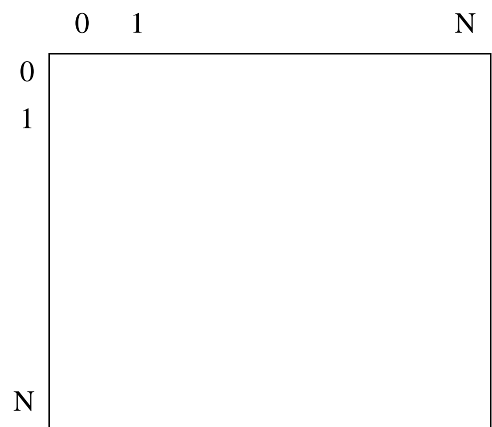
Εικόνα 3: Γραφική απεικόνιση ενός quad tree

Ένα χρήσιμο μέγεθος για τη λειτουργία του αλγορίθμου είναι το **domain**. Με τον όρο αυτό ουσιαστικά αντιστοιχίζεται το πλήθος γραμμών και στηλών του πίνακα (στο συγκεκριμένο πρόγραμμα είναι τα ίδια γιατί λειτουργεί για τετραγωνικούς πίνακες), στον αριθμό 1, έτσι ώστε με κάθε διάσπαση να προκύπτει ένα κλάσμα του αρχικού. Η αντιστοίχιση αυτή βοηθάει ώστε το σπάσιμο σε subdomains να μένει ανεπηρέαστο από το μέγεθος της εισόδου. Εφόσον έχουμε δομή quad tree, κάθε domain θα πρέπει να διασπάται σε 4 ίσα subdomains. Επομένως το μέγεθος κάθε subdomain που δημιουργείται, θα έχει το ½ του μεγέθους του αρχικού του κατά γραμμές και το ½ επίσης κατά στήλες. Στο παράδειγμα που ακολουθεί γίνεται πιο κατανοητή η έννοια του domain και του πως αυτό διαιρείται.

Έστω μια είσοδος μεγέθους NXN.

Βήμα 1:

Η διάσταση του συγκεκριμένου domain ορίζεται ως [0 , 1] x [0 , 1]



Βήμα 2:

Αν αυτό ικανοποιεί τις προϋποθέσεις διάσπασης και σπάσει σε 4 subdomains τότε αυτά θα έχουν διαστάσεις domain όπως φαίνονται στο σχήμα:

Subdomain(1) = [0 , ½] x [0 , ½]

Subdomain(2) = [0 , ½] x [½ , 1]

Subdomain(3) = [½ ,1] x [0 , ½]

Subdomain(4) = [½ ,1] x [½ , 1]

Κάθε subdomain εδώ έχει πραγματικές διαστάσεις N/2 x N/2

Βήμα 3:

Αν τώρα υποθέσουμε ότι το $1^{ο}$ subdomain του προηγούμενου βήματος πρέπει να διασπαστεί, τότε τα subdomains που προκύπτουν, όπως φαίνεται και στο σχήμα, έχουν διαστάσεις:

Subdomain(1.1) = [0 , ¼] x [0 , ¼]

Subdomain(1.2) = [0 , ¼] x [¼, ½]

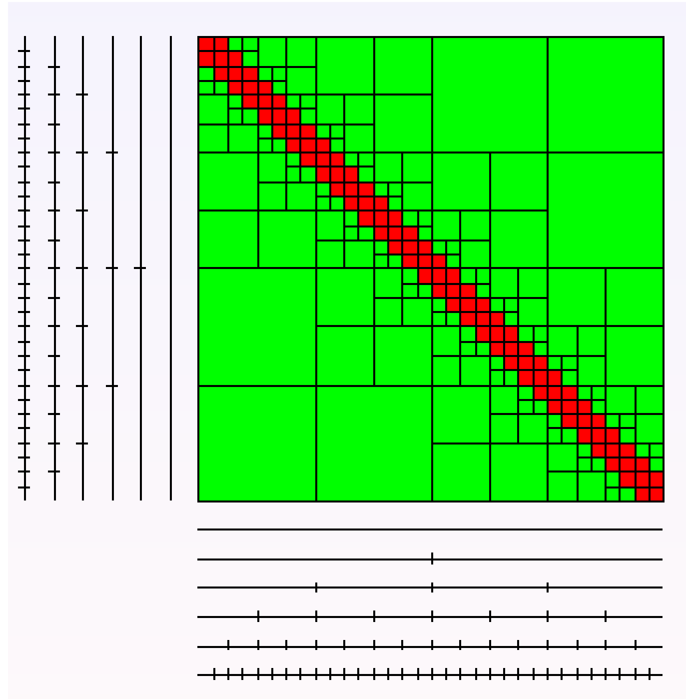
Subdomain(1.3) = [¼, ½] x [0 , ½]

Subdomain(1.4) = [¼, ½] x [¼, ½]

Και πραγματικές διαστάσεις N/4 x N/4

Η διάσπαση μπορεί να συνεχιστεί σε όλα τα κομμάτια μέχρι να φτάσουμε σε κάποιο μέγιστο όριο επιτρεπτής διάσπασης που ορίζεται από το πρόβλημα, ή αν φτάσουμε σε subdomains με πραγματικό μέγεθος ενός στοιχείου, όποτε και περαιτέρω διάσπαση δεν έχει νόημα.

Η εικόνα του πίνακα μετά το τέλος της διάσπασης είναι αυτή που φαίνεται στο σχήμα. Με πράσινο χρώμα υποδηλώνονται τα admissible blocks και με κόκκινο τα inadmissible.



