Πανεπιστήμιο Ιωαννίνων
Τμήμα Πληροφορικής

Μεταπτυχιακή Εργασία Ειδίκευσης

ΔΕΝΔΡΟΕΙΔΕΙΣ ΔΟΜΕΣ ΓΙΑ ΟΥΡΕΣ ΠΡΟΤΕΡΑΙΟΤΗΤΑΣ ΣΕ ΣΥΣΤΗΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΗΣ ΔΙΑΚΡΙΤΩΝ ΓΕΓΟΝΟΤΩΝ

Αικατερίνη Ασδρέ

Επιβλέπων Καθηγητής
Σταύρος Δ. Νικολόπουλος

Σεπτέμβριος 2001

ΕΥΧΑΡΙΣΤΙΕΣ

Με την αποπεράτωση της παρούσης εργασίας θα ήθελα να απευθύνω τις ιδιαίτερες ευχαριστίες μου στον επιβλέποντα καθηγητή Σταύρο Δ. Νικολόπουλο, χάρη στην προτρεπτική υπομονή του οποίου ολοκληρώθηκε, πιστεύω επιτυχώς, αυτή η προσπάθεια. Βασιζόμενη στην πολύτιμη γνώση του για το αντικείμενο και στην προθυμία του για βοήθεια, θεωρώ τη συμβολή του ουσιαστική για το πέρας αυτής της μελέτης. Ελπίζω ότι το αποτέλεσμα της συνεργασίας μας θα αποβεί εποικοδομητικό και θα αποτελέσει τη βάση για περαιτέρω έρευνα.

ΠΕΡΙΛΗΨΗ

Στόχος αυτής της εργασίας είναι η μελέτη, τόσο θεωρητικά όσο και πειραματικά, των αλγορίθμων και των δομών δεδομένων που χρησιμοποιούνται σε συστήματα προσομοίωσης διακριτών γεγονότων. Τέτοιου είδους συστήματα έχουν σαν σκοπό τη βελτιστοποίηση του χρόνου απόκρισης σε συστήματα με ουρές προτεραιότητας. Για το σκοπό αυτό μελετώνται πειραματικά και θεωρητικά δενδροειδείς δομές και αλγόριθμοι και γίνεται σύγκριση της απόδοσής τους και της συμπεριφοράς τους με σκοπό να προσδιοριστούν εκείνοι οι αλγόριθμοι και οι δομές δεδομένων οι οποίες θα είναι οι πιο κατάλληλες (αποτελεσματικές) για να προσομοιώσουν μια ουρά προτεραιότητας σε ένα σύστημα προσομοίωσης διακριτών γεγονότων. Για την εκτίμηση της αποτελεσματικότητας των αλγορίθμων χρησιμοποιείται το μοντέλο HOLD και η τεχνική Event Horizon. Το σύνολο των δενδροειδών δομών δεδομένων που μελετήθηκε για να χρησιμοποιηθεί για την αποθήκευση του συνόλου γεγονότων αποτελείται από ήδη υπάρχουσες δομές αλλά και από νέες που προτείνονται. Συγκεκριμένα, οι προτεινόμενες δομές είναι επεκτάσεις της δομής του σωρού και του *P*-tree που ονομάζονται *Multiple* heap, *n*-heap και *Indexed P*-tree και *Multiple P*-tree, αντίστοιχα. Τέλος, μελετάται και η παράλληλη προσομοίωση διακριτών γεγονότων και προτείνονται παράλληλοι αλγόριθμοι που χρησιμοποιούν της δομές που μελετήθηκαν ακολουθιακά.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Η μελέτη συστημάτων με μαθηματικές μεθόδους απαιτεί αφενός πλήρη γνώση του υπάρχοντος ή προτεινόμενου συστήματος, αφετέρου δυνατότητα αναπαράστασης του συστήματος με μαθηματικά μοντέλα. Οι παραπάνω όμως προϋποθέσεις σχεδόν ποτέ δεν πληρούνται σε πολύπλοκα συστήματα, γεγονός που οδήγησε στην ανάπτυξη άλλων μεθόδων για τη μελέτη και ανάλυσή τους, οι οποίες αν και δεν είναι τόσο ακριβείς όσο οι μαθηματικές μέθοδοι προσφέρουν σημαντικά πλεονεκτήματα. Μια τέτοια μέθοδος είναι η προσομοίωση, η οποία αποτελεί μία πειραματική μέθοδο που έχει σαν σκοπό τη βελτιστοποίηση, ανάλυση και μελέτη της λειτουργίας των συστημάτων. Η προσομοίωση εξαρτάται από το μοντέλο που θα χρησιμοποιηθεί για το σύστημα καθώς και από την επιλογή των παραμέτρων που απαιτούνται για την εξαγωγή αξιόπιστων και χρήσιμων συμπερασμάτων. Στο Κεφάλαιο αυτό γίνεται μια εισαγωγή στα χαρακτηριστικά των συστημάτων και των μοντέλων προσομοίωσης καθώς και στα βασικά χαρακτηριστικά της μεθοδολογίας προσομοίωσης. Αναλύονται επίσης οι τύποι των μοντέλων που χρησιμοποιούνται για προσομοίωση και οι μέθοδοι προσομοίωσης που αντιστοιχούν σε κάθε τύπο.

1.1 Προσομοίωση και Εξομοίωση

Ο όρος προσομοίωση (simulation) συγχέεται συχνά με τον όρο εξομοίωση (emulation) αν και οι όροι αυτοί δηλώνουν τελείως διαφορετικές μεθοδολογίες. Η **προσομοίωση** είναι μέθοδος μελέτης ενός συστήματος και εξοικείωσης με τα χαρακτηριστικά του, με τη βοήθεια ενός άλλου συστήματος το οποίο στις περισσότερες περιπτώσεις είναι ηλεκτρονικός υπολογιστής. Η **εξομοίωση** είναι μέθοδος αναπαραγωγής ενός συστήματος εντός ή μέσω ενός άλλου συστήματος παρόμοιου με το πρώτο. Είναι λοιπόν εμφανές ότι κατά την προσομοίωση δεν πρέπει να υπάρχει ούτε η εντύπωση ούτε η επιθυμία υλοποίησης του πραγματικού συστήματος αφού σκοπός είναι η μελέτη του συστήματος και όχι η χρήση του. Αντίθετα, κατά την εξομοίωση, υπάρχει η εντύπωση υλοποίησης στο πραγματικό σύστημα αφού σκοπός είναι η χρήση του.

1.2 Ορισμός του Συστήματος και Ιδιότητες Συστημάτων

Η προσομοίωση χρησιμοποιείται, όπως προαναφέρθηκε, για τη μελέτη συστημάτων. Είναι επομένως απαραίτητο να ορισθεί η έννοια του συστήματος και τα συστατικά του στοιχεία καθώς και να εξεταστούν οι ιδιότητές του.

Σύστημα είναι ένα σύνολο αλληλεπιδρώντων στοιχείων τα οποία συνεργάζονται μεταξύ τους ή λειτουργούν συλλογικά για την επίτευξη κάποιου σκοπού. Η μελέτη συστημάτων αφορά τόσο στην ανάλυσή τους, όταν πρόκειται για υπάρχοντα συστήματα, όσο και στη σύνθεσή τους, όταν πρόκειται για συστήματα που σχεδιάζονται. Η ανάλυση ορίζεται ως ο καθορισμός της εξόδου του συστήματος όταν δοθεί σε αυτό η είσοδος. Η μεθοδολογία αυτή χρησιμοποιείται επομένως όταν είναι γνωστά τα στοιχεία του συστήματος και επιδιώκεται να διαπιστωθεί η λειτουργία του και να καθορισθεί η αξιοπιστία του. Η σύνθεση ορίζεται ως ο καθορισμός των στοιχείων του συστήματος όταν δοθούν οι είσοδοι και οι έξοδοι που αντιστοιχούν σε αυτές τις εισόδους. Η μεθοδολογία αυτή χρησιμοποιείται κατά το σχεδιασμό ενός συστήματος.

Τα συστήματα αποτελούνται από οντότητες, χαρακτηριστικά και ιδιότητες. **Οντότητα** είναι κάθε αντικείμενο του συστήματος που ενδιαφέρει το μελετητή. Ανάλογα με την περίπτωση και τους σκοπούς της μελέτης ακόμη και το ίδιο το σύστημα αποτελεί μια οντότητα. Οι ιδιότητες των οντοτήτων αποτελούν τα **χαρακτηριστικά**. **Δραστηριότητα** ονομάζεται κάθε διεργασία που προκαλεί αλλαγές στο σύστημα.

Ένα πολύ σημαντικό στοιχείο που χαρακτηρίζει ένα σύστημα είναι η **κατάσταση του συστήματος**, που ορίζεται ως η συνολική περιγραφή των οντοτήτων, των χαρακτηριστικών τους και των δραστηριοτήτων σε μια δεδομένη χρονική στιγμή. Η προσομοίωση ασχολείται ακριβώς με την παρακολούθηση της κατάστασης ενός συστήματος, όπως αυτή μεταβάλλεται με την πάροδο του χρόνου. Η κατάσταση ενός συστήματος όμως μπορεί να μην εξαρτάται μόνο από τις δραστηριότητες που λαμβάνουν χώρα μέσα στο σύστημα αλλά και από δραστηριότητες εκτός του συστήματος. Για το λόγο αυτό ορίζεται ως **περιβάλλον του συστήματος** το σύνολο των μεταβολών που συμβαίνουν εκτός του συστήματος. Τα συστήματα χωρίζονται σε κατηγορίες ανάλογα με τις μεταβολές της κατάστασής τους ή τη

σχέση τους με το περιβάλλον.

Στα **συνεχή συστήματα** οι μεταβολές της κατάστασης είναι κατά κύριο λόγο ομαλές. Οι δραστηριότητες δηλαδή μεταβάλλουν συνεχώς την κατάσταση του συστήματος και όχι μόνο όταν τελειώσουν. Παράδειγμα τέτοιου συστήματος είναι ένα αυτοκίνητο.

Στα **διακριτά συστήματα** οι μεταβολές είναι κυρίως ασυνεχείς πράγμα που σημαίνει ότι η κατάσταση του συστήματος αλλάζει μόλις τελειώσει μια δραστηριότητα. Παράδειγμα διακριτού συστήματος είναι μια τράπεζα.

Στην πραγματικότητα βέβαια όλα τα συστήματα είναι ασυνεχή στη φύση. Κατά τη μελέτη των συστημάτων όμως, πολλές φορές έχουν ενδιαφέρον οντότητες και χαρακτηριστικά που εμφανίζουν ασυνεχείς μεταβολές. Στο παράδειγμα της τράπεζας, ενδιαφέρον παρουσιάζει ο συνολικός χρόνος αναμονής ενός πελάτη στην ουρά. Το χαρακτηριστικό αυτό αποκτά τιμή μόνο όταν ο πελάτης βγει από την ουρά και αρχίσει να εξυπηρετείται. Επομένως η κατάσταση του συστήματος μεταβάλλεται μόνο σε διακριτές χρονικές στιγμές, μια από τις οποίες είναι η έναρξη εξυπηρέτησης ενός πελάτη.

1.3 Μοντέλα Συστημάτων

Η μελέτη των συστημάτων, είτε με μαθηματικές μεθόδους είτε με προσομοίωση, δε γίνεται με το ίδιο το σύστημα αλλά με ένα μοντέλο του συστήματος. Μερικοί από τους λόγους για τους οποίους κατασκευάζεται ένα μοντέλο είναι η ανεύρεση εναλλακτικών λύσεων και η βελτιστοποίηση της απόδοσης του συστήματος, η πρόβλεψη της συμπεριφοράς του συστήματος και η διευκόλυνση στην κατανόησή του.

Μοντέλο είναι το σύνολο των πληροφοριών ενός συστήματος που έχει συγκεντρωθεί με σκοπό τη μελέτη του συστήματος. Το μοντέλο θα πρέπει να αντιπροσωπεύει το σύστημα όσο πιο πιστά γίνεται έτσι ώστε τα συμπεράσματα που θα εξαχθούν από τη μελέτη του μοντέλου να αντιστοιχούν σε συμπεράσματα για το σύστημα. Τα μοντέλα μπορούν να ταξινομηθούν σε μερικές κατηγορίες με βάση συγκεκριμένα κριτήρια. Έτσι υπάρχουν φυσικά και μαθηματικά μοντέλα, τα μαθηματικά μπορεί να είναι αριθμητικά ή αναλυτικά ενώ τόσο τα φυσικά όσο και τα μαθηματικά μπορεί να είναι είτε στατικά είτε δυναμικά.

1.4 Δημιουργία Μοντέλων Προσομοίωσης

Τα μοντέλα προσομοίωσης είναι κυρίως αριθμητικά δυναμικά μαθηματικά μοντέλα. Όλα τα μοντέλα προσομοίωσης είναι περιγραφικά μοντέλα, με την έννοια ότι χρησιμοποιούνται αλγόριθμοι, υπολογιστικές μέθοδοι και διεργασίες για να περιγράψουν τη λειτουργία του συστήματος. Σχεδόν όλα τα μοντέλα προσομοίωσης αποτελούνται από κάποιο συνδυασμό των παρακάτω στοιχείων:

1. Συστατικά
2. Μεταβλητές
3. Παραμέτρους
4. Λειτουργικές Σχέσεις
5. Περιορισμούς
6. Συναρτήσεις Κριτηρίων

Η προσομοίωση συνεπώς δεν είναι κάποια θεωρία αλλά μια απλή μεθοδολογία επίλυσης προβλημάτων, μια από τις πολλές που υπάρχουν.

Δημιουργία Μοντέλων Προσομοίωσης

Η δημιουργία μοντέλων προσομοίωσης είναι μια δύσκολη τεχνική γιατί πρέπει να εξισορροπήσει αντικρουόμενους παράγοντες. Αφενός το μοντέλο θα πρέπει να είναι απλό ώστε να είναι δυνατόν να κατασκευαστεί και να μελετηθεί, αφετέρου θα πρέπει να είναι αρκετά πολύπλοκο έτσι ώστε να αντιπροσωπεύει όσο πιο πιστά γίνεται το σύστημα που πρόκειται να μελετηθεί.

Η ισορροπία αυτή μπορεί να επιτευχθεί με προσεκτική ανάλυση του μοντέλου. Αν το μοντέλο που έχει κατασκευαστεί είναι απλό και κατανοητό τότε μπορεί να εμπλουτιστεί, αντίθετα αν είναι πολύπλοκο και δυσνόητο μπορεί να απλοποιηθεί. Ο στόχος είναι τελικά να κατασκευαστεί ένα μοντέλο το οποίο προσαρμόζεται και εξελίσσεται εύκολα.

1.5 Βήματα και Χαρακτηριστικά της Προσομοίωσης

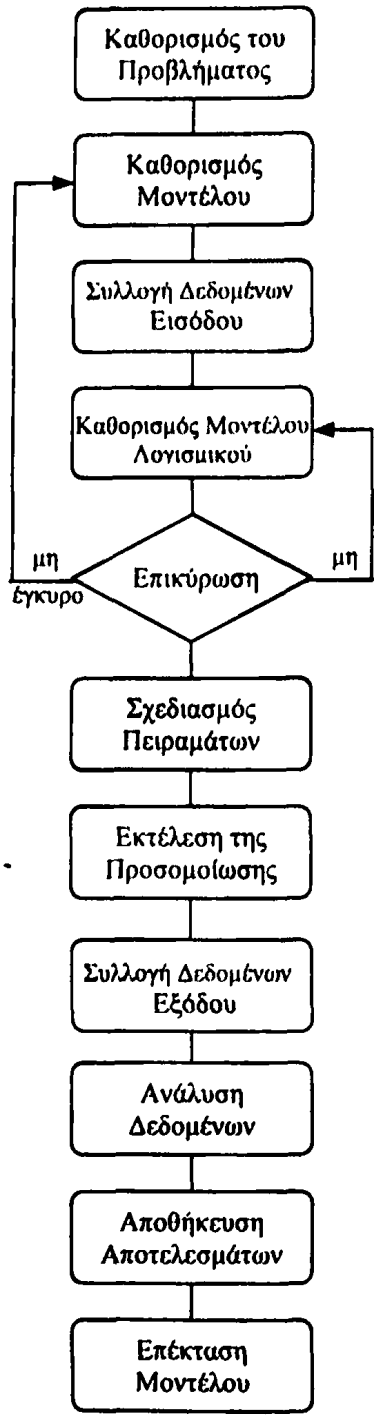
Περιγραφή των Βημάτων της Διαδικασίας της Προσομοίωσης

Η διαδικασία της προσομοίωσης περιλαμβάνει τις εξής φάσεις, οι οποίες παρουσιάζονται στο Σχήμα 1.1.

Καθορισμός του προβλήματος. Το πρώτο βήμα της διαδικασίας είναι ο καθορισμός του προβλήματος που επιλύει το μοντέλο. Καθορίζεται ακριβώς ο χώρος που ενδιαφέρει καθώς και η ζητούμενη ακρίβεια των αποτελεσμάτων.

Καθορισμός μοντέλου. Αυτή η φάση περιλαμβάνει τον καθορισμό των αλγορίθμων που θα χρησιμοποιηθούν προκειμένου να καθοριστεί ο τρόπος που θα περιγραφεί το σύστημα, τα δεδομένα εισόδου και τα δεδομένα που θα προκύψουν. Περιγράφεται ακόμη ο συνολικός χρόνος, το προσωπικό και τα μέσα που θα χρειαστούν.

Συλλογή δεδομένων εισόδου. Σε αυτή τη φάση συλλέγονται πληροφορίες που θα χρησιμοποιηθούν είτε ως παράμετροι εισόδου είτε στην αξιολόγηση της απόδοσης της προσομοίωσης είτε θα βοηθήσουν στην ανάπτυξη των αλγορίθμων. Αποτελεί μία από τις πιο δύσκολες φάσεις αλλά και από τις πιο ευάλωτες σε λάθη.



Σχήμα 1.1

Καθορισμός μοντέλου λογισμικού. Μαθηματικές και λογικές περιγραφές του πραγματικού συστήματος σε αυτή τη φάση κωδικοποιούνται έτσι ώστε να μπορούν να εκτελεστούν από υπολογιστή. Η δημιουργία και η υλοποίηση ενός προγράμματος προσομοίωσης γίνεται με βάση τις αρχές της τεχνολογίας λογισμικού.

Επικύρωση. Ένα σημαντικό βήμα κατά το οποίο επιβεβαιώνεται ότι οι αλγόριθμοι του μοντέλου, τα δεδομένα εισόδου και οι παραδοχές είναι όλα ορθά.

Σχεδιασμός πειραμάτων. Σε αυτή τη φάση αναζητούνται παραγωγικές και ακριβείς μέθοδοι εκτέλεσης της προσομοίωσης, έτσι ώστε να προκύψουν τα επιθυμητά αποτελέσματα.

Εκτέλεση της προσομοίωσης. Πρόκειται για τη φάση κατά την οποία εκτελείται το πρόγραμμα προσομοίωσης.

Συλλογή δεδομένων εξόδου. Κατά τη διάρκεια της εκτέλεσης του προγράμματος τα αποτελέσματα που θα προκύψουν συγκεντρώνονται, ομαδοποιούνται και τέλος αποθηκεύονται.

Ανάλυση δεδομένων. Τα αποτελέσματα της εκτέλεσης του προγράμματος επεξεργάζονται, παίρνουν κατάλληλες μορφές προκειμένου να προκύψουν οι ζητούμενες πληροφορίες.

Αποθήκευση αποτελεσμάτων. Σε αυτό το σημείο τα αποτελέσματα διανέμονται στους ενδιαφερόμενους και εξετάζεται κατά πόσο έχουν απαντηθεί τα αρχικά ερωτήματα.

Επέκταση μοντέλου. Τα μοντέλα προσομοίωσης είναι ακριβά και υλοποιούνται δύσκολα. Για το λόγο αυτό, αφού πραγματοποιηθεί ένα μοντέλο, θα τροποποιηθεί και θα επεκταθεί προκειμένου να χρησιμοποιηθεί και για την επίλυση άλλων παρόμοιων προβλημάτων.

Η κατασκευή του μοντέλου αποτελεί ίσως το πιο σημαντικό βήμα για την προσομοίωση του συστήματος, αφού η ποιότητα και η αξιοπιστία του καθορίζουν και την αξιοπιστία της προσομοίωσης. Στο επόμενο Κεφάλαιο δίνεται αναλυτικά η μεθοδολογία ανάπτυξης μοντέλων για διακριτά συστήματα, τα οποία είναι τα πιο κατάλληλα για προσομοίωση. Στη συνέχεια αυτού του κεφαλαίου αναπτύσσεται η μεθοδολογία ελέγχου του χρόνου καθώς το μοντέλο εξελίσσεται κατά τη διάρκεια της προσομοίωσης. Θα πρέπει να σημειωθεί ότι ο χρόνος που χρησιμοποιείται κατά την προσομοίωση αποτελεί μοντελοποίηση του χρόνου του συστήματος. Επομένως, ο προσομοιωμένος χρόνος δεν έχει καμία σχέση με τον πραγματικό χρόνο που παρέρχεται όταν εκτελείται η προσομοίωση.

Μηχανισμοί Διαχείρισης Χρόνου

Η προσομοίωση αποτελεί ένα μοντέλο του συστήματος στο οποίο μελετώνται οι χρονικές μεταβολές της κατάστασης του συστήματος. Είναι απαραίτητο επομένως να χρησιμοποιηθεί κάποιος μηχανισμός ο οποίος θα καταγράφει την πάροδο του χρόνου και θα ελέγχει αυτές τις μεταβολές.

Οι μηχανισμοί χρόνου βασίζονται στα γεγονότα που συμβαίνουν κατά την προσομοίωση. **Γεγονός** είναι μια αλλαγή της κατάστασης του συστήματος, η οποία λαμβάνει χώρα σε κάποια συγκεκριμένη χρονική στιγμή. Το τμήμα της προσομοίωσης που ασχολείται με την παρέλευση του χρόνου ονομάζεται **Μηχανισμός Ροής Χρόνου** (ΜΡΧ). Το τμήμα αυτό, αφενός αυξάνει τον προσομοιωμένο χρόνο και αφετέρου προσφέρει τον απαιτούμενο συγχρονισμό ανάμεσα στα υπόλοιπα τμήματα της προσομοίωσης καθώς εμφανίζονται τα διάφορα γεγονότα. Οι δύο βασικοί ΜΡΧ που υπάρχουν είναι:

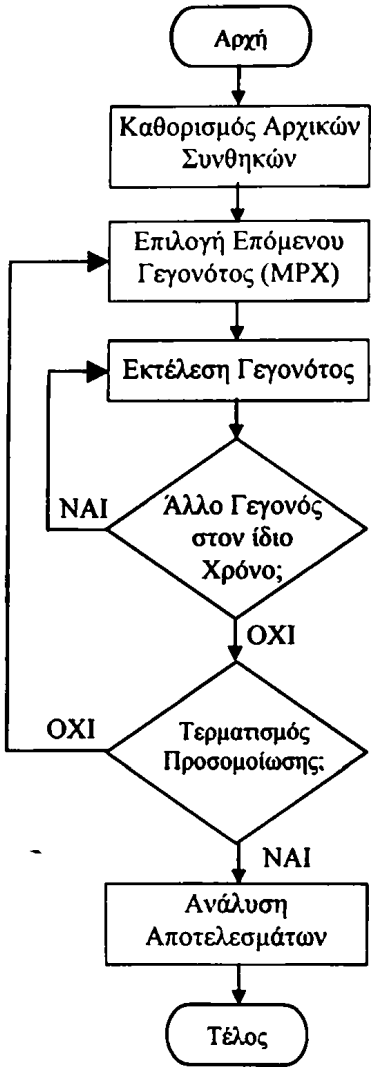
1. **Μηχανισμός Επόμενου Γεγονότος**. Με τον μηχανισμό αυτό, καθορίζεται η χρονική στιγμή κατά την οποία θα συμβεί το επόμενο γεγονός και το ρολόι της προσομοίωσης προχωρά σε αυτή τη χρονική στιγμή προσπερνώντας όλο τον ενδιάμεσο χρόνο, κατά τον οποίο δε συμβαίνει τίποτα. Η εφαρμογή της μεθόδου αυτής περιλαμβάνει την ύπαρξη ενός "καταλόγου" ή "λίστας" γεγονότων όπου καταγράφονται τα γεγονότα που πρόκειται να συμβούν στο μέλλον.

2. **Μηχανισμός Σταθερού Χρονικού Διαστήματος.** Σύμφωνα με το μηχανισμό αυτό, ο χρόνος, δηλαδή το ρολόι της προσομοίωσης, αυξάνει κατά ένα μικρό και σταθερό χρονικό διάστημα. Όλα τα γεγονότα που εμφανίζονται τη συγκεκριμένη χρονική στιγμή, καθώς και όσα συνέβησαν κατά το διάστημα που μεσολάβησε από την προηγούμενη χρονική στιγμή μέχρι αυτή, θεωρούνται ότι συμβαίνουν κατά τη συγκεκριμένη χρονική στιγμή.

Το γενικό διάγραμμα ΜΡΧ δίνεται στο Σχήμα 1.2. Ο μηχανισμός ροής χρόνου του επόμενου γεγονότος χρησιμοποιείται κυρίως σε διακριτά συστήματα ενώ ο μηχανισμός του σταθερού διαστήματος χρησιμοποιείται για την προσομοίωση συνεχών συστημάτων.

Για την καλύτερη μελέτη και σύγκριση των δύο μηχανισμών χρόνου εξετάζεται το παράδειγμα ενός απλού συστήματος ουράς. Στο σύστημα αυτό υπάρχει ένας εξυπηρετητής, για την εξυπηρέτηση των πελατών και η ουρά που σχηματίζεται μπροστά από αυτόν. Οι πελάτες φτάνουν στο σύστημα και παραμένουν στην ουρά μέχρι να εξυπηρετηθούν. Μόλις τελειώσει η εξυπηρέτηση ενός πελάτη, αυτός φεύγει από το σύστημα και αρχίζει η εξυπηρέτηση του επόμενου πελάτη που βρίσκεται στην ουρά. Η κατανομή του χρόνου που μεσολαβεί ανάμεσα σε δύο αφίξεις, καθώς και του χρόνου εξυπηρέτησης θεωρούνται γνωστές. Επίσης, η ουρά είναι FIFO, δηλαδή πρώτος αφικνούμενος πρώτος εξυπηρετούμενος και θεωρείται ότι έχει άπειρο μήκος. Οι συμβολισμοί που χρησιμοποιούνται στα διαγράμματα που ακολουθούν παρουσιάζονται στον Πίνακα 1.1.

Στο Σχήμα 1.3 δίνεται το διάγραμμα ροής της προσομοίωσης στην οποία χρησιμοποιείται ο μηχανισμός ροής χρόνου του επόμενου γεγονότος. Στο διάγραμμα αυτό δε φαίνεται μόνο ο μηχανισμός ροής χρόνου αλλά και ολόκληρη η προσομοίωση, στην οποία η ποσότητα που πρέπει να υπολογισθεί είναι ο μέσος χρόνος των πελατών στο σύστημα.



