Τεχνολογικο Εκπαιδευτικο Ιδρυμα Πελοποννησου

Σχολη Τεχνολογικων Εφαρμογων

Τμημα Μηχανικων Πληροφορικης Τ.Ε.

Πολλαπλασιασμός Αλληλουχίας Πινάκων

Πτυχιακη Εργασια

του

ΑΡΓΥΡΙΑΔΗ ΧΡΗΣΤΟΥ

Επιβλέπων: Γρηγόρης Καραγιώργος

Επίκουρος Καθηγητής

Σπάρτη, Ιούνιος 2016

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Πελοποννήσου
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

Πολλαπλασιασμός Αλληλουχίας Πινάκων

Πτυχιακη Εργασια

του

ΑΡΓΥΡΙΑΔΗ ΧΡΗΣΤΟΥ

Επιβλέπων: Γρηγόρης Καραγιώργος
Επίκουρος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Φ.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γρηγόρης Καραγιώργος	-	-
Επίκουρος Καθηγητής	-	-

Σπάρτη, Ιούνιος 2016

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Πελοποννήσου
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας, και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής Τ.Ε. του ΤΕΙ Πελοποννήσου.

(Υπογραφή)

.....

Χρήστος Αργυριάδης

Περίληψη

Μία αλληλουχία πινάκων αποτελείται από ένα σύνολο πινάκων, οι οποίοι πληρούν τις προϋποθέσεις του πολλαπλασιασμού πινάκων. Ο πολλαπλασιασμός αλληλουχίας πινάκων είναι η διαδικασία με την οποία πολλαπλασιάζουμε τους πίνακες της αλληλουχίας, μέχρις ότου, το αποτέλεσμα να είναι ένας τελικός πίνακας, ο οποίος είναι το γινόμενο όλων των πινάκων που αποτελλούσαν την αλληλουχία. Η εξέλιξη της τεχνολογίας και η φυσική ανάγκη του ανθρώπου να θέλει όλο και περισσότερη ταχύτητα, μας οδήγησαν στην εύρεση της βέλτιστης λύσης του πολλαπλασιασμού αλληλουχίας πινάκων. Αντιληφθήκαμε λοιπόν, ότι με τη χρήση του Δυναμικού Προγραμματισμού βελτιώνεται κατά πολύ ο χρόνος επίλυσης του συγκεκριμμένου προβλήματος. Ο Δυναμικός προγραμματισμός είναι ένα πολύ χρήσιμο, σχεδόν απαραίτητο εργαλείο όσον αφορά όχι μόνο τους αλγόριθμους, αλλά και γενικά στην επίλυση προβλημάτων. Εκτός από το δυναμικό προγραμματισμό, τεράστια σημασία στην επίλυση του αποτελέσματος έχει η σειρά με την οποία θα πολλαπλασιαστούν οι πίνακες μέχρις ότου φτάσουμε στον τελικό πίνακα. Ο υπολογισμός της βέλτιστης σειράς των πράξεων έχει επίσης τεράστια σημασία στον χρόνο επίλυσης του προβλήματος.

Στόχος της διπλωματικής εργασίας είναι α) η κατανόηση σε βάθος του πολλαπλασιασμού αλληλουχίας πινάκων β) η κατανόηση της σημασίας του δυναμικού προγραμματισμού γ) η κατανόηση της σημασίας της βέλτιστης σειράς πράξεων και δ) η υλοποίηση στην πράξη όλων των παραπάνω μέσω της ανάπτυξης δύο προγραμμάτων στη γλώσσα C.

στους γονείς μου

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή Δρ. Γρηγόριο Καραγιώργο για την επίβλεψη αυτής της διπλωματικής εργασίας. Επίσης θα ήθελα να ευχαριστήσω τους γονείς μου για την καθοδήγηση και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια.

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάση επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δε μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

Όνομα και Επώνυμο Συγγραφέα(Με κεφαλαία):

.....

Υπογραφή(Ολογράφως, χωρίς μονογραφή):

.....

Ημερομηνία(Ημέρα-Μήνας-Έτος):

.....

Περιεχόμενα

Κατάλογος Σχημάτων

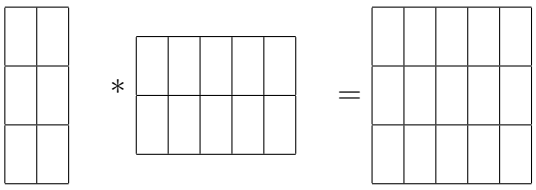
Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Τι είναι πολλαπλασιασμός πινάκων

Ο πολλαπλασιασμός δυο πινάκων ορίζεται αν και μόνο αν ο αριθμός των στηλών του αριστερού πίνακα είναι ίδιος με τον αριθμό των γραμμών του δεξιού πίνακα. Εάν το A είναι ένας μ-ν πίνακας και B είναι ένας ν-ρ πίνακας, τότε το το γινόμενο του πίνακα AB είναι ο μ-ρ πίνακας του οποίου τα στοιχειά δίνονται από το γινόμενο της αντίστοιχης σειράς του A και της αντίστοιχης στήλης B.Για να βρούμε τον αριθμό των πράξεων που θα χρειαστεί να πραγματοποιήσουμε για να βρούμε το αποτέλεσμα, αρκεί να βρούμε το γινόμενο $\mu * \nu * \rho$.

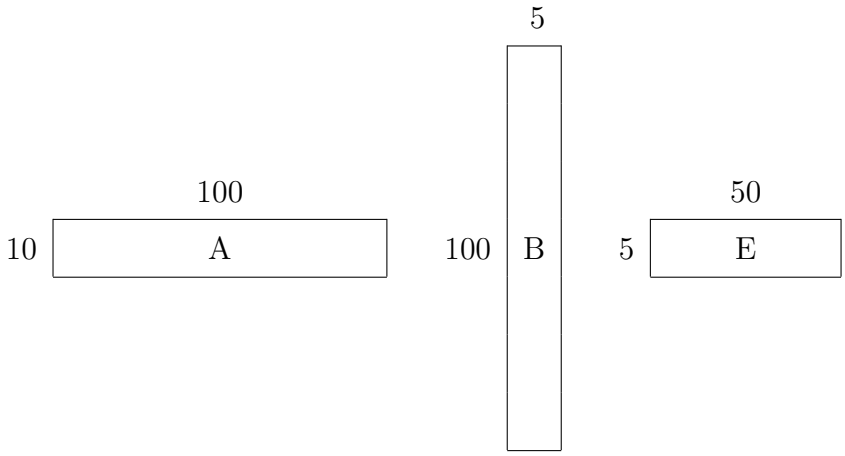


1.2 Τι είναι πολλαπλασιασμός αλληλουχίας πινάκων

Πολλαπλασιασμός αλληλουχίας πινάκων, είναι ο πολλαπλασιασμός μιας σειράς με 3 ή περισσότερους πίνακες.

Το πρόβλημα που παρουσιάζεται στον πολλαπλασιασμό αλληλουχίας πινάκων είναι

το εξής: ενώ ισχύει η προσεταιριστική ιδιότητα, αλλάζει ο αριθμός των πράξεων που πρέπει να πραγματοποιηθούν για να φτάσουμε στο αποτέλεσμα, ανάλογα με την σειρά που θα πολλαπλασιάσουμε τους πίνακες.



Στο παραπάνω παράδειγμα παρατηρούμε ότι, υπάρχουν 2 τρόποι πολλαπλασισμού αλληλουχίας των τριών πινάκων α) $(A*B)*E$ και β) A*(B*E) . Παρακάτω υπολογίζουμε τον αριθμό πράξεων που πρέπει να πραγματοποιηθούν σε κάθε μία από τις περιπτώσεις

$$\alpha) 10 * 100 * 5 + 10 * 5 * 50 = 7.500$$

$$\beta) 100 * 5 * 50 + 10 * 100 * 50 = 75.000$$

Παρατηρούμε ότι, ακόμα και σε ένα απλό πολλαπλασιασμό αλληλουχίας τριών πινάκων έχει μεγάλη σημασία με ποια σειρά θα πολλαπλασιαστούν οι πίνακες.

Για να βρεθεί ο βέλτιστος αριθμός πράξεων για τον πολλαπλασιασμό μιας αλληλουχίας πινάκων υπάρχουν 2 τρόποι. Ο πρώτος τρόπος είναι να πραγματοποιήσουμε όλες τις δυνατές πράξεις μεταξύ των πινάκων και στο τέλος να βρούμε τη βέλτιστη λύση. Αντιλαμβανόμαστε ότι ο πρώτος τρόπος είναι χρονοβόρος και πραγματοποιεί επαναλαμβανόμενες πράξεις οι οποίες μπορούν να αποφευχθούν. Ο δεύτερος τρόπος είναι να χρησιμοποιήσουμε το δυναμικό προγραμματισμό έτσι ώστε να βρούμε τη βέλτιστη λύση.