3.3 Admissibility Condition

Το Admissibility Condition, όπως αναφέρθηκε και πιο πάνω, είναι ένα κριτήριο το οποίο καθορίζει αν το sub-block που εξετάζεται μπορεί να αναπαρασταθεί στην παραγοντοποιημένη (rkmatrix) μορφή, οπότε χαρακτηρίζεται ως **admissible**, ή πρέπει να διασπαστεί (αν αυτό είναι εφικτό και δεν έχει φτάσει στο μέγιστο επίπεδο διάσπασης) σε 4 μικρότερα με τον τρόπο που φαίνεται στο προηγούμενο παράδειγμα, οπότε χαρακτηρίζεται ως **inadmissible**. Ένα τυπικό admissibility condition, το οποίο χρησιμοποιήθηκε και στη συγκεκριμένη εφαρμογή, βασίζεται στη γεωμετρία του πίνακα και όχι στο περιεχόμενό του και εξετάζει την Ευκλείδεια διάμετρο (diam) ενός block και την ελάχιστη Ευκλείδεια απόσταση (dist). Αν diam ≤ dist για ένα block τότε αυτό μαρκάρεται ως **admissible** αλλιώς είναι **inadmissible.**

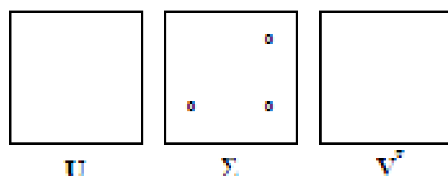
Έστω ένα block με συντεταγμένες domain [domainrowsfrom , domainrowsto] x [domaincolsfrom , domaincolsto]. Η Ευκλείδεια διάμετρος και η ελάχιστη Ευκλείδεια απόσταση υπολογίζονται ως εξής:

Diam = |domainrowsfrom – domainrowsto|

Dist = min (|domainrowsfrom- domaincolsfrom| , |domainrowsfrom – domaincolsto|, |domainrowsto - domaincolsfrom| , | domainrowsto - domaincolsto|)

3.4 Singular Value Decomposition

Για να αποθηκευτεί ένας πίνακας σε rkmatrix μορφή, απαιτείται να υποστεί μια διαδικασία γνωστή ως **Singular Value Decomposition** ή πιο σύντομα **SVD**. Η διαδικασία αυτή μετατρέπει έναν πίνακα A, σε γινόμενο τριών πινάκων U, S, V, με τους U και V να είναι ορθοκανονικοί, και τον S να είναι διαγώνιος, με στοιχεία της διαγωνίου του τις ιδιάζουσες τιμές του πίνακα A και μάλιστα σε φθίνουσα σειρά. Αν και πολλές φόρες συμπίπτουν, οι ιδιάζουσες τιμές που υπολογίζονται ΔΕΝ είναι απαραίτητα οι ιδιοτιμές του πίνακα. Η παραγοντοποίηση αυτή έχει ορισμένα σημαντικά πλεονεκτήματα.



Εικόνα 3: SVD ανάλυση

1. Οι πίνακες U και V συνήθως έχουν περισσότερα μηδενικά, άρα είναι πιο πιθανό να είναι αραιοί σε σχέση με τον αρχικό.

2. Ο S εφόσον είναι διαγώνιος, μπορεί να αποθηκευτεί ως ένα διάνυσμα

3. Επειδή οι ιδιάζουσες τιμές φθίνουν και μάλιστα πολλές φορές γρήγορα, κάποιες πολύ μικρές μπορούν να πασαλειφθούν αντικαθιστώντας τες με μηδενικά, αφού το σφάλμα που προκύπτει είναι αμελητέο. Έτσι κατά την αποθήκευση μπορούν να μικρύνουν και οι U και V καθώς μπορούν να πασαλειφθούν οι γραμμές/στήλες που πολλαπλασιάζονται με τις αντίστοιχες ιδιάζουσες τιμές που μηδενίστηκαν, αφού είναι προφανές ότι το αποτέλεσμά τους θα είναι 0.

Κεφάλαιο 4

Ανάπτυξη και Λειτουργία του Προγράμματος

4.1 Αρχεία Κώδικα

Το πρόγραμμα γράφτηκε σε περιβάλλον C και δέχεται ως είσοδο 2 αρχεία, με το 1ο να περιλαμβάνει τον πίνακα εισόδου και το 2ο το διάνυσμα με το οποίο θα γίνει ο πολλαπλασιασμός, και περιλαμβάνει συναρτήσεις χωρισμένες σε διάφορα αρχεία, ανάλογα με τη λειτουργικότητά τους.

- **global.h:** περιλαμβάνει τις δηλώσεις των global μεταβλητών που χρησιμοποιήθηκαν καθώς και τις δηλώσεις των δομών που απαιτούνται όπως fullmatrix, rkmatrix, αλλά και της struct του δέντρου.
- **List func.c:** περιέχει βοηθητικές συναρτήσεις για την κατασκευή του δέντρου, όπως είναι η αρχικοποίηση ενός κόμβου, η προσθήκη του στο δέντρο και η αρχικοποίηση του δέντρου.
- **Functions.c:** έχει τις συναρτήσεις αρχικοποίησης και δέσμευσης μνήμης των fullmatrix και rkmatrix και της διαγραφής τους όταν αυτοί δε χρειάζονται.
- **Print.c:** έχει τις συναρτήσεις εκτύπωσης ενός πίνακα και των πεδίων ενός κόμβου του δέντρου (η εκτύπωση του κόμβου χρειάστηκε καθαρά και μόνο για debugging)
- **Parser.c:** εδώ γίνεται η ανάγνωση των αρχείων εισόδου και η αποθήκευση των δεδομένων σε ένα fullmatrix ώστε να ξεκινήσει η διαδικασία του σπασίματος.
- **Simple break.c:** στο αρχείο αυτό γίνεται η δημιουργία του quad tree με αναδρομικό τρόπο και ταυτόχρονα αποφασίζεται και το admissibility κάθε κόμβου
- **Multiply.c:** υλοποιεί τον πολλαπλασιασμό ενός ιεραρχικού πίνακα με διάνυσμα και επιστρέφει το αποτέλεσμα
- **Main.c:** η κλήση της main η οποία ελέγχει τα αρχεία εισόδου αν είναι σωστά, καλεί τον parser για να αποθηκεύσει τα δεδομένα εισόδου και κατόπιν καλεί τη **Simple break** για να δημιουργήσει το δέντρο και τέλος πολλαπλασιάζει με το διάνυσμα που δίνεται στο 2ο αρχείο εισόδου και τυπώνει το αποτέλεσμα.
- **Run.sh:** βοηθητικό αρχείο το οποίο τρέχει την εντολή compile με τους κατάλληλους linkers. Από την εκτέλεση προκύπτει ένα εκτελέσιμο με όνομα main_file.