Σχήμα 1.2. Διάγραμμα Ροής της Προσομοίωσης

T	Χρόνος
AI	Χρόνος μεταξύ αφίξεων (τυχαία μεταβλητή)
A(I)	Χρόνος άφιξης του I πελάτη
D(I)	Χρόνος τέλους εξυπηρέτησης του I πελάτη
ST	Χρόνος εξυπηρέτησης (τυχαία μεταβλητή)
N	Αριθμός πελατών στο σύστημα
NMAX	Αριθμός πελατών που θα εξυπηρετηθούν
TSUM	Άθροισμα των χρόνων στο σύστημα
TMEAN	Εκτίμηση του μέσου χρόνου στο σύστημα
DT	Χρόνος τέλους εξυπηρέτησης του πελάτη
M	Αριθμός των πελατών που εξυπηρετήθηκαν

Πίνακας 1.1

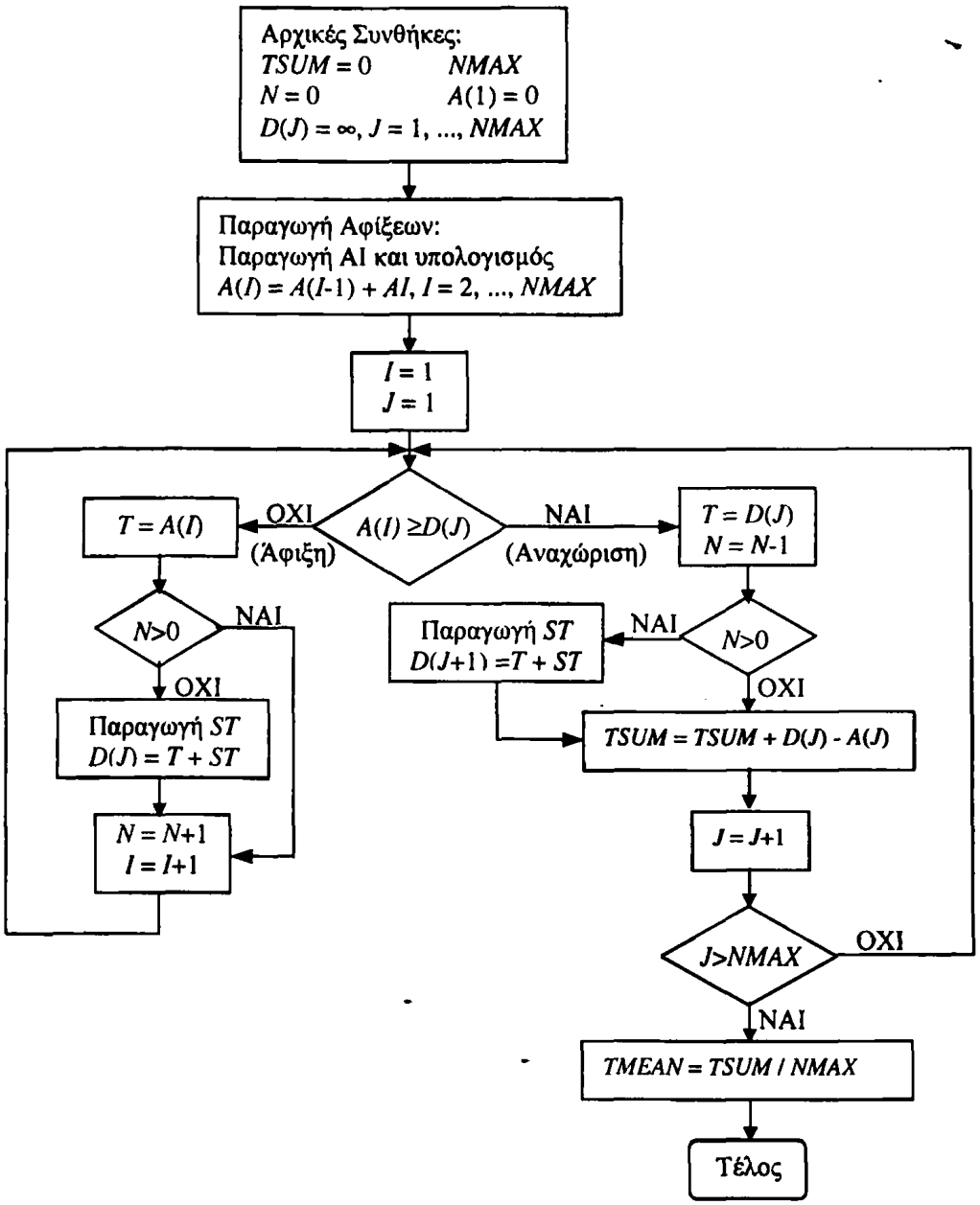
Έστω ότι οι αρχικές τιμές των χρόνων άφιξης και αναχώρησης των πελατών (που υπολογίζονται κατά την αρχικοποίηση της προσομοίωσης) είναι αυτές που δίνονται στον Πίνακα 1.2

I	*A(I)*	*D(I)*
1	0	∞
2	6	∞
3	15	∞
4	33	∞
.	.	.
.	.	.
.	.	.

Πίνακας 1.2

Έστω επίσης ότι οι χρόνοι εξυπηρέτησης είναι: $ST = \{9, 14, 11, ...\}$. Σύμφωνα με το μηχανισμό επόμενου γεγονότος, το ρολόι της προσομοίωσης (T) σταματά μόνο στις χρονικές στιγμές κατά τις οποίες συμβαίνουν γεγονότα. Επομένως, στην περίπτωση αυτή, το ρολόι θα σταματήσει μόνο στις χρονικές στιγμές αφίξεων πελατών στο σύστημα και στις χρονικές στιγμές αναχωρήσεων πελατών από το σύστημα.

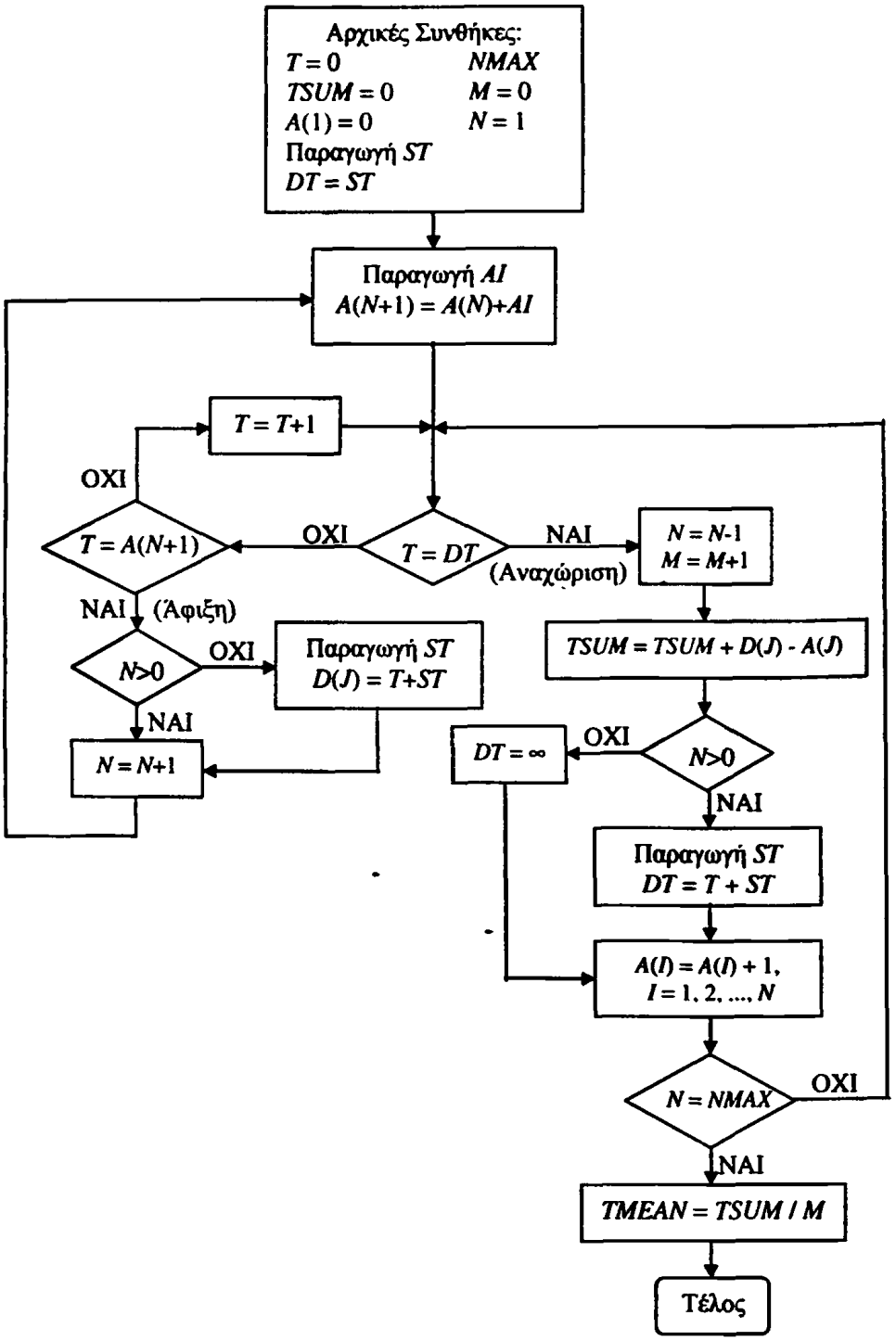
Στην περίπτωση του μηχανισμού ροής χρόνου σταθερού διαστήματος, το ρολόι της προσομοίωσης αυξάνει κάθε φορά κατά δεδομένο χρονικό διάστημα. Το διάστημα αυτό θα πρέπει να είναι αρκετά μικρό ώστε να ελαχιστοποιείται η πιθανότητα να συμβούν δύο ανεξάρτητα γεγονότα κατά τη διάρκεια του διαστήματος αυτού. Διαφορετικά δεν θα προσομοιωθεί σωστά η αλληλουχία δύο ή περισσότερων γεγονότων που συμβαίνουν στο χρονικό αυτό διάστημα επειδή δεν θα είναι γνωστό ποιο συνέβη πρώτο και ποιο δεύτερο. Αν για την προσομοίωση υποτεθεί ότι οι χρόνοι λαμβάνουν μόνο ακέραιες τιμές, τότε το διάστημα μπορεί να τεθεί ίσο με μια χρονική μονάδα. Έτσι διατηρείται η σωστή αλληλουχία των γεγονότων, αφού δύο γεγονότα θα συμβούν είτε ταυτόχρονα είτε σε διαφορετικά χρονικά διαστήματα.



Σχήμα 1.3. Διάγραμμα ροής του μηχανισμού επόμενου γεγονότος

Στο Σχήμα 1.4 δίνεται το διάγραμμα ροής της προσομοίωσης στην οποία χρησιμοποιείται ο MPX σταθερού χρονικού διαστήματος. Η προσομοίωση χρησιμοποιείται πάλι για τον υπολογισμό του μέσου χρόνου που καταναλώνουν οι πελάτες στο σύστημα. Όπως φαίνεται και στο διάγραμμα, οι χρόνοι άφιξης δεν υπολογίζονται όλοι κατά την αρχικοποίηση, αλλά κατά τη διάρκεια εκτέλεσης της προσομοίωσης. Με το MPX σταθερού διαστήματος, το ρολόι της προσομοίωσης σταματά σε όλες τις χρονικές στιγμές που καθορίζονται από το μέγεθος του σταθερού διαστήματος, ανεξάρτητα από το αν συμβαίνει κάποιο γεγονός ή όχι.

Το βασικό μειονέκτημα του δεύτερου σε σχέση με το μηχανισμό επόμενου γεγονότος είναι η άσκοπη αύξηση του ρολογιού προσομοίωσης κατά το σταθερό χρονικό διάστημα και ο συνεχής έλεγχος για να διαπιστωθεί αν κάποιο γεγονός έχει συμβεί. Αντίθετα, ο μηχανισμός επόμενου γεγονότος μειονεκτεί έναντι του μηχανισμού σταθερού διαστήματος γιατί περιλαμβάνει πιο πολύπλοκο μηχανισμό προγραμματισμού των επόμενων γεγονότων.



Σχήμα 1.4. Διάγραμμα ροής του μηχανισμού σταθερού διαστήματος

Παραγωγή Τυχαίων Αριθμών

Πολλά μοντέλα απαιτούν τη χρήση τυχαίων αριθμών λόγω της στατιστικής περισσότερο παρά ντετερμινιστικής παρουσίασης των γεγονότων. Οι γεννήτορες τυχαίων αριθμών χρησιμοποιούνται από τους υπολογιστές για τη δημιουργία μιας ακολουθίας αριθμών που είναι

τυχαίοι και ανεξάρτητοι ο ένας από τον άλλον. Συνήθως οι αλγόριθμοι πρέπει να είναι γρήγοροι, οικονομικοί από άποψη μνήμης και συνήθως παράγουν ομοιόμορφα κατανεμημένους αριθμούς στο διάστημα [0,1]. Για τη δημιουργία μεταβλητών από άλλες κατανομές, οι ομοιόμορφοι τυχαίοι αριθμοί γίνονται είσοδοι σε έναν άλλο αλγόριθμο που παράγει κανονικά ή εκθετικά κατανεμημένους αριθμούς, ή αριθμούς σύμφωνα με κατανομές όπως η Beta, Poisson, Gamma, Binomial .

1.6 Η Τεχνολογία στην Προσομοίωση

Η ανάγκη για τη δημιουργία πολύπλοκων μοντέλων προσομοίωσης συνήθως προηγείται της ικανότητας αναπαράστασης του υλικού και του λογισμικού. Οι εφαρμογές της προσομοίωσης διευρύνονται συνεχώς αφού παρέχονται εργαλεία αρκετά ισχυρά ώστε να αναπαριστώνται τα προβλήματα. Μερικά χρήσιμα επιτεύγματα της τεχνολογίας περιγράφονται παρακάτω.

Δίκτυα. Η ικανότητα να κατανεμηθεί η προσομοίωση σε ένα δίκτυο υπολογιστών οδηγεί στην κατασκευή πιο πολύπλοκων και λεπτομερών μοντέλων.

Παράλληλη Επεξεργασία. Η παράλληλη επεξεργασία παρέχει πολλά από τα πλεονεκτήματα των κατανεμημένων προσομοιώσεων. Μερικά προβλήματα μπορούν να διαιρεθούν σε πολλές διαφορετικές διεργασίες αλλά η επικοινωνία μεταξύ αυτών είναι τόσο συχνή που εισάγονται μεγάλες καθυστερήσεις. Σε αυτές τις περιπτώσεις η παράλληλη επεξεργασία μπορεί να αποτελεί μία αποδοτική λύση.

Τεχνητή Νοημοσύνη. Η αναπαράσταση της ανθρώπινης συμπεριφοράς έχει γίνει πολύ απαραίτητη σε μερικές περιπτώσεις προσομοίωσης. Όταν η προσομοίωση περιλαμβάνει τεχνητή νοημοσύνη, έμπειρα συστήματα ή νευρωνικά δίκτυα η αναπαράσταση αυτή γίνεται πιο σωστά και πιο ρεαλιστικά.

Γραφικά. Η διεπαφή χρήστη (user interface) μπορεί να παρέχει εύκολη υλοποίηση και λειτουργία του μοντέλου προσομοίωσης, εύκολη ανάλυση και παρουσίαση δεδομένων. Αυτά τα εργαλεία κάνουν πιο ελκυστική τη διαδικασία της προσομοίωσης.

Βάσεις Δεδομένων. Η διαδικασία της προσομοίωσης μπορεί να παράγει έναν πολύ μεγάλο αριθμό πληροφοριών που χρειάζονται ανάλυση ή ακόμη μπορεί να απαιτεί ως είσοδο μεγάλο όγκο δεδομένων. Οι βάσεις δεδομένων καθιστούν ευκολότερη και πιο αποδοτική τη διαδικασία.

World Wide Web. Η ανάπτυξη του διαδικτύου (internet) έχει οδηγήσει σε πειράματα με προσομοιώσεις που είτε είναι κατανεμημένες μέσα στο internet είτε προσπελάσιμες από αυτό. Έτσι οι χρήστες δεν χρειάζεται να κατέχουν έναν συγκεκριμένο υπολογιστή προκειμένου να εκτελέσουν το πρόγραμμα της προσομοίωσης αρκεί να αποκτήσουν πρόσβαση σε μία συγκεκριμένη μηχανή για προσομοίωση που είναι συνδεδεμένη στο δίκτυο.

Γλώσσες για την Προσομοίωση Διακριτών Γεγονότων. Μερικά παραδείγματα γλωσσών προγραμματισμού που χρησιμοποιούνται για την προσομοίωση διακριτών

γεγονότων (discrete event simulation) είναι η Simula, η GPSS/H, η SIMSCRIPT II.5, η SIMAN/Cinema, η SLAM II και η MODSIM.

Γλώσσες για την Προσομοίωση Συνεχούς Χρόνου. Οι επικρατέστερες μέχρι τώρα γλώσσες προγραμματισμού που χρησιμοποιούνται για την προσομοίωση συνεχούς χρόνου (continuous simulation) είναι η ACSL (Advanced Continuous Simulation Language και η CSMP (Continuous System Modeling Program).

Γλώσσες για Διαλογική Προσομοίωση. Μερικά παραδείγματα γλωσσών προγραμματισμού που χρησιμοποιούνται για την διαλογική προσομοίωση (interactive simulation), η οποία χρησιμοποιείται κυρίως σε στρατιωτικές εφαρμογές είναι η VRLink, η ITEMS, η FLAMES και η MultiGenII.

1.7 Στόχος της Εργασίας

Στόχος αυτής της εργασίας είναι η μελέτη, τόσο θεωρητικά όσο και πειραματικά, των αλγορίθμων και των δομών δεδομένων που χρησιμοποιούνται σε συστήματα προσομοίωσης διακριτών γεγονότων. Τέτοιου είδους συστήματα έχουν σαν σκοπό τη βελτιστοποίηση του χρόνου απόκρισης σε συστήματα με ουρές προτεραιότητας.

Για το σκοπό αυτό μελετώνται δενδροειδείς δομές και αλγόριθμοι με σκοπό να προσδιοριστούν εκείνοι οι αλγόριθμοι και οι δομές δεδομένων οι οποίες θα είναι οι πιο κατάλληλες (αποτελεσματικές) για να προσομοιώσουν μια ουρά προτεραιότητας σε ένα σύστημα προσομοίωσης διακριτών γεγονότων.

1.8 Θέματα Μελέτης της Εργασίας

Για την εκτίμηση της αποτελεσματικότητας των αλγορίθμων προγραμματισμού γεγονότων χρησιμοποιείται το μοντέλο HOLD και η τεχνική Event Horizon. Περιγράφεται ένα σύνολο δενδροειδών δομών δεδομένων που χρησιμοποιούνται για την αποθήκευση του συνόλου γεγονότων. Χρησιμοποιούνται επίσης κατανομές για την παραγωγή τυχαίων αριθμών με συγκεκριμένα χαρακτηριστικά. Συγκεκριμένα, πρόκειται για τις κατανομές που παρουσιάζονται στο Σχήμα 1.5 και οι οποίες επιλέχθηκαν γιατί, λόγω των χαρακτηριστικών τους, μπορεί κανείς να μελετήσει την αποτελεσματικότητα ενός αλγόριθμου υπό πολύ διαφορετικές κάθε φορά συνθήκες.

Οι αλγόριθμοι και οι δομές μελετώνται πειραματικά και θεωρητικά και γίνεται σύγκριση της απόδοσής τους και της συμπεριφοράς τους. Έχουν μελετηθεί γνωστές δομές, όπως η δομή του Leftist heap ή του *P*-tree αλλά προτείνονται και κάποιες νέες δομές που αποτελούν επεκτάσεις της δομής του σωρού (static heap) και του *P*-tree. Οι δομές που προτείνονται είναι *Multiple* heap, *n*-heap, *Indexed P*-tree και *Multiple P*-tree. Όλες οι δομές που μελετήθηκαν αναφέρονται στο επόμενο κεφάλαιο.

Τέλος μελετάται και η παράλληλη προσομοίωση διακριτών γεγονότων και προτείνονται παράλληλοι αλγόριθμοι που χρησιμοποιούν τις δομές που μελετήθηκαν και σε ακολουθιακό περιβάλλον.

(1) Συνεχείς Κατανομές

- Uniform στο διάστημα [0,2].
- Uniform στο διάστημα [0.9,1.1].
- Exponential (με $\lambda = 1$).
- Bimodal
  με πιθανότητα 0.9 ομοιόμορφη στο διάστημα [0,S],
  με πιθανότητα 0.1 ομοιόμορφη στο διάστημα [100*S,101*S],
  όπου το S επιλέγεται έτσι ώστε η μέση τιμή να είναι 1.

(2) Διακριτές Κατανομές

- Discrete με τιμή 1.
- Discrete με τιμές 0,1 και 2.

Σχήμα 1.5: Κατανομές

ΚΕΦΑΛΑΙΟ 2

ΠΡΟΣΟΜΟΙΩΣΗ ΔΙΑΚΡΙΤΩΝ ΓΕΓΟΝΟΤΩΝ

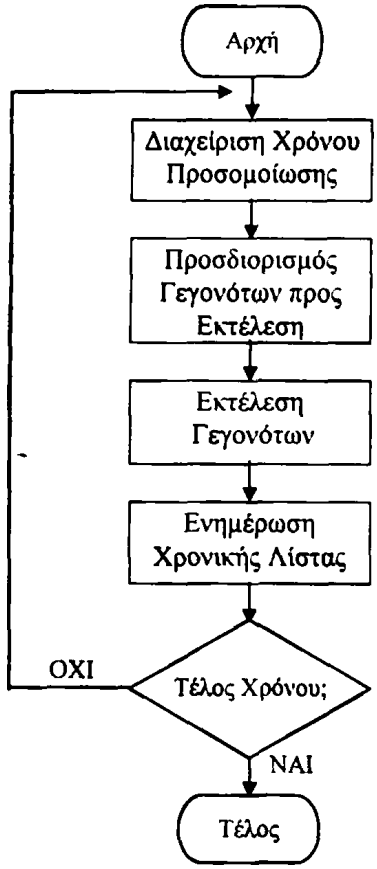
2.1 Προσομοίωση Γεγονότων

Η προσομοίωση γεγονότων παρουσιάστηκε για πρώτη φορά από τον Markowitz το 1962 κατά την ανάπτυξη της γλώσσας προσομοίωσης SIMSCRIPT. Το βασικό στοιχείο της προσομοίωσης γεγονότων είναι ο ορισμός και χρονοδρομολόγηση των γεγονότων του μοντέλου.

Κατά την προσομοίωση γεγονότων ο διαχειριστής είναι υπεύθυνος κυρίως για τρεις λειτουργίες:

1. τον έλεγχο του χρόνου,
2. τον προσδιορισμό των γεγονότων και
3. την εκτέλεση των γεγονότων

Ο έλεγχος του χρόνου γίνεται μέσω του ρολογιού προσομοίωσης, ενώ ο προσδιορισμός των γεγονότων προς εκτέλεση γίνεται μέσω μιας χρονικής λίστας γεγονότων στην οποία τοποθετούνται τα γεγονότα που πρόκειται να εκτελεσθούν σε κάποια μελλοντική χρονική στιγμή. Το ρολόι της προσομοίωσης δείχνει σε κάθε χρονική στιγμή το χρόνο της προσομοίωσης, ο οποίος αντιστοιχεί στα γεγονότα που εκτελούνται αυτή τη χρονική στιγμή. Η διαχείρισή του γίνεται από το πρόγραμμα ελέγχου, σύμφωνα με το μηχανισμό ροής χρόνου επόμενου γεγονότος. Ένα γενικό διάγραμμα του προγράμματος ελέγχου γεγονότων δίνεται στο Σχήμα 2.1.



Σχήμα 2.1. Διάγραμμα του προγράμματος ελέγχου προσομοίωσης γεγονότων

Η χρονική λίστα των γεγονότων είναι μια δυναμική λίστα, η οποία περιέχει τα γεγονότα που πρόκειται να εκτελεσθούν. Η εισαγωγή γεγονότων στη λίστα και η διαγραφή τους από αυτή γίνεται τόσο από το πρόγραμμα ελέγχου όσο και από τα προγράμματα εκτέλεσης των γεγονότων, σύμφωνα με την εξής διαδικασία:

- αμέσως μόλις το πρόγραμμα ελέγχου αυξήσει το ρολόι της προσομοίωσης στη νέα του τιμή, που αντιστοιχεί στο χρόνο του αμέσως επόμενου γεγονότος που πρόκειται να εκτελεστεί, ελέγχει τα γεγονότα της λίστας για να προσδιορίσει αυτά που πρέπει να εκτελεσθούν κατά τη χρονική αυτή στιγμή. Τα γεγονότα αυτά δίνονται προς εκτέλεση στα αντίστοιχα προγράμματα και αφαιρούνται από τη χρονική λίστα γεγονότων.

- κατά την εκτέλεση ενός γεγονότος από το πρόγραμμα εκτέλεσής του, είναι δυνατόν να προγραμματισθεί ένα άλλο γεγονός για το μέλλον. Το γεγονός αυτό εισάγεται στη χρονική λίστα γεγονότων για να εκτελεσθεί σε κάποια επόμενη επανάληψη του κύκλου του προγράμματος ελέγχου. Φυσικά, για απλούστερη και καθαρότερη δομή των προγραμμάτων προσομοίωσης, η εισαγωγή αυτή γίνεται πάντα από το πρόγραμμα ελέγχου μετά από αίτηση του προγράμματος εκτέλεσης, έτσι ώστε η διαχείριση της λίστας γεγονότων να πραγματοποιείται κεντρικά.

Ο κύκλος του Σχήματος 2.1 επαναλαμβάνεται μέχρι να τελειώσει η προσομοίωση. Η διαχείριση της χρονικής λίστας γεγονότων είναι αρκετά κρίσιμη για την ταχύτητα της προσομοίωσης, ιδίως όταν το μοντέλο είναι πολύπλοκο και προγραμματίζονται πολλά γεγονότα για το μέλλον. Για το λόγο αυτό στα προγράμματα προσομοίωσης χρησιμοποιούνται συνήθως χρονικά διατεταγμένες λίστες, έτσι ώστε να είναι εύκολη τόσο η προσθήκη και αφαίρεση γεγονότων όσο και η αναζήτηση του επόμενου γεγονότος που πρόκειται να εκτελεσθεί. Η αναζήτηση αυτή είναι στοιχειώδης σε μια χρονικά διατεταγμένη λίστα γιατί το επόμενο γεγονός που πρόκειται να εκτελεσθεί βρίσκεται στην αρχή της λίστας.

2.2 Βασικοί Όροι

Event Notice. Ο όρος αναφέρεται σε έναν κόμβο που αντιστοιχεί σε ένα γεγονός και αποτελείται από δύο πεδία, έστω *t* και *a*, όπου *a* είναι η ενέργεια (activity) που θα πραγματοποιηθεί κατά το χρόνο *t* (event time).

Χρόνος Δρομολόγησης (Event Time ή Scheduling Time). Ο όρος χρόνος δρομολόγησης αντιστοιχεί στον προγραμματισμένο χρόνο εκτέλεσης ενός γεγονότος.