Κεφάλαιο 2

Δυναμικός Προγραμματισμός

Στο κεφάλαιο αυτό εξηγείται τι είναι δυναμικός προγραμματισμός και παρουσιάζονται κάποιες εφαρμογές του.

2.1 Τι είναι δυναμικός προγραμματισμός·

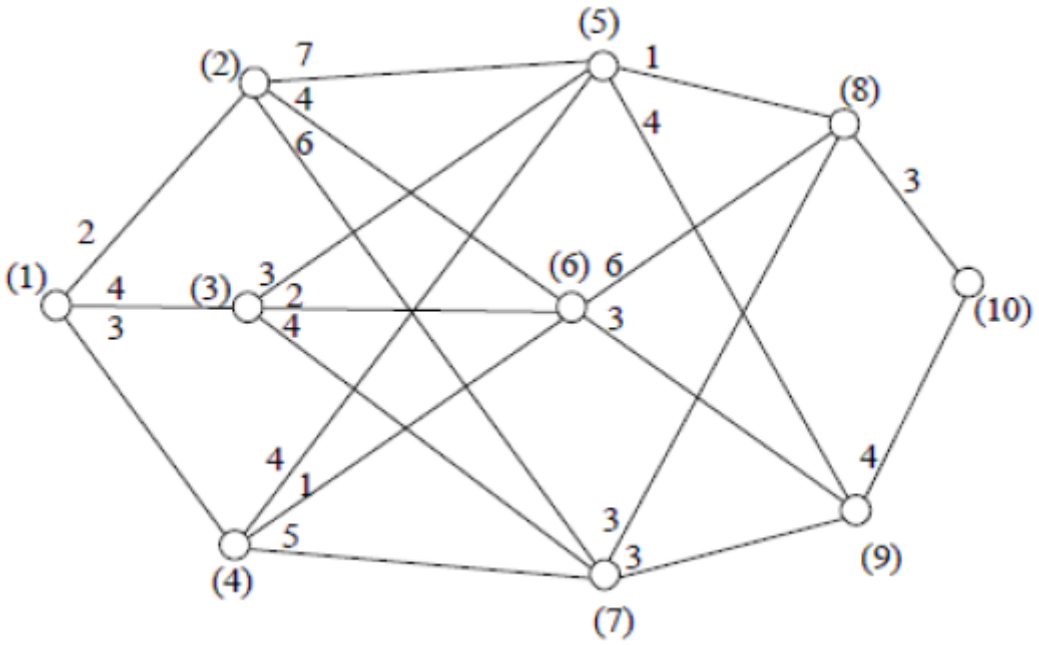
Ο δυναμικός προγραμματισμός είναι μια υπολογιστική μέθοδος η οποία εφαρμόζεται όταν πρόκειται να ληφθεί μια σύνθετη απόφαση, η οποία προκύπτει από τη σύνθεση επιμέρους αποφάσεων που αλληλοεξαρτώνται. Επιλύει προβλήματα χωρίζοντας τις λύσεις σε υποπροβλήματα τα οποία συνδέονται μεταξύ τους με τη βοήθεια αναδρομικών σχέσεων. Στη συνέχεια επιλύει κάθε υποπρόβλημα ξεχωριστά και αποθηκεύει κάθε λύση ξεχωριστά έτσι ώστε να μπορεί να την επαναχρησιμοποιήσει στο μέλλον, χωρίς να χρειαστεί να ξαναυπολογιστεί το υποπρόβλημα. Η αποθήκευση των λύσεων είναι και η κύρια διαφορά της μεθόδου του δυναμικού προγραμματισμού με τη μέθοδο διαίρει και βασίλευε.

2.2 Εφαρμογή του δυναμικού προγραμματισμού: ¨Αρχή της βελτιστότητας' (Υπολογισμός ελαχίστου μονοπατιού)

Μιλώντας για βέλτιστη υποδομή αναφερόμαστε στο ότι οι βέλτιστες λύσεις των υποπροβλημάτων αποτελούν το κλειδί για την βέλτιστη λύση του γενικότερου προβλήματος. Ο δυναμικός προγραμματισμός δεν μπορεί να εφαρμοστεί αν δε διατηρείται η αρχή της βελτιστότητας. Η πρώτη κίνηση όταν χρησιμοποιούμε τον δυναμικό προγραμματισμό, είναι η δημιουργία της αναδρομικής εξίσωσης για την εύρεση της βέλτιστης λύσης με τη χρήση της αρχής της βελτιστότητας. Αυτή δηλώνει ότι ανεξάρτητα ποια θα είναι η πρώτη απόφαση, όλες οι υπόλοιπες θα πρέπει να προσαρμοστούν και να είναι βέλτιστες σε σχέση με την κατάσταση που θα προκύψει από την πρώτη απόφαση. Αφού λυθεί η αναδρομική εξίσωση - για την τιμή της βέλτιστης λύσης - εκτελείται ένα βήμα αναγωγής όπου κατασκευάζεται η λύση. Στο δίκτυο του παρακάτω σχήματος ζητείται να βρεθεί η βέλτιστη διαδρομή από το σημείο 1 στο σημείο 10. Οι αριθμοί στην αρχή κάθε κλάδου μας δείχνουν τις επιμέρους αποστάσεις μεταξύ των κόμβων.

Ο υπολογισμός του συνολικού μήκους των διαδρομών είναι μια χρονοβόρα διαδικασία ακόμα και για ένα μικρού μεγέθους πρόβλημα, όπως είναι αυτό που εξετάζουμε. Για να υπολογιστούν όλα τα μονοπάτια πρέπει να υπολογίσουμε $3 * 3 * 2 = 18$ διαδρομές. Εάν χρειαζόταν να υπολογίσουμε όλες τις περιπτώσεις και υπήρχαν π.χ. 10 διαδρομές και κάθε διαδρομή αποτελούταν από τρεις περιπτώσεις θα έπρεπε να υπολογίσουμε 3^9 περιπτώσεις. Άρα ψάχνουμε κάποιον πιο αποτελεσματικό τρόπο για την επίλυση του προβλήματος.

Ένας πιο αποτελεσματικός τρόπος από τον υπολογισμό του συνολικού μήκους των διαδρομών από το 1 έως το 10 είναι να χρησιμοποιήσουμε το δυναμικό προγραμματισμό και να "σπάσουμε" το πρόβλημα σε μικρότερα υποπροβλήματα τα οποία λύνουμε διαδοχικά, αποθηκεύουμε τις λύσεις τους και τέλος τις συνδέουμε για να φτάσουμε στο τελικό αποτέλεσμα. Η μέθοδος που θα ακολουθήσουμε είναι να αντιμετωπίσουμε το πρόβλημα σε χωριστά βήματα, όπου το καθένα αποτελεί την επίληση του επόμενου προβλήματος, μέχρις ότου να φτάσουμε στο αρχικό πρόβλημα.



Σχήμα 2.1: Μονοπατι (Υπολογισμός Ελαχίστου Μονοπατιού)

Αρχίζοντας λοιπόν από το τέλος, βρίσκουμε την ελάχιστη διαδρομή από το 8 στο 10 είναι 3 και η ελάχιστη απόσταση από το 9 στο 10 είναι 4. Πηγαίνοντας πίσω ένα ακόμη βήμα, βρίσκουμε την ελάχιστη απόσταση από το 5 στο 10 επαναχρησιμοποιώντας τα προηγούμενα αποτελέσματα.

Η ελάχιστη απόσταση από το 5 στο 10 είναι

$\min 1 + 3, 4 + 4 = 4$

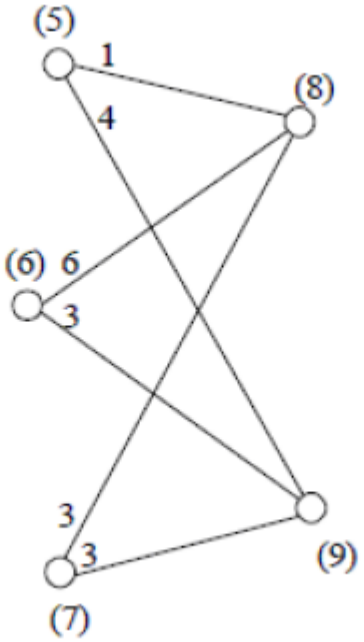
και η διαδρομή είναι μέσω του 8.