4.2 Δομές του προγράμματος

Οι δομές που κατασκευάστηκαν, είναι αυτές των fullmatrix, rkmatrix, και της struct του δέντρου που ονομάστηκε list.

4.2.1 Fullmatrix

Ο fullmatrix είναι η πιο απλή και πιο μικρή από τις δομές και περιέχει 2 πεδία int για το πλήθος των γραμμών και στηλών του πίνακα καθώς και ένα δείκτη για τον ίδιο τον πίνακα και ο ορισμός του σε κώδικα είναι:

```
typedef struct _fullmatrix fullmatrix;
typedef fullmatrix *pfullmatrix;
 struct _fullmatrix
 {
            int rows;
            int cols;
            double* f;
};
```

Rows
Cols
*f

Εικόνα 5: fullmatrix

4.2.2 Rkmatrix

Η δομή του rkmatrix είναι λίγο πιο σύνθετη καθώς έχει 2 πίνακες a και τις μεταβλητές k και kt οι οποίες είναι για την τάξη του πίνακα. Χρησιμεύουν για υλοποιήσεις και ελέγχους σε πίνακες μη τετραγωνικούς επομένως στην παρούσα υλοποίηση δεν είναι και τόσο χρήσιμες.

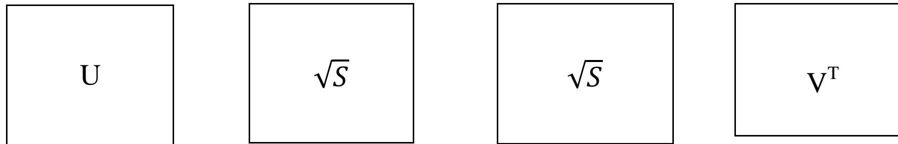
```
typedef struct _rkmatrix rkmatrix;
typedef rkmatrix *prkmatrix;
struct _rkmatrix
 {
            int k;
            int kt;
            int rows;
            int cols;
            double* a;
            double* b;
};
```

rows
cols
k
kt
*a
*b

Εικόνα 4: rkmatrix

Οι πίνακες a και b περιέχουν το αποτέλεσμα της SVD ανάλυσης. Ο a περιέχει το γινόμενο του U που προκύπτει από την SVD πολλαπλασιασμένο από δεξιά με την τετραγωνική ρίζα του S και ο b έχει το γινόμενο του V^T πολλαπλασιασμένο από αριστερά πάλι με την

τετραγωνική ρίζα του S για λόγους συμμετρίας και ομοιομορφίας. Έχουν αποθηκευτεί δηλαδή τα γινόμενα $U\sqrt{S}$ και $\sqrt{S}V^T$.



Εικόνα 7: r->a και r->b

Για να ανακτηθεί ένα στοιχείο του αρχικού πίνακα (π.χ. για πολλαπλασιασμό η για οποιαδήποτε άλλη λειτουργία), απαιτείται ο πολλαπλασιασμός της αντίστοιχης γραμμής του a με την αντίστοιχη στήλη του b.

Η SVD ανάλυση ενός πίνακα, πραγματοποιείται από τη συνάρτηση gsl_linalg_SV_decomp(), η οποία βρίσκεται στο μαθηματικό πακέτο gsl και χρειάζεται να γίνει #include το κομμάτι που αφορά τους πίνακες της gsl που βρίσκεται στο <gsl/gsl_matrix.h>. Επίσης για να δουλέψει σωστά, χρειάζεται κατά το compile, link με τις –lgsl –lgslcblas.

4.2.3 Stuct List

Τελευταία είναι η δομή για τη δημιουργία του δέντρου, κάθε κόμβος του οποίου περιέχει πληροφορίες για το block που αντιπροσωπεύει

Τα πεδία domainrowsfrom, domainrowsto, domaincolsfrom, domaincolsto, περιέχουν πληροφορίες για τη θέση και το μέγεθος του block με βάση το domain.