Σύνολο Γεγονότων (Event Set). Το σύνολο γεγονότων είναι η δομή που χρησιμοποιείται για την αποθήκευση των μελλοντικών γεγονότων (events).

Αλγόριθμος Δρομολόγησης Γεγονότων (Event Set Algorithm ή Event Scheduling Algorithm). Αλγόριθμος δρομολόγησης γεγονότων είναι ο αλγόριθμος για τη διαχείριση των γεγονότων σε ένα σύστημα προσομοίωσης διακριτών γεγονότων. Πραγματοποιεί κυρίως τις εξής λειτουργίες:

α) Αφαίρεση από το Event Set του Event Notice με το μικρότερο χρόνο εκτέλεσης.
β) Δημιουργία του νέου Event Notice με κατάλληλο χρόνο εκτέλεσης.
γ) Εισαγωγή του νέου γεγονότος στο Event Set.
δ) Συγχώνευση δύο συνόλων γεγονότων (Event Set).
ε) Ακύρωση (cancel) ενός γεγονότος, δηλαδή αφαίρεσή του από το Event Set.

2.3 Το Μοντέλο HOLD και η Τεχνική Event Horizon

Το βασικό χαρακτηριστικό της διαδικασίας της προσομοίωσης διακριτών γεγονότων είναι ότι τα γεγονότα πραγματοποιούνται σε διακριτές και γενικά απρόβλεπτες χρονικές στιγμές. Όταν γίνει γνωστός ο χρόνος πραγματοποίησης ενός γεγονότος ένας κατάλληλος αλγόριθμος εξασφαλίζει ότι το γεγονός θα πραγματοποιηθεί κατά τον προγραμματισμένο χρόνο. Ο αλγόριθμος αυτός περιλαμβάνει μερικά χαρακτηριστικά στοιχεία, όπως είναι η δομή που χρησιμοποιείται για την αποθήκευση του συνόλου μελλοντικών γεγονότων καθώς και κάποιες λειτουργίες, όπως για παράδειγμα η εισαγωγή ενός νέου γεγονότος στη δομή. Το είδος των λειτουργιών που θα πραγματοποιηθούν καθώς και ο τρόπος πραγματοποίησής τους προσδιορίζει το είδος του μοντέλου που χρησιμοποιείται προκειμένου να μελετηθεί η αποτελεσματικότητα των αλγορίθμων που χρησιμοποιούνται κατά την προσομοίωση συστημάτων διακριτών γεγονότων.

Το μοντέλο που χρησιμοποιήθηκε, που διαδραματίζει καθοριστικό ρόλο για την πραγματοποίηση της μελέτης της αποτελεσματικότητας των αλγορίθμων, είναι το HOLD. Δεν αποτελεί μοντέλο προσομοίωσης, όπως ορίστηκε στο Κεφάλαιο 1, αλλά ένα μέτρο σύγκρισης, μια κοινή βάση προκειμένου να γίνει η σύγκριση των αλγορίθμων πιο αποδοτική. Το μοντέλο

HOLD καθώς και η τεχνική Event Horizon, η οποία χρησιμοποιείται για την βελτίωση της αποτελεσματικότητας των αλγορίθμων περιγράφονται στη συνέχεια.

Μοντέλο Hold

Για την προσομοίωση ενός κόμβου γεγονότος (event notice), όπως έχει ήδη αναφερθεί, χρησιμοποιείται μια δομή record (ή struct στην γλώσσα προγραμματισμού C), που περιέχει πληροφορίες όπως το είδος της ενέργειας που θα πραγματοποιηθεί και το χρόνο εκτέλεσής της (event time).

Οι πιο συνήθεις λειτουργίες που πραγματοποιούνται από ένα σύστημα προσομοίωσης διακριτών γεγονότων είναι η διαγραφή του γεγονότος με το μικρότερο χρόνο εκτέλεσης από το event set και εισαγωγή ενός νέου κόμβου στο σύνολο γεγονότων.

Η διαδικασία HOLD αρχικά εντοπίζει το γεγονός με το μικρότερο χρόνο εκτέλεσης, t, και αφαιρεί αυτόν τον κόμβο από το σύνολο γεγονότων (διαδικασία διαγραφής - delete_min). Στη συνέχεια αυξάνεται κατά T ο χρόνος δρομολόγησης (event time) t, όπου T είναι μια τυχαία τιμή χρόνου που προέρχεται από κάποια κατανομή. Τέλος εισάγεται ο νέος κόμβος με τη νέα τιμή ξανά στο σύνολο γεγονότων (διαδικασία εισαγωγής - insert). Επομένως διαδοχικές κλήσεις της διαδικασίας HOLD προκαλούν μια σειρά από διαδοχικές διαγραφές του κόμβου με τη μικρότερη τιμή και εισαγωγές ενός κόμβου, που εναλλάσσονται. Περιλαμβάνει δηλαδή τις εξής διαδικασίες:

(α) διαγραφή του κόμβου με την μικρότερη τιμή, έστω *min*,
(β) δημιουργία ενός νέου κόμβου με χρόνο δρομολόγησης $min + T$, όπου T προέρχεται από κάποια κατανομή και είναι η παράμετρος του μοντέλου, και τέλος
(γ) εισαγωγή του νέου κόμβου στο σύνολο γεγονότων.

Οι διαδικασίες εκτελούνται με τη σειρά.

Για την πραγματοποίηση της διαδικασίας της εισαγωγής, κατά την υλοποίηση του μοντέλου HOLD, λαμβάνεται υπόψη και η μικρότερη τιμή χρόνου που βρίσκεται ήδη αποθηκευμένη στη δομή που χρησιμοποιείται για την προσομοίωση του συνόλου γεγονότων. Αυτό εξασφαλίζει τη σωστή προσομοίωση της ροής του χρόνου. Αφού η μικρότερη τιμή χρόνου αντιστοιχεί στον τρέχοντα χρόνο, ένα νέο γεγονός δεν είναι δυνατόν να έχει ακόμη μικρότερο προγραμματισμένο χρόνο εκτέλεσης διότι κάτι τέτοιο θα σήμαινε ότι το συγκεκριμένο γεγονός πρέπει να εκτελεστεί στο παρελθόν.

Ο παράγοντας που θα κρίνει την αποτελεσματικότητα των αλγορίθμων είναι ο χρόνος που απαιτείται για την εκτέλεσή τους, δηλαδή ο χρόνος επεξεργαστή (CPU time). Ο χρόνος αυτός προφανώς επηρεάζεται καθοριστικά από τη δομή που χρησιμοποιείται για την αποθήκευση του συνόλου γεγονότων. Έτσι κάποιες δομές έχουν αποδοτική διαδικασία εισαγωγής η οποία εκτελείται σε σύντομο χρόνο και μη αποδοτική διαδικασία καθορισμού του στοιχείου με το μικρότερο χρόνο εκτέλεσης, ενώ σε κάποιες άλλες είναι δυνατόν να συμβαίνει το αντίθετο. Η αποδοτικότητα μιας συγκεκριμένης δομής δεν εξαρτάται από το μοντέλο HOLD.

Μερικοί ακόμη καθοριστικοί παράγοντες για την αποδοτικότητα των αλγορίθμων που όμως διαδραματίζουν τον ίδιο ρόλο ανεξάρτητα με το μοντέλο το οποίο χρησιμοποιείται, είναι το

μέγεθος του συνόλου γεγονότων (event set), ή αλλιώς ο αριθμός *N* των κόμβων γεγονότων που αποτελούν το event set και φυσικά η επιλογή της κατανομής. Η κατανομή που θα επιλεγεί χρησιμοποιείται για την αρχική δημιουργία του συνόλου γεγονότων πριν αρχίσει η μελέτη της συμπεριφοράς του, καθώς επίσης και για την ανάθεση μιας τιμής χρόνου σε ένα νέο γεγονός που θα εισαχθεί, αφού προσφέρει την τιμή *T*.

Παρά το γεγονός ότι το μοντέλο υλοποιεί τις πιο συνήθεις λειτουργίες που απαιτούνται από ένα σύστημα προσομοίωσης διακριτών γεγονότων, δηλαδή τις διαδικασίες εισαγωγής και διαγραφής, υπάρχουν και άλλες λειτουργίες που μπορεί να πραγματοποιηθούν σε ένα σύνολο γεγονότων. Μερικές από αυτές τις λειτουργίες τροποποιούν κάποιες από τις παραμέτρους που επηρεάζουν την αποτελεσματικότητα του αλγόριθμου. Για παράδειγμα η λειτουργία ακύρωσης (cancel), η οποία αφαιρεί ένα γεγονός από το σύνολο γεγονότων ώστε αυτό να μην πραγματοποιηθεί ποτέ, μεταβάλλει το μέγεθος του event set.

Το μοντέλο HOLD δεν είναι αρκετά ρεαλιστικό αφού πραγματοποιεί διαδοχικές διαγραφές και εισαγωγές. Διατηρεί σταθερό το μέγεθος του συνόλου γεγονότων, κάτι που επίσης δεν είναι ρεαλιστικό, αφού σε πραγματικές προσομοιώσεις το πλήθος των γεγονότων που βρίσκονται κάθε στιγμή στο event set αλλάζει δυναμικά. Ένας ακόμη περιορισμός που υπάρχει, μπορεί όμως εύκολα να αντιμετωπισθεί, είναι ότι οι προγραμματισμένοι χρόνοι εκτέλεσης όλων των γεγονότων προέρχονται από την ίδια κατανομή. Παρά την ύπαρξη όμως των περιορισμών που εντοπίστηκαν, τα αποτελέσματα που προέκυψαν είναι έγκυρα, αφού όλοι οι αλγόριθμοι μελετήθηκαν κάτω από τις ίδιες συνθήκες.

Τεχνική Event Horizon

Ο όρος Event Horizon αντιστοιχεί σε μία τεχνική εξαιρετικά χρήσιμη κατά την παράλληλη προσομοίωση. Όπως προέκυψε από τη μελέτη που πραγματοποιήθηκε είναι εξίσου αποδοτική και για την προσομοίωση διακριτών γεγονότων, όχι απαραίτητα σε παράλληλο περιβάλλον επεξεργασίας.

Η τεχνική Event Horizon περιλαμβάνει τη χρήση μιας βοηθητικής, δευτερεύουσας δομής προκειμένου να αποθηκεύονται προσωρινά τα γεγονότα, η οποία συνήθως είναι διπλά συνδεδεμένη λίστα. Η μικρότερη πληροφορία που βρίσκεται αποθηκευμένη στη βοηθητική δομή είναι κάθε χρονική στιγμή γνωστή και δεν χρειάζεται να είναι ταξινομημένη. Έτσι, αν το νέο γεγονός που θα πρέπει κάποια στιγμή να επεξεργαστεί, δηλαδή να αφαιρεθεί από την ουρά προτεραιότητας, βρίσκεται στη βοηθητική δομή, αυτή ταξινομείται και συγχωνεύεται με τη βασική δομή που υλοποιεί την ουρά προτεραιότητας.

Σκοπός της παραπάνω διαδικασίας είναι η όσο το δυνατόν μείωση του χρόνου που απαιτείται προκειμένου να πραγματοποιηθεί η διαδικασία της εισαγωγής ενός νέου γεγονότος στην ουρά προτεραιότητας. Απαιτεί ιδιαίτερη προσοχή, όμως, η διαδικασία της ταξινόμησης της βοηθητικής δομής καθώς και η συγχώνευσή της με τη βασική δομή, αφού σε κάποιες περιπτώσεις μπορεί να αποβεί τελικά όλη η διαδικασία μη αποδοτική.

2.4 Γεννήτορας Αριθμών και Κατανομές

Κατά την προσομοίωση απαιτείται η δημιουργία μιας σειράς τυχαίων μεταβλητών οι οποίες θα προσομοιώνουν είτε τη μη προκαθορισμένη άφιξη οντοτήτων στο μοντέλο είτε τη μη προκαθορισμένη συμπεριφορά των οντοτήτων του μοντέλου. Οι τυχαίες αυτές μεταβλητές ακολουθούν συνήθως κατανομές οι οποίες έχουν προσδιοριστεί είτε από μετρήσεις στο πραγματικό σύστημα είτε από υποθέσεις για το πραγματικό σύστημα. Οι υποθέσεις βασίζονται συνήθως σε εμπειρία για τις κατανομές που ακολουθούν ορισμένες κλάσεις μεταβλητών. Στην περίπτωση που η κατανομή μιας τυχαίας μεταβλητής δεν είναι προκαθορισμένη είναι δυνατόν να επαναληφθεί η προσομοίωση με διάφορες κατανομές για να καθοριστεί η απόκριση του συστήματος σε όλες τις δυνατές κατανομές εισόδων. Η παραγωγή των τυχαίων μεταβλητών βασίζεται στην παραγωγή τυχαίων αριθμών και τη μετατροπή τους σε τυχαίες μεταβλητές που ακολουθούν την απαιτούμενη κατανομή.

Οι αριθμοί που προκύπτουν από τις κατανομές χρησιμοποιούνται για τον προσδιορισμό της παραμέτρου T, που αποτελεί την προσαύξηση στις τιμές του χρόνου και καθορίζει το χρόνο εκτέλεσης του νέου γεγονότος που εισάγεται στο event set.

Οι κατανομές που χρησιμοποιήθηκαν, οι οποίες παρουσιάζονται αναλυτικά στη συνέχεια, έχουν επιλεγεί διότι είναι αντιπροσωπευτικές αφού αποκαλύπτουν τα μειονεκτήματα και τα πλεονεκτήματα κάθε μιας από τις δομές δεδομένων που μελετήθηκαν. Η αποτελεσματικότητα κάθε δομής εξαρτάται από τα δεδομένα τα οποία αποθηκεύει, δηλαδή από το πεδίο τιμών των στοιχείων της. Εξάλλου πρόκειται για κατανομές που έχουν ήδη μελετηθεί σε ορισμένους αλγόριθμους, γεγονός που επιτρέπει την ύπαρξη μέτρου σύγκρισης για τα αποτελέσματα αυτής της μελέτης.

Τυχαίοι και Ψευδοτυχαίοι Αριθμοί

Η παραγωγή πραγματικά τυχαίων αριθμών δεν είναι δυνατή με ψηφιακούς ηλεκτρονικούς υπολογιστές. Αυτό προφανώς οφείλεται στο γεγονός ότι οι ηλεκτρονικοί υπολογιστές εκτελούν πάντα μια αλληλουχία συγκεκριμένων εντολών η οποία αν επαναληφθεί με τις ίδιες αρχικές συνθήκες θα δημιουργεί πάντα τα ίδια αποτελέσματα. Οι μόνες μέθοδοι που παράγουν τυχαίους αριθμούς βασίζονται σε φυσικά φαινόμενα τα οποία, λόγω του πολύ μεγάλου αριθμού παραμέτρων που υπεισέρχονται, δίνουν τυχαία αποτελέσματα. Είναι όμως δυνατή η δημιουργία τυχαίων αριθμών και με ηλεκτρονικά μέσα όπως οι γεννήτριες λευκού θορύβου και οι ηλεκτρονικές λυχνίες. Τυχαίοι αριθμοί που έχουν δημιουργηθεί με παρόμοιες μεθόδους είναι διαθέσιμοι με τη μορφή πινάκων οι οποίοι μπορούν να εισαχθούν ως δεδομένα σε ένα πρόγραμμα. Για παράδειγμα, η εταιρεία RAND Corporation διαθέτει πίνακα 1000000 τυχαίων αριθμών. Η χρήση όμως αυτών των πινάκων δεν ενδείκνυται για δύο λόγους. Ο πρώτος είναι η αδυναμία αποθήκευσης των τεραστίων αυτών πινάκων στην περιορισμένη μνήμη ενός υπολογιστή αφού αυτό θα αποτελούσε σπατάλη πολύτιμων πόρων. Αν υποτεθεί ότι αποθηκεύονται σε δευτερεύουσα μνήμη, η χαμηλή ταχύτητα πρόσβασης σε αυτή θα καθιστούσε την προσομοίωση απαράδεκτα αργή. Ο δεύτερος λόγος είναι ότι σε πολλές περιπτώσεις απαιτείται η δημιουργία πολύ περισσότερων από 1000000 τυχαίων αριθμών ενώ η επαναχρησιμοποίηση των ίδιων αριθμών θα προκαλεί στατιστικά απαράδεκτη πόλωση στα αποτελέσματα.

Οι σειρές των τυχαίων αριθμών παράγονται στους ψηφιακούς υπολογιστές χρησιμοποιώντας απλές επαναληπτικές μεθόδους. Οι αριθμοί αυτοί ονομάζονται ψευδοτυχαίοι αφού στην πραγματικότητα δεν είναι εντελώς τυχαίοι. Ξεκινώντας από τις ίδιες αρχικές συνθήκες παράγεται πάντα η ίδια σειρά τυχαίων αριθμών επειδή η μέθοδος παραγωγής είναι συγκεκριμένη. Το μειονέκτημα όμως αυτό αποτελεί στην ουσία πλεονέκτημα για την προσομοίωση. Αν μια σειρά τυχαίων αριθμών ικανοποιεί τα στατιστικά τεστ τυχαιότητας, η επανάληψη αυτής της σειράς είναι επιθυμητό χαρακτηριστικό για την προσομοίωση, αφού μια από τις σημαντικότερες εφαρμογές της είναι ο έλεγχος υποθέσεων, δηλαδή η εξέταση ενός συστήματος κάτω από διαφορετικές συνθήκες ή με διαφορετικές παραμέτρους. Στην περίπτωση αυτή ενδιαφέρει η μεταβολή μόνο των παραμέτρων του συστήματος και όχι των εξωτερικών παραγόντων, οι οποίοι θα πρέπει να είναι οι ίδιοι για κάθε εκτέλεση της προσομοίωσης. Οι σειρές των τυχαίων αριθμών είναι ένας εξωτερικός παράγοντας που πρέπει να διατηρηθεί σταθερός για να μην επηρεάσει τα αποτελέσματα. Επιπλέον, η ιδιότητα αυτή είναι σημαντική ακόμη και όταν δεν συγκρίνονται αποτελέσματα προσομοίωσης του ίδιου συστήματος με διαφορετικές παραμέτρους αλλά απαιτείται η επανάληψη της ίδιας σειράς τυχαίων δειγμάτων σε διαφορετικά συστήματα.

Γεννήτορας Αριθμών

Έχει ήδη αναφερθεί ότι προκειμένου να παραχθεί μια ακολουθία αριθμών που ακολουθούν μια συγκεκριμένη κατανομή, αρχικά παράγονται τυχαίοι αριθμοί ομοιόμορφα κατανεμημένοι στο διάστημα (0,1). Είναι όμως ευκολότερο να παραχθεί ακολουθία ακεραίων $\{X_n\}$ μεταξύ του μηδενός και ενός αριθμού m και στη συνέχεια να προκύψουν οι ομοιόμορφοι αριθμοί U_n ως εξής:

$$U_n = X_n / m$$

Συνήθως ο αριθμός m έχει σχέση με το μέγεθος της λέξης του επεξεργαστή που χρησιμοποιείται. Σε δυαδικούς επεξεργαστές ο αριθμός m έχει την μορφή $m = 2^b$, όπου b είναι ο αριθμός των bits που χρησιμοποιούνται για την αναπαράσταση των ακεραίων.

Ο επόμενος αριθμός της ακολουθίας X_n προκύπτει μετά από έναν μετασχηματισμό f(.) που υφίσταται ο προηγούμενος. Ξεκινώντας επομένως από έναν αριθμό που ονομάζεται seed, παράγεται η ακολουθία

$$X_0 , X_1 = f(X_0), \ldots , X_{n+1} = f(X_n), \ldots$$

Υπάρχει μία ποικιλία μεθόδων παραγωγής κάθε αριθμού από τον προηγούμενό του. Ο γεννήτορας όμως που χρησιμοποιείται συχνότερα είναι αυτός που προτάθηκε από τον Lehmer:

$$X_{i+1} = (k X_i + C) \bmod m \qquad \text{με} \quad i = 0, 1, 2, \ldots$$

όπου οι X_0, k, C, m είναι ακέραιοι και X_0, k, $C < m$. Η ακολουθία $\{X_i / m\}$ αποτελεί την ακολουθία των ομοιόμορφα κατανεμημένων αριθμών. Προφανώς κάποια στιγμή η ακολουθία θα αρχίσει να επαναλαμβάνεται η ίδια από την αρχή αλλά με κατάλληλη επιλογή των αριθμών X_0, k, C, m παράγεται ένας ικανοποιητικά μεγάλος κύκλος.

Θεωρία Αριθμών (Number Theory)

Το παρακάτω θεώρημα (FERMAT, 1640) είναι βασικό στη θεωρία αριθμών:

Έστω p ένας πρώτος αριθμός και k ένας θετικός ακέραιος που ικανοποιεί τη σχέση:

$$k \neq 0 \bmod p$$

και έστω d ο μικρότερος από τους θετικούς ακεραίους n που ικανοποιεί τη σχέση:

$$k^n = 1 \bmod p$$

Τότε ο αριθμός d διαιρεί το p-1.

Έστω τώρα ο γεννήτορας $X_{n+1} = k X_n \bmod p$ με $n = 0, 1, 2, \ldots$ (1)

όπου ο p είναι πρώτος αριθμός. Τότε

$$X_n = k^n X_0 \bmod p \quad (2)$$

Η ακολουθία των X που ικανοποιεί την (1) αρχίζει να επαναλαμβάνεται από την τιμή του n για την οποία ισχύει

$$k^n X_0 = X_0 \bmod p \quad (3)$$

Έτσι ο αριθμός ($k^n X_0 - X_0$) διαιρείται με το p και αφού και το X_0 διαιρείται με το p (αν δεν ίσχυε αυτό η (2) θα έδινε ότι $X_1 = X_2 = \ldots = 0$) ο αριθμός $k^n - 1$ θα πρέπει επίσης να διαιρείται με το p, δηλαδή θα ισχύει

$$k^n = 1 \bmod p \quad (4)$$

Το μέγεθος του κύκλου δίνεται από το μικρότερο n που ικανοποιεί την σχέση (4). Αν αυτός ο αριθμός είναι ο d τότε λέγεται περίοδος του γεννήτορα. Από το θεώρημα του FERMAT προκύπτει ότι η μέγιστη δυνατή τιμή της περιόδου είναι (p-1).

Παραγωγή Τυχαίων Αριθμών (Random Number Generation)

Λαμβάνοντας υπόψη τα παραπάνω επιλέγεται p = 67099547 και η τιμή του k ως 2^{13}= 8192. Έτσι ο γεννήτορας του Lehmer που δημοσιεύθηκε από τους Downham και Roberts

$$X_{k+1} = 8192\, X_k \bmod (67099547)$$

παράγει έναν πλήρη κύκλο και η ακολουθία $\{X_k / 67099547\}$ μπορεί να χρησιμοποιηθεί σαν ακολουθία ομοιόμορφα κατανεμημένων τυχαίων αριθμών.

Στη γλώσσα προγραμματισμού C, τα παραπάνω υλοποιούνται ως εξής:

void *Starter*(int *f)

```
{
  *f = 7*(*f);
  if(*f>=m) *f = (*f) % m;
  *f = (*f)*13;
  if(*f>=m) *f = (*f) % m;
  *f = (*f)*15;
  if(*f>=m) *f = (*f) % m;
  *f = (*f)*27;
  if(*f>=m) *f = (*f) % m;
}


int Random(int *i)
 {
   int counter;
   for(counter=1; counter<=2; counter++)
    {
     *i = 32*(*i);
     if(*i>=m) *i = (*i) % m;
    }
   *i = (*i)*8;
   if(*i>=m) *i = (*i) % m;
   return(*i);
 }


float RNG(int *i)
 {
   float f;
   *i = Random(i);
   f = (float)(*i) / (float)m;
   return(f);
 }
```

Συνεχείς Κατανομές

➢ *Uniform(a,b)*

Έστω η συνεχής τυχαία μεταβλητή x με πεδίο ορισμού το διάστημα $[a,b]$, όπου a και b είναι πραγματικοί αριθμοί. Τότε αν όλα τα ισομήκη υποδιαστήματα του $[a,b]$ έχουν την ίδια πιθανότητα να περιέχουν μία τιμή της x τότε η τυχαία μεταβλητή x ακολουθεί την ομοιόμορφη (uniform) κατανομή και συμβολικά: $x \sim U[a,b]$.

Η συνάρτηση πυκνότητας πιθανότητας της x είναι

$$f(x) = \frac{1}{b-a} \quad \text{εαν } b \geq x \geq a, \quad \text{αλλιώς } f(x) = 0$$

Η αθροιστική συνάρτηση κατανομής της $x \sim U[a, b]$ είναι η

$$F(x) = \int_a^x f(t)dt = \frac{x-a}{b-a}$$

Η μέση τιμή της $x \sim U[a, b]$ δίνεται από τη σχέση

$$E(x) = \int_a^b tf(t)dt = \frac{b+a}{2}$$

Ο κώδικας που υλοποιεί την ομοιόμορφη κατανομή είναι:

```
float Uniform(float a, float b, int *x)
 {
    float ans, r;
    r = RNG(x);
    ans = a + (b-a)*r;
    return(ans);
 }
```

➢ *Exponential(λ)*

Μία συνεχής τυχαία μεταβλητή ακολουθεί την εκθετική κατανομή αν η συνάρτηση πυκνότητας πιθανότητας είναι

$$f(x) = \lambda e^{-\lambda x} \quad \text{αν } x \geq 0, \text{ άλλιώς } f(x) = 0$$

όπου $\lambda > 0$ και είναι ο μέσος αριθμός συμβάντων στη μονάδα του χρόνου, η οποία μπορεί να είναι για παράδειγμα ένα δευτερόλεπτο, μία ώρα ή ένας χρόνος. Στη μελέτη που έχει γίνει θεωρήθηκε το $\lambda = 1$, δηλαδή ο μέσος αριθμός συμβάντων στη μονάδα του χρόνου είναι ένα. Επομένως θα ισχύει $f(x) = e^{-x}$.