Μπορούμε να επεκταθούμε προς τα πίσω, βήμα-βήμα. Έτσι έχουμε:

- Η ελάχιστη απόσταση από το 6 στο 10 είναι 7 μέσω του 9.
- Η ελάχιστη απόσταση από το 7 στο 10 είναι 6 μέσω του 8.
- Η ελάχιστη απόσταση από το 2 στο 10 είναι 11 μέσω του 5 ή του 6.
- Η ελάχιστη απόσταση από το 3 στο 10 είναι 7 μέσω του 5.
- Η ελάχιστη απόσταση από το 4 στο 10 είναι 8 μέσω του 5 ή του 6.

Τέλος η ελάχιστη απόσταση από το 1 στο 10 είναι 11 μέσω του 3 ή του 4.

Από τα αποτελέσματα αυτά και μόνο μπορεί να προσδιοριστεί η βέλτιστη διαδρομή:



Σχήμα 2.2: Τμήμα σχήματος (Υπολογισμός Ελαχίστου Μονοπατιού)

Από το 1 η πρώτη απόφασή, μας φέρνει στο 3 ή στο 4. Εάν πάμε στο 3, τότε το επόμενο βήμα είναι στο 5, ύστερα στο 8 και τέλος στο 10. Εάν πάμε στο 4 τότε το επόμενο βήμα είναι στο 5 ή στο 6. Εάν πάμε στο 5, τότε τα επόμενα βήματα είναι 8,10. Εάν πάμε στο 6 τότε προχωράμε μέσω 9 στο 10. Έτσι υπάρχουν τρεις βέλτιστες διαδρομές.

$$1 - 3 - 5 - 8 - 10$$

$$1 - 4 - 5 - 8 - 10$$

$$1 - 4 - 6 - 9 - 10$$

Και οι τρεις διαδρομές έχουν, φυσικά το ίδιο μήκος και αυτό είναι το 11. [1]

Κεφάλαιο 3

Εφαρμογή Πολλαπλασιασμού Αλληλουχίας Πινάκων

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά, ένα παράδειγμα πολλαπλασιασμού αλληλουχίας έξι (6) πινάκων.

3.1 Παράδειγμα Πολλαπλασιασμού Αλληλουχίας 6 Πινάκων

Παρακάτω θα εξετάσουμε ένα παράδειγμα δυναμικού προγραμματισμού και έναν αλγόριθμο που επιλύει το πρόβλημα του πολλαπλασιασμού μιας αλληλουχίας πινάκων (Α, Β, Γ, Δ, Ε, Ζ) και μας ζητείται να υπολογίσουμε το γινόμενο $\mathrm{A} * \mathrm{B} * \Gamma * \Delta * \mathrm{E} * \mathrm{Z}$ Αρχικά βλέπουμε τους 6 πίνακες σε σειρά τους οποίους θα πολλαπλασιάσουμε.

$$A_{40,45} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,45} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,45} \\ \vdots & \vdots & \ddots & \vdots \\ a_{40,1} & a_{40,2} & \cdots & a_{40,45} \end{pmatrix} \quad B_{45,15} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,15} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ a_{45,1} & a_{45,2} & \cdots & a_{45,15} \end{pmatrix}$$

$$\Gamma_{15,10} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,10} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ a_{15,1} & a_{15,2} & \cdots & a_{40,10} \end{pmatrix} \quad \Delta_{10,20} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,20} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,20} \\ \vdots & \vdots & \ddots & \vdots \\ a_{10,1} & a_{10,2} & \cdots & a_{10,20} \end{pmatrix}$$

$$\mathrm{E}_{20,5} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,5} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,5} \\ \vdots & \vdots & \ddots & \vdots \\ a_{20,1} & a_{20,2} & \cdots & a_{20,5} \end{pmatrix} \mathrm{Z}_{5,25} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,25} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,25} \\ \vdots & \vdots & \ddots & \vdots \\ a_{5,1} & a_{5,2} & \cdots & a_{5,25} \end{pmatrix}$$

3.1.1 Διατύπωση του προβλήματος του πολλαπλασιασμού αλληλουχίας πινάκων

Για μια δεδομένη αλληλουχία πινάκων $(A_1, A_2, ..., A_n)$ όπου για κάθε $i = 1, 2, ...n$ ο πίνακας A_i έχει διαστάσεις $p_{(i-1)} * p_i$, εκφράστε το γινόμενο $A_1 * A_2 * .... * A_n$ σε πλήρως παρενθετική μορφή τέτοια ώστε να ελαχιστοποιείται το πλήθος των βαθμωτών πολλαπλασιασμών. Το ζητούμενο του προβλήματος του πολλαπλασιασμού αλληλουχίας πινάκων δεν είναι η πράξη του πολλαπλασιασμού των πινάκων, αλλά να προσδιοριστεί μια διάταξη του πολλαπλασιασμού των πινάκων η οποία να δίνει το ελάχιστο δυνατό κόστος. Ο χρόνος που αναλώνεται για τον προσδιορισμό της βέλτιστης διάταξης υπεραντισταθμίζεται από τον χρόνο που εξοικονομείται αργότερα κατά τον πολλαπλασιασμό των πινάκων.

3.1.2 Μεθοδος επίλυσης εξαντλητική αναζήτηση ("brute force")

Πριν προχωρήσουμε στην επίλυση του πολλαπλασιασμού αλληλουχίας πινάκων με τη μέθοδο του δυναμικού προγραμματισμού ας ελέγξουμε την προφανή μέθοδο του εξαντλητικού ελέγχου (brute force), για να βεβαιωθούμε οτι δεν ενδίκνυται και ότι δεν μπορεί να μας δώσει δραστικό αλγόριθμο. Ας συμβολίσουμε το πλήθος των εναλλακτικών ομαδοποιήσεων μιας αλληλουχίας ν πινάκων με $P(n)$. Για $n = 1$ υπάρχει μόνο ένας πίνακας, άρα μόνο ένας τρόπος έκφρασης του γινομένου σε πλήρως παρενθετική μορφή. Για $n \geq 2$, το γινόμενο πινάκων είναι το γινόμενο δύο υπογινομένων πινάκων εκφρασμένων σε πλήρως παρενθετική μορφή, και η ¨διαχωριστική γραμμή' μεταξύ των δύο υπογινομένων θα βρίσκεται μεταξύ του k-οστού και του $(k+1)$-οστού πίνακα όπου $k = 1, 2, ..., n-1$. Άρα η σχέση (1)είναι η παρακάτω:

$$(1) \quad \Pi(i,j) = \begin{cases} 1 \text{ Εάν } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) \text{ Εάν } n \geq 2 \end{cases}$$

Το πλήθος των ομαδοποιήσεων αυξάνεται εκθετικά σε συνάρτηση με το n, άρα η μέθοδος εξαντλητικής αναζήτησης (brute force) δεν ενδίκνυται για τον προσδιορισμό της βέλτιστης παρενθετικής ομαδοποίησης μιας αλληλουχίας πινάκων και κατ΄ επέκταση τη λύση του προβλήματος του πολλαπλασιασμού αλληλουχίας πινάκων.

3.1.3 Η δομή μιας βέλτιστης παρενθετικής ομαδοποίησης

Το πρώτο βήμα στην επίλυση προβλημάτων μέσω του δυναμικού προγραμματισμού είναι η εύρεση της βέλτιστης υποδομής και στη συνέχεια μέσω των βέλτιστων λύσεων υποπροβλημάτων η εύρεση της κύριας βέλτιστης λύσης.
Αρχικά υιοθετούμε το συμβολισμό $A_{i...j}$ όπου $i \leq j$ για να δηλώσουμε τον πίνακα που προκύπτει από τον πολλαπλασιασμό της αλληλουχίας των πινάκων που έχουμε. Παρατηρούμε ότι για κάποια τιμή του k με $i < k < j$ υπολογίζουμε αρχικά τους πίνακες $A_{i..k}$ και $A_{k+1...j}$ και στη συνέχεια τους πολλαπλασιάζουμε για να πάρουμε το τελικό γινόμενο $A_{i..j}$. Άρα το κόστος αυτής της ομαδοποίησης είναι ίσο με το κόστος υπολογισμού του $A_{i..k}$, συν το κόστος του υπολογισμού του $A_{k+1...j}$, συν το κόστος του πολλαπλασιασμού των δύο αυτών πινάκων. Ας υποθέσουμε ότι μια βέλτιστη ομαδοποίηση της ακολουθίας $A_i * A_{i+1}$ διαχωρίζει το γινόμενο ανάμεσα στον πίνακα A_k και A_{k+1}. Άρα συνεχίζοντας πρέπει να βρεθεί η βέλτιστη ομαδοποίηση της ακολουθίας $A_i, A_{i+1}, ...A_k$ και στη συνέχεια η βέλτιστη ομαδοποίηση της ακολουθίας A_{k+1}, $A_{k+2}, ...A_j$. Εδώ βλέπουμε πως μπορούμε να διαιρέσουμε το πρόβλημά μας σε υποπροβλήματα και τα υποπροβλήματα σε νέα υποπροβλήματα κ.ο.κ. Στο τέλος θα πρέπει να συνδυάσουμε τις βέλτιστες λύσεις όλων των υποπροβλημάτων και στο τέλος του συνδυασμού αυτού να φτάσουμε στη βέλτιστη λύση του αρχικού προβλήματος.