Τα πεδία rowsfrom, rowsto, colsfrom, colsto περιέχουν την ίδια πληροφορία με τα αντίστοιχα domains που περιγράφονται πάνω, αυτή τη φορά όμως σε καρτεσιανές συντεταγμένες. Ο υπολογισμός τους είναι απλός και προκύπτει από το γινόμενο του domain με το πλήθος γραμμών/στηλών του αρχικού πίνακα

Π.χ. rowsfrom = domainrowsfrom * initrows

Το πεδίο admissible παίρνει τιμές -1, 0, 1 και όπως είναι προφανές, δηλώνει αν το block είναι admissible (1) ή όχι (0). Η τιμή -1 δίνεται κατά την αρχικοποίηση και αν κάποιο block βρεθεί με τέτοια τιμή σημαίνει ότι δεν έχει εξεταστεί ακόμα.

rows
cols
domainrowsfrom
domainrowsto
domaincolsfrom
domaincolsto
rowsfrom
rowsto
colsfrom
colsto
admissible
prkmakrix *r
pfullmakrix *f
*son1, *son2, *son3, *son4

Εικόνα 8: Κόμβος του δέντρου

Οι δείκτες r και f δείχνουν σε ένα rkmatrix και ένα fullmatrix αντίστοιχα και είναι NULL για όλους τους ενδιάμεσους κόμβους του δέντρου, ενώ για τα φύλλα του, αν είναι admissible τότε είναι γεμάτος ο rkmatrix ενώ στην αντίθετη περίπτωση ο fullmatrix.

Όσο για τα *son1, *son2, *son3, *son4, είναι δείκτες σε κόμβους παιδιά του κόμβου που εξετάζεται. Αν ένας κόμβος δεν έχει παιδιά τότε σημαίνει ότι είναι φύλλο και περιέχει είτε ένα fullmatrix είτε ένα rkmatrix.

4.3 Παράδειγμα Τρεξίματος

Έστω ότι έχουμε ως είσοδο ένα πίνακα 8X8

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 9: Πίνακας εισόδου 8X8

Το δέντρο αρχικοποιείται κατά την κλήση της simple_break με αρχικές τιμές στα πεδία του αυτές που φαίνονται στην Εικόνα 9.

Βήμα 1

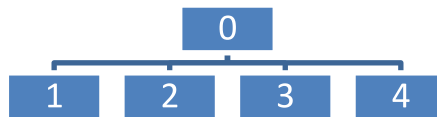
Εξετάζοντας το Admissibility condition, είναι προφανές ότι ο πίνακας δεν είναι admissible οπότε θα διασπαστεί σε 4 κομμάτια όπως φαίνεται στην Εικόνα 10 και το δέντρο παίρνει τη μορφή της εικόνας 12 με 0 να είναι η ρίζα του.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 10: 1ο στάδιο διάσπασης δέντρου

Rows=8
Cols=8
Domainrowsfrom=0
Domainrowsto=1
Domaincolsfrom=0
Domaincolsto=1
Rowsfrom=0
Rowsto=7
Colsfrom=0
Colsto=7
Admissible=-1
prkmakrix *r=NULL
pfullmakrix *f=NULL
*son1, *son2, *son3, *son4

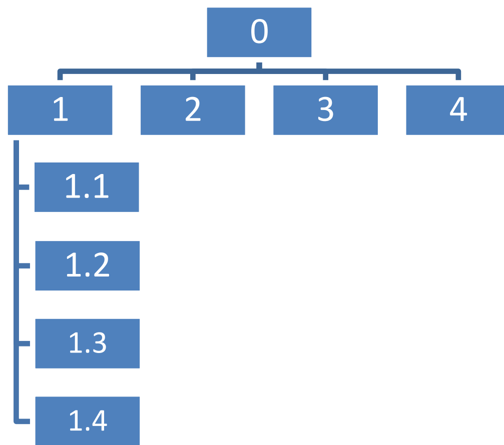
Εικόνα 11: ρίζα του



Εικόνα 12: δέντρο στο 1ο βήμα

Βήμα 2

Εξετάζεται ο κόμβος 1 και διαπιστώνεται ότι δεν είναι admissible και πρέπει να διασπαστεί (Εικόνα 14), καθώς diam[0, ½] = ½ και dist[0, ½][0, ½] = 0 επεκτείνοντας και το δέντρο (Εικόνα 13)



Εικόνα 13

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

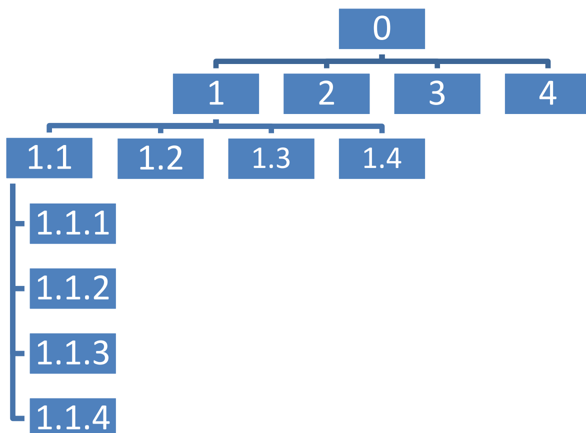
Εικόνα 14

Βήμα 3

Η αναδρομή συνεχίζεται εξετάζοντας τον κόμβο 1.1 και διαπιστώνεται ότι

Diam[0, ¼] = ¼ Dist[0, ¼][0, ¼] = 0

Επομένως το block είναι inadmissible και θα διασπαστεί περαιτέρω και ο πίνακας με το δέντρο θα έχουν τις μορφές των εικόνων 15 και 16



Εικόνα 15

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 16

Βήμα 4

Βρίσκοντας πάλι με τον ίδιο τρόπο τα μεγέθη diam και dist των block 1.1.1 ως 1.1.4, διαπιστώνουμε ότι είναι πάλι inadmissible αλλά έχουν φτάσει το μέγιστο επιτρεπτό όριο διάσπασης, επομένως τα αντίστοιχα δεδομένα θα αποθηκευτούν σε fullmatrix μορφή, αφήνοντας το δέντρο ανεπηρέαστο. Στον πίνακα μαρκάρονται με μπλε χρώμα τα inadmissible blocks που είναι αποθηκευμένα σε fullmatrix μορφή.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 17