Η αθροιστική συνάρτηση κατανομής δίνεται ως εξής:

$$F(x) = \int_0^x f(t)dt = 1 - e^{-\lambda x} \quad , \text{για } x \geq 0$$

και η μέση (ή αναμενόμενη) τιμή δίνεται από τη σχέση

$$E(x) = \int_0^{\infty} x\lambda\, e^{-\lambda x}dx = \frac{1}{\lambda}$$

Η εκθετική κατανομή υλοποιείται ως εξής:

```
float Expon(float ex, int *x)
 {
   float r;
   r = RNG(x);
   return(-(ex*log(r)));
 }
```

➢ *Bimodal*

Η κατανομή bimodal παρουσιάζει δύο "κορυφαίες τιμές" στην συνάρτηση πυκνότητας πιθανότητας. Στη μελέτη που έγινε θεωρήθηκε μία bimodal κατανομή, η οποία περιλαμβάνει δύο ομοιόμορφες κατανομές. Με πιθανότητα 0.9 παράγει ομοιόμορφη τυχαία μεταβλητή στο διάστημα (0,*S*) και με πιθανότητα 0.1 παράγει ομοιόμορφη τυχαία μεταβλητή στο διάστημα (100 *S*, 101 *S*). Ο αριθμός *S* επιλέγεται έτσι ώστε το μέσο να είναι 1 και υπολογίζεται από τον τύπο:

$$0.5\ S\ 0.9 + 0.5\ 100\ S\ 0.1 = 1$$

Η κατανομή Bimodal υλοποιείται ως εξής:

```
float Bimodal(int *x, int *y, int *z)
   {
     float r1;
     r1 = RNG(x);
     if(r1<0.9) return(RNG(y)*s);
     else return((100+RNG(z))*s);
   }
```

Διακριτές κατανομές

➢ *Discrete(k)*

Αν x είναι μια τυχαία μεταβλητή η οποία παίρνει τις ακέραιες τιμές στο διάστημα [*k1*, *k2*] με σταθερή πιθανότητα *p*, τότε η *x* ακολουθεί τη διακριτή κατανομή. Στη μελέτη που έγινε θεωρήθηκε ότι *k1* = 0 και *k2* = *k*, οπότε ισχύει *p* = 1 / *k*.

Η συνάρτηση πιθανότητας της διακριτής τυχαίας μεταβλητής δίνεται από τη σχέση:

$$p(x) = \frac{1}{k} \qquad \text{με } x = 0, 1, 2, ..., k$$

Η συνάρτηση κατανομής υπολογίζεται από τη σχέση:

$$F(x) = \frac{x}{k}$$

και η μέση τιμή:

$$E(x) = \sum_{i=0}^{k} i\, x_i$$

Η διακριτή κατανομή υλοποιείται ως εξής:

```
float Discrete(int a, int b, int *x)
    {
      float ans, r;
      r = RNG(x);
      ans = a + (int)(r*(b-a+1));
      return(ans);
    }
```

2.5 Δομές Δεδομένων

Οι δομές που μελετήθηκαν ανήκουν στην κατηγορία των δενδροειδών δομών. Συγκεκριμένα έχει μελετηθεί η δομή του σωρού (heap) με στατική και δυναμική αναπαράσταση μνήμης καθώς και οι νέες δομές *n*-Heap και *M*-Heap που προτείνονται και που αποτελούν επέκταση της δομής του σωρού. Άλλες δομές σωρού που μελετώνται είναι οι Binomial και Leftist Heap. Επιπλέον θα εξετασθεί η αποτελεσματικότητα της δομής SPEEDES QHeap που χρησιμοποιεί την τεχνική Event Horizon καθώς και η δομή Calendar Queues. Ακόμη θα μελετηθεί εκτενέστερα η δομή του *P*-δέντρου (*P*-tree) καθώς και τρόποι βελτίωσης της αποτελεσματικότητάς της. Με βάση τις ιδιότητες της δομής *P*-tree και την τεχνική Event Horizon δημιουργείται και μελετάται μια πολύ αποτελεσματική δομή που ονομάζεται *Indexed P*-tree (*IP*-tree) ενώ συνδυάζοντας κατάλληλα έναν αριθμό από *P*-trees δημιουργείται η δομή *Multiple P*-tree (*MP*-tree) της οποίας η αποτελεσματικότητα μελετάται επίσης. Αναλυτικά:

- ***Στατική Αναπαράσταση του Σωρού (Static Heap)***
- ***Δυναμική Αναπαράσταση του Σωρού (Dynamic Heap)***
- ***Διωνυμικός Σωρός (Binomial Heap)***
- ***SPEEDES QHeap***
- ***n-Σωρός (n-Heap)***
- ***M-Σωρός (M-Heap)***
- ***Leftist Heap***
- ***Calendar Queues***

- ***P-δέντρο (P-tree)***
- ***Indexed P-tree (IP-tree)***
- ***Multiple P-tree (MP-tree)***

Κάθε μια από αυτές τις δομές περιγράφεται αναλυτικά στα επόμενα κεφάλαια.

2.6 Συνθήκες Πειραματικής Μελέτης

Σαν τμήμα ενός συστήματος προσομοίωσης, ένας αλγόριθμος του συνόλου γεγονότων απαιτείται να λειτουργεί κάτω από μια πληθώρα διαφορετικών συνθηκών. Το μοντέλο HOLD που συνδυάζει τις λειτουργίες της εισαγωγής και της διαγραφής, έχει επιλεγεί προκειμένου να καθοριστεί η μέση πολυπλοκότητα καθενός από τους αλγορίθμους.

Για κάθε μία δομή προσομοίωσης του συνόλου γεγονότων και με δεδομένο το μέγεθος του και την κατανομή από την οποία προέρχονται οι χρόνοι δρομολόγησης των γεγονότων εκτελείται ένας βρόχος 16000 φορές, δηλαδή η διαδικασία hold εκτελείται 16000 φορές άρα πραγματοποιούνται συνολικά 16000 εισαγωγές και διαγραφές.

Μια παράμετρος που προφανώς καθορίζει την αποτελεσματικότητα ενός αλγόριθμου είναι ο αριθμός των κόμβων γεγονότων, N, που αποτελούν το event set. Οι τιμές του N που έχουν ελεγχθεί είναι $N = 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192$ και 16384 κόμβοι.

Με την επιλογή της κατανομής που χρησιμοποιείται κάθε φορά επιδιώκεται να προσδιοριστούν τα πλεονεκτήματα αλλά και τα μειονεκτήματα της κάθε δομής δεδομένων που χρησιμοποιήθηκε. Οι κατανομές που έχουν επιλεγεί, διακριτές και συνεχείς, διαφέρουν κατά μεγάλο βαθμό μεταξύ τους, έχουν όμως μέσον το ένα.

Για την πραγματοποίηση καθενός από τους ελέγχους δημιουργείται το σύνολο γεγονότων με τον απαιτούμενο αριθμό γεγονότων. Οι χρόνοι δρομολόγησης των γεγονότων που αποτελούν αρχικά το event set προέρχονται από μια συγκεκριμένη κάθε φορά κατανομή.

Αξίζει να σημειωθεί επίσης το γεγονός ότι κατά τον υπολογισμό του χρόνου που απαιτείται από τον επεξεργαστή προκειμένου να ολοκληρώσει τη μελέτη κάποιου αλγόριθμου, υπεισέρχεται και μία ακόμη παράμετρος η οποία ίσως και να μην είναι επιθυμητή. Πρόκειται για το χρόνο που απαιτείται για την παραγωγή τυχαίων αριθμών που θα αντιστοιχούν στους χρόνους δρομολόγησης των γεγονότων καθώς και στην κλήση της συνάρτησης hold. Τα αποτελέσματα όμως που προκύπτουν από τη μελέτη σχετικά με την αποτελεσματικότητα των αλγορίθμων εξακολουθούν να είναι έγκυρα αφού η παραπάνω παράμετρος υπεισέρχεται σε κάθε έναν από τους αλγορίθμους. Κρίνεται όμως απαραίτητο να μελετηθεί και ο αριθμός των συγκρίσεων μεταξύ των στριχείων που πραγματοποιεί ο κάθε αλγόριθμος προκειμένου να ολοκληρωθούν οι διαδικασίες της εισαγωγής και της διαγραφής. Παρέχονται κατά αυτόν τον τρόπο ακριβέστερα αποτελέσματα αφού η αποτελεσματικότητα μερικών αλγορίθμων σχεδόν

ταυτίζεται όταν αυτή κρίνεται με βάση το χρόνο που απαιτείται από τον επεξεργαστή προκειμένου να πραγματοποιήσει τις διαδικασίες που προαναφέρθηκαν.

Το κυρίως πρόγραμμα (main program) του κάθε αλγόριθμου είναι σε πολλά σημεία κοινό, ανεξάρτητα από τη δομή δεδομένων που χρησιμοποιείται κάθε φορά. Έτσι, δημιουργείται κάθε φορά το σύνολο γεγονότων τόσο με τη χρήση συναρτήσεων που ποικίλλουν ανάλογα με τη δομή δεδομένων που χρησιμοποιείται όσο και με τις συναρτήσεις που υλοποιούν τις διάφορες κατανομές. Στη συνέχεια ακολουθεί μια διαδικασία που είναι γνωστή ως προετοιμασία του συστήματος. Αυτό που ουσιαστικά γίνεται είναι η εκτέλεση της διαδικασίας που θα εκτελεστεί με βάση το HOLD μοντέλο, χωρίς να υπολογίζεται ο χρόνος που θα απαιτηθεί, τόσες φορές όσο είναι το πλήθος των γεγονότων που αποτελούν κάθε φορά το event set. Η προετοιμασία αυτή του συστήματος πραγματοποιείται προκειμένου τα αποτελέσματα που θα προκύψουν τελικά από κάθε αλγόριθμο να είναι εγκυρότερα. Ακόμη, όλοι οι αλγόριθμοι έχουν υλοποιηθεί σε Sun Sparc-4 επεξεργαστή με 32 MB μνήμη.
Η μορφή του κυρίως προγράμματος παρουσιάζεται καλύτερα στο Σχήμα 2.2. Η συνάρτηση clock() που προσφέρεται από τη γλώσσα προγραμματισμού C, επιστρέφει το χρόνο επεξεργαστή που καταναλώθηκε από την αρχή της εκτέλεσης του προγράμματος μέχρι τη στιγμή της κλήσης της συνάρτησης. Η διαίρεση του χρόνου με τη σταθερά CLOCKS_PER_SEC, η οποία επίσης παρέχεται από τη C, μετατρέπει το χρόνο επεξεργαστή από κτύπους του ρολογιού σε δευτερόλεπτα.

```
main program
 begin
   for i = 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 και 16384 do
     begin
      για κάθε μια από τις κατανομές εκτέλεσε:
        begin
          1. Δημιούργησε το σύνολο γεγονότων με χρήση
             συγκεκριμένης δομής δεδομένων.
          2. Για k = 1 μέχρι i εκτέλεσε:
             κλήση της συνάρτησης HOLD.
          3. a = clock( ).
          4. Για k = 1 μέχρι 16000 εκτέλεσε:
             κλήση της συνάρτησης HOLD.
          5. b = clock( ).
          6. Ο χρόνος επεξεργαστή που καταναλώθηκε, σε
             δευτερόλεπτα, υπολογίζεται ως εξής:
             CPU_Time = (b - a) / CLOCKS_PER_SEC.
        end.
     end.
 end.
```

Σχήμα 2.2. Το Κυρίως Πρόγραμμα (Main Program)

ΚΕΦΑΛΑΙΟ 3

ΒΑΣΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΓΙΑ ΤΗΝ ΠΡΟΣΟΜΟΙΩΣΗ ΤΟΥ ΣΥΝΟΛΟΥ ΓΕΓΟΝΟΤΩΝ

3.1 Η Δομή του Σωρού

Μια από τις δομές που χρησιμοποιήθηκαν για την προσομοίωση του συνόλου γεγονότων είναι η δομή του σωρού (heap). Συγκεκριμένα υλοποιήθηκε η στατική και η δυναμική αναπαράσταση της δομής του σωρού ελαχίστων. Στην συνέχεια παρατίθενται ο ορισμός καθώς και οι ιδιότητες της δομής του σωρού ελαχίστων.

ΟΡΙΣΜΟΣ 3.1: **Σωρός Ελαχίστων** είναι το σχεδόν πλήρες (left complete) δυαδικό δέντρο που έχει την ιδιότητα η τιμή της πληροφορίας κάθε κόμβου να είναι μικρότερη ή ίση από τις τιμές των πληροφοριών των δύο παιδιών.

Σχεδόν πλήρες (left complete) δυαδικό δέντρο είναι το δέντρο του οποίου κάθε επίπεδο είναι πλήρες εκτός ίσως από το τελευταίο, το οποίο περιλαμβάνει κόμβους διατεταγμένους διαδοχικά από τα αριστερά προς τα δεξιά.

Σωρός - Στατική Αναπαράσταση (Static Heap)

Για την αποθήκευση της δομής του σωρού είναι δυνατόν να χρησιμοποιηθεί ένας μονοδιάστατος πίνακας, έστω H, το μέγεθος N του οποίου ορίζεται να είναι τέτοιο ώστε το πλήθος των κόμβων που αποτελούν τη δομή να μην το υπερβαίνει. Έτσι, αν η μικρότερη πληροφορία που υπάρχει στη δομή βρίσκεται αποθηκευμένη στην θέση $H[1]$, τότε για κάθε κόμβο που βρίσκεται στη θέση i του πίνακα θα ισχύει ότι το δεξί παιδί του βρίσκεται αποθηκευμένο στη θέση $H[2\ i]$ και το αριστερό παιδί του στη θέση $H[2\ i + 1]$, για κάθε i τέτοιο ώστε $1 \leq i \leq N$. Ακόμη θα ισχύει $H[i] \geq H[i \text{ div } 2]$ για κάθε i τέτοιο ώστε $2 \leq i \leq N$, αφού το στοιχείο $H[i \text{ div } 2]$ είναι ο πατέρας του στοιχείου $H[i]$.

Η διαδικασία της διαγραφής του μικρότερου στοιχείου από τη δομή πραγματοποιείται διαγράφοντας αρχικά την τιμή που είναι αποθηκευμένη στην πρώτη θέση ($H[1]$). Στη συνέχεια, αν N είναι το πλήθος των στοιχείων που είναι αποθηκευμένα στη δομή, το στοιχείο που βρίσκεται στη θέση $H[N]$, δηλαδή στην τελευταία θέση, τοποθετείται στην πρώτη θέση του πίνακα, δηλαδή στη θέση $H[1]$. Τέλος, αν η τιμή αυτή είναι μεγαλύτερη από οποιοδήποτε από τα δύο παιδιά της ρίζας, ανταλλάσσονται οι θέσεις των δύο στοιχείων. Στην περίπτωση που η τιμή που έχει τοποθετηθεί στη ρίζα είναι μεγαλύτερη και από τα δύο παιδιά, ανταλλάσσεται με το παιδί που έχει το μικρότερο χρόνο δρομολόγησης. Η διαδικασία αυτή συνεχίζεται μέχρι να τοποθετηθεί το στοιχείο στη σωστή θέση ώστε να ισχύουν ξανά οι ιδιότητες της δομής ή μέχρι να φτάσει σε τερματικό κόμβο (φύλλο).

Η διαδικασία της εισαγωγής ενός νέου κόμβου στη δομή αρχικοποιείται με την τοποθέτηση της νέας τιμής στην τελευταία θέση ($H[N + 1]$, με δεδομένο ότι η δομή περιέχει ήδη N στοιχεία). Συγκρίνεται, στη συνέχεια, η τιμή αυτή με την τιμή που περιέχεται στον πατέρα του νέου στοιχείου και αν βρεθεί μικρότερη ανταλλάσσονται. Η διαδικασία αυτή συνεχίζεται μέχρι ο πατέρας κάποιου κόμβου να περιέχει μικρότερη πληροφορία ή μέχρι να φτάσει η νέα τιμή στη ρίζα, δηλαδή στη θέση $H[1]$.

Σωρός - Δυναμική Αναπαράσταση (Dynamic Heap)

Η δυναμική αναπαράσταση της δομής του σωρού περιλαμβάνει τη χρήση δεικτών για τον εντοπισμό των στοιχείων του δέντρου. Οι κόμβοι του δέντρου περιέχουν την πληροφορία που αντιστοιχεί στον χρόνο δρομολόγησης καθώς και τρεις δείκτες που υποδεικνύουν τη θέση μνήμης των δύο παιδιών και του πατέρα του κάθε κόμβου. Φυσικά οποιοσδήποτε από τους δύο δείκτες προς τα δύο παιδιά μπορεί να είναι null. Ο δείκτης προς τον πατέρα του κόμβου της ρίζας δείχνει στον ίδιο τον κόμβο, στη ρίζα.

Οι διαδικασίες της εισαγωγής ενός νέου κόμβου και της διαγραφής του κόμβου με τη μικρότερη πληροφορία πραγματοποιούνται κατά τον τρόπο που περιγράφηκαν στην παρουσίαση της στατικής υλοποίησης της δομής του σωρού. Αυτό που αξίζει να σημειωθεί είναι ότι η αρχική εισαγωγή ενός νέου κόμβου γίνεται έτσι ώστε να διατηρείται η ιδιότητα του σχεδόν πλήρους δέντρου. Προκειμένου να είναι εφικτό κάτι τέτοιο υπάρχει ένας δείκτης που αντιστοιχεί στο πιο αριστερό φύλλο του δέντρου.

Θεωρητική Μελέτη της Δομής του Σωρού

Έστω ότι κάθε κόμβος της δομής έχει 2 παιδιά. Τότε το πλήθος των κόμβων που βρίσκονται στα πρώτα k επίπεδα είναι:

$$C_k = 1 + 2 + 2^2 + \ldots + 2^{k-1}$$

Δηλαδή,

$$C_k = (2^k - 1)$$

Το τελευταίο επίπεδο συνήθως δεν είναι πλήρες. Με βάση τα παραπάνω, το πλήθος των επιπέδων ενός σωρού n κόμβων είναι:

$$L = \log_2 (1 + n)$$

Έτσι, η πολυπλοκότητα της διαδικασίας της εισαγωγής είναι:

$$CI = 1 + 2 (L - h)$$

όπου h είναι το επίπεδο στο οποίο τελικά θα εισαχθεί ο νέος κόμβος. Η πολυπλοκότητα στη χειρίστη περίπτωση της διαδικασίας της εισαγωγής είναι $1+2L$ και αντιστοιχεί στην τιμή $h = 0$.

Προκειμένου να πραγματοποιηθεί η διαδικασία της διαγραφής, το στοιχείο που βρίσκεται στην πρώτη θέση, η μικρότερη δηλαδή πληροφορία, αντικαθίσταται από το στοιχείο που είναι αποθηκευμένο στην τελευταία θέση, όπως έχει ήδη περιγραφεί για τη δομή του σωρού είτε σε στατική είτε σε δυναμική αναπαράσταση. Στη συνέχεια η πληροφορία αυτή προωθείται προς τα φύλλα ώστε να τοποθετηθεί στη σωστή θέση και να ολοκληρωθεί έτσι η διαδικασία της διαγραφής. Αν η θέση που θα σταματήσει το στοιχείο είναι στο επίπεδο έστω h, τότε η χρονική πολυπλοκότητα της όλης διαδικασίας είναι ίση με:

$$CR = 1 + 3 (h + 1)$$

Η πολυπλοκότητα της διαδικασίας της διαγραφής στη χειρίστη περίπτωση είναι $1 + 3 (L+1)$ όταν ισχύει $h = L$.

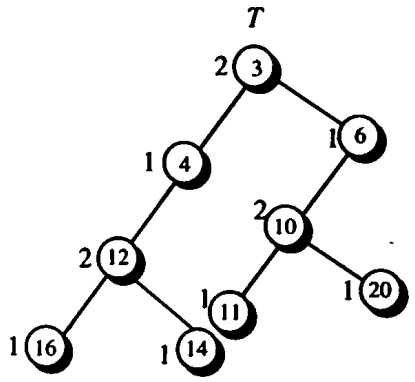
3.2 Η Δομή Leftist Heap

Η δομή Leftist Heap, η οποία παρουσιάζεται στη συνέχεια, παρέχει την δυνατότητα της εύκολης συνένωσης (melding). Αρχικά ορίζεται ο όρος *rank*.

Αν x είναι ένας κόμβος σε ένα πλήρες δυαδικό δέντρο, *rank* του κόμβου x ορίζεται να είναι το ελάχιστο μήκος ενός μονοπατιού από τον κόμβο x σε έναν εξωτερικό κόμβο. Οι εξωτερικοί κόμβοι δεν περιέχουν πληροφορία και θεωρείται ότι είναι NULL. Έτσι, ισχύει $rank(x) = 0$ αν ο κόμβος x είναι εξωτερικός και $rank(x) = 1 + \min\{rank(left(x)), rank(right(x))\}$ αν ο κόμβος x

είναι εσωτερικός. Ένα πλήρες δυαδικό δέντρο είναι leftist αν ισχύει *rank*(*left*(*x*)) ≥ *rank*(*right*(*x*)) για κάθε εσωτερικό κόμβο *x*. Σε ένα leftist δέντρο το δεξί μονοπάτι είναι το πιο σύντομο μονοπάτι από τη ρίζα σε έναν εξωτερικό κόμβο. Αποδεικνύεται με επαγωγή ότι αυτό το μονοπάτι έχει μήκος το πολύ log*n* [37].

Leftist heap ορίζεται να είναι ένα leftist tree που περιέχει στοιχεία τοποθετημένα στους εσωτερικούς κόμβους κατά τέτοιο τρόπο ώστε να ικανοποιείται η ιδιότητα του σωρού ελαχίστων, δηλαδή κάθε κόμβος να μην περιέχει τιμή μεγαλύτερη από εκείνη των παιδιών του. Θεωρείται ότι οι εξωτερικοί κόμβοι δεν περιέχουν πληροφορία και επομένως είναι NULL και ισχύει *rank*(NULL) = 0. Παράδειγμα μιας δομής leftist heap παρουσιάζεται στο Σχήμα 3.1.



Σχήμα 3.1

Κάθε κόμβος της δομής, εκτός από την πληροφορία, περιέχει και τρεις δείκτες, δύο προς τις θέσεις μνήμης των παιδιών του και έναν προς τον πατέρα καθώς και ένα πεδίο για την αποθήκευση της τιμής *rank*. Η μορφή του κάθε κόμβου είναι:

```
struct node {
        float information;
      int rank;
        struct node *right_child;
        struct node *left_child;
        struct node *parent;
      };
```

Η βασικότερη λειτουργία της δομής leftist heap είναι η συνένωση, για την πραγματοποίηση της οποίας συγχωνεύονται τα δεξιά μονοπάτια των δύο δομών έτσι ώστε οι πληροφορίες των κόμβων να ταξινομηθούν κατά αύξουσα σειρά. Στη συνέχεια υπολογίζονται οι νέες τιμές *rank* των κόμβων του νέου μονοπατιού και ανταλλάσσεται το δεξί με το αριστερό παιδί όπου είναι απαραίτητο. Όλη η διαδικασία ολοκληρώνεται σε O(log*n*) χρόνο, όπου *n* είναι το πλήθος των κόμβων των δύο δομών Leftist Heap. Στο Σχήμα 3.2 παρουσιάζεται το αποτέλεσμα *T3* της συνένωσης δύο δομών leftist heap *T1* και *T2*. Η διαδικασία της συνένωσης σε αλγοριθμική μορφή είναι η εξής:

Meld(heap *h1*, heap *h2*):

1. Αν *h1* = NULL return(*h2*);
2. Αν *h2* = NULL return(*h1*);
3. Αν *h1.key* > *h2.key* swap(*h1*, *h2*);
4. Αν *h1.right* = NULL τότε *h1.right* = *h2*,
   αλλιώς *Meld*(*h1.right*, *h2*);
5. Αν *rank*(*h1.left*) < *rank*(*h1.right*) τότε swap(*h1.left*, *h1.right*);
6. *rank*(*h1*) = *rank*(*h1.right*) + 1;
7. return(*h1*);

όπου *h.key*, *h.left*, *h.right* συμβολίζουν την πληροφορία, το αριστερό και το δεξί υποδέντρο του κόμβου *h*, ενώ *rank*(*h*) αντιστοιχεί στην τιμή *rank* του κόμβου *h*.

Η διαδικασία της εισαγωγής ενός νέου στοιχείου σε ένα leftist heap πραγματοποιείται με τη βοήθεια της συνένωσης. Το νέο στοιχείο θεωρείται σαν leftist heap και συγχωνεύεται με τη δομή. Αλγοριθμικά:

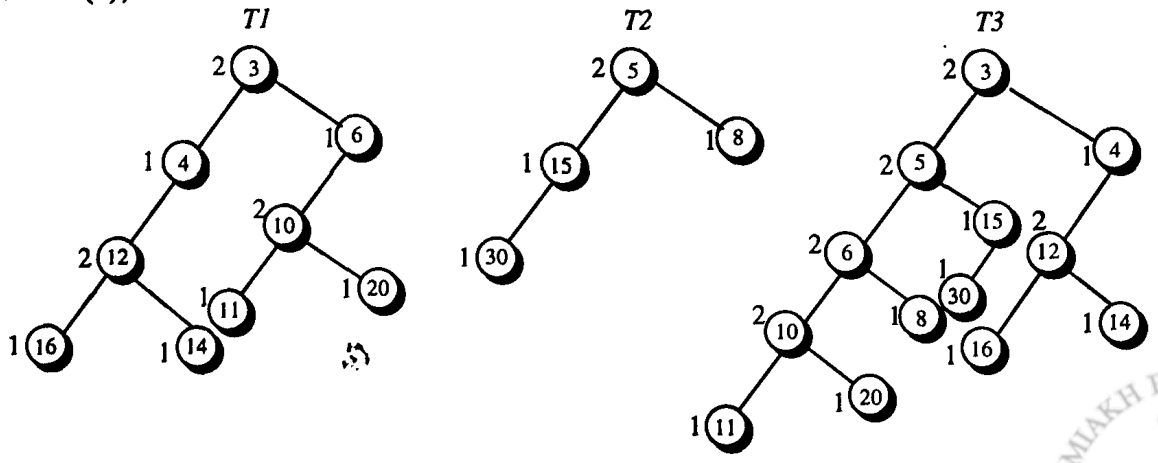
Insert(node *x*, heap *h*):

1. *x.left* = NULL;
2. *x.right* = NULL;
3. *rank*(*x*) = 1;
4. *h* = *Meld*(*x*, *h*);

Για τη διαγραφή του στοιχείου με τη μικρότερη πληροφορία, αφαιρείται ο κόμβος-ρίζα και συνενώνονται το δεξί και το αριστερό υποδέντρο. Τόσο η διαδικασία της εισαγωγής όσο και αυτή της διαγραφής του στοιχείου με τη μικρότερη τιμή απαιτούν χρόνο τάξης O(log*n*). Ακολουθεί η αλγοριθμική περιγραφή της διαδικασίας της διαγραφής.

Delete_minimum(heap *h*)

1. *x* = *h*;
2. *h* = *Meld*(*h.left*, *h.right*);
3. return(*x*);



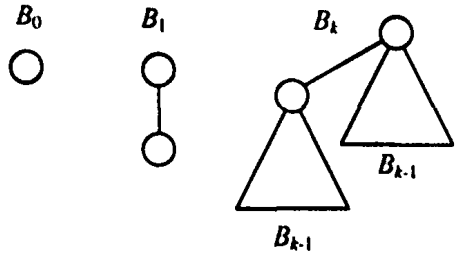
Σχήμα 3.2

3.3 Η Δομή Binomial Heap

Η δομή binomial heap είναι επίσης μια δομή η οποία έχει αποδοτική διαδικασία συνένωσης. Μια δομή binomial heap είναι μια συλλογή από binomial trees (διωνυμικά δέντρα). Στη συνέχεια ορίζεται η δομή binomial tree.