3.1.4 Δημιουργία τύπου - ορισμού

Για το πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων, επιλέγουμε ως υποπροβλήματα τα προβλήματα του προσδιορισμού του ελάχιστου κόστους ομαδοποίησης των γινομένων $A_i * A_{i+1}....A_j$ για $1 \leq i \leq j \leq n$. Έστω $m[i, j]$ το ελάχιστο πλήθος βαθμωτών πολλαπλασιασμών που απαιτουνται για τον πολλαπλασιασμό του γινομένου $A_{i...j}$. Για το πλήρες πρόβλημα το κόστος θα είναι $A[1, n]$. Το $m[i, j]$ μπορεί να οριστεί αναδρομικά ως εξής. Εάν $i = j$ η αλληλουχία αποτελείται μόνο από ένα πίνακα Ai, άρα δεν απαιτούνται πράξεις για να βρεθεί η αλληλουχία των πινάκων και άρα $m[i, i] = 0$. Εάν $i < j$,έστω οτι η ομαδοποίηση χωρίζει το γινόμενο ως εξής : $A_i * A_{i+1}....A_k kai A_{k+1} * A_{k+2}....A_j$ με $i \leq k \leq j$. Σε αυτή την περίπτωση το $m[i, j]$ ισούται με το ελάχιστο κόστος για τον υπολογισμό των δύο υπογινόμενων $A_{i..k}$ και $A_{k+1...j}$ συν το κόστος του πολλαπλασιασμού των δύο πινάκων που θα προκύψουν από τα υπογινόμενα. Κάθε πίνακας έχει διαστάσεις $p_{i-1} * p_i$, άρα ο υπολογισμός του γινομένου των πινάκων $A_{i...k}A_{k+1..j}$ απαιτεί $pi - 1 * p_k * pj$ βαθμωτούς πολλαπλασιασμούς. Άρα έχουμε:

$$(2) \quad [i, j] = m[i, k] + m[k + 1, j] + p_{i-1} * p_k * p_j$$

Ο τύπος αυτός προϋποθέτει οτι ξέρουμε την τιμή του k, την οποία αγνοούμε. Ξέρουμε όμως ότι οι πιθανές τιμές του k είναι $j - i$ και πιο συγκεκριμένα $k = i, i + 1, ..., j - 1$. Συμπερασματικά πρέπει να εξετάσουμε όλες τις τιμές του k για να βρούμε την βέλτιστη λύση. Ο τελικός μας τύπος είναι:

$$(3) \quad \mathrm{m}[i, j] = \begin{cases} 0 \text{ Εάν } i = j \\ min\{m[i, k] + m[k + 1, j] + p_{i-1} * p_k * p_j \text{ Εάν } i < j \\ i \leq k < j \end{cases}$$

3.1.5 Υπολογισμός ελαχίστου κόστους

Για τον υπολογισμό του ελαχίστου κόστους θα μπορούσαμε να χρησιμοποιήσουμε την παραπάνω εξίσωση, για να κατασκευάσουμε έναν αναδρομικό αλγόριθμο. Ο αναδρομικός αλγόριθμος αυτός όμως θα ήταν το ίδιο προβληματικός με την προαναφερθείσα μέθοδο εξαντλητικής αναζήτησης, γιατί έχουν και οι δύο εκθετικό χρόνο εκτέλεσης. Έχουμε ένα υποπρόβλημα για κάθε ζεύγος των i και j, το οποίο ικανοποιεί τη σχέση $1 \leq i \leq j \leq n$. Παρατηρούμε λοιπόν, ότι ο αριθμός αυτός των υποπροβλημάτων είναι σχετικά μικρός. Ωστόσο παρατηρούμε και ότι μέσω του αναδρομικού αλγορίθμου, ο αλγόριθμος θα συναντήσει το ίδιο υποπρόβλημα πολλές φορές και θα το επιλύσει κάθε φορά που θα το συναντήσει. Εδώ καταλαβαίνουμε ότι αυτή η μέθοδος δεν ενδίκνυται, αφού χρησιμοποιούνται πόροι για την επίλυση του ίδιου υποπροβλήματος πολλές φορές, ενώ γνωρίζουμε ότι υπάρχει δυνατότητα γρηγορότερης λύσης μέσω του δυναμικού προγραμματισμού. Αντί, λοιπόν να χρησιμοποιήσουμε αναδρομικό αλγόριθμο μέσω της εξίσωσης (3), θα χρησιμοποιήσουμε τη μέθοδο του δυναμικού προγραμματισμού και θα χρησιμοποιήσουμε πίνακες για την αποθήκευση των αποτελεσμάτων, έτσι ώστε να αποφευχθεί η επαναχρησιμοποίηση πόρων για τη λύση του ίδιου προβλήματος. Παρακάτω βλέπουμε τον αλγόριθμο για την εύρεση της σωστής διάταξης αλληλουχίας πινάκων:

ΔΙΑΤΑΞΗ ΑΛΛΗΛΟΥΧΙΑΣ ΠΙΝΑΚΩΝ (p)

1 n $\leftarrow$ μήκος[p] -1

2 **για** i $\leftarrow$ 1 **έως** n

3 m[i,j] $\leftarrow$ 0

4 **για** l $\leftarrow$ 2 **έως** n $\quad \triangleright$ l είναι το μήκος της αλληλουχίας

5 **για** i $\leftarrow$ **έως** $n - l + 1$

6 $j \leftarrow i + l - 1$

7 $m[i, j] \leftarrow \infty$

8 **για** $k \leftarrow i$ **έως** $j - 1$

9 $q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$

10 **αν** $q < m[i, j]$

11 **τότε** $m[i, j] \leftarrow q$

12 $s[i, j] \leftarrow k$

13 **επιστροφή** m και s

[3]

Επεξήγηση του αλγορίθμου:

Ο αλγόριθμος είναι βασιμένος στη σχέση (3). Αρχικά (γραμμές 2-3) υπολογίζει τα ελάχιστα κόστη για τις αλληλουχίες μοναδιαίου μήκους (την πρώτη διαγώνιο του πίνακα 3.1). Το αποτέλεσμα σε αυτές τις περιπτώσεις είναι πάντα ¨0¨, αφού χρειάζονται μηδέν πράξεις για να βρεθεί ένας ήδη υπάρχων πίνακας. Αφού ο αλγόριθμος εισέθλει στον βρόχγο (γραμμή 4) για πρώτη φορά, υπολογίζει τα κόστη για τις αλληλουχίες μήκους l=2 (δεύτερη διαγώνιος του πίνακα 3.1). Στην επόμενη επανάληψη του βρόχγου θα υπολογίσει τις αλληλουχίες μήκους l=3 κ.ο.κ. . Παρακάτω (γραμμές 8-12) ο αλγόριθμος χρησιμοποιεί τον τύπο (3).Μέσω του τύπου αυτού βρίσκουμε την ελάχιστη τιμή του m[i,j], όπου αρχικά έχουμε θέσει ως άπειρο για την ομαλή λειτουργία του αλγορίθμου και παράλληλα βρίσκουμε και το 'k' της βέλτιστης λύσης, το οποίο αποθηκεύεται στον πίνακα s.

Παρακάτω βλέπουμε τον πίνακα αριθμών πράξεων που πραγματοποιούνται ανάλογα με ποιούς πίνακες πολλαπλασιάζονται μέσα στην παραπάνω αλληλουχία.