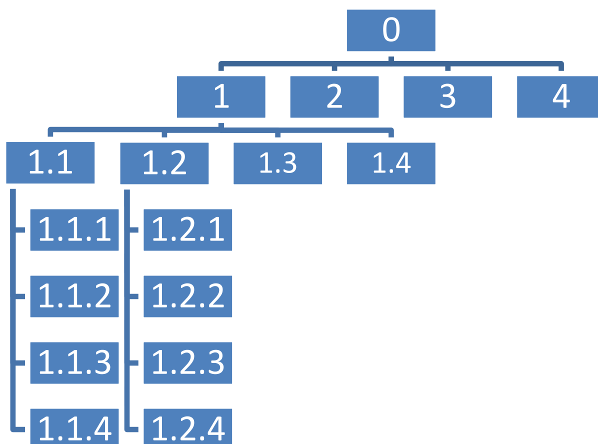
Βήμα 5

Η αναδρομή επιστρέφει στο προηγούμενο επίπεδο και εξετάζει τώρα το $2^{ο}$ παιδί του τρέχοντος κόμβου δηλαδή το 1.2

Diam[¼ , ½] = ¼

Dist[0, ¼][¼ , ½] = 0

Και αυτό το κομμάτι είναι inadmissible οπότε διασπάται όπως φαίνεται στις εικόνες 18-19



Εικόνα 18

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 19

Βήμα 6

Εδώ εμφανίζεται η $1^{η}$ διαφοροποίηση καθώς το block του στοιχείου 11 είναι μεν inadmissible όπως τα προηγούμενα, αλλά τα υπόλοιπα δεν είναι καθώς

Diam[¼ , 3/8] = 1/8 Dist[¼ , 3/8] [0 , 1/8] = 1/8

Diam[3/8 , ½] = 1/8 Dist[3/8 , ½] [0 , 1/8] = ¼

Diam[3/8 , ½] = 1/8 Dist[3/8 , ½] [1/8 , ¼] = 1/8

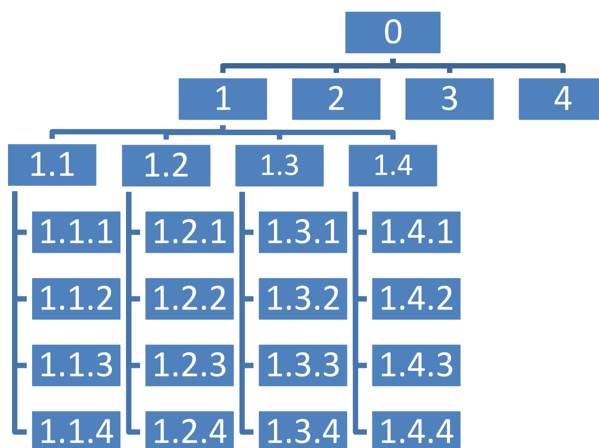
Επομένως το block του στοιχείου 11 θα αποθηκευτεί σε fullmatrix μορφή ενώ τα υπόλοιπα 3 blocks σε rkmatrix

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 20

Βήμα 7

Με τον ίδιο τρόπο και λόγω συμμετρίας αναλύονται τα blocks 1.1.3 και 1.1.4 (Εικόνα 21)



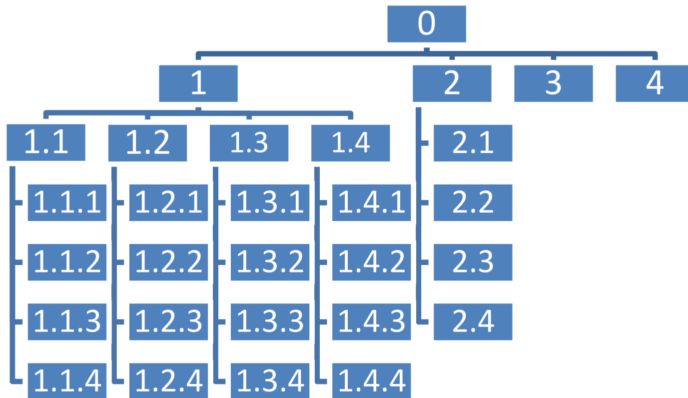
Εικόνα 21

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 22

Βήμα 8

Το επόμενο block που εξετάζεται είναι το 2. όπως και στο βήμα 2, διαπιστώνεται ότι δεν είναι admissible και χωρίζεται σε 4 sub-blocks με τις ανάλογες προσθήκες στο ιεραρχικό δέντρο.



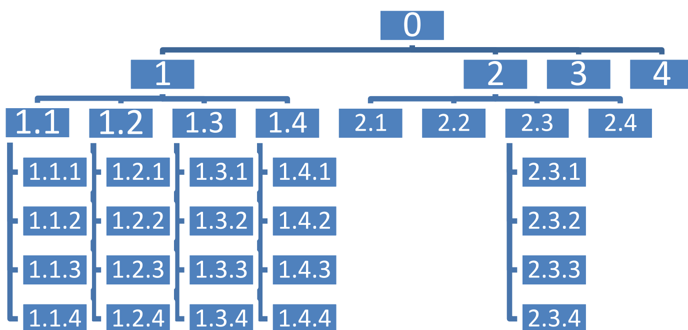
Εικόνα 23

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 24

Βήμα 9

Διαπιστώνουμε εδώ ότι τα blocks 2.1 , 2.2 και 2.4 είναι admissible οπότε γίνεται η μετατροπή τους σε rk μορφή και αποθηκεύονται στα κατάλληλα πεδία των κόμβων τους. Αντίθετα, το 2.3 διασπάται ακόμα μια φορά (Εικόνες 25-26)