Binomial trees

Το binomial tree B_k είναι ένα ταξινομημένο δέντρο που ορίζεται αναδρομικά. Έτσι, το binomial tree B_0 αποτελείται από έναν και μόνο κόμβο. Το δέντρο B_k αποτελείται από δύο B_{k-1} δέντρα τα οποία είναι συνδεδεμένα μεταξύ τους ως εξής: η ρίζα του ενός είναι το πιο αριστερό παιδί της ρίζας του άλλου. Ο αναδρομικός ορισμός του διωνυμικού δέντρου B_k απεικονίζεται στο Σχήμα 3.3.



Σχήμα 3.3

Μερικές από τις ιδιότητες των διωνυμικών δέντρων δίνονται από το παρακάτω λήμμα.

Λήμμα 3.1. Για το binomial tree B_k ισχύουν τα εξής:

1. αποτελείται από 2^k κόμβους,
2. το ύψος του δέντρου είναι k,
3. υπάρχουν ακριβώς $\binom{k}{i}$ κόμβοι σε βάθος i, για $i = 0, 1, \ldots, k$ και
4. η ρίζα έχει βαθμό k, μεγαλύτερο από το βαθμό οποιουδήποτε άλλου κόμβου και επιπλέον αν θεωρηθεί ότι τα παιδιά της ρίζας αριθμούνται από αριστερά προς τα δεξιά με k-1, k-2, …, 0 τότε το παιδί i είναι η ρίζα του υποδέντρου B_i.

Από τις ιδιότητες 1 και 4 του παραπάνω λήμματος προκύπτει επίσης ότι ο μέγιστος βαθμός οποιουδήποτε κόμβου σε ένα binomial tree n κόμβων είναι $\log n$.

Binomial heaps

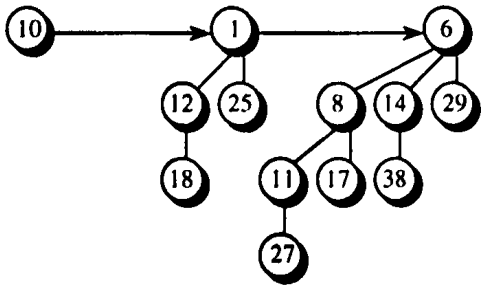
Ένας διωνυμικός σωρός (binomial heap) H είναι ένα σύνολο από διωνυμικά δέντρα και ικανοποιεί τις ακόλουθες ιδιότητες:

1. Κάθε διωνυμικό δέντρο στο H ικανοποιεί την ιδιότητα του σωρού, δηλαδή η τιμή ενός κόμβου είναι μεγαλύτερη ή ίση από την τιμή του κόμβου-πατέρα.
2. Υπάρχει το πολύ ένα διωνυμικό δέντρο στο H του οποίου η ρίζα έχει τον ίδιο βαθμό.

Η δεύτερη ιδιότητα υποδηλώνει ότι ένας διωνυμικός σωρός n κόμβων αποτελείται από το πολύ $\lfloor \log n \rfloor + 1$ διωνυμικά δέντρα. Αυτό ισχύει αφού η δυαδική αναπαράσταση του n έχει $\lfloor \log n \rfloor + 1$ bits, έστω $\langle b_{\lfloor \log n \rfloor}, b_{\lfloor \log n \rfloor - 1}, \ldots, b_0 \rangle$ έτσι ώστε:

$$n = \sum_{i=0}^{\lfloor \log n \rfloor} b_i 2^i$$

Επομένως, από την ιδιότητα 1 του Λήμματος 3.1, το διωνυμικό δέντρο B_i εμφανίζεται στη δομή αν και μόνον αν το bit $b_i = 1$. Έτσι, ο διωνυμικός σωρός περιέχει το πολύ $\lfloor \log n \rfloor + 1$ διωνυμικά δέντρα.



Σχήμα 3.4

Στο Σχήμα 3.4 παρουσιάζεται ένα παράδειγμα διωνυμικού σωρού 13 κόμβων. Η δυαδική αναπαράσταση του 13 είναι <1101> και ο σωρός H αποτελείται από τα δέντρα B_3, B_2 και B_0 που έχουν 8, 4 και 1 κόμβο αντίστοιχα. Οι ρίζες των διωνυμικών δέντρων είναι συνδεδεμένες σε λίστα, που ονομάζεται λίστα ριζών (root list). Οι βαθμοί των ριζών αυξάνονται αυστηρά καθώς διασχίζεται η λίστα. Από τη δεύτερη ιδιότητα της δομής προκύπτει ότι σε μια δομή n κόμβων οι βαθμοί των ριζών είναι υποσύνολο του $\{0, 1, \ldots, \lfloor \log n \rfloor\}$. Κάθε κόμβος περιέχει, εκτός από ένα πεδίο για την πληροφορία που του αντιστοιχεί, ένα επιπλέον πεδίο για το βαθμό του και τρεις δείκτες, προς τον πατέρα, το πιο αριστερό παιδί και τον αδερφό του κόμβου. Έτσι, η μορφή του κόμβου είναι:

```
struct node {
        float key;
       int degree;
        struct node *child;
        struct node *sibling;
        struct node *parent;
       };
```

Πριν την περιγραφή των διαδικασιών της εισαγωγής ενός νέου στοιχείου και της διαγραφής του στοιχείου με τη μικρότερη τιμή, κρίνεται απαραίτητο να περιγραφεί η διαδικασία της συνένωσης (*union*) δύο δομών διωνυμικού σωρού.

Κατά τη διαδικασία της συνένωσης συνδέονται τα διωνυμικά δέντρα των οποίων οι ρίζες έχουν τον ίδιο βαθμό. Η παρακάτω διαδικασία συνδέει το B_{k-1} δέντρο με ρίζα τον κόμβο y με το B_{k-1} δέντρο με ρίζα τον κόμβο z, δηλαδή ο κόμβος z γίνεται πατέρας του κόμβου y. Έτσι, ο κόμβος z γίνεται η ρίζα ενός B_k δέντρου.

Link(y, z):
1. $y.parent = z$;
2. $y.sibling = z.child$;
3. $z.child = y$;
4. $z.degree = z.degree + 1$;

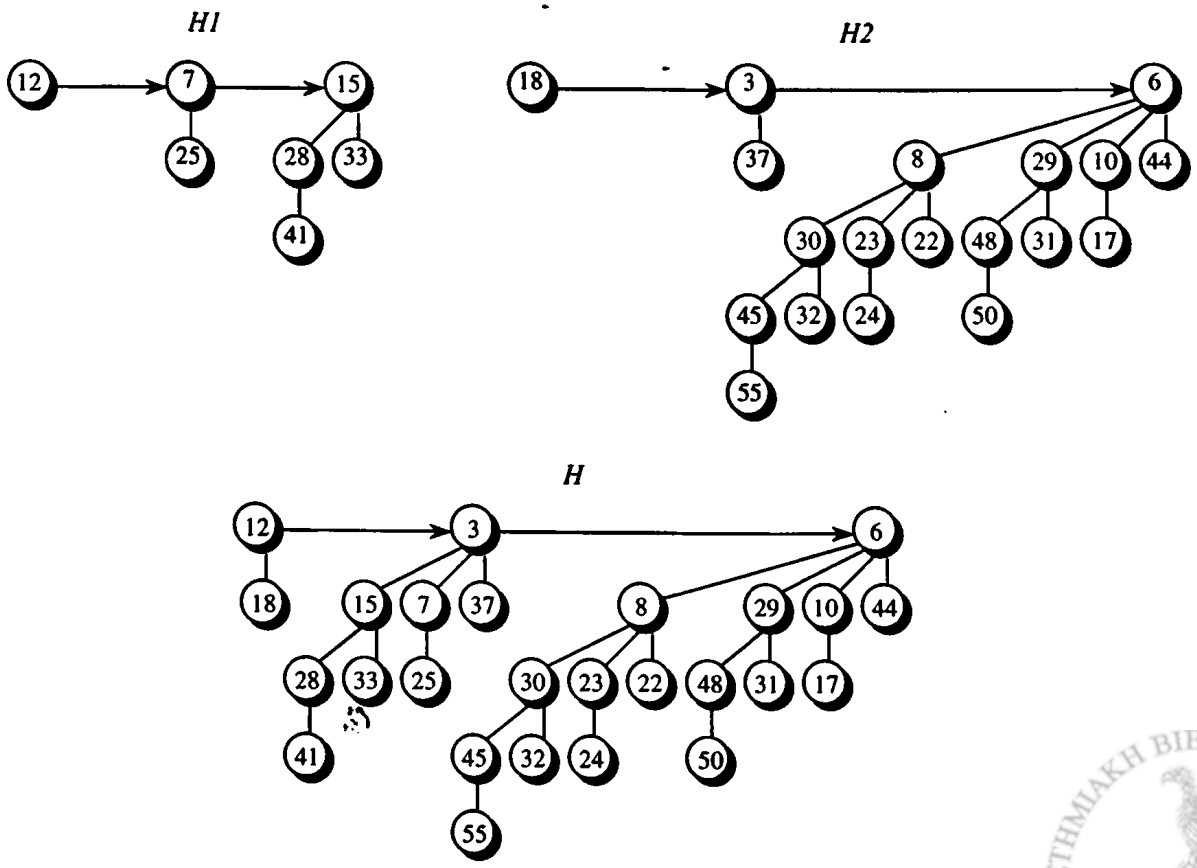
όπου *u.parent*, *u.sibling*, *u.child* και *u.degree* είναι ο πατέρας, ο αδερφός, το παιδί και ο βαθμός του κόμβου u, αντίστοιχα.

Η διαδικασία της συνένωσης περιγράφεται αλγοριθμικά ως εξής:

Union(*H1*, *H2*):

1. H = *Merge*(*H1*, *H2*);
2. If H = NULL return(H);
3. *prev_x* = NULL;
4. $x = H$;
5. *next_x* = *x.sibling*;
6. while *next_x* ≠ NULL
   if *x.degree* ≠ *next_x.degree* or
   (*next_x.sibling* ≠ NULL and *next_x.sibling.degree* = *x.degree*)
   then {*prev_x* = *x*, *x* = *next_x*}
   else if *x.key* ≤ *next_x.key*
   then {*x.sibling* = *next_x.sibling*, *Link*(*next_x*, *x*)}
   else if *prev_x* = NULL then H = *next_x* else *prev_x.sibling* = *next_x*
   Link(*x*, *next_x*)
   x = *next_x*
   next_x = *x.sibling*;
7. return H;

Η παραπάνω διαδικασία έχει δύο φάσεις. Κατά την πρώτη φάση, καλείται η συνάρτηση Merge η οποία συγχωνεύει τις λίστες των ριζών (root lists) των σωρών *H1* και *H2* σε μια λίστα, πρώτος κόμβος της οποίας είναι ο κόμβος H, η οποία είναι ταξινομημένη με βάση το βαθμό κατά μονοτονικά αύξουσα σειρά. Υπάρχει βέβαια η πιθανότητα να υπάρχουν περισσότερες από μια ρίζες (το πολύ δύο) που έχουν τον ίδιο βαθμό κι έτσι κατά τη δεύτερη φάση συνδέονται ρίζες που έχουν τον ίδιο βαθμό μέχρι να υπάρχει μόνο μια ρίζα με συγκεκριμένο βαθμό. Το αποτέλεσμα H της συνένωσης των σωρών *H1* και *H2* παρουσιάζεται στο Σχήμα 3.5.



Σχήμα 3.5

Η πολυπλοκότητα χρόνου της διαδικασίας που περιγράφηκε είναι O(log*n*), όπου *n* είναι ο συνολικός αριθμός των κόμβων που περιέχουν οι δύο σωροί, *H1* και *H2*. Το αποτέλεσμα αυτό προκύπτει ως εξής:

Έστω ότι οι σωροί *H1* και *H1* περιλαμβάνουν n_1 και n_2 κόμβους, αντίστοιχα, έτσι ώστε $n_1 + n_2 = n$. Τότε ο σωρός *H1* περιλαμβάνει το πολύ $\lfloor \log n_1 \rfloor + 1$ ρίζες και ο σωρός *H2* το πολύ $\lfloor \log n_2 \rfloor + 1$ ρίζες. Έτσι, για το αποτέλεσμα *H* θα υπάρχουν το πολύ $\lfloor \log n_1 \rfloor + \lfloor \log n_2 \rfloor + 2 \leq 2 \lfloor \log n \rfloor + 2 = O(\log n)$ ρίζες αμέσως μετά την κλήση της διαδικασίας Merge. Ο χρόνος επομένως που απαιτείται για τη συνάρτηση Merge είναι O(log*n*). Κάθε επανάληψη του while απαιτεί σταθερό χρόνο και το πλήθος των επαναλήψεων είναι $\lfloor \log n_1 \rfloor + \lfloor \log n_2 \rfloor + 2$, αφού κάθε επανάληψη είτε μετακινεί τους βοηθητικούς δείκτες κατά μια θέση δεξιά στην λίστα των ριζών (root list) είτε διαγράφει από την τελευταία μια ρίζα. Επομένως ο συνολικός χρόνος που απαιτείται είναι της τάξης του O(log*n*).

Για την εισαγωγή ενός νέου στοιχείου σε μια δομή διωνυμικού σωρού, θεωρείται ότι το νέο αυτό στοιχείο αποτελεί σωρό ενός κόμβου οπότε ενώνεται με τη δομή *n* κόμβων σε χρόνο O(log*n*) σύμφωνα με τη διαδικασία που έχει ήδη περιγραφεί. Αλγοριθμικά η διαδικασία της εισαγωγής περιγράφεται ως εξής:

Insert (heap *H*, node *x*):

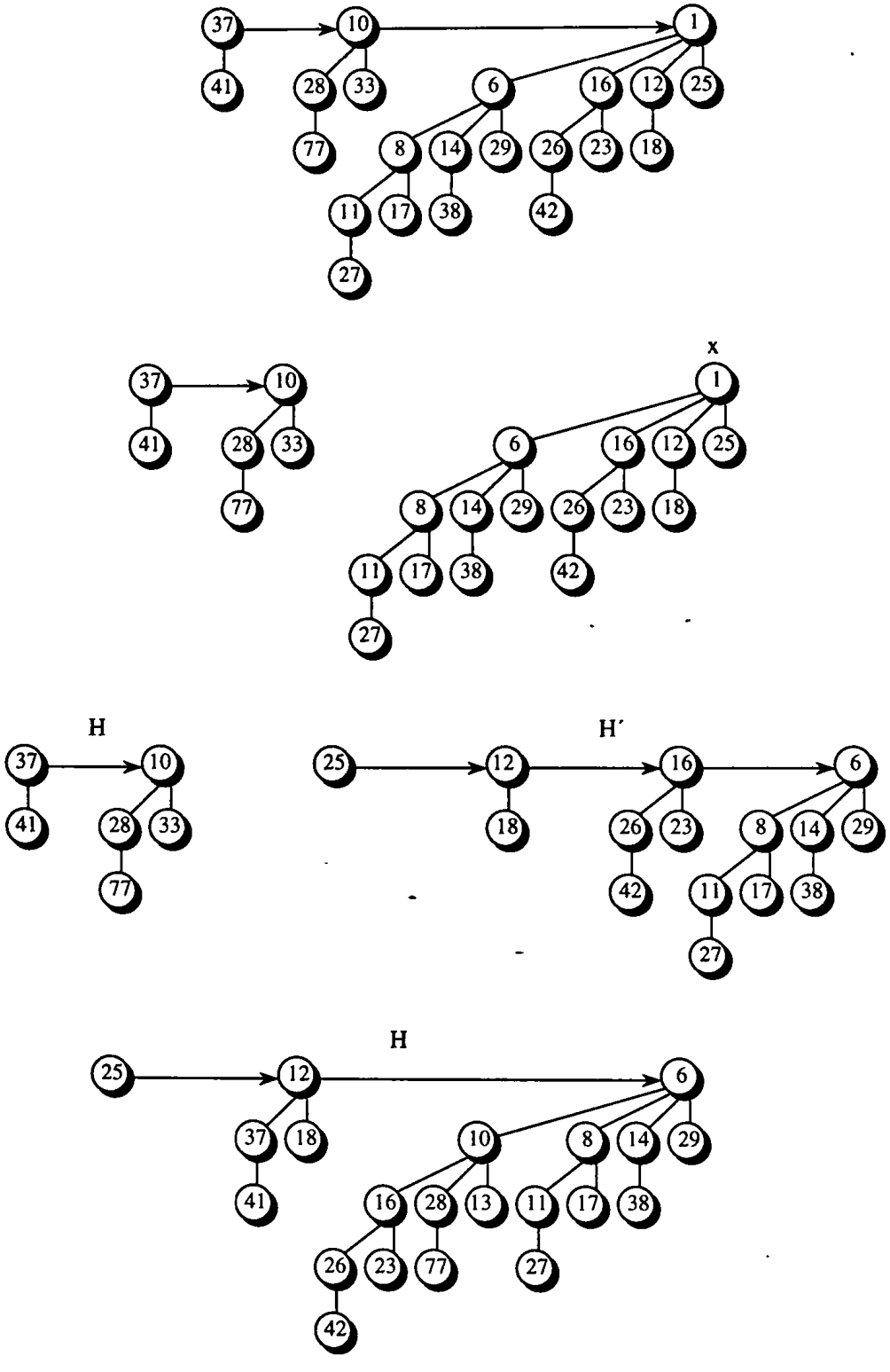
1. *x.parent* = NULL;
2. *x.child* = NULL;
3. *x.sibling* = NULL;
4. *x.degree* = NULL;
5. *H'* = *x*;
6. *Union*(*H*, *H'*);

Η διαδικασία της διαγραφής του στοιχείου με τη μικρότερη τιμή, η οποία ολοκληρώνεται σε χρόνο O(log*n*), πραγματοποιείται ως εξής:

Delete_minimum(*H*):

1. Εντοπίζεται και αφαιρείται από τη λίστα των ριζών η ρίζα *x* με τη μικρότερη τιμή;
2. Αντιστρέφεται η σειρά των κόμβων που αποτελούν τη λίστα των παιδιών του κόμβου *x* και δημιουργείται έτσι ο σωρός *H'*;
3. *H* = *Union*(*H*, *H'*);
4. return *x*;

Η παραπάνω διαδικασία πραγματοποιείται σε O(log*n*) χρόνο σε μια δομή που αποτελείται από *n* κόμβους. Στο Σχήμα 3.6 παρουσιάζεται αναλυτικά όλη η διαδικασία της διαγραφής του στοιχείου με τη μικρότερη τιμή από ένα διωνυμικό σωρό 22 κόμβων.



Σχήμα 3.6

3.4 Η Δομή *P*-tree

Η δομή *P*-tree (Priority Tree) είναι ένα δυαδικό δέντρο το οποίο είναι είτε κενό είτε αποτελείται από ένα αριστερό μονοπάτι που είναι μια φθίνουσα ακολουθία από κόμβους κάθε ένας από τους οποίους, εκτός από τον τελευταίο, έχει σαν δεξί υποδέντρο ένα *P*-tree. Ο κόμβος με τη μεγαλύτερη πληροφορία βρίσκεται πάντα στη ρίζα. Οι διάδοχοι κάθε κόμβου έχουν πληροφορία μικρότερη από την πληροφορία του κόμβου αυτού και οι τιμές του δεξιού υποδέντρου του είναι μεγαλύτερες από όλες τις πληροφορίες του αριστερού υποδέντρου. Έτσι, δεν υπάρχει δεξί υποδέντρο σε κάποιον κόμβο, αν δεν υπάρχει αριστερό. Επιπλέον, ο τερματικός κόμβος στο πιο αριστερό μονοπάτι περιέχει τη μικρότερη πληροφορία.

Το Σχήμα 3.7 παρουσιάζει ένα παράδειγμα της δομής *P*-tree που αποτελείται από 11 κόμβους.

Κάθε κόμβος του δέντρου, εκτός από την πληροφορία, περιέχει και τρεις δείκτες, δύο προς τις θέσεις μνήμης των παιδιών του και έναν προς τον πατέρα. Η μορφή του κάθε κόμβου είναι:

```
struct node {
        float information;
        struct node *right_child;
        struct node *left_child;
        struct node *parent;
      };
```

Η διαδικασία της εισαγωγής ενός νέου στοιχείου x σε ένα *P*-tree T πραγματοποιείται αναδρομικά ως εξής:

P-insert(x, T):

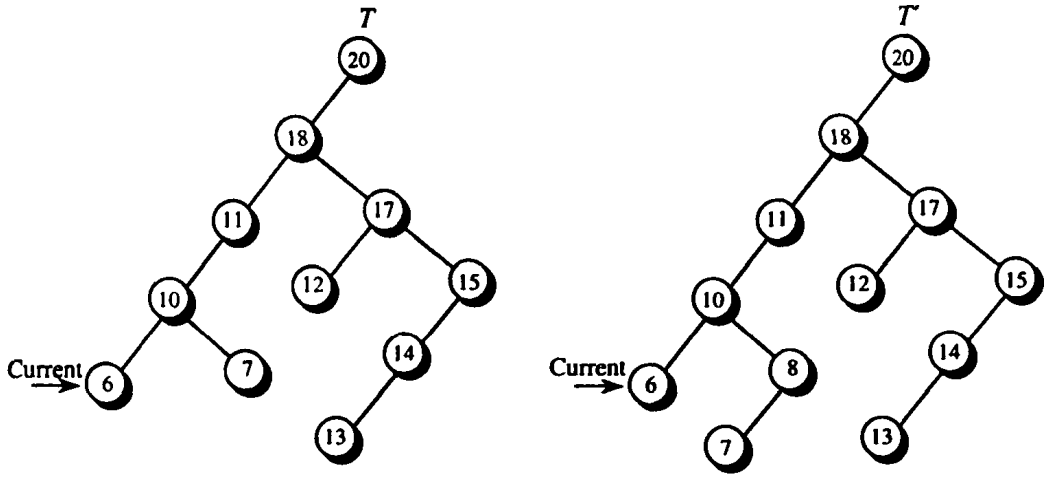
1. Αν $T = \emptyset$ ή $x.value \geq T.value$, τότε ο κόμβος x γίνεται η νέα ρίζα και το T γίνεται αριστερό του υποδέντρο;
2. Αλλιώς, αναζητείται στο αριστερό μονοπάτι του T, ξεκινώντας από τη ρίζα, ο πρώτος κόμβος y, αν υπάρχει, τέτοιος ώστε $y.value \leq x.value$;
   2.1 Αν δεν υπάρχει, ο κόμβος x γίνεται το νέο αριστερό φύλλο;
   2.2 Αλλιώς, $y.value \leq x.value \leq z.value$, όπου z είναι ο πατέρας του y ($y = z.left$);
   P-insert(x, $z.right$);

όπου τα x, y και z δηλώνουν κόμβους και τα $u.left$, $u.right$ και $u.value$ δηλώνουν το αριστερό υποδέντρο, το δεξί υποδέντρο και την τιμή του κόμβου u, αντίστοιχα.

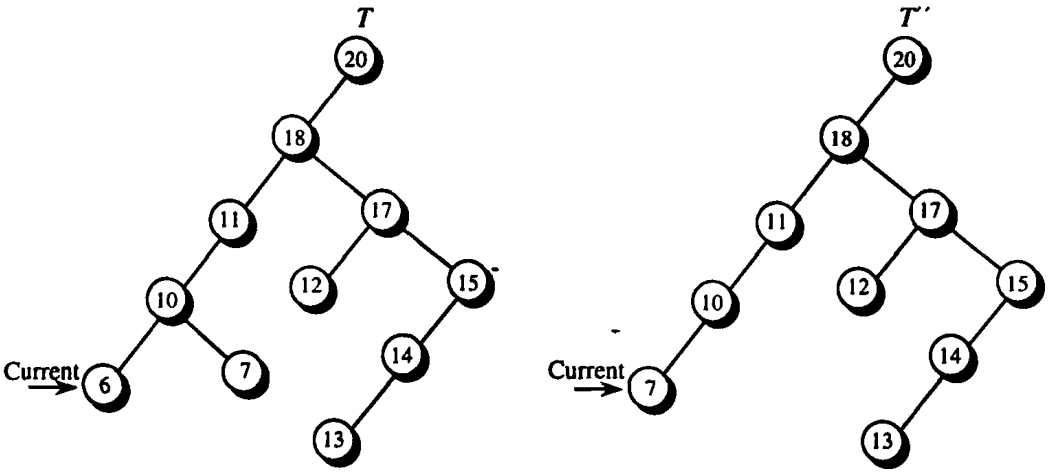
Στο Σχήμα 3.7 παρουσιάζεται η δομή T' που προκύπτει αν εισαχθεί ένας νέος κόμβος u με την πληροφορία $u.value = 8$ στο *P*-tree T που παρουσιάζεται στο ίδιο σχήμα.

Ο εντοπισμός του κόμβου με τη μικρότερη πληροφορία μπορεί να πραγματοποιηθεί σε σταθερό χρόνο, εφόσον υπάρχει μία επιπλέον μεταβλητή τύπου δείκτη που αντιστοιχεί στον τερματικό κόμβο του πιο αριστερού μονοπατιού. Μετά την αφαίρεση αυτού του κόμβου, το τελευταίο δεξί υποδέντρο, δηλαδή αυτό που αντιστοιχεί στον πατέρα του κόμβου που αφαιρέθηκε, αν δεν είναι κενό, παίρνει τη θέση του κόμβου που αφαιρέθηκε, σαν αριστερό

υποδέντρο. Αν πραγματοποιηθεί η διαδικασία της διαγραφής στη δομή T του Σχήματος 3.8, θα προκύψει η δομή T''.



Σχήμα 3.7



Σχήμα 3.8

Θεωρητική Μελέτη της Δομής P-tree

Η θεωρητική μελέτη της δομής του *P*-tree που παρατίθεται στη συνέχεια πραγματοποιήθηκε από τους A. Jonassen και O.J. Dahl. Συγκεκριμένα, πρόκειται για τη μελέτη με θέμα Analysis of an Algorithm for Priority Queue Administration που υπάρχει στο *BIT* 15 (1975) 409-422.

Έστω μια ακολουθία n στοιχείων που αποτελούν τα στοιχεία μιας δομής *P*-tree. Χωρίς περιορισμό της γενικότητας, οποιαδήποτε ακολουθία n στοιχείων μπορεί να παρασταθεί σαν μετάθεση των 1, 2, ..., n. Έστω $\Pi^{(n)}$, $n \geq 0$, το σύνολο όλων των μεταθέσεων των αριθμών 1, 2, ..., n και έστω $P^{(n)}$ το σύνολο των *P*-trees που αποτελούνται από n στοιχεία και που προκύπτουν αν πραγματοποιηθούν διαδοχικές εισαγωγές των στοιχείων κάθε μιας από αυτές τις μεταθέσεις. Θεωρείται ότι κάθε μετάθεση έχει την ίδια πιθανότητα να επιλεγεί, η οποία είναι

$1/n!$. Αφού γενικά δεν ισχύει ότι διαφορετικές μεταθέσεις από το σύνολο $\Pi^{(n)}$ αντιστοιχούν και σε διαφορετικά δέντρα του συνόλου $P^{(n)}$ οι πιθανότητες που αντιστοιχούν στα δέντρα (στοιχεία) του συνόλου $P^{(n)}$ είναι διαφορετικές μεταξύ τους.