Πίνακας 3.1: Πίνακας Αριθμών Πράξεων

	1	2	3	4	5	6
6	19125	10750	3625	2250	2500	0
5	14125	5125	1750	1000	0	
4	32750	15750	3000	0		
3	24750	6750	0			
2	27000	0				
1	0					

Πίνακας 3.2: Πίνακας σημείου διαχωρισμού μεταξύ των πινάκων

	1	2	3	4	5
6	5	5	5	5	5
5	1	2	3	4	
4	3	3	3		
3	1	2			
2	1				

Χρησιμοποιώντας τον τύπο (3), θα εξετάσουμε πως συμπληρώνονται οι παραπάνω πίνακες βήμα βήμα.

Όσον αφορά τον πίνακα 3.1, είναι κατανοητό, ότι η μεγαλύτερη διαγώνιος του πίνακα (l=1) θα αποτελείται από μηδενικά αφού χρειάζονται 0 πράξεις για να βρεθούν οι ήδη υπάρχοντες πίνακες. Σύμφωνα με τον τύπο υπολογισμού του ελάχιστου αριθμού πράξεων πρέπει να υπολογίζουμε μία μία τις διαγωνίους του πίνακα, αφού για να υπολογιστεί κάποιος αριθμός της επόμενης διαγωνίου, είναι απαιραίτητοι αριθμοί από την προηγούμενη διαγώνιο.

Όσον αφορά τον πίνακα 3.2, συμπληρώνεται με την τιμή του 'k' στην ελάχιστη τιμή της κάθε περίπτωσης.

Ξεκινώντας, εφαρμόζουμε τον τύπο(3), για τον υπολογισμό του (1,2):

$\Pi(1,1) + \Pi(2,2) + 40 * 45 * 15 = 27000$

min= 27000, k = 1

Στη συνέχεια του (2,3):

$\Pi(2,2) + \Pi(3,3) + 45 * 15 * 10 = 6750$

min= 6750, k = 2

Ύστερα το (3,4):

$\Pi(3,3) + \Pi(4,4) + 15 * 10 * 20 = 3000$

min= 3000, k = 3

Μετά το (4,5):

$\Pi(4,4) + \Pi(5,5) + 10 * 20 * 5 = 1000$

min= 1000, k = 4

Και τέλος το (5,6):

$\Pi(5,5) + \Pi(6,6) + 20 * 5 * 25 = 2500$

min= 2500, k = 5

Στη συνέχεια θα υπολογίσουμε τη δεύτερη διαγώνιο του πίνακα.

Ξεκινάμε την αντικατάσταση του τύπου από το (1,3):

$\Pi(1,1) + \Pi(2,3) + 40 * 45 * 10 = 24750$

$\Pi(1,2) + \Pi(3,3) + 40 * 15 * 10 = 33000$

min = 24750, k = 1

Συνεχίζουμε με το (2,4):

$\Pi(2,2) + \Pi(3,4) + 45 * 15 * 20 = 16500$

$\Pi(2,3) + \Pi(4,4) + 45 * 10 * 20 = 15750$

min = 15750, k = 3

Ύστερα υπολογίζουμε το (3,5):

$\Pi(3,3) + \Pi(4,5) + 15 * 10 * 5 = 1750$

$\Pi(3,4) + \Pi(5,5) + 15 * 20 * 5 = 4500$

min = 1750, k = 3

Και τέλος το (4,6):

$\Pi(4,4) + \Pi(5,6) + 10 * 20 * 25 = 7500$

$\Pi(4,5) + \Pi(6,6) + 10 * 5 * 25 = 2250$

min = 2250, k = 5

Στις περιπτώσεις (1,3),(2,4),(3,5),(4,6) παρατηρούμε οτι έχουμε δύο αποτελέσματα, άρα πρέπει να επιλέξουμε το μικρότερο από τα δύο

Στη συνέχεια θα υπολογίσουμε την τρίτη διαγώνιο του πίνακα.

Ξεκινάμε την αντικατάσταση του τύπου από το (1,4):

$\Pi(1,1) + \Pi(2,4) + 40 * 45 * 20 = 51750$

$\Pi(1,2) + \Pi(3,4) + 40 * 15 * 20 = 42000$

$\Pi(1,3) + \Pi(4,4) + 40 * 10 * 20 = 32750$

min = 32750, k = 3

Στη συνέχεια υπολογίζουμε το (2,5):

$\Pi(2,2) + \Pi(3,5) + 45 * 15 * 5 = 5125$

$\Pi(2,3) + \Pi(4,5) + 45 * 10 * 5 = 10000$

$\Pi(2,4) + \Pi(5,5) + 45 * 20 * 5 = 20250$

min = 5125, k = 2

Και τέλος το (3,6):

$\Pi(3,3) + \Pi(4,6) + 15 * 10 * 25 = 6000$

$\Pi(3,4) + \Pi(5,6) + 15 * 20 * 25 = 13000$

$\Pi(3,5) + \Pi(6,6) + 15 * 5 * 25 = 3625$

min = 3625, k = 5

Στις περιτώσεις αυτές : (1,4),(2,5),(3,6) παρατηρούμε οτι έχουμε τρία αποτελέσματα και πρέπει να επιλέξουμε το μικρότερο από τα τρία.

Ομοίως θα υπολογίσουμε την τέταρτη διαγώνιο του πίνακα.
Ξεκινάμε την αντικατάσταση του τύπου από το (1,5):
$\Pi(1,1) + \Pi(2,5) + 40 * 45 * 5 = 14125$
$\Pi(1,2) + \Pi(3,5) + 40 * 15 * 5 = 31750$
$\Pi(1,3) + \Pi(4,5) + 40 * 10 * 5 = 29000$
$\Pi(1,4) + \Pi(5,5) + 40 * 20 * 5 = 36750$
min = 14125, k = 1

Και τελειώνουμε με το (2,6):
$\Pi(2,2) + \Pi(3,6) + 45 * 15 * 25 = 20500$
$\Pi(2,3) + \Pi(4,6) + 45 * 10 * 25 = 20250$
$\Pi(2,4) + \Pi(5,6) + 45 * 20 * 25 = 40750$
$\Pi(2,5) + \Pi(6,6) + 45 * 5 * 25 = 10750$
min = 10750, k = 5

Σε αυτές τις περιπτώσεις παρατηρούμε οτι έχουμε τέσσερα αποτελέσματα και θα πρέπει να βρούμε το ελάχιστο από αυτά τα τέσσερα αποτελέσματα.

Τέλος μας μένει να υπολογίσουμε το (1,6) το οποίο είναι και η κορυφή του πίνακα και είναι το αποτέλεσμα των πράξεων ολόκληρης της αλληλουχίας:
Υπολογίζουμε το (1,6):
$\Pi(1,1) + \Pi(2,6) + 40 * 45 * 25 = 55750$
$\Pi(1,2) + \Pi(3,6) + 40 * 15 * 25 = 45625$
$\Pi(1,3) + \Pi(4,6) + 40 * 10 * 25 = 37000$
$\Pi(1,4) + \Pi(5,6) + 40 * 20 * 25 = 60250$
$\Pi(1,5) + \Pi(6,6) + 40 * 5 * 25 = 19125$
min = 19125, k = 5

Συμπεράσματα του παραδείγματος:

Σε αυτό το παράδειγμα φαίνονται τα στοιχεία του δυναμικού προγραμματισμού. Τα αποτελέσματα που βρέθηκαν αποθηκευτηκαν στον πίνακα, έτσι ώστε την επόμενη φορά που τα χρειαστήκαμε να μην χρειάζεται να υπολογιστούν ξανά, σπαταλώντας άσκοπα χρόνο υπολογισμού.

Επίσης παρατηρούμε οτι για να βρεθεί το (1,2) το οποίο είναι αποτέλεσμα του πολλαπλασιασμού του πίνακα A με τον πίνακα B, χρειάστηκαν **27000** πράξεις, ενώ για τον υπολογισμό ολόκληρης της αλληλουχίας (1,6) χρειαστήκαμε **μόνο 19125** πράξεις! Εδώ βλέπουμε ακόμα μία χρήση του δυναμικού προγραμματισμού στον πολλαπλασιασμό αλληλουχίας πινάκων, ακόμα και στο παράδειγμά μας που αποτελείται μόνο από 6 πίνακες.