Εικόνα 25

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 26

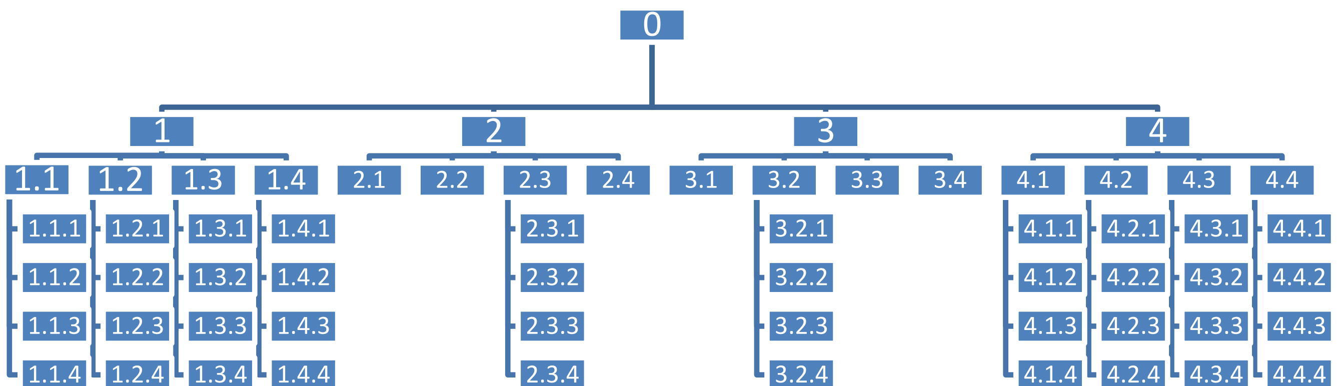
Βήμα 10

Τα blocks 2.3.1, 2.3.3 και 2.3.4 διαπιστώνεται ότι δεν είναι admissible και θα αποθηκευτούν σε fullmatrix μορφή, αντίθετα με το 2.3.2 που είναι.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 27

Τα υπόλοιπα βήματα είναι ακριβώς συμμετρικά ως προς τις 2 διαγωνίους του πίνακα, κι έτσι καταλήγει στο τέλος της διαδικασίας το δέντρο και ο πίνακας να έχουν τη μορφή στις Εικόνες 28-29.



Εικόνα 28

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Εικόνα 29

4.4 Πολλαπλασιασμός με διάνυσμα

Όπως είναι γνωστό από τη γραμμική άλγεβρα, ο πολλαπλασιασμός ενός πίνακα A με ένα διάνυσμα b, μας δίνει ως αποτέλεσμα ένα διάνυσμα c. Κάθε στοιχείο του διανύσματος c προκύπτει από τον πολλαπλασιασμό της αντίστοιχης γραμμής του A με το διάνυσμα b. Στην περίπτωση όμως των ιεραρχικών πινάκων, μια γραμμή του A είναι αποθηκευμένη σε διαφορετικά σημεία της δομής, επομένως μια λύση θα ήταν σε κάθε βήμα, να δημιουργείται η γραμμή που χρειάζεται και κατόπιν να γίνεται ο πολλαπλασιασμός και ο υπολογισμός του στοιχείου του αποτελέσματος. Αυτό όμως δε φαίνεται να είναι ιδιαίτερα αποδοτικό γιατί για να ανακτηθεί μια γραμμή του αρχικού πίνακα, χρειάζονται αρκετές αναδρομικές αναζητήσεις μέσα στη δομή του δέντρου οι οποίες κοστίζουν σε χρόνο και επιπλέον, όταν βρεθεί τελικά το block που χρειάζεται, δε χρησιμοποιούνται όλα τα στοιχεία του, παρά μόνο μια γραμμή του. Επιπλέον, το ίδιο block σε επόμενο βήμα θα ανακτηθεί και πάλι για να συνεισφέρει στη δημιουργία μιας επόμενης γραμμής.

Για παράδειγμα, στον πίνακα της εικόνας 26, για να ανακτηθεί η $3^{η}$ γραμμή του πίνακα, θα πρέπει να γίνουν 7 διατρέξεις του δέντρου για να εντοπιστούν τα 8 στοιχεία. Επιπλέον, στην τελευταία αναζήτηση για τα στοιχεία 23 24, έρχεται στη μνήμη όλο το block, όμως χρησιμοποιείται μόνο το μισό. Στο επόμενο βήμα που θα δημιουργηθεί η $4^{η}$ γραμμή, το block αυτό θα ανακτηθεί ξανά, αυτή τη φορά για να χρησιμοποιήσει μόνο τα στοιχεία 31 32.

Αυτό έρχεται σε αντίθεση με το σκοπό της δημιουργίας όλης της δομής γιατί δεν εκμεταλλεύεται σε ικανοποιητικό βαθμό τα δεδομένα που φέρνει στη μνήμη. Και αν δεν είναι τόσο φανερό εδώ γιατί είναι μικρό το παράδειγμα, σε ένα πίνακα της τάξεως 3000X3000, ένας rkmatrix του μπορεί να είναι 200X200. Το συγκεκριμένο block λοιπόν θα αναζητηθεί 200 φορές, και κάθε φορά θα χρησιμοποιείται μόνο 1/200 γραμμές.

Ας δούμε όμως ένα μικρό παράδειγμα πολλαπλασιασμού πίνακα με διάνυσμα.