Το σύνολο $P^{(n)}$ μπορεί να θεωρηθεί σαν το σύνολο όλων των δυαδικών δέντρων T μεταθεματικής (postfix) διάταξης που αποτελούνται από n κόμβους. Σε κάθε δέντρο T ανατίθεται η πιθανότητα επιλογής του:

$$\text{Prob}(T) = N(T) / n!$$

όπου $N(T)$ είναι το πλήθος των μεταθέσεων που ανήκουν στο σύνολο $\Pi^{(n)}$ που αντιστοιχεί στο T.

Η αποτελεσματικότητα της διαδικασίας της εισαγωγής μπορεί να εκτιμηθεί από το πλήθος των συγκρίσεων (S_n) που πραγματοποιούνται μεταξύ των στοιχείων προκειμένου να εισαχθεί ένα τυχαίο στοιχείο x που ανήκει στο σύνολο $X^{(n)} = \{1/2, 3/2, ..., (n+1)/2\}$ σε ένα τυχαίο δέντρο του συνόλου $P^{(n)}$. Σημειώνεται ότι κάθε στοιχείο έχει την ίδια πιθανότητα να επιλεγεί, η οποία είναι ίση με $1/(n+1)$. Ακόμη, για το πλήθος των συγκρίσεων ισχύει ότι $S_n = V_n + W_n$ (1) όπου V_n είναι το μέσο πλήθος συγκρίσεων που πραγματοποιούνται με στοιχεία αριστερού μονοπατιού και W_n είναι το μέσο πλήθος συγκρίσεων με στοιχεία δεξιού μονοπατιού. Οι δύο όροι υπολογίζονται στη συνέχεια.

➢ Το Αριστερο Μονοπατι

Ο υπολογισμός του V_n απλοποιείται αν παρατηρήσει κανείς ότι ένα αριστερό μονοπάτι μήκους t:

$$L = (q_1, q_2, ..., q_t),$$

όπου $2 \le t \le n$ και $n = q_1 > q_2 > ... > q_t = 1$, εμφανίζεται στο $P^{(n)}$ με την ίδια πιθανότητα με την οποία εμφανίζεται το μονοπάτι

$$L' = (n + 1 - q_t, n + 1 - q_{t-1}, ..., n + 1 - q_1).$$

Η παραπάνω πρόταση αποδεικνύεται ως εξής: ένα στοιχείο q_k μιας μετάθεσης $p = (p_1, p_2, ..., p_n) \in \Pi^{(n)}$ ανήκει στο αριστερό μονοπάτι του αντίστοιχου δέντρου T αν και μόνο αν είναι είτε μεγαλύτερο είτε μικρότερο από όλα τα στοιχεία $p_1, p_2, ..., p_{k-1}$ της μετάθεσης p. Έστω μια μετάθεση $p' = (p_1', p_2', ..., p_n') \in \Pi^{(n)}$ και το αντίστοιχο δέντρο T' όπου $p_i' = n + 1 - p_i$, $i = 1, 2, ..., n$. Αν το στοιχείο p_k βρίσκεται στο αριστερό μονοπάτι του T, τότε το στοιχείο p_k' είναι είτε μικρότερο είτε μεγαλύτερο από όλα τα στοιχεία της p που βρίσκονται αριστερά του, που σημαίνει ότι το p_k' βρίσκεται στο αριστερό μονοπάτι του T'. Έτσι, το αριστερό μονοπάτι του T' είναι το L'.

Χρησιμοποιώντας αυτήν την ιδιότητα της συμμετρίας προκύπτει ότι το μέσο πλήθος συγκρίσεων αριστερού μονοπατιού είναι:

$$V_n = \frac{(\lambda_n / 2 + 1)(n-1) + \lambda_n + 1}{n+1} = \frac{\lambda_n}{2} + \frac{n}{n+1} \quad (2)$$

όπου λ_n είναι το μέσο μήκος αριστερού μονοπατιού στο $P^{(n)}$.

Έστω $a_{n,k}$ η πιθανότητα να επιλεγεί ένα δέντρο από το $P^{(n)}$ με μήκος αριστερού μονοπατιού k. Η εισαγωγή ενός στοιχείου $x \in X^{(n)}$ σε ένα τέτοιο δέντρο αυξάνει το μήκος του αριστερού μονοπατιού κατά 1 στην περίπτωση κατά την οποία $x = 1/2$ ή $x = n + 1/2$, ενώ σε κάθε άλλη περίπτωση δεν αλλάζει το μήκος. Έτσι ισχύει η ακόλουθη αναδρομική σχέση:

$$(n+1)\, a_{n+1,k} = (n-1)\, a_{n,k} + 2\, a_{n,k-1}. \qquad (3)$$

Αν οριστεί $G_n(z) = \sum_{k=2}^{n} \alpha_{n,k} z^k$ θα ισχύει $\lambda_n = G'_n(1)$. Από τη σχέση (3) προκύπτει:

$$(n+1)\, G_{n+1}(z) = (n - 1 + 2z)\, G_n(z) \qquad (n \geq 2)$$
$$G_2(z) = z^2$$

Παραγωγίζοντας και θέτοντας $z = 1$ προκύπτει

$$\lambda_{n+1} = \lambda_n + 2/(n+1) \qquad (n \geq 2)$$
$$\lambda_2 = 2$$

Από την οποία προκύπτει η σχέση (4):

$$\lambda_n = 2H_n - 1 \quad (n \geq 2) \qquad (4)$$

όπου $H_n = \sum_{k=1}^{n} 1/k$. Η σχέση (4) ισχύει και για $n = 1$.

➢ Τα Δεξια Υποδεντρα

Έστω ότι το σύνολο $B_{a,k}^{(n)}$ όλων των εμφανίσεων δεξιών υποδέντρων που περιέχουν τους εξής k κόμβους: $a + 1$, $a + 2$, ..., $a + k$ αποτελείται από πλήρεις εμφανίσεις του $P^{(k)}$. Τα στοιχεία κάθε $P^{(k)}$ έχουν τις σωστές σχετικές πιθανότητες (αλλά τα στοιχεία τροποποιούνται με την πρόσθεση του a). Προκειμένου να γίνει αυτό κατανοητό, έστω $p \in \Pi^{(n)}$, με αντίστοιχο δέντρο $T \in P^{(n)}$ που περιέχει ένα τέτοιο δεξί υποδέντρο B. Ο κόμβος-πατέρας του B του αριστερού μονοπατιού είναι αυτός που περιέχει το στοιχείο $a + k + 1$, του οποίου αριστερός διάδοχος είναι το στοιχείο a.

Στην μετάθεση p, τα στοιχεία $a + 1$, ..., $a + k$ εμφανίζονται όλα στα δεξιά του a και του $a + k + 1$. Αν διατηρηθεί το σύνολο των θέσεων στην p αυτών των στοιχείων σταθερό αλλά μετατεθούν εσωτερικά, οι μόνες αλλαγές στο T θα πραγματοποιηθούν στο υποδέντρο B και θα δημιουργηθεί πλήρες $P^{(k)}$ από όλες τις $k!$ μεταθέσεις των $a + 1$, ..., $a + k$. Είναι φανερό ότι η παραπάνω διαδικασία διαμερίζει εξαντλητικά το σύνολο των δέντρων $T \in P^{(n)}$ που περιέχουν ένα δεξί υποδέντρο $B \in B_{a,k}^{(n)}$ σε ξένα υποσύνολα που κάθε ένα από αυτά είναι ισόμορφο με το $P^{(k)}$. Συνεπώς, ο μέσος αριθμός συγκρίσεων στο B για την εισαγωγή του x, $a < x < a + k + 1$, είναι ίσος με S_k και για τον υπολογισμό του W_n υπολογίζεται η πιθανότητα $q_{a,k}^{(n)}$ οι κόμβοι a και $a + k + 1$ να είναι γείτονες στο αριστερό μονοπάτι στο $P^{(n)}$. Οι συνθήκες για να ισχύει αυτό είναι:

1. αν $a + k + 1$ είναι στα δεξιά του a στην p, τότε όλα τα στοιχεία στα αριστερά του $a + k + 1$ πρέπει να είναι μικρότερα ή ίσα με το a, και
2. αν το a είναι στα δεξιά του $a + k + 1$ στην p, τότε όλα τα στοιχεία στα αριστερά του a πρέπει να είναι μεγαλύτερα ή ίσα με το $a + k + 1$.

Για την πιθανότητα $q_{a,k}^{(n)}$ ισχύει:

$$q_{a,k}^{(n)} = \frac{1}{(n-a)(n-a+1)} + \frac{1}{(a+k)(a+k+1)}, (1 \le a < n, 0 \le k < n - a) \quad (5)$$

Έτσι, θα ισχύει: $W_n = \sum_{a=1}^{n-1} \sum_{k=0}^{n-a-1} \frac{k+1}{n+1} q_{a,k}^{(n)} S_k$ (6)

Αν η πιθανότητα να ισχύει $a < x < a + k + 1$ είναι $(k + 1) / (n + 1)$.

➢ ΑΝΑΜΕΝΟΜΕΝΟ ΠΛΗΘΟΣ ΣΥΓΚΡΙΣΕΩΝ

Από τις σχέσεις (1), (2), (4), (5) και (6) προκύπτει ότι:

$$S_n = H_n + 2^{-1}(n-1)/(n+1) + [2/(n(n+1))] \sum_{k=0}^{n-1} (n-k-1) S_k \quad (n \ge 1) \quad (7)$$

Ισχύει $S_0 = 0$, $S_1 = 1$ οπότε:

$$S_n = (1/3) H_{n+1}^2 + (10/9) H_{n+1} - (1/3) H_{n+1}^{(2)} - 28/27 \quad (n \ge 2)$$

όπου $H_{n+1}^{(2)} = \sum_{k=1}^{n+1} k^{-2}$ και $\lim_{n \to \infty} H_{n+1}^{(2)} = \pi^2 / 6$.

Η σχέση (7) μπορεί να αποδειχθεί με επαγωγή . Σημειώνεται ακόμη ότι ισχύει $S_n = O(\log^2 n)$.

➢ ΑΝΑΜΕΝΟΜΕΝΟ ΒΑΘΟΣ ΤΗΣ ΑΝΑΔΡΟΜΗΣ

Έστω R_n το αναμενόμενο βάθος της αναδρομής κατά τη διαδικασία της εισαγωγής, το οποίο είναι κατά ένα μεγαλύτερο από το αναμενόμενο πλήθος των «βημάτων προς τα δεξιά». Έτσι, ισχύει:

$$R_n = 1 + [2/(n(n+1))] \sum_{k=0}^{n-1} (n-1-k) R_k \qquad (n \ge 1)$$
$$R_0 = 1$$

και η λύση είναι:

$$R_n = (2/3) H_{n+1} + 1/9 \qquad (n \ge 2)$$
$$R_0 = R_1 = 1.$$

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

Έστω ότι κάθε κόμβος του *P*-tree έχει μια μεταβλητή-δείκτη που αντιστοιχεί στον τελευταίο κόμβο του πιο αριστερού μονοπατιού του δεξιού του υποδέντρου. Αυτό διευκολύνει την ολοκλήρωση της διαδικασίας της διαγραφής, αφού μετά τη διαγραφή του μικρότερου στοιχείου πρέπει να εντοπισθεί το νέο μικρότερο στοιχείο που υπάρχει στη δομή.

Το αναμενόμενο μήκος του αριστερού μονοπατιού του τελευταίου δεξιού υποδέντρου, RL_n, είναι ίσο με:

$$RL_n = 3/2 - 4/n + 1/(n(n-1)) \quad (n \geq 2).$$

Αφού το RL_n φράσσεται από μια σταθερά, ακόμη και στην περίπτωση που δεν υπάρχει η μεταβλητή-δείκτης που προαναφέρθηκε, η διαδικασία της διαγραφής του μικρότερου στοιχείου εξακολουθεί να πραγματοποιείται σε σταθερό χρόνο.

Σημειώνεται ότι αν πραγματοποιηθεί η διαδικασία της διαγραφής σε ένα τυχαίο δέντρο του $P^{(n)}$ δεν προκύπτει ένα τυχαίο δέντρο του $P^{(n-1)}$. Έτσι, αν εναλλάσσονται οι διαδικασίες της εισαγωγής και της διαγραφής προκύπτουν αποτελέσματα διαφορετικά από την (7). Όμως πειραματικά αποτελέσματα δείχνουν ότι οι διαφορές αυτές είναι αμελητέες.

➢ ΤΟ ΜΗΚΟΣ ΤΟΥ ΔΕΞΙΟΥ ΜΟΝΟΠΑΤΙΟΥ

Το πλήθος των κόμβων του δεξιού μονοπατιού, συμπεριλαμβάνοντας και τον κόμβο-ρίζα, φράσσεται από μια σταθερά. Συνεπώς, η ρίζα ενός τυχαίου *P*-tree μπορεί να διαγραφεί με τη χρήση κάποιου αλγόριθμου που εκτελείται σε σταθερό χρόνο. Μετά τη διαγραφή, όλοι οι κόμβοι του δεξιού μονοπατιού μετακινούνται μια θέση προς τη ρίζα και το τελευταίο αριστερό υποδέντρο, αν δεν είναι κενό, προσαρτάται στο δεξί μονοπάτι. Για το αναμενόμενο μήκος του δεξιού μονοπατιού ρ_n ισχύει:

$$\begin{cases} \rho_0 = 0 \\ \rho_1 = \rho_2 = 1 \\ \rho_n = 1 + \sum_{k=0}^{n-2} \rho_k q^{(n)}_{n-k-1,n} \quad (n \geq 3) \end{cases}$$

όπου $q^{(n)}_{a,k}$ είναι η ποσότητα που δίνεται από τη σχέση (5). Έχει αποδειχθεί ότι ισχύει

$$\lim_{n\to\infty} \rho_n = \sum_{j=0}^{\infty} 2^j / [(j+1)!]^2 = 1.6261754$$

➢ ΤΟ ΠΛΗΘΟΣ ΤΩΝ ΔΙΑΦΟΡΕΤΙΚΩΝ *P*-TREES

Όπως έχει ήδη αναφερθεί παραπάνω, περισσότερες από μια μεταθέσεις είναι δυνατόν να δημιουργούν, μετά από διαδοχικές εισαγωγές των στοιχείων τους, το ίδιο P-tree. Έχει αποδειχθεί ότι για το πλήθος N_n των διαφορετικών δέντρων που ανήκουν στο $P^{(n)}$ ισχύει:

$$\begin{cases} N_n + N_{n-1} = \sum_{k=0}^{n-1} N_k N_{n-1-k} & (n \geq 2) \\ N_0 = N_1 = 1 \end{cases}$$

με λύση:

$$N_n = \frac{(-1)^n}{2^{2n+3}} \sum_{k=0}^{n+1} (-3)^k \frac{\binom{2k}{k}\binom{2n-2k+2}{n-k+1}}{(2k-1)(2n-2k+1)} \qquad (n \geq 1)$$

Έχει επίσης αποδειχθεί ότι $\lim_{n\to\infty} N_{n+1} / N_n = 3$. Οι τρεις προηγούμενες σχέσεις είναι σε αντιστοιχία με τις γνωστές ακόλουθες σχέσεις για το πλήθος K_n των διαφορετικών δυαδικών δέντρων.

$$\begin{cases} K_n = \sum_{k=0}^{n-1} K_k K_{n-1-k} & (n \geq 1) \\ K_0 = 1 \end{cases}$$

$$K_n = \frac{\binom{2n}{n}}{n+1} \qquad (n \geq 0)$$

και τέλος: $\lim_{n\to\infty} K_{n+1} / K_n = 4$.

ΚΕΦΑΛΑΙΟ 4

ΣΥΝΘΕΤΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ: SPEEDES QHEAP ΚΑΙ CALENDAR QUEUES

4.1 Η Δομή SPEEDES Qheap

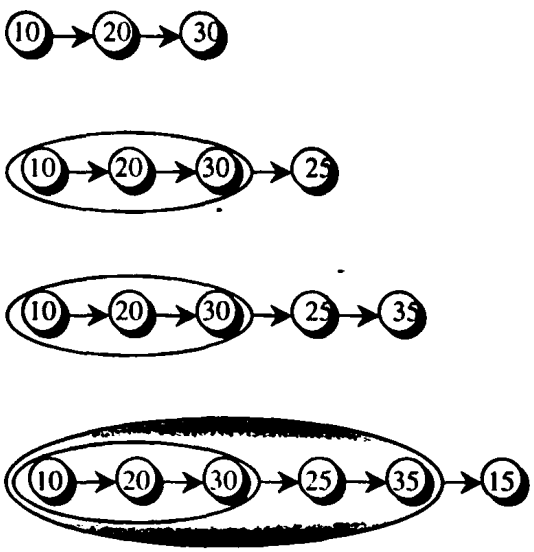
Στην παράγραφο αυτή παρουσιάζεται μια δομή δεδομένων για την προσομοίωση του συνόλου γεγονότων, της οποίας η συμπεριφορά μελετήθηκε θεωρητικά και πειραματικά. Αυτή είναι η δομή SPEEDES Qheap η οποία υλοποιεί τη δομή του σωρού χωρίς όμως να χρησιμοποιείται κάποιος από τους δύο τρόπους που περιγράφηκαν στο προηγούμενο κεφάλαιο (στατική ή δυναμική αναπαράσταση) και έχει εξαιρετικά καλή συμπεριφορά.

Για την υλοποίηση της δομής χρησιμοποιείται η δομή της διπλά συνδεδεμένης λίστας τόσο για την αποθήκευση των κόμβων που περιέχουν πληροφορία όσο και για την οργάνωση των μετα-κόμβων, οι οποίοι, όπως θα περιγραφεί αναλυτικά στη συνέχεια της παραγράφου, αποτελούν ομαδοποιημένα σύνολα κόμβων. Επιπλέον, στη δομή μπορεί να χρησιμοποιηθεί αποτελεσματικά η τεχνική Event Horizon η οποία έχει ήδη περιγραφεί στο Κεφάλαιο 2. Υπενθυμίζεται ότι περιλαμβάνει τη χρήση μιας βοηθητικής, δευτερεύουσας δομής προκειμένου να αποθηκεύονται προσωρινά τα γεγονότα, που στη συγκεκριμένη περίπτωση είναι διπλά συνδεδεμένη λίστα. Η μικρότερη πληροφορία που βρίσκεται αποθηκευμένη στη βοηθητική δομή είναι γνωστή κάθε χρονική στιγμή και δεν χρειάζεται να είναι ταξινομημένη. Έτσι, αν το νέο γεγονός που θα πρέπει να επεξεργαστεί, δηλαδή να αφαιρεθεί από την ουρά

προτεραιότητας, βρίσκεται στη βοηθητική δομή, αυτή ταξινομείται και συγχωνεύεται με τη βασική δομή που υλοποιεί την ουρά προτεραιότητας.

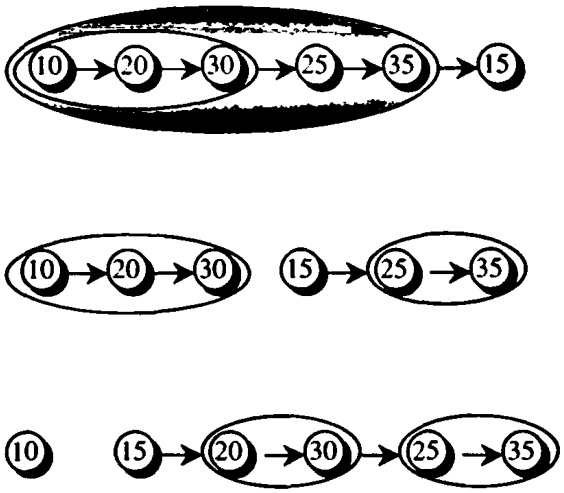
Όπως προαναφέρθηκε, οι κόμβοι της δομής είναι οργανωμένοι σε συνδεδεμένη λίστα Q η οποία διατηρείται ταξινομημένη και δεν επιτρέπεται να έχει μήκος μεγαλύτερο από μια προκαθορισμένη τιμή S. Το μέγεθος S πρέπει να επιλεγεί έτσι ώστε να εντοπίζει το σημείο στο οποίο η απευθείας εισαγωγή ενός στοιχείου σε μια ήδη ταξινομημένη συνδεδεμένη λίστα είναι πιο αποδοτική από άλλες λογαριθμικές τεχνικές. Αν ισχύει $S = 2$ η δομή είναι ουσιαστικά ένας δυαδικός σωρός. Τυπικές τιμές για το S είναι από 20 μέχρι 80 [36].

Τα νέα στοιχεία εισάγονται στη λίστα Q με το γνωστό τρόπο που γίνεται η εισαγωγή σε μια ταξινομημένη λίστα. Αν όμως το πλήθος των στοιχείων που υπάρχουν στη δομή είναι ίσο με το S, πριν την εισαγωγή του νέου στοιχείου, η λίστα αναδιοργανώνεται και σχηματίζεται ένας μετα-κόμβος (metaitem) η τιμή του οποίου θεωρείται ότι είναι αυτή του πρώτου του στοιχείου, δηλαδή του πρώτου στοιχείου στη λίστα Q. Η παραπάνω διαδικασία έχει σαν αποτέλεσμα να περιέχει η λίστα Q τόσο κόμβους με πραγματική πληροφορία όσο και μετα-κόμβους. Επιπλέον, είναι δυνατόν ένας μετα-κόμβος να αποτελείται από στοιχεία που είναι επίσης μετα-κόμβοι. Έτσι, η Q μπορεί να θεωρηθεί σαν αναδρομική λίστα που σχετίζεται με την ιδιότητα του σωρού. Στο Σχήμα 4.1 παρουσιάζεται η δομή που προκύπτει από την αρχική αν εισαχθούν διαδοχικά στοιχεία με τιμές 25, 35 και 15 και ισχύει $S = 3$.



Σχήμα 4.1

Η διαδικασία της διαγραφής του μικρότερου στοιχείου είναι περισσότερο πολύπλοκη από εκείνη της εισαγωγής αφού το πρώτο στοιχείο, το οποίο θα πρέπει να αφαιρεθεί, είναι πιθανόν να περιέχεται σε μετα-κόμβο. Αν ισχύει αυτό, τότε το πρώτο στοιχείο (μετα-κόμβος) διασπάται καθώς αφαιρείται ο πρώτος κόμβος του ενώ οι υπόλοιποι δημιουργούν ένα νέο μετα-κόμβο που εισάγεται ξανά στη λίστα Q με τον τρόπο που περιγράφηκε παραπάνω. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να βρεθεί ένας απλός κόμβος. Στο Σχήμα 4.2 παρουσιάζονται τα βήματα που πρέπει να πραγματοποιηθούν ώστε να αφαιρεθεί το μικρότερο στοιχείο από τη δομή, δηλαδή ο κόμβος με την τιμή 10.



Σχήμα 4.2

Η τεχνική Event Horizon εφαρμόζεται ως εξής: κάθε φορά που εισάγεται ένα στοιχείο τοποθετείται σε μια άλλη λίστα Q_{temp} η οποία δεν είναι ταξινομημένη. Όταν κατά τη διαδικασία της διαγραφής το στοιχείο που πρέπει να αφαιρεθεί βρίσκεται στη δευτερεύουσα λίστα Q_{temp}, αυτή ταξινομείται και έτσι διαγράφεται το πρώτο της στοιχείο. Στη συνέχεια, το υπόλοιπο της Q_{temp} δημιουργεί ένα μετα-κόμβο ο οποίος εισάγεται στη λίστα Q. Προφανώς, ένα πλεονέκτημα της χρήσης της δευτερεύουσας λίστας είναι ότι δημιουργεί μετα-κόμβους που περιέχουν μεγάλο πλήθος απλών κόμβων χωρίς κανένα μετα-κόμβο, γεγονός που καθιστά τη διαδικασία της διαγραφής από την Q αποδοτικότερη, αφού μειώνει τα βήματα που θα απαιτούνται για την αναδιοργάνωσή της. Η δομή SPEEDES Qheap χρησιμοποιείται στο λειτουργικό σύστημα SPEEDES.

Ακολουθεί η αλγοριθμική περιγραφή των διαδικασιών της εισαγωγής και της διαγραφής (με χρήση της τεχνικής Event Horizon) που περιγράφηκαν παραπάνω.

SPEEDES Qheap Insertion:

1. Το νέο στοιχείο τοποθετείται στο τέλος της λίστας Q_{temp}.
2. Ενημερώνεται η τιμή της μεταβλητής T_{min} αν το νέο στοιχείο έχει τη μικρότερη τιμή από εκείνες των στοιχείων που υπάρχουν στην Q_{temp}.

SPEEDES Qheap Removal:

1. Αν $T_{min} < min$, όπου min η μικρότερη τιμή στη λίστα Q εκτελούνται τα βήματα 1.1 ως 1.6, αλλιώς, εκτελείται το βήμα 2.
   1.1. Ταξινομείται η λίστα Q_{temp} και τίθεται $T_{min} = \infty$.
   1.2. Αφαιρείται το πρώτο στοιχείο της Q_{temp} και επιστρέφεται σαν *NextEvent*.
   1.3. Τα υπόλοιπα στοιχεία της Q_{temp} σχηματίζουν ένα νέο μετα-κόμβο που ονομάζεται $meta_{temp}$.
   1.4. Αν η λίστα Q περιέχει ήδη S στοιχεία, σχηματίζεται ένας νέος μετα-κόμβος από αυτά τα στοιχεία και έτσι η Q περιέχει ένα μόνο στοιχείο.

1.5. Το στοιχείο $meta_{temp}$ εισάγεται στη λίστα *Q*.
1.6. Return *NextEvent*.

2. Αφαιρείται το *NextItem*, το πρώτο στοιχείο από τη λίστα *Q*. Τα βήματα 2.1 ως 2.4 επαναλαμβάνονται μέχρι το *NextItem* να μην είναι μετα-κόμβος. Μόλις συμβεί αυτό, επιστρέφεται σαν *NextEvent*.
   2.1 Αφαιρείται το πρώτο στοιχείο από το *NextItem* και ονομάζεται *NewItem*. Αν το *NextItem* έχει μόνο ένα στοιχείο, παύει να θεωρείται μετα-κόμβος.
   2.2 Αν η λίστα *Q* περιέχει *S* στοιχεία, αυτά σχηματίζουν ένα μετα-κόμβο που εισάγεται στην *Q* σαν το μοναδικό της στοιχείο.
   2.3 Το *NextItem* εισάγεται στην *Q*.
   2.4 *NextItem = NewItem*.

4.2 Η Δομή Calendar Queues

Η δομή calendar queue ακολουθεί τη λογική ενός ημερολογίου. Ένα γεγονός προγραμματίζεται να πραγματοποιηθεί μια συγκεκριμένη ημέρα και γράφεται στη συγκεκριμένη σελίδα του ημερολογίου. Η διαδικασία αυτή αντιστοιχεί στην εισαγωγή ενός στοιχείου στη δομή, του οποίου η τιμή είναι η ημερομηνία πραγματοποίησης του γεγονότος. Η διαδικασία της διαγραφής του μικρότερου στοιχείου από τη δομή αντιστοιχεί στην πραγματοποίηση ενός γεγονότος, σύμφωνα με την ημερομηνία που αυτό έχει προγραμματιστεί στο ημερολόγιο.