3.1.6 Κατασκευή μιας βέλτιστης λύσης

Ο αλγόριθμος "Διάταξη αλληλουχίας πινάκων" προσδιορίζει το ελάχιστο πλήθος πολλαπλασιασμών που μπορούν να πραγματοποιηθούν για την εύρεση του γινομένου μιας αλληλουχίας πινάκων. Ωστόσο δεν υπολογίζει άμεσα τη σειρά που θα πολλαπλασιαστούν οι πίνακες για να φτάσουμε στην επιθυμητή βέλτιστη λύση. Για την κατασκευή της βέλτιστης αυτής λύσης θα χρειαστούμε τον πίνακα 3.2. Στην παρακάτω αναδρομική διαδικασία που ακολουθεί, χρησιμοποιείται ο πίνακας 3.2(s) και εκτυπώνεται η βέλτιστη ομαδοποίηση της αλληλουχίας $(A_i, A_{i+1,....,A_j})$. Αρχικά η διαδικασία ξεκινάει με (s,1,n) ή στο παράδειγμά μας με (s,1,6) και η τελική εκτύπωση θα είναι η ομαδοποίηση για τη βέλτιστη αλληλουχία πινάκων.

ΕΚΤΥΠΩΣΗ ΒΕΛΤΙΣΤΗΣ ΟΜΑΔΟΠΟΙΗΣΗΣ (s,i,j)

```
1   αν i=j
2       τότε εκτύπωση 'A'
3       αλλιώς εκτύπωση '('
4           ΕΚΤΥΠΩΣΗ ΒΕΛΤΙΣΤΗΣ ΟΜΑΔΟΠΟΙΗΣΗΣ (s,i,s[i,j])
5           ΕΚΤΥΠΩΣΗ ΒΕΛΤΙΣΤΗΣ ΟΜΑΔΟΠΟΙΗΣΗΣ (s,s[i,j] + 1,j)
6           εκτύπωση ')'
```

[3]

Κεφάλαιο 4

Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζονται δύο προγράμματα σε γλώσσα C τα οποία πραγματοποιούν εφαρμογές του Πολλαπλασιασμού Αλληλουχίας Πινάκων

4.1 Πρόγραμμα Πολλαπλασιασμού Αλληλουχίας τριών Πινάκων εισαγόμενων από το χρηστη

4.1.1 Περίληψη του προγράμματος και ανάδειξη των στοιχείων που αποκομίζει ο χρήστης από αυτό

Το πρόγραμμα αυτό, αρχικά δημιουργεί τρεις πίνακες, των οποίων οι γραμμές και οι στήλες εισάγονται από το χρήστη και στη συνέχεια πραγματοποιεί τον πολλαπλασιασμό των τριών αυτών πινάκων, με τη σειρά που θα αποφασίσει ο χρήστης.

Μέσω του προγράμματος κατανοούμε τα παρακάτω όσον αφορά τον πολλαπλασιασμό αλληλουχίας πινάκων:
α) Έχουμε μια οπτική εικόνα όσον αφορά τον πολλαπλασιασμό αλληλουχίας πινάκων και μέσω της εικόνας αυτής κατανοούμε καλύτερα τη διαδικασία.
β) Κατανοούμε οτι στον πολλαπλασιασμό αλληλουχίας πινάκων τηρούνται οι κανόνες του πολλαπλασιασμού πινάκων, άρα έχουμε αρκετους περιορισμούς όσον αφορά τις γραμμές και τις στήλες των πινάκων που θα χρησιμοποιήσουμε

γ) Λόγω του ότι επιλέγουμε τη σειρά που θα πολλπλασιαστούν οι πίνακες, μπορούμε να δούμε οτι υπάρχει διαφορά όσον αφορά τη σειρά με την οποία πολλαπλασιάζονται οι πίνακες.

δ) Μπορούμε να δούμε πρακτικά τη διαφορά της σειράς με την οποία πολλαπλασιάζονται οι πίνακες.

4.1.2 Λειτουργία του προγράμματος

Λόγω των κανόνων του πολλαπλασιασμού πινάκων, θεωρήθηκε καλύτερη πρακτική να μην αφεθεί πλήρως στην κρίση του χρήστη η χρήση των γραμμών και των στηλών των πινάκων. Για να επιτευχεί αυτός ο στόχος, το πρόγραμμα έχει φτιαχτεί έτσι ώστε ο χρήστης να επίλέγει τις γραμμές και τις στήλες του πρώτου και του τρίτου πίνακα και στη συνέχεια, όσον αφορά τον δεύτερο πίνακα να του δίνει το πρόγραμμα τις επιλογές που υπάρχουν, έτσι ώστε να ισχύουν οι κανόνες πολλαπλασιασμού πινάκων.

Αρχικά το πρόγραμμα καλεί τη συνάρτηση 'arrayLinesRows' για τον πρώτο και τον τρίτο πίνακα

```
//Γραμμές και στήλες πρώτου πίνακα
arrayLinesRows(&a1_l, &a1_r, &arrayCounter);

//Γραμμές και στήλες δεύτερου πίνακα
arrayLinesRows(&a2_l, &a2_r, &arrayCounter);
```

Στη συνέχεια βλέπουμε τη συνάρτηση 'arrayLinesRows", όπου ρωτάται ο χρήστης πόσες γραμμές και πόσες στήλες θέλει να είναι ο πίνακας. Περιορίζει, ωστόσο, τον χρήστη στις 10 γραμμές και στήλες έτσι ώστε να υπάρξει ένας περιορισμός στα όρια των πινάκων.

Στην παρακάτω εικόνα βλέπουμε το πρόγραμμα να προτρέπει από το χρήστη να επιλέξει, ποιες θα είναι οι γραμμές και οι στήλες του δεύτερου πίνακα.

```
void arrayLinesRows(int *a_l, int *a_r, int *counter){

    do{
    *counter += 1;
    printf("Πληκτρολογήστε πόσες γραμμές και πόσες στήλες θέλετε να είναι ο %doς πίνακας (Μέχρι 10 γραμμές/στήλες)\n", *counter);
    printf("Πληκτρολογήστε γραμμές: \n");
    scanf("%d", a_l);
    printf("Πληκτρολογήστε στήλες: \n");
    scanf("%d", a_r);
    if(*a_l > 11 || *a_r >11){
        printf("Παρακαλούμε να εισάγετε πίνακες που οι γραμμές και οι στήλες τους είναι μικρότερες από '10'. Ευχαριστούμε.\n");
        *counter -= 1;
        }
    }while(*a_l > 11 || *a_r > 11 );
```

```
    do{
        printf("Ο τρίτος πίνακας μπορεί να είναι \nα) (%d,%d) (%d,%d) \nβ) (%d,%d) (%d,%d)\n", a2_r,a1_l, a1_r, a2_l);
        printf("Πατήστε '1' για την πρώτη επιλογή '2' για την δεύτερη\n");
        scanf("%d", &Array3LinesRows);

        if(Array3LinesRows == 1){
            a3_l = a2_r;
            a3_r = a1_l;
        }else if(Array3LinesRows == 2){
            a3_l = a1_r;
            a3_r = a2_l;
        }else{
            printf("Πρέπει να πληκτρολογήσετε '1' ή '2'.\n");
        }
    }while(Array3LinesRows != 1 && Array3LinesRows != 2);
```

Παρακάτω, το πρόγραμμα γεμίζει τον πρώτο πίνακα με τυχαίες τιμές από το 1 έως και το 10.

```
/* Γέμισμα ΠΡΩΤΟΥ πίνακα με τυχαίους αριθμούς*/
printf("Πίνακας 1");
for (i=1; i<=a1_l; i++){
    printf("\n");
    for (j=1; j<=a1_r; j++){
        array1 [i][j] = rand()%10;
        printf("%d ", array1[i][j]);
    }
}
printf("\n\n");
```

Ομοίως και τον δεύτερο.

```
/* Γέμισμα ΔΕΥΤΕΡΟΥ πίνακα με τυχαίους αριθμούς */
printf("Πίνακας 2");
 for (i=1; i<=a2_l; i++){
    printf("\n");
    for (j=1; j<=a2_r; j++){
        array2 [i][j] = rand()%10;
        printf("%d ", array2 [i][j]);
    }
}

printf("\n\n");
```

Τέλος γεμίζει ο τρίτος πίνακας.

```
/* Γέμισμα ΤΡΙΤΟΥ πίνακα με τυχαίους αριθμούς */
printf("Πίνακας 3");
 for (i=1; i<=a3_l; i++){
    printf("\n");
    for (j=1; j<=a3_r; j++){
        array3 [i][j] = rand()%10;
        printf("%d ", array3 [i][j]);
    }
}


printf("\n\n");
```