Έστω ο πίνακας A 3X3 ο οποίος πρέπει να πολλαπλασιαστεί με το διάνυσμα [10 11 12]

1	2	3
4	5	6
7	8	9

Πίνακας A

Το αποτέλεσμα c θα είναι αναλυτικά:

1*10 + 2*11 +3*12
4*10 + 5*11 +6*12
7*10 + 8*11 +9*12

Μπορεί εύκολα να παρατηρηθεί εδώ ότι κάθε στοιχείο του A, συμμετέχει στο μερικό άθροισμα μόνο της γραμμής που βρίσκεται. Σε λίγο πιο κωδικοποιημένη μορφή αυτό σημαίνει ότι το στοιχείο A_{ij} πολλαπλασιάζεται με το b_j και προστίθεται στο μερικό άθροισμα του c_i.

Αν γίνει η αντιστοίχηση με έναν ιεραρχικό πίνακα, τότε για κάθε block που έρχεται στη μνήμη, όλα τα στοιχεία του θα πολλαπλασιάζονται με τα κατάλληλα στοιχεία του διανύσματος, και το αποτέλεσμα θα προστίθεται στο αντίστοιχο μερικό άθροισμα του διανύσματος αποτελέσματος. Με τον τρόπο αυτό απαιτείται μια αναδρομική διατρέξει του δέντρου, όπως ακριβώς και στη δημιουργία του, κατά την οποία για κάθε κόμβο που εξετάζεται και περιέχει πίνακες είτε full είτε rk χρησιμοποιούνται όλα τα στοιχεία τους. Όταν εξεταστεί και ο τελευταίος κόμβος του δέντρου, τα μερικά αθροίσματα στο διάνυσμα του αποτελέσματος θα περιέχουν το τελικό αποτέλεσμα.

<u>Παράδειγμα</u>

Έστω ότι ο πίνακας της εικόνας 26 θέλουμε να πολλαπλασιαστεί με το διάνυσμα [65 66 67 68 69 70 71 72] και η διαδικασία έχει φτάσει στο σημείο όπου εξετάζεται ο κόμβος 3.4 και στο διάνυσμα αποτελέσματος υπάρχουν τα μερικά αθροίσματα που έχουν προκύψει από προηγούμενα βήματα [Σ1 Σ2 Σ3 Σ4 Σ5 Σ6 Σ7 Σ8]

51	52
59	60

65
66
67
68
69
70
71
72

=

Σ1
Σ2
Σ3
Σ4
Σ5
Σ6
Σ7+51*67+52*68
Σ8+59*67+60*68

Εικόνα 30

Κεφάλαιο 5

Κώδικας και Πηγές

5.1 Κώδικας

Παρουσιάζεται το περιεχόμενο των αρχείων **simple_break.c** και **multiply.c** τα οποία περιέχουν τη διαδικασία διάσπασης του πίνακα και κατασκευής του δέντρου, καθώς και τον πολλαπλασιασμό πίνακα με διάνυσμα.

```
void breakit(struct list *current, int initrows, int initcols)

{

int i,j;

if(diam(current->domainrowsfrom, current->domainrowsto) > dist(current- >domainrowsfrom,
current->domainrowsto, current->domaincolsfrom, current->domaincolsto) )

{

current->admissible=0;


if(current->rows>minSubDomain && current->cols>minSubDomain)

{

struct list *node1= malloc(sizeof(struct list));

init_node(node1);

struct list *node2= malloc(sizeof(struct list));

init_node(node2);

struct list *node3= malloc(sizeof(struct list));

init_node(node3);

struct list *node4= malloc(sizeof(struct list));

init_node(node4);


node1->domainrowsfrom = current-> domainrowsfrom;

node1->domainrowsto = current->domainrowsfrom + (current->domainrowsto - current-
>domainrowsfrom)/2;

node1->domaincolsfrom = current-> domaincolsfrom;

node1->domaincolsto = current->domaincolsfrom + (current->domaincolsto - current-
>domaincolsfrom)/2;

node1->rows = (node1->domainrowsto - node1->domainrowsfrom) * initrows;

node1->cols = (node1->domaincolsto - node1->domaincolsfrom) * initcols;
```

```
node1->rowsfrom = (node1-> domainrowsfrom * initrows);

node1->rowsto = (node1->domainrowsto * initrows);

node1->colsfrom = node1->domaincolsfrom * initcols;

node1->colsto = node1->domaincolsto * initcols;

node1->admissible = -1;


node2->domainrowsfrom = current-> domainrowsfrom;

node2->domainrowsto = current->domainrowsfrom + (current->domainrowsto - current-
>domainrowsfrom)/2;

node2->domaincolsfrom = current->domaincolsfrom + (current->domaincolsto - current-
>domaincolsfrom)/2;

node2->domaincolsto = current-> domaincolsto;

node2->rows = (node2->domainrowsto - node2->domainrowsfrom) * initrows;

node2->cols = (node2->domaincolsto - node2->domaincolsfrom) * initcols;

node2->rowsfrom = (node2-> domainrowsfrom * initrows);

node2->rowsto = (node2->domainrowsto * initrows);

node2->colsfrom = node2->domaincolsfrom * initcols;

node2->colsto = node2->domaincolsto * initcols;

node2->admissible = -1;


node3->domainrowsfrom = current->domainrowsfrom + (current->domainrowsto - current-
>domainrowsfrom)/2;

node3->domainrowsto = current-> domainrowsto;

node3->domaincolsfrom = current-> domaincolsfrom;

node3->domaincolsto = current->domaincolsfrom + (current->domaincolsto - current-
>domaincolsfrom)/2;

node3->rows = (node3->domainrowsto - node3->domainrowsfrom) * initrows;

node3->cols = (node3->domaincolsto - node3->domaincolsfrom) * initcols;

node3->rowsfrom = (node3-> domainrowsfrom * initrows);

node3->rowsto = (node3->domainrowsto * initrows);

node3->colsfrom = node3->domaincolsfrom * initcols;

node3->colsto = node3->domaincolsto * initcols;

node3->admissible = -1;
```

```
node4->domainrowsfrom = current->domainrowsfrom + (current->domainrowsto - current->domainrowsfrom)/2;

node4->domainrowsto = current-> domainrowsto;

node4->domaincolsfrom = current->domaincolsfrom + (current->domaincolsto - current->domaincolsfrom)/2;

node4->domaincolsto = current-> domaincolsto;

node4->rows = (node4->domainrowsto - node4->domainrowsfrom) * initrows;

node4->cols = (node4->domaincolsto - node4->domaincolsfrom) * initcols;

node4->rowsfrom = (node4-> domainrowsfrom * initrows);

node4->rowsto = (node4->domainrowsto * initrows);

node4->colsfrom = node4->domaincolsfrom * initcols;

node4->colsto = node4->domaincolsto * initcols;

node4->admissible = -1;


list_add(node1, current, 1);

list_add(node2, current, 2);

list_add(node3, current, 3);

list_add(node4, current, 4);


breakit(current->son1, initrows, initcols);

breakit(current->son2, initrows, initcols);

breakit(current->son3, initrows, initcols);

breakit(current->son4, initrows, initcols);

}

else

{

current->admissible=1;

current->f = new_fullmatrix(current->rows, current->cols);

for(i=0; i<current->f->rows; i++)

{

for(j=0; j<current->f->cols; j++)

{

current->f-> e[(int)(i*(current->f->rows)+j)] =ptr->e[(int)((initrows*current->domainrowsfrom+i)*(ptr->rows)+(initcols*current->domaincolsfrom+j))];
```

```
}
}

// print_vector(current->f->e, current->f->rows, current->f->cols);
}

}
else
{
current->admissible = 0;
current->r = new_rkmatrix(current->rows, current->rows, current->cols);
gsl_matrix *A, *V;
gsl_vector *work, *S;
A = gsl_matrix_alloc (current->rows, current->cols);
V = gsl_matrix_alloc (current->rows, current->cols);
S = gsl_vector_alloc (current->rows);
work = gsl_vector_alloc (current->rows);

for(i=0; i<current->r->rows; i++)
{
for(j=0; j<current->r->cols; j++)
{
A->data[i*(current->r->rows)+j] = ptr->e[(int)((initrows*current->domainrowsfrom+i)*(ptr-
>rows)+(initcols*current->domaincolsfrom+j))];
}
}

gsl_linalg_SV_decomp (A, V, S, work);

for (i=0; i<S->size; i++){S->data[i] = sqrt(S->data[i]);}  //

for(i=0; i<current->r->rows; i++)
{
```

```
for(j=0; j<current->r->cols; j++)
{
current->r->a[i*(current->r->rows)+j] = A->data[i*A->size1+j] * S->data[j];
current->r->b[i*(current->r->rows)+j] = V->data[j*V->size1+i] * S->data[i];       //transposed V
}
}
gsl_matrix_free (A);
gsl_matrix_free (V);
gsl_vector_free (S);
gsl_vector_free (work);

return;
}
}
```