Κάθε σελίδα του ημερολογίου υλοποιείται από μια ταξινομημένη συνδεδεμένη λίστα που περιλαμβάνει τα γεγονότα που πρέπει να πραγματοποιηθούν τη συγκεκριμένη ημέρα. Ο εντοπισμός της λίστας των γεγονότων μιας συγκεκριμένης ημέρας γίνεται με τη βοήθεια ενός μονοδιάστατου πίνακα *bucket* που περιέχει δείκτες για κάθε ημέρα (λίστα) του έτους. Για παράδειγμα, το στοιχείο *bucket*[12] είναι ένας δείκτης για τη λίστα των γεγονότων που έχουν προγραμματιστεί για τη $12^η$ ημέρα του έτους. Επομένως, ολόκληρο το ημερολόγιο αποτελείται από έναν πίνακα από 365 δείκτες, δηλαδή από 365 συνδεδεμένες λίστες.

Θεωρώντας το ημερολόγιο κυκλικό, είναι δυνατόν να προγραμματίζονται γεγονότα για ένα ολόκληρο έτος από την τρέχουσα ημέρα. Για παράδειγμα, αν η τρέχουσα ημερομηνία είναι 5 Δεκεμβρίου, είναι δυνατόν, γυρίζοντας στη λίστα που αντιστοιχεί στις 12 Νοεμβρίου, να προγραμματίσει κανείς γεγονότα για το επόμενο έτος. Μετά τη διαγραφή του τελευταίου γεγονότος που έχει προγραμματιστεί για 31 Δεκεμβρίου του τρέχοντος έτους, η διαδικασία συνεχίζεται από την $1^η$ Ιανουαρίου του επόμενου έτους. Έτσι, το ίδιο ημερολόγιο μπορεί να χρησιμοποιηθεί επ' αόριστον.

Τα γεγονότα μπορούν να προγραμματίζονται για περισσότερα από ένα μελλοντικά χρόνια αν κανείς σημειώνει την ημερομηνία που αντιστοιχεί σε κάθε λίστα. Έτσι, πριν τη διαγραφή ενός γεγονότος ελέγχεται η ημερομηνία που αντιστοιχεί στη συγκεκριμένη λίστα προκειμένου να εξακριβωθεί ότι το γεγονός είναι προγραμματισμένο για το τρέχον έτος. Αν ισχύει αυτό, πραγματοποιείται η διαδικασία της διαγραφής, αλλιώς το γεγονός αγνοείται.

Το μέγεθος ενός έτους, δηλαδή η διάσταση του πίνακα *bucket*, επιλέγεται να είναι αρκετά μεγάλο έτσι ώστε τα περισσότερα γεγονότα, τουλάχιστον το 75%, να πραγματοποιηθούν μέσα σε ένα έτος το πολύ από τη στιγμή που εισάγονται. Επιπλέον, αυτό έχει σαν αποτέλεσμα η λίστα των γεγονότων να μην είναι πολύ μεγάλη. Τόσο το πλήθος των ημερών του έτους όσο και το μέγεθος της λίστας που αντιστοιχεί σε κάθε μέρα αλλάζουν και προσαρμόζονται ανάλογα με το πλήθος των γεγονότων, των στοιχείων δηλαδή που υπάρχουν στη δομή. Στο Σχήμα 4.3 παρουσιάζεται ένα παράδειγμα μιας δομής calendar queue με έτος που αποτελείται από οκτώ μέρες. Το μέγεθος του χρόνου είναι 4 χρονικές μονάδες ενώ το μήκος της ημέρας είναι 0.5 χρονικές μονάδες. Το γεγονός που πρέπει να αφαιρεθεί έχει τιμή 14.5 και η τρέχουσα ημερομηνία είναι στη θέση *bucket*[5]. Σημειώνεται ότι το στοιχείο με τιμή 19.1 του *bucket*[6] θα αγνοηθεί την πρώτη φορά και θα αφαιρεθεί σε επόμενο έτος. Το έτος του συγκεκριμένου παραδείγματος αρχίζει με την τιμή 12.0 και τελειώνει με την τιμή 16.0. Αν το έτος 0 ξεκινά τη στιγμή 0.0, το έτος 1 τη στιγμή 4.0 κτλ, τότε το τρέχον έτος είναι το έτος με αριθμό 4.

	Bucket 0:	16.2			/*12.0 – 12.5*/
	Bucket 1:	16.6			/*12.5 – 13.0*/
	Bucket 2:				/*13.0 – 13.5*/
	Bucket 3:	17.8			/*13.5 – 14.0*/
	Bucket 4:				/*14.0 – 14.5*/
→	*Bucket* 5:	14.5	14.7	14.8	/*14.5 – 15.0*/
	Bucket 6:	15.2	15.3	19.1	/*15.0 – 15.5*/
	Bucket 7:	15.9			/*15.5 – 16.0*/

Σχήμα 4.3

Αν το πλήθος των στοιχείων της δομής είναι κατά πολύ μικρότερο ή μεγαλύτερο από το πλήθος των buckets, η δομή δεν θα είναι αποτελεσματική, αφού αν κάθε λίστα περιέχει μεγάλο πλήθος στοιχείων, ο χρόνος της εισαγωγής ενός νέου στοιχείου θα είναι απαγορευτικά μεγάλος. Αντίθετα, αν μόνο ένα bucket στα 100 περιέχει κάποιο στοιχείο, η διαδικασία της διαγραφής δεν είναι αποδοτική.

Έστω μια δομή που σταδιακά μεγαλώνει και τελικά περιλαμβάνει 10000 στοιχεία. Τότε το πλήθος των buckets θα πρέπει να είναι κοντά στο 10000. Όσο όμως το πλήθος των στοιχείων είναι μικρό, περίπου κοντά στο 10 μόνο ένα στα 1000 buckets θα περιέχει κάποιο στοιχείο. Έτσι, μεγάλος πλήθος από buckets καθιστά τη δομή μη αποδοτική. Η λύση επομένως είναι να μεταβάλλεται το πλήθος των buckets, όπως προαναφέρθηκε. Ένας τρόπος είναι αρχικά το πλήθος των buckets να είναι μικρός και να αντιγράφονται τα στοιχεία σε μια άλλη δομή κάθε φορά που το πλήθος τους γίνεται μεγαλύτερο από το διπλάσιο του πλήθους των buckets. Όμοια, αν το πλήθος των στοιχείων ελαττωθεί τόσο ώστε να γίνει μικρότερο από το μισό του πλήθους των buckets, τα στοιχεία αντιγράφονται σε ένα μικρότερο ημερολόγιο. Αν το πλήθος των buckets διπλασιάζεται όταν το πλήθος των στοιχείων γίνει ίσο με το διπλάσιο του πλήθους των buckets, η νέα δομή θα έχει τόσα στοιχεία όσα είναι και τα buckets. Αντίστοιχα, μειώνοντας στο μισό το πλήθος των buckets όταν το πλήθος των στοιχείων γίνει υποδιπλάσιός του θα έχει σαν αποτέλεσμα η νέα δομή να έχει τόσα στοιχεία όσα και τα buckets.

Έστω μια δομή η οποία ξεκινώντας από 0 έχει τελικά 1024 στοιχεία και έστω ότι αρχικά είχε 2 buckets. Τα στοιχεία αντιγράφηκαν σε νέα δομή όταν το πλήθος τους έγινε ίσο με 5, 9, 17, 33, 65, 129 κτλ. Δηλαδή, η αντιγραφή πραγματοποιήθηκε κάθε φορά που το πλήθος των στοιχείων ήταν κατά ένα μεγαλύτερο από μια δύναμη του 2. Έτσι, τα μισά στοιχεία δεν αντιγράφηκαν καθόλου, αφού εισήχθησαν μετά τον τελευταίο διπλασιασμό του πλήθους των buckets. Τα μισά από τα υπόλοιπα 512 στοιχεία εισήχθησαν μετά τον προτελευταίο και πριν το τελευταίο διπλασιασμό, με αποτέλεσμα να έχουν αντιγραφεί μια φορά. Έτσι, προκύπτει τελικά ότι 1/4 N στοιχεία αντιγράφηκαν μια φορά, 1/8 N στοιχεία αντιγράφηκαν δύο φορές, 1/16 N στοιχεία αντιγράφηκαν τρεις φορές κτλ, όπου N = 1024. Επομένως, ένα στοιχείο αντιγράφεται κατά μέσο όρο (1/4) + 2(1/8) + 3(1/16) + 4(1/32) + 5(1/64) + ... φορές. Ο τελευταίος όρος του αθροίσματος είναι αυτός που αντιστοιχεί στο αρχικό πλήθος των στοιχείων που σε αυτή την περίπτωση πρόκειται για δύναμη του δύο. Η χειρίστη περίπτωση είναι όταν το πλήθος των στοιχείων είναι κατά ένα μεγαλύτερο από μια δύναμη του δύο. Όταν γίνει ίσο με 1025, όλα τα στοιχεία αντιγράφονται σε μια άλλη δομή ενώ το μέσο στοιχείο αντιγράφεται το πολύ δύο φορές.

Έστω ότι το μέγεθος της δομής είναι μεταβλητό. Αυτό σημαίνει ότι το πλήθος των στοιχείων αυξάνεται ή μειώνεται είτε τυχαία είτε περιοδικά. Η χειρίστη περίπτωση είναι να αυξάνεται απότομα σε μια τιμή λίγο μεγαλύτερη από μια δύναμη του δύο και στη συνέχεια να πέφτει λίγο κάτω από την αμέσως μικρότερη δύναμη του δύο και η διαδικασία αυτή να επαναλαμβάνεται συνέχεια.

Το μέγεθος της ημέρας, δηλαδή το μήκος της λίστας, πρέπει να προσαρμόζεται κάθε φορά που η δομή αντιγράφεται σε μια άλλη. Αν είναι πολύ μεγαλύτερο από τη μέση διαφορά μεταξύ δύο γειτονικών στοιχείων, τα στοιχεία θα μοιραστούν σε ένα μικρό πλήθος λιστών (buckets) που είναι κοντά στην τρέχουσα ημέρα ενώ οι υπόλοιπες λίστες θα είναι άδειες. Αν, αντίθετα, είναι πολύ μικρό, τα περισσότερα στοιχεία θα αντιστοιχούν σε μελλοντικά έτη. Ο χρόνος που απαιτείται για τις διαδικασίες της εισαγωγής και της διαγραφής είναι ο μικρότερος όταν το μήκος της λίστας είναι κοντά στη μέση διαφορά γειτονικών στοιχείων. Αφού αυτή η τιμή γενικά μειώνεται καθώς νέα στοιχεία εισάγονται στη δομή, το μέγεθος των buckets πρέπει να προσαρμόζεται κάθε φορά που η δομή αντιγράφεται σε μια άλλη. Τα buckets που περιέχουν τα στοιχεία που πρόκειται να διαγραφούν συνήθως περιέχουν το μεγαλύτερο πλήθος στοιχείων αφού δεν έχει γίνει πρόσφατα (μέσα στο τρέχον έτος) διαγραφή από αυτά. Ένας εύκολος τρόπος να υπολογίσει κανείς το νέο μέγεθος των buckets είναι να υπολογίσει τη μέση διαφορά μερικών στοιχείων.

Ένα τελευταίο πρόβλημα που πρέπει να αντιμετωπιστεί είναι η περίπτωση κατά την οποία τα στοιχεία έχουν τέτοιες τιμές ώστε να ομαδοποιούνται και να μοιράζονται γύρω από δύο ή περισσότερα σημεία. Για παράδειγμα, αν τα μισά στοιχεία ανήκουν στο διάστημα [0, 0.1] και τα υπόλοιπα στο διάστημα [5.6, 5.7] και η δομή περιέχει N = 1000 στοιχεία, η μέση διαφορά μεταξύ των στοιχείων σε οποιοδήποτε από τα δύο διαστήματα θα είναι 0.0002. Αν το μέγεθος του bucket είναι 0.0002 τότε το μήκος του έτους θα είναι 0.1024 (υποθέτοντας ότι υπάρχουν 512 buckets). Η απόσταση μεταξύ των δύο διαστημάτων είναι 54 έτη με 512 ημέρες για κάθε έτος. Τα πρώτα 500 στοιχεία θα διαγραφούν γρήγορα ενώ θα πρέπει να περάσουν 54 έτη προκειμένου η αναζήτηση να οδηγήσει στα υπόλοιπα στοιχεία τα οποία ανήκουν στο διάστημα [5.6, 5.7] που στη συνέχεια θα μπορούν να διαγραφούν αμέσως.

Μια απλή λύση στο παραπάνω πρόβλημα θα ήταν να αναζητείται κατευθείαν το μικρότερο στοιχείο κάθε φορά που περνά ένα έτος χωρίς να εντοπιστεί το στοιχείο που πρέπει να

διαγραφεί (direct search). Αφού τα στοιχεία σε κάθε bucket είναι ταξινομημένα, αρκεί να εξετάσει κανείς το πρώτο μόνο στοιχείο σε κάθε λίστα. Ένα πλεονέκτημα αυτής της μεθόδου είναι ότι αποτρέπει την επιλογή ενός μη αποδοτικού αρχικού εύρους για τα buckets. Αν η δομή αρχικοποιείται με δύο buckets, η μέση διαφορά μεταξύ των στοιχείων δεν θα υπολογιστεί μέχρι το πλήθος των στοιχείων να γίνει ίσο με 5, οπότε μέχρι τότε το εύρος του bucket θα έχει την αρχική τιμή του. Αν αυτή είναι πολύ μεγάλη τότε όλα τα στοιχεία θα ανήκουν σε ένα bucket. Αυτό, στη συγκεκριμένη περίπτωση, δεν είναι πρόβλημα αφού θα υπάρχουν μόνο 4 στοιχεία στη δομή. Αν όμως η αρχική αυτή τιμή είναι πολύ μικρή, τα στοιχεία μπορεί να απέχουν μεταξύ τους εκατοντάδες ή χιλιάδες έτη, κάτι που θα αποτελεί σοβαρό πρόβλημα αν δεν εφαρμοστεί η τεχνική που περιγράφηκε παραπάνω, η οποία για 2 buckets και 4 στοιχεία είναι αποδοτική. Επομένως, η τιμή 1.0 αποτελεί καλή επιλογή για το εύρος των buckets.

Ακολουθεί η αλγοριθμική περιγραφή των διαδικασιών της εισαγωγής και της διαγραφής:

Insert(*item*):

1. Υπολογίζεται ο αριθμός του bucket στο οποίο θα γίνει η εισαγωγή:
   bucket = (*item* / *width*) mod *number_of_buckets*.
2. Εισάγεται το στοιχείο *item* στη λίστα του bucket με νούμερο *bucket*.
3. *number_of_nodes* = *number_of_nodes* + 1.
4. Αν *number_of_nodes* > *top_threshold* η δομή αλλάζει και το πλήθος των buckets διπλασιάζεται.

Η μεταβλητή *top_threshold* έχει συνήθως τιμή που ισούται με το διπλάσιο του τρέχοντος πλήθους των buckets, ενώ η μεταβλητή *width* αντιστοιχεί στο εύρος των buckets.

Delete_minimum:

1. Για κάθε bucket *i* πραγματοποιούνται τα εξής:
   Αν η λίστα του *bucket[i]* δεν είναι κενή και *bucket[i].key* < *buckettop*, πραγματοποιούνται τα εξής:
   1.1 Αφαιρείται το πρώτο στοιχείο της λίστας.
   1.2 *lastbucket* = *i*.
   1.3 *lastitem* = *item*.
   1.4 *number_of_nodes* = *number_of_nodes* – 1.
   1.5 Ελέγχεται αν πρέπει να υποδιπλασιαστεί η δομή:
   Αν *number_of_nodes* < *bot_threshold* υποδιπλασιάζεται το πλήθος των buckets.
   1.6 return(*item*).
   Αλλιώς, πραγματοποιούνται τα εξής:
   1.1 *i* = *i* +1.
   1.2 Αν *i* = *number_of_buckets*, τότε *i* = 0.
   1.3 *buckettop* = *buckettop* + *width*.
   1.4 Αν *i* = *lastbucket* πραγματοποιείται η διαδικασία *direct_search*.

Στη συνέχεια περιγράφεται αναλυτικά η μέθοδος direct search που περιγράφηκε παραπάνω:

direct_search:

1. Εντοπίζεται το μικρότερο στοιχείο της δομής, έστω *minimum*, ελέγχοντας τα πρώτα στοιχεία κάθε λίστας.
2. *lastbucket* = ο αριθμός του bucket που περιέχει το στοιχείο *minimum*.
3. *lastitem* = *minimum*.
4. Ενημερώνεται η μεταβλητή *buckettop* με βάση το στοιχείο *minimum*.
5. return(*Delete_minimum*).

Οι τρεις μεταβλητές, *lastbucket*, *lastitem* και *buckettop* χρησιμοποιούνται στον εντοπισμό της θέσης που βρισκόταν το τελευταίο στοιχείο που αφαιρέθηκε. Συγκεκριμένα, η μεταβλητή *lastbucket* αντιστοιχεί στον αριθμό του bucket από το οποίο αφαιρέθηκε το στοιχείο ώστε να εντοπίζεται το σημείο από το οποίο πρέπει να αρχίσει η επόμενη αναζήτηση, η μεταβλητή *buckettop* αντιστοιχεί στην μεγαλύτερη πληροφορία που θα μπορούσε να αποθηκεύσει το συγκεκριμένο bucket ώστε να αναγνωρίζεται να το στοιχείο πρέπει να αφαιρεθεί σε κάποιο επόμενο έτος, ενώ *lastitem* είναι η τιμή του στοιχείου που αφαιρέθηκε. Οι μεταβλητές *lastbucket* και *lastitem* ενημερώνονται κάθε φορά πριν επιστραφεί το μικρότερο στοιχείο ενώ η *buckettop* κάθε φορά που η αναζήτηση συνεχίζεται σε κάποιο άλλο bucket.

4.3 Θεωρητική Μελέτη των Αλγορίθμων

➢ Η ΔΟΜΗ SPEEDES QHEAP

Για τη δομή SPEEDES Qheap ο συνολικός χρόνος που απαιτείται για την εισαγωγή και τη διαγραφή ενός γεγονότος είναι:

$$T = C_1 B S + C_2$$

όπου το C_1 αντιστοιχεί στο χρόνο που απαιτείται για τη διάσχιση της συνδεδεμένης λίστας της δομής που αποτελείται από S μετα-κόμβους, το B στο μέσο ποσοστό γεγονότων που διασχίζονται και εξαρτάται από την εκάστοτε κατανομή, και το C_2 αντιστοιχεί στο χρόνο που απαιτεί η διαδικασία της διαγραφής.

Η διαδικασία της συγχώνευσης της δευτερεύουσας δομής με την κύρια είναι ουσιαστικά ισοδύναμη με την εισαγωγή ενός στοιχείου σε μια ταξινομημένη λίστα μήκους το πολύ S, αφού πριν την συγχώνευση η δευτερεύουσα δομή αναδιοργανώνεται ώστε να αποτελέσει ένα μετα-κόμβο. Επομένως η διαδικασία αυτή απαιτεί χρόνο τάξης O(log$S$). Είναι γνωστό ακόμη ότι η διαδικασία της ταξινόμησης m γεγονότων απαιτεί χρόνο της τάξης O($m$log$m$). Έτσι, αφού ο αλγόριθμος περιλαμβάνει τις δύο παραπάνω διαδικασίες, οι οποίες πραγματοποιούνται για κάθε m γεγονότα, η σχέση που ακολουθεί αντιστοιχεί στο χρόνο που απαιτείται για την εισαγωγή ενός γεγονότος:

$$T = C_1 + C_2 \log_2(m) + C_3 \frac{S}{m} \qquad (1)$$

όπου εδώ το C_1 αντιστοιχεί στο σταθερό χρονικό κόστος για την εισαγωγή στη δευτερεύουσα δομή ενός γεγονότος, το C_2 αντιστοιχεί στο χρονικό κόστος για την ταξινόμηση m γεγονότων,

και τέλος, το C_3 αναπαριστά το κόστος από τη συγχώνευση m γεγονότων της δευτερεύουσας λίστας (δηλαδή ενός μετακόμβου) με S γεγονότα της κύριας δομής.

Για την διαδικασία της διαγραφής στη χειρίστη περίπτωση, δηλαδή στην περίπτωση κατά την οποία n-1 στοιχεία αποτελούν τον πρώτο μετα-κόμβο της δομής, απαιτείται χρόνος τάξης $O([\lfloor n\text{-}1/S \rfloor+(n \bmod S)] \log S)$. Στη χειρίστη περίπτωση δηλαδή, προκειμένου να διαγραφεί το μικρότερο στοιχείο, απαιτούνται $\lfloor n\text{-}1/S \rfloor+(n \bmod S)$ διασπάσεις του πρώτου μετα-κόμβου, όσα δηλαδή είναι και τα στοιχεία (μετα-κόμβοι) που τον αποτελούν. Σημειώνεται ότι όλοι οι μετακόμβοι (πριν την πραγματοποίηση της διαγραφής) έχουν το ίδιο πρώτο στοιχείο, το μικρότερο στοιχείο της δομής.

Παραγωγίζοντας την σχέση (1) ως προς m και μηδενίζοντας την παράγωγο, υπολογίζεται εύκολα ότι ο καλύτερος χρόνος για την διαδικασία της εισαγωγής επιτυγχάνεται όταν το m είναι ίσο με:

$$m_{\text{optimal}} = (C_3 / C_2)\, S \log_e(2)$$

➢ Η ΔΟΜΗ CALENDAR QUEUES

Οι δυο παράμετροι που διαδραματίζουν καθοριστικό ρόλο στην αποτελεσματικότητα της δομής calendar queues είναι το εύρος (width) δ και το πλήθος M των buckets. Η επιλογή των τιμών των δυο αυτών παραμέτρων εξαρτάται από το πλήθος N των στοιχείων της δομής καθώς και από την κατανομή σύμφωνα με την οποία παράγονται τα στοιχεία.

Όπως έχει ήδη αναφερθεί κατά την περιγραφή της δομής, αν η τιμή της παραμέτρου M είναι πολύ μικρή ή αν η τιμή της δ είναι είτε πολύ μικρή είτε πολύ μεγάλη, η δομή είναι αναποτελεσματική. Αν $t(e)$ είναι η τιμή που αντιστοιχεί στο γεγονός e, τότε θεωρείται ότι η ποσότητα $jump = t(e) - t(e_0)$ είναι μια τυχαία μεταβλητή που ακολουθεί μια κατανομή που έχει μέσο μ, όπου e_0 είναι το γεγονός με το μικρότερο χρόνο δρομολόγησης. Η επιλογή της τιμής δ εξαρτάται από τις τιμές μ και N. Συγκεκριμένα, καθώς αυξάνει το μ πρέπει και το δ να αυξάνει, ενώ όσο αυξάνει το N, το δ πρέπει να μειώνεται.

Υποθέτοντας ότι το πλήθος των buckets είναι πολύ μεγάλο ($M \to \infty$), η τιμή του δ εξαρτάται και από άλλες τρεις παραμέτρους της υλοποίησης, b, c και d, όπου b είναι ο χρόνος επεξεργασίας ενός άδειου bucket, c είναι ο χρόνος διάσχισης ενός κόμβου σε μια λίστα προκειμένου να βρεθεί το μικρότερο στοιχείο και d είναι ο σταθερός χρόνος επεξεργασίας ενός γεγονότος. Αν m άδεια buckets διασχίζονται πριν εντοπιστεί ένα bucket που περιλαμβάνει n γεγονότα ($n \geq 1$), τότε ο χρόνος επεξεργασίας ενός γεγονότος είναι $bm + cn + d$. Έστω $K_N(\delta)$ η αναμενόμενη τιμή του χρόνου επεξεργασίας ενός γεγονότος. Αν το πλήθος των buckets είναι πεπερασμένο, όπως συμβαίνει στην πραγματικότητα, έστω $K'_N(\delta)$ ο αναμενόμενος χρόνος επεξεργασίας ενός γεγονότος. Ισχύει τότε $K'_N(\delta) \geq K_N(\delta)$, αφού απαιτείται περισσότερος χρόνος διότι κάποια γεγονότα δεν επεξεργάζονται αν ο χρόνος δρομολόγησής τους δεν ανήκει στο δ. Αποδεικνύεται [15] ότι η τιμή $K_N(\delta)$ είναι ελάχιστη για τιμή δ_{opt} που ισούται με:

$$\delta_{opt} = \sqrt{\frac{2b}{c}}\frac{\mu}{N} + O(N^{-3/2})$$

Ακόμη αποδεικνύεται [15] ότι η απόδοση του αλγορίθμου είναι βέλτιστη όταν ισχύει $M = O(N)$.

ΚΕΦΑΛΑΙΟ 5

ΕΠΕΚΤΑΣΕΙΣ ΤΗΣ ΔΟΜΗΣ ΤΟΥ ΣΩΡΟΥ: *n*-HEAP ΚΑΙ *M*-HEAP

5.1 Εισαγωγή

Όπως έχει περιγραφεί στο Κεφάλαιο 3, για την αποθήκευση της δομής του σωρού είναι δυνατόν να χρησιμοποιηθεί ένας μονοδιάστατος πίνακας, έστω H, το μέγεθος N του οποίου ορίζεται να είναι τέτοιο ώστε το πλήθος των κόμβων που αποτελούν τη δομή να μην το υπερβαίνει. Έτσι, αν η μικρότερη πληροφορία που υπάρχει στη δομή βρίσκεται αποθηκευμένη στην θέση $H[1]$, τότε για κάθε κόμβο που βρίσκεται στη θέση i του πίνακα θα ισχύει ότι το δεξί παιδί του βρίσκεται αποθηκευμένο στη θέση $H[2i]$ και το αριστερό παιδί του στη θέση $H[2i+1]$. Ακόμη θα ισχύει $H[i] \geq H[i \text{ div } 2]$ για κάθε i τέτοιο ώστε $2 \leq i \leq N$. Η διαδικασία της διαγραφής του μικρότερου στοιχείου από τη δομή πραγματοποιείται διαγράφοντας αρχικά την τιμή που είναι αποθηκευμένη στην πρώτη θέση ($H[1]$) και στη συνέχεια πραγματοποιείται η αναδιοργάνωση της δομής του δέντρου ώστε να επανακτηθούν οι ιδιότητες του σωρού. Με όμοιο τρόπο αναδιοργανώνεται η δομή του δέντρου μετά την εισαγωγή στην τελευταία θέση του πίνακα ενός νέου στοιχείου. Στο ίδιο κεφάλαιο περιγράφηκε επίσης και η δυναμική αναπαράσταση της δομής του σωρού, η οποία περιλαμβάνει τη χρήση δεικτών για τον εντοπισμό των στοιχείων του δέντρου. Για τις διαδικασίες της εισαγωγής ενός νέου κόμβου και της διαγραφής του κόμβου με τη μικρότερη πληροφορία ισχύουν όσα περιγράφηκαν στην παρουσίαση της στατικής υλοποίησης της δομής του σωρού. Πραγματοποιούνται όμως με τέτοιο τρόπο ώστε το δέντρο να παραμένει σχεδόν πλήρες (left complete).