Ακολούθως, ο χρήστης καλείται να επιλέξει τη σειρά με την οποία θα πολλαπλασιαστούν οι τρεις πίνακες

```
do{
    printf("Με ποιά σειρά θέλετε να πολλαπλασιάσετε τους πίνακες? \n"
           "Εάν θέλετε να πολλαπλασιάσετε ΠΡΩΤΑ τον 1ο με τον 3ο και ΜΕΤΑ τον 2ο πατήστε : 1 \n"
           "Εάν θέλετε να πολλαπλασιάσετε ΠΡΩΤΑ τον 2ο με τον 3ο και ΜΕΤΑ τον 1ο πατήστε : 2 \n"
           "Εάν θέλετε να βγείτε από το πρόγραμμα πατήστε 'exit' ή 'x' \n");

    scanf("%d", &order);
```

Παρακάτω βλέπουμε την περίπτωση που ο χρήστης έχει επιλέξει το ¨1'

```
/************** ΠΡΩΤΗ ΠΕΡΙΠΤΩΣΗ (1*3)*2 **************/



        if(order == 1){// Επιβεβαίωση οτι έχει πατηθεί το "1"
            printf("Πατήσατε 1!\n\n");
```

Σε αυτό το σημείο, το πρόγραμμα κάνει έναν έλεγχο, για να πραγματοποιηθεί ο πολλαπλασιασμός πινάκων, σύμφωνα με τους κανόνες πολλαπλασιασμού πινάκων. Έτσι, το πρόγραμμα ελέγχει εάν οι γραμμές του πρώτου πίνακα είναι ίσες με τις στήλες

του τρίτου ή εάν οι στήλες του πρώτου πίνακα είναι ίσες με τις γραμμές του τρίτου. Με αυτό τον έλεγχο, το πρόγραμμα βάζει στη σωστή σειρά τους πίνακες που πρόκειται να πολλαπλασιαστούν, έτσι ώστε να μπορεί να γίνει πολλαπλασιασμός μεταξύ τους.

Στη συνέχεια, βλέπουμε τον πολλαπλασισμό των πινάκων, με τη σειρά που έχει επιλέξει ο χρήστης. Αρχικά πραγματοποιείται ο πολλαπλασιασμός του πρώτου πίνακα με τον τρίτο και εμφανίζεται ο πίνακας που προκύπτει από την πράξη στην οθόνη. Παρακάτω πολλαπλασιάζεται ο πίνακας γινόμενο, με τον εναπομείνωντα πίνακα και εμφανίζεται το τελικό αποτέλεσμα στην οθόνη.

Στην παρακάτω εικόνα, βλέπουμε την πρώτη περίπτωση, όπου οι γραμμές του πρώτου πίνακα είναι ίσες με τις στήλες του τρίτου.

```
/******** ΟΙ ΓΡΑΜΜΕΣ ΤΟΥ 1ΟΥ ΠΙΝΑΚΑ ΕΙΝΑΙ ΙΣΕΣ ΜΕ ΤΟΥ 3ΟΥ  *********/

    for (i=1; i<=a3_l; i++){
        printf("\n");
        for(j=1; j<=a1_r; j++){
            sum = 0;
            for(k=1; k<=a1_l; k++){
            sum = sum + (array1[k][j])*(array3[i][k]);
            }
            multipliedArray[i][j] = sum;
            printf("%03d ", multipliedArray[i][j]);
        }
    }


    printf("\n\n");
    for (i=1; i<=a2_l; i++){
        printf("\n");
        for(j=1; j<=a1_r; j++){
            sum = 0;
            for(k=1; k<=a3_l; k++){
            sum = sum + (multipliedArray[k][j])*(array2[i][k]);
            }
            finalArray[i][j] = sum;
            printf("%04d ", finalArray[i][j]);
        }
    }
```

Επιπλέον, στην οθόνη εμφανίζεται και ο αριθμός πράξεων που έχει πραγματοποιηθεί, έτσι ώστε να ξέρει ο χρήστης πόσες πράξεις έχουν πραγματοποιηθεί με την περίπτωση που έχει επιλέξει. Στη συνέχεια ο χρήστης μπορεί να τρέξει το πρόγραμμα με τους ίδιους πίνακες, αλλά να ακολουθήσει διαφορετική σειρά στον πολλαπλασιασμό των πινάκων και να βγάλει τα συμπεράσματά του, σύμφωνα με τον αριθμό των πράξεων που έχουν γίνει για να φτάσει στο αποτέλεσμα.

```
num_of_mult = a1_r * a3_l * a1_l + a1_r * a2_l * a2_r;
printf("\n\nO αριθμός των πολλαπλασιασμών που πραγματοποιήθηκαν για αυτό το αποτέλεσμα είναι : "
       "%d\n", num_of_mult);
```

Εδώ βλέπουμε την δεύτερη περίπτωση όπου οι στήλες του πρώτου πίνακα είναι ίσες με τις γραμμές του τρίτου. Το πρόγραμμα βάζει τους πίνακες στη σωστή σειρά έτσι ώστε να γίνει ο πολλαπλασιασμός μεταξύ τους.

```
}else if (a1_r == a3_l){

/********* ΟΙ ΣΤΗΛΕΣ ΤΟΥ 1ΟΥ ΠΙΝΑΚΑ ΕΙΝΑΙ ΙΣΕΣ ΜΕ ΤΟΥ 3ΟΥ  *********/
```

Όπως και στην πρώτη περίπτωση, αρχικά πολλαπλασιάζονται οι δύο πίνακες που έχει επιλέξει ο χρήστης με τη σειρά που επιτρέπεται από τους κανόνες πολλαπλασιασμού πινάκων και εμφανίζεται ο πίνακας γινόμενο. Στη συνέχεια ο πίνακας αυτός πολλαπλασιάζεται με τον πίνακα που έχει απομείνει και εμφανίζεται το τελικό αποτέλεσμα.

Επίσης, εμφανίζεται και ο αριθμός πράξεων που πραγματοποιήθηκαν γι αυτό το αποτέλεσμα.

Παρακάτω βλέπουμε την περίπτωση όπου ό χρήστης έχει επιλέξει την δεύτερη περίπτωση, δηλαδή να πολλαπλασιαστεί ο δεύτερος πίνακας με τον τρίτο και στη συνέχεια ο πρώτος με τον πίνακα γινόμενο.

Όπως και παραπάνω, το πρόγραμμα κάνει έλεγχο για την επιλογή του χρήστη και στη συνέχεια ελέγχει εάν οι γραμμές του δεύτερου πίνακα είναι ίσες με τις στήλες του τρίτου.

Εάν η δήλωση είναι αληθής, το πρόγραμμα προχωράει στην υλοποίση των πράξεων και στην εμφάνιση των αποτελεσμάτων.

Ύστερα εμφανίζει τον αριθμό των πράξεων που πραγματοποιήθηκαν

```
for (i=1; i<=a3_l; i++){
    printf("\n");
    for(j=1; j<=a1_r; j++){
        sum = 0;
        for(k=1; k<=a1_l; k++){
        sum = sum + (array1[k][j])*(array3[i][k]);
        }
        multipliedArray[i][j] = sum;
        printf("%03d ", multipliedArray[i][j]);
    }
}


printf("\n\n");
for (i=1; i<=a2_l; i++){
    printf("\n");
    for(j=1; j<=a1_r; j++){
        sum = 0;
        for(k=1; k<=a3_l; k++){
        sum = sum + (multipliedArray[k][j])*(array2[i][k]);
        }
        finalArray[i][j] = sum;
        printf("%04d ", finalArray[i][j]);
    }
}
```

```
num_of_mult = a1_r * a3_l * a1_l + a1_r * a2_l * a2_r;
printf("\n\nΟ αριθμός των πολλαπλασιασμών που πραγματοποιήθηκαν για αυτό το αποτέλεσμα είναι : "
       "%d\n", num_of_mult);
```

```
/*** ------------- ΔΕΥΤΕΡΗ ΠΕΡΙΠΤΩΣΗ (2*3)*1 ------------- ***/

        }else if(order == 2){
            printf("Πατήσατε 2! \n");


if (a2_l == a3_r){

/******** ΟΙ ΓΡΑΜΜΕΣ ΤΟΥ 1ΟΥ ΠΙΝΑΚΑ ΕΙΝΑΙ ΙΣΕΣ ΜΕ ΤΟΥ 3ΟΥ  *********/
```

Η δεύτερη περίπτωση όπου οι στήλες του δεύτερου πίνακα είναι ίσες με τις γραμμές του τρίτου