void simple_break (int rows, int cols)   //initialize tree and start recursion

```
{
list_init();
root->domainrowsfrom = 0;
root->domainrowsto = 1;
root->domaincolsfrom = 0;
root->domaincolsto = 1;

root->rowsfrom = 0;
root->rowsto = rows;
root->colsfrom = 0;
root->colsto = cols;
root->rows = rows;
root->cols = cols;
root->admissible = -1;
struct list *current;
int counter=0;
```

```
breakit(root, rows, cols);
}
```

void matrix_vec_mul(struct list *current)

```
{
int ro, co;
if(current->f != NULL)
{
int i, j, rfrom, cfrom;
double *fullmatr = current->f->e;
ro=current->f->rows;
co=current->f->cols;
rfrom=current->rowsfrom;
cfrom=current->colsfrom;
for(i=0; i<ro; i++)
{
for(j=0; j<co; j++)
{
result[rfrom+i] += fullmatr[i*ro+j] * mvec[j+ cfrom];
}
}
return;
}
else if(current->r != NULL)
{
int i, j, k, rfrom, cfrom, ro1, co1;
double *temp = (double*)malloc(current->r->rows*current->r->cols*sizeof(double));
ro=current->r->rows;
co=current->r->cols;
ro1=current->rows;
co1=current->cols;
```

```
rfrom=current->rowsfrom;
cfrom=current->colsfrom;
double *a = current->r->a;
double *b = current->r->b;
for(i=0; i<ro*co; i++)    {    temp[i] = 0;                                   }

for(i=0; i<ro; i++)
{
for(j=0; j<co; j++)
{
for(k=0; k<ro; k++)
{    temp[i*ro+j] += a[i*ro1+k] * b[k*ro1 +j];      }
}
}
for(i=0; i<ro; i++)
{
for(j=0; j<co; j++)
{result[i+rfrom] += temp[i*ro1+j] * mvec[j+cfrom];}
}
free(temp);
return;
}
matrix_vec_mul(current->son1);
matrix_vec_mul(current->son2);
matrix_vec_mul(current->son3);
matrix_vec_mul(current->son4);
}
```

5.2 Πηγές

http://www.disigma.gr/media/blfa_files/chapter_ARITHMITIKH_GRAMMIKH_ALGEBRA.pdf

https://en.wikipedia.org/wiki/Singular_value_decomposition

http://www.mis.mpg.de/preprints/ln/lecturenote-2103.pdf

http://www.inf.uth.gr/wp-content/uploads/formidable/manona_406.pdf

http://nanodft09.iyte.edu.tr/pdf/KG-L2.pdf