Στη συνέχεια αυτού του κεφαλαίου παρουσιάζονται δύο επεκτάσεις της δομής του σωρού με σκοπό τη βελτίωση της αποτελεσματικότητάς του. Η πρώτη, η δομή *n*-heap, περιλαμβάνει ένα σχεδόν πλήρες δυαδικό δέντρο, όπως και η δομή του απλού σωρού, με τη διαφορά όμως ότι οι κόμβοι του δεν περιλαμβάνουν ένα μόνο στοιχείο αλλά *n*, τα οποία διατηρούνται και ταξινομημένα. Η δομή *M*-heap, η δεύτερη επέκταση της δομής του σωρού που θα παρουσιαστεί σε επόμενη παράγραφο, περιλαμβάνει *M* σχεδόν πλήρη δυαδικά δέντρα τα οποία ικανοποιούν τις ιδιότητες του σωρού.

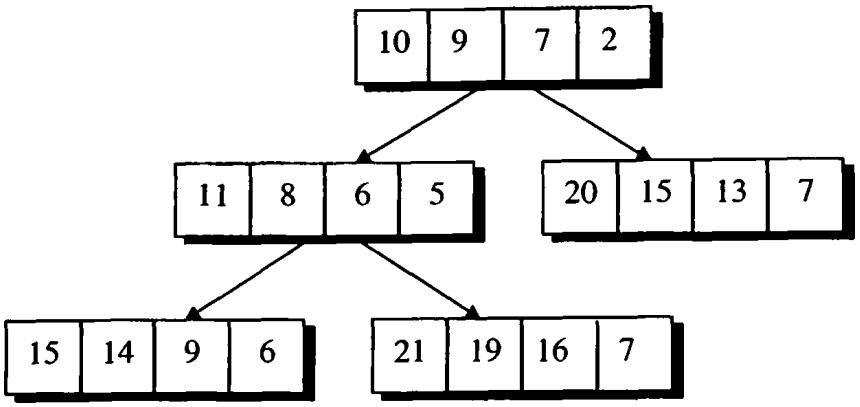
5.2 Η Δομή *n*-heap

Όπως αναφέρθηκε παραπάνω, η δομή *n*-heap αποτελεί επέκταση της δομής του σωρού. Πρόκειται για ένα σχεδόν πλήρες δυαδικό δέντρο του οποίου οι κόμβοι περιέχουν αρχικά *n*, αντί ενός, στοιχεία. Κάθε κόμβος περιλαμβάνει, εκτός από τους τρεις δείκτες που αντιστοιχούν στον πατέρα, το δεξί και το αριστερό παιδί του κόμβου, έναν μονοδιάστατο πίνακα *A* για την αποθήκευση των στοιχείων και μία μεταβλητή *position* που αντιστοιχεί στο πλήθος των στοιχείων που υπάρχουν στο συγκεκριμένο κόμβο. Η μορφή του κόμβου επομένως είναι:

```
struct node {
        float A[n];
    int position;
        struct node *right_child;
        struct node *left_child;
        struct node *parent;
      };
```

Τα στοιχεία σε κάθε κόμβο είναι ταξινομημένα κατά φθίνουσα τάξη. Επιπλέον, οι κόμβοι του δέντρου είναι τοποθετημένοι κατά τέτοιο τρόπο ώστε για τα μικρότερα στοιχεία που περιέχουν οι κόμβοι να ισχύουν οι ιδιότητες του σωρού. Έτσι, για τη διαγραφή του μικρότερου στοιχείου που υπάρχει στη δομή πραγματοποιείται ως εξής: το στοιχείο που πρέπει να αφαιρεθεί βρίσκεται στην τελευταία θέση του πρώτου κόμβου, δηλαδή στη ρίζα του δέντρου. Αφού αφαιρεθεί το στοιχείο και ενημερωθεί η μεταβλητή *position* του κόμβου, η δομή αναδιοργανώνεται ώστε η ρίζα να περιέχει εκείνο το σύνολο των στοιχείων που περιέχει το μικρότερο στοιχείο από όλα όσα υπάρχουν στη δομή. Πρόκειται για τη διαδικασία που πραγματοποιείται και στην απλή δομή του σωρού. Ένα παράδειγμα της δομής *n*-heap, όπου $n = 4$, παρουσιάζεται στο Σχήμα 5.1.

Η διαδικασία της εισαγωγής αρχικοποιείται με την τοποθέτηση του νέου στοιχείου στον κόμβο-ρίζα του δέντρου και την ενημέρωση της μεταβλητής *position* του κόμβου αυτού. Η διαδικασία ολοκληρώνεται με την εισαγωγή του στοιχείου στην κατάλληλη θέση του πίνακα ώστε ο τελευταίος να περιέχει τα στοιχεία κατά φθίνουσα τάξη. Αναδιοργάνωση της δομής δεν χρειάζεται αφού αν το νέο στοιχείο είναι μικρότερο από το μικρότερο στοιχείο που υπήρχε στη δομή τότε θα εξακολουθεί να ισχύει η ιδιότητα του σωρού επειδή αυτό τοποθετείται στον κόμβο-ρίζα. Στην περίπτωση που το στοιχείο που εισάγεται δεν είναι μικρότερο από το τελευταίο στοιχείο της ρίζας, δηλαδή το μικρότερο, η μικρότερη τιμή που υπάρχει στη δομή δεν αλλάζει και βρίσκεται στον κόμβο-ρίζα του δέντρου.

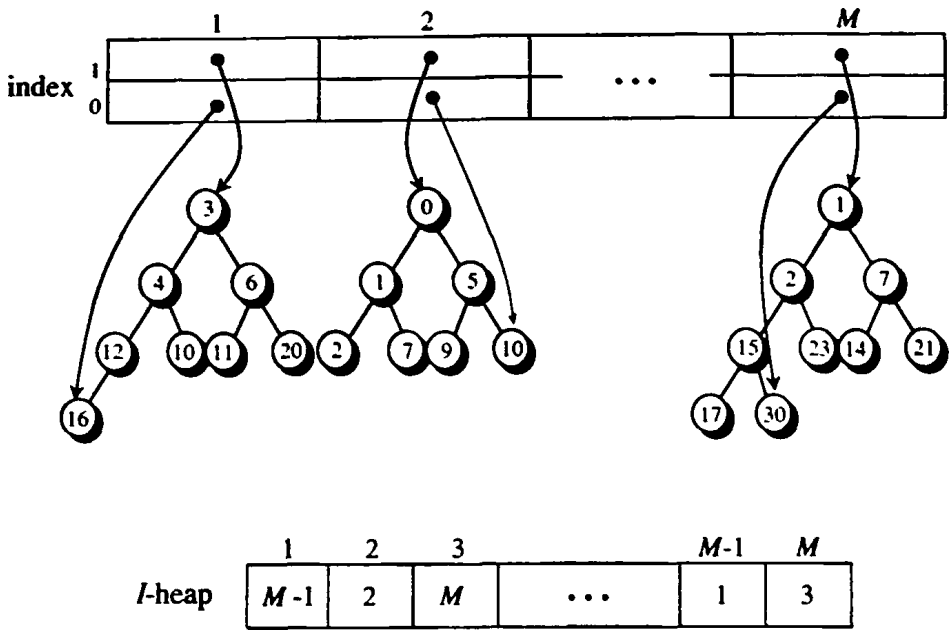


Σχήμα 5.1

Αξίζει να σημειωθεί ότι το μήκος του πίνακα σε κάθε έναν από τους κόμβους μπορεί να αλλάζει. Αυτό ισχύει γιατί μετά τη διαγραφή του μικρότερου στοιχείου, η δομή αναδιοργανώνεται με αποτέλεσμα η διαδικασία της εισαγωγής να μην πραγματοποιείται πάντα στον ίδιο κόμβο από τον οποίο είχε γίνει η διαγραφή. Επιπλέον, η εισαγωγή ενός νέου στοιχείου μπορεί να πραγματοποιηθεί σε κόμβο που περιέχει ήδη *n* στοιχεία. Όμως, αφού στο μοντέλο hold η διαδικασία της διαγραφής του μικρότερου στοιχείου ακολουθείται πάντα από τη διαδικασία της εισαγωγής, το πλήθος των στοιχείων που θα περιέχει τελικά κάθε ένας από τους κόμβους θα είναι κατά ένα λιγότερο ή κατά ένα περισσότερο από *n*.

5.3 Η Δομή *M*-heap

Η δομή *M*-heap (Multiple heap) αποτελεί, όπως έχει προαναφερθεί, επέκταση της δομής του σωρού. Αποτελείται από ένα σταθερό πλήθος *M* από σωρούς και έναν πίνακα *index* δύο διαστάσεων ο οποίος διατηρεί δείκτες προς τη ρίζα και προς το πιο αριστερό φύλλο κάθε ενός από τους σωρούς. Επιπλέον, περιλαμβάνει και μια δομή *I*-heap, μια στατική δηλαδή αναπαράσταση της δομής του σωρού. Προφανώς, τόσο ο πίνακας *index* όσο και ο πίνακας *I*-heap έχουν σταθερό μήκος *M*, όπου *M* είναι το πλήθος των δομών του σωρού που αποτελούν τη δομή *M*-heap, όπως φαίνεται και στο Σχήμα 5.2. Έτσι, θα μπορούσε κανείς να αριθμήσει τους σωρούς σύμφωνα με τη θέση (index) των δεικτών που αντιστοιχούν σε κάθε έναν από αυτούς στη δομή *index*. Η δομή *I*-heap περιέχει ακεραίους που αντιστοιχούν στη θέση-αριθμό κάθε ενός από τους σωρούς. Η θέση κάθε αριθμού, δηλαδή η θέση κάθε σωρού, εξαρτάται από το μικρότερο στοιχείο που υπάρχει στο σωρό στον οποίο αντιστοιχεί ο συγκεκριμένος αριθμός καθώς και την κατάταξη αυτού του στοιχείου σε σχέση με τα υπόλοιπα αντίστοιχα στοιχεία των υπολοίπων σωρών. Με άλλα λόγια, ικανοποιούνται οι ιδιότητες του σωρού για τα μικρότερα στοιχεία που είναι αποθηκευμένα σε κάθε δομή. Έτσι, το μικρότερο στοιχείο που υπάρχει στη δομή *M*-heap θα βρίσκεται σε εκείνο το σωρό που υποδεικνύεται από την πρώτη θέση του πίνακα *I*-heap, δηλαδή από τη ρίζα του σωρού *I*-heap.



	1	2	3		M-1	M
I-heap	M -1	2	M	• • •	1	3

Σχήμα 5.2

Σημειώνεται ότι η δομή *M*-heap δημιουργείται κατά τέτοιο τρόπο ώστε το πλήθος, έστω *n*, των στοιχείων κάθε σωρού να είναι σχεδόν ίδιος για όλους. Συγκεκριμένα, αν *N* είναι το πλήθος των στοιχείων, θα ισχύει $n \approx N / M$. Η σχέση αυτή, όπως θα καταστεί φανερό και στη συνέχεια, ισχύει και κατά τη διάρκεια και μετά το τέλος της εκτέλεσης οσωνδήποτε διαδικασιών hold, επομένως, με άλλα λόγια το πλήθος των στοιχείων σε κάθε σωρό είναι σχεδόν σταθερό.

Έστω ότι στο παράδειγμα του Σχήματος 5.2 ισχύει $M = 10$ και $N = 80$. Η διαδικασία της διαγραφής του μικρότερου στοιχείου αρχικοποιείται με τον εντοπισμό του σωρού που το περιέχει. Ο ζητούμενος σωρός είναι αυτός που υποδεικνύεται από την πρώτη θέση του πίνακα *I*-heap και στη συγκεκριμένη περίπτωση ο $9^{ος}$ σωρός. Η θέση του στοιχείου είναι προφανώς στη ρίζα του συγκεκριμένου σωρού. Η διαδικασία της διαγραφής συνεχίζεται κατά τα γνωστά στον $9^{ο}$ σωρό, και η μικρότερη τιμή που περιέχει αλλάζει. Έτσι, ενημερώνεται η δομή *I*-heap ώστε η πρώτη θέση (ρίζα) να περιέχει και πάλι το σωρό που αποθηκεύει τη μικρότερη τιμή της δομής *M*-heap. Αυτό πραγματοποιείται με όμοιο τρόπο με αυτόν που πραγματοποιείται η διαδικασία της εισαγωγής σε μία δομή σωρού. Η τιμή που θεωρείται ότι εισάγεται είναι η νέα μικρότερη τιμή που υπάρχει στον $9^{ο}$ σωρό, παρόλο που τελικά αποθηκεύεται ο αριθμός 9.

Κατά τη διαδικασία της εισαγωγής ενός νέου στοιχείου στη δομή *M*-heap πρέπει αρχικά να καθοριστεί ο σωρός στον οποίο αυτό θα τοποθετηθεί. Έτσι, η εισαγωγή γίνεται στον σωρό που υποδεικνύεται από την πρώτη θέση της δομής *I*-heap, δηλαδή στο προηγούμενο παράδειγμα στον $9^{ο}$ σωρό. Ο πίνακας *I*-heap δεν χρειάζεται ενημέρωση αφού ο $9^{ος}$ σωρός θα εξακολουθεί να περιέχει τη μικρότερη τιμή της δομής *M*-heap.

Τέλος, σημειώνεται ότι υπάρχει περίπτωση η διαδικασία της διαγραφής να έχει σαν αποτέλεσμα κάποιος από τους σωρούς να καταστεί κενός. Προκειμένου να αποφευχθεί μια τέτοια κατάσταση, αν το στοιχείο που διαγράφεται είναι το τελευταίο που υπάρχει σε κάποιο

σωρό, η εισαγωγή γίνεται στον ίδιο σωρό. Ακόμη, όπως αναφέρθηκε παραπάνω, η διαδικασία της εισαγωγής δεν επηρεάζει τη δομή *I*-heap. Έτσι, αν *t* είναι ο αριθμός (index) που αντιστοιχεί στο σωρό στον οποίο πραγματοποιήθηκε η τελευταία εισαγωγή, σύμφωνα με τα παραπάνω και με τη διαδικασία hold, το στοιχείο που θα διαγραφεί, αυτό δηλαδή με τη μικρότερη τιμή, θα βρίσκεται στον ίδιο σωρό, στον t^{th} σωρό. Για το λόγο αυτό, το πλήθος των στοιχείων των σωρών παραμένει σχεδόν σταθερό.

5.4 Θεωρητική Μελέτη των Αλγορίθμων

Για τον υπολογισμό της πολυπλοκότητας χρόνου που απαιτείται για την πραγματοποίηση των διαδικασιών της εισαγωγής και της διαγραφής για τις δομές *n*-heap και *M*-heap, αρκεί να λάβει κανείς υπόψη του τη χρονική πολυπλοκότητα των αντίστοιχων διαδικασιών της απλής δομής του σωρού. Στη συνέχεια παρουσιάζεται η θεωρητική μελέτη της αποτελεσματικότητας των δύο αλγορίθμων.

➢ Η ΔΟΜΗ *n*-HEAP

Αφού η δομή αποτελείται από ένα σχεδόν πλήρες δυαδικό δέντρο του οποίου οι κόμβοι περιέχουν *n* στοιχεία, προκύπτει ότι, αν *N* είναι το πλήθος των στοιχείων που υπάρχουν στη δομή, για το ύψος *h* του δέντρου θα ισχύει $h = O(\log(N/n))$. Έτσι, σύμφωνα με όσα αναφέρθηκαν παραπάνω, το στοιχείο με τη μικρότερη τιμή που υπάρχει στη δομή, εντοπίζεται και διαγράφεται σε σταθερό χρόνο. Απαιτείται όμως επιπλέον χρόνος για την αναδιοργάνωση της δομής μετά τον εντοπισμό και τη διαγραφή του συγκεκριμένου στοιχείου. Πρόκειται για τη διαδικασία που πραγματοποιείται και στην απλή δομή του σωρού μετά τη διαγραφή του στοιχείου από τον κόμβο-ρίζα. Επομένως ο χρόνος που απαιτείται για την διαδικασία της διαγραφής είναι τάξης $O(\log(N/n))$.

Με βάση τον τρόπο που πραγματοποιείται η διαδικασία της εισαγωγής προκύπτει ότι απαιτείται χρόνος μόνο για την εισαγωγή του νέου στοιχείου στην κατάλληλη θέση του πίνακα του κόμβου-ρίζα. Έτσι, η εισαγωγή ολοκληρώνεται με χρονική πολυπλοκότητα τάξης $O(\log n)$.

➢ Η ΔΟΜΗ *M*-HEAP

Έστω μια δομή *M*-heap που αποτελείται από *M* σωρούς. Επιπλέον, περιλαμβάνει και μια δομή *I*-heap, μια στατική δηλαδή αναπαράσταση της δομής του σωρού, η οποία, προφανώς, είναι ένας πίνακας μήκους *M*. Αν *n* είναι το πλήθος των στοιχείων κάθε σωρού και *N* είναι το πλήθος των στοιχείων που υπάρχουν στη δομή, θα ισχύει $n \approx N / M$. Αφού το μικρότερο στοιχείο που υπάρχει στη δομή *M*-heap θα βρίσκεται σε εκείνο το σωρό που υποδεικνύεται από την πρώτη θέση του πίνακα *I*-heap, δηλαδή από τη ρίζα του σωρού *I*-heap, ο σωρός από τον οποίο θα πραγματοποιηθεί η διαδικασία της διαγραφής εντοπίζεται σε σταθερό χρόνο. Η διαγραφή του μικρότερου στοιχείου ακολουθείται από τη διαδικασία της αναδιοργάνωσης του σωρού *I*-heap, η οποία προφανώς απαιτεί χρόνο $O(\log M)$. Έτσι, η διαδικασία της διαγραφής για τη δομή *M*-heap πραγματοποιείται συνολικά σε χρόνο τάξης $O(\log M + \log n)$.

Η εισαγωγή ενός νέου στοιχείου στη δομή *M*-heap γίνεται στον σωρό που υποδεικνύεται από την πρώτη θέση της δομής *I*-heap. Ο πίνακας *I*-heap δεν χρειάζεται ενημέρωση συνεπώς η διαδικασία απαιτεί πολυπλοκότητα χρόνου O(log*n*).

ΚΕΦΑΛΑΙΟ 6

ΕΠΕΚΤΑΣΕΙΣ ΤΗΣ ΔΟΜΗΣ ΤΟΥ *P*-TREE: *INDEXED P*-TREE ΚΑΙ *MULTIPLE P*-TREE

6.1 Εισαγωγή

Όπως έχει περιγραφεί στο Κεφάλαιο 5, ένα *P*-tree ορίζεται αναδρομικά. Έτσι, είτε είναι άδειο είτε είναι μια ταξινομημένη, μη αύξουσα ακολουθία από κόμβους που αποτελούν το αριστερό μονοπάτι, τέτοια ώστε σε κάθε έναν από αυτούς, εκτός από τον τελευταίο, να αντιστοιχεί ένα *P*-tree σαν δεξί υποδέντρο (πιθανόν και άδειο). Επιπλέον, οι κόμβοι του δεξιού υποδέντρου του κόμβου *x* έχουν τιμές μεταξύ της τιμής του κόμβου *x* και της τιμής του αριστερού παιδιού του. Τέλος, για να είναι ένα δέντρο *P*-tree θα πρέπει κάθε κόμβος που έχει δεξί να έχει και αριστερό παιδί.

Υπενθυμίζεται ότι η διαδικασία της διαγραφής του μικρότερου στοιχείου πραγματοποιείται αφαιρώντας τον τελευταίο κόμβο του πιο αριστερού μονοπατιού και τοποθετώντας σαν αριστερό παιδί του πατέρα του το δεξί υποδέντρο του πατέρα του. Για τη διαδικασία της εισαγωγής η αναζήτηση της κατάλληλης θέσης αρχικοποιείται από τη ρίζα και συνεχίζεται πολλές φορές αναδρομικά σε κάποιο δεξί υποδέντρο.

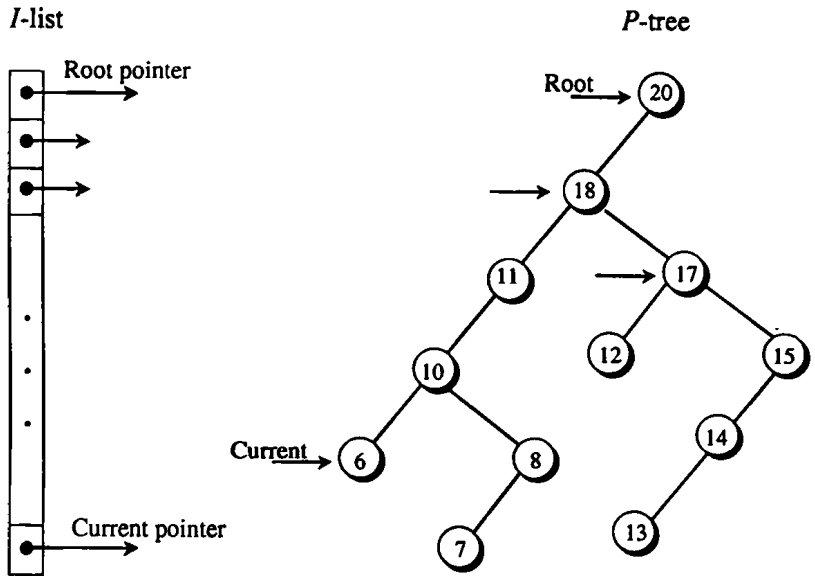
Στη συνέχεια αυτού του κεφαλαίου παρουσιάζονται δύο επεκτάσεις της δομής του *P*-tree με σκοπό τη βελτίωση της αποτελεσματικότητάς του. Η πρώτη, η δομή *Indexed P*-tree, συνδυάζει

τα πλεονεκτήματα της δομής του *P*-tree και της στατικής αναπαράστασης της δομής της λίστας, που ονομάζεται *I*-list. Η τελευταία έχει σταθερό μήκος και τα στοιχεία της είναι μεταβλητές τύπου δείκτη που αντιστοιχούν σε κόμβους του *P*-tree. Ο συνδυασμός των δύο δομών και η εφαρμογή της τεχνικής Event Horizon δημιουργεί μια αποτελεσματική δομή για την προσομοίωση του συνόλου γεγονότων. Το βασικό χαρακτηριστικό της δομής είναι η αποτελεσματικότητα της διαδικασίας της συγχώνευσης, δηλαδή της διαδικασίας της ταξινόμησης της δευτερεύουσας δομής και της εισαγωγής των στοιχείων της στην κύρια δομή, δηλαδή στη δομή του *P*-tree.

Η δεύτερη επέκταση της δομής του *P*-tree που θα παρουσιαστεί σε επόμενη παράγραφο, η δομή *Multiple P*-tree για την προσομοίωση του συνόλου γεγονότων, περιλαμβάνει *M P*-trees και τη στατική αναπαράσταση της δομής του σωρού που ονομάζεται *I*-heap. Η τελευταία δεν περιλαμβάνει κόμβους που αντιστοιχούν σε γεγονότα αλλά παρέχει τη δυνατότητα του εύκολου εντοπισμού του *P*-tree που περιέχει το μικρότερο στοιχείο. Το βασικό χαρακτηριστικό της δομής είναι ότι διαιρεί το σύνολο των στοιχείων, τα οποία μοιράζονται σε έναν ικανοποιητικά μεγάλο αριθμό από *P*-trees, με αποτέλεσμα να καθίσταται αποδοτικότερη η διαδικασία της εισαγωγής ενός νέου στοιχείου στη δομή.

6.2 Η Δομή *Indexed P*-tree (*IP*-tree)

Η δομή *indexed P*-tree, ή απλά *IP*-tree, αποτελείται από μια δομή *P*-tree και μια δομή που ονομάζεται *I*-list η οποία είναι στατική αναπαράσταση της δομής της λίστας. Τα στοιχεία της δομής *I*-list είναι δείκτες σε κόμβους της δομής του *P*-tree, όπως φαίνεται και στο Σχήμα 6.1.



Σχήμα 6.1

Χρησιμοποιείται η τεχνική Event Horizon και επομένως κάθε φορά που η μικρότερη τιμή που υπάρχει στη δομή *IP*-tree βρίσκεται στη δευτερεύουσα δομή, η τελευταία ταξινομείται και τα

στοιχεία της επανατοποθετούνται στο *P*-tree. Έτσι θα πρέπει να βρεθεί ένας αποδοτικός τρόπος προκειμένου να πραγματοποιείται αποδοτικά η διαδικασία της συγχώνευσης.

Έστω ότι πρέπει να εισαχθεί στη δομή του *P*-tree ένα σύνολο στοιχείων. Κατά τη διαδικασία της εισαγωγής ενός στοιχείου στη δομή αναζητείται η κατάλληλη θέση διασχίζοντας το δέντρο ξεκινώντας από τη ρίζα. Στη δομή *IP*-tree μπορεί κανείς να εκμεταλλευτεί το γεγονός ότι τα στοιχεία της δευτερεύουσας δομής που εισάγονται διαδοχικά είναι ταξινομημένα κατά φθίνουσα τάξη καθώς και ότι η δομή *I*-list καθορίζει τμήματα του δέντρου (υποδέντρα). Υπενθυμίζεται ότι κάθε υποδέντρο της δομής του *P*-tree αποτελεί επίσης ένα *P*-tree. Έτσι, για την εισαγωγή ενός στοιχείου δεν είναι απαραίτητο η αναζήτηση για την κατάλληλη θέση να ξεκινήσει από τη ρίζα, αλλά από ένα συγκεκριμένο υποδέντρο που εντοπίζεται με τη βοήθεια της δομής *I*-list. Αυτό οφείλεται στο γεγονός ότι είναι γνωστό ότι το αμέσως προηγούμενο στοιχείο που τοποθετήθηκε στο *P*-tree είχε μεγαλύτερη τιμή από το νέο και παρέχεται έτσι η δυνατότητα να τοποθετούνται δείκτες σε συγκεκριμένους κόμβους στο μονοπάτι που ακολούθησε το προηγούμενο στοιχείο προκειμένου να διευκολυνθεί η εισαγωγή του νέου. Εξάλλου τα μονοπάτια που ακολουθούνται στο δέντρο για την εισαγωγή των στοιχείων θα είναι μέχρι κάποιο σημείο κοινά, γεγονός που οφείλεται στη σχέση που έχουν τα δύο στοιχεία μεταξύ τους.

Αφού εντοπισθεί το κατάλληλο υποδέντρο, η διαδικασία της εισαγωγής ολοκληρώνεται στο συγκεκριμένο *P*-tree σύμφωνα με τον τρόπο που περιγράφηκε στην προηγούμενη παράγραφο. Για παράδειγμα, έστω *a* το στοιχείο του οποίου η εισαγωγή μόλις ολοκληρώθηκε και *r* το τελευταίο στοιχείο του πιο αριστερού μονοπατιού με το οποίο συγκρίθηκε το στοιχείο *a* προτού η αναζήτηση της κατάλληλης θέσης προχωρήσει σε δεξί υποδέντρο. Αν το νέο στοιχείο που πρέπει να εισαχθεί, έστω *b*, έχει μικρότερη τιμή, αρκεί η διαδικασία της εισαγωγής να ξεκινήσει από το υποδέντρο (*P*-tree) με ρίζα τον κόμβο *r*. Έτσι, είναι απαραίτητο