Η υλοποίηση και εμφάνιση της δεύτερης περίπτωσης

Τέλος, η εμφάνιση του αριθμού πράξεων του προγράμματος.

```
for (i=1; i<=a3_l; i++){
    printf("\n");
    for(j=1; j<=a2_r; j++){
        sum = 0;
        for(k=1; k<=a2_l; k++){
        sum = sum + (array2[k][j])*(array3[i][k]);
        }
        multipliedArray[i][j] = sum;
        printf("%03d ", multipliedArray[i][j]);
    }
}


printf("\n\n");
for (i=1; i<=a1_l; i++){
    printf("\n");
    for(j=1; j<=a2_r; j++){
        sum = 0;
        for(k=1; k<=a1_r; k++){
        sum = sum + (multipliedArray[k][j])*(array1[i][k]);
        }
        finalArray[i][j] = sum;
        printf("%04d ", finalArray[i][j]);
    }
}

num_of_mult = a3_l * a2_r * a3_r + a1_l * a2_r * a1_r;
printf("\n\nΟ αριθμός των πολλαπλασιασμών που πραγματοποιήθηκαν για αυτό το αποτέλεσμα είναι : "
       "%d\n", num_of_mult);


}else if (a2_r == a3_l){

/******** ΟΙ ΣΤΗΛΕΣ ΤΟΥ 1ΟΥ ΠΙΝΑΚΑ ΕΙΝΑΙ ΙΣΕΣ ΜΕ ΤΟΥ 3ΟΥ  *********/
```

4.2 Πρόγραμμα εύρεσης ελάχιστου αριθμού πράξεων για μια αλληλουχία τριών έως εννέα πινάκων

Αρχικά το πρόγραμμα παροτρύνει το χρήστη να εισάγει τον αριθμό των πινάκων που αποτελούν την αλληλουχία. Στη συνέχεια, αφού πρόκειται για αλληλουχία, ο αριθμός γραμμών του επόμενου πίνακα ισούται με τον αριθμό στηλών του προηγούμενου.

```
for (i=1; i<=a2_l; i++){
    printf("\n");
    for(j=1; j<=a3_r; j++){
        sum = 0;
        for(k=1; k<=a3_l; k++){
        sum = sum + (array2[i][k])*(array3[k][j]);
        }
        multipliedArray[i][j] = sum;
        printf("%03d ", multipliedArray[i][j]);
    }
}


printf("\n\n");
for (i=1; i<=a2_l; i++){
    printf("\n");
    for(j=1; j<=a1_r; j++){
        sum = 0;
        for(k=1; k<=a1_l; k++){
        sum = sum + (multipliedArray[i][k])*(array1[k][j]);
        }
        finalArray[i][j] = sum;
        printf("%04d ", finalArray[i][j]);
    }
}

num_of_mult = a2_l * a3_r * a2_r + a2_l * a1_r * a1_l;
printf("\n\nΟ αριθμός των πολλαπλασιασμών που πραγματοποιήθηκαν για αυτό το αποτέλεσμα είναι : "
       "%d\n", num_of_mult);
```

Γι αυτό το πρόγραμμα ζητάει μόνο τον αριθμό γραμμών των πινάκων και τέλος τον αριθμό στηλών του τελευταίου πίνακα.

Στη συνέχεια το πρόγραμμα υπολογίζει τον ελάχιστο αριθμό πράξεων για οποιονδήποτε συνδυασμό πολλαπλασιασμών της αλληλουχίας και εμφανίζει τα αποτελέσματα σε μορφή πίνακα.

```
// Ζητάμε τον αριθμό των πινάκων από το χρήστη
do{
    printf("Δώστε τον αριθμό των πινάκων που θέλετε να γίνει υπολογισμός (Από το 1 μέχρι το 9)");
    scanf("%d", &number_of_arrays);
}while(number_of_arrays<1 || number_of_arrays>9);
```

```
// Ζητάμε από το χρήστη τον αριθμό των γραμμών των πινάκων
for (i=0; i<number_of_arrays; i++){
    do{
        arrayCounter += 1;
        printf("Πληκτρολογήστε τις γραμμές του %dου πίνακα \n", arrayCounter);
        scanf("%d", &number_of_arrays_array[i]);
        if(number_of_arrays_array[i] < 1 || number_of_arrays_array[i] > 500){
            printf("Παρακαλούμε εισάγετε ξανά τις γραμμές του %dου πίνακα\n", arrayCounter);
            arrayCounter-=1;
        }
    }while(number_of_arrays_array[i] < 1 || number_of_arrays_array[i] > 500);
}

printf("Πληκτρολογήστε τις στήλες του τελευταίου πίνακα\n");
    scanf("%d", &number_of_arrays_array[number_of_arrays]);

printf("\n");
for (i=0; i<10; i++){
    printf("%d ", number_of_arrays_array[i]);
}
```

```
b = number_of_arrays;

for (i=0; i<number_of_arrays; i++){
    for (j=0; j<b; j++){
        if (i+j == (number_of_arrays-1)){
            number_of_multiplies_array[i][j] = 000000;
        }else{

            number_of_multiplies_array[i][j] = 111111;

        }
    }
    b-=1;
}
```

```
do{
    for(i=number_of_arrays - count; i>=0; i--){

        for(j=0; j<=number_of_arrays - count; j++){

            printf("\n");
            loops = number_of_arrays - (i+j);
            for(k=1; k<loops; k++){
                //printf("%d ", number_of_multiplies_array[i+loops-k][j] + number_of_multiplies_array[i][j+k] + number_of_arrays_array[j] * number_of_arrays_array[j+k] * number_of_arrays_array[j+
                if((number_of_multiplies_array[i+loops-k][j] + number_of_multiplies_array[i][j+k] + number_of_arrays_array[j] * number_of_arrays_array[j+k] * number_of_arrays_array[j+loops]) < mi

                    min = number_of_multiplies_array[i+loops-k][j] + number_of_multiplies_array[i][j+k] + number_of_arrays_array[j] * number_of_arrays_array[j+k] * number_of_arrays_array[j+loops]

                }

            }

            number_of_multiplies_array[i][j] = min;
            min = 10000000000000;
            i--;
        }
    }
    count+=1;
}while(count<=number_of_arrays);

printf("\n\n");
```

```
for (i=0; i<number_of_arrays; i++){
     printf("\n");
     for (j=0; j<number_of_arrays; j++){
          if(i+j<number_of_arrays){
          printf("%08d ", number_of_multiplies_array[i][j]);
          }
     }
}
```

Κεφάλαιο 5

Επίλογος

5.1 Συμπεράσματα

Αυτό που συμπεραίνουμε όσον αφορά τον πολλαπλασιασμό αλληλουχίας πινάκων, είναι η σημαντικότητα της εύρεσης του ελάχιστου αριθμού πράξεων, μέσω του δυναμικού προγραμματισμού. Όπως παρατηρούμε από τα παραδείγματα, ακόμη και σε μικρές αλληλουχίες πινάκων υπάρχει μεγάλη διαφορά στον αριθμό των πράξεων που πραγματοποιούνται. Όσον αφορά τον πολλαπλασιασμό πολύ μεγαλύτερων αλληλουχιών πινάκων, είναι απαραίτητη η χρήση του δυναμικού προγραμματισμού για την εύρεση του αποτελέσματος, αφού όπως είναι κατανοητό η διαφορά του χρόνου υπολογισμού ανεβαίνει κατακόρυφα.

Βιβλιογραφία

[1] Φωτάκης Δ. & Σπυράκης Π. (2001), *Αλγόριθμοι και Πολυπλοκότητα*

[2] Σταύρος Νικολόπουλος. (2015), *Σχεδιασμός και Ανάλυση Αλγορίθμων*

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. (2007), *Εισαγωγή στους Αλγορίθμους*

[4] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, Algorithms, McCraw-Hill, 2008

[5] G. J. E. Rawlings, *Αλγόριθμοι: Ανάλυση και Σύγκριση*. Εκδόσεις Κριτική, 2004

[6] A. Levitin., *Ανάλυση και Σχεδίαση Αλγορίθμων* Εκδόσεις Τζιόλα, 2007

[7] D. Kozen. *The Design and Analysis of Algorithms* Springer, 1991

[8] J. Kleinberg, E. Tardos. *Σχεδιασμός Αλγορίθμων. Κλειδάριθμος* 2008.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *Introduction to Algorithms, 2nd Edition*, MIT Press, 2001.

Συντομογραφίες - Αρκτικόλεξα - - Ακρωνύμια

βλπ	βλέπε
κ.λπ.	και λοιπά
κ.ο.κ	και ούτω καθεξής
ΤΕΙ	Τεχνολογικό Εκπαιδευτικό Ίδρυμα

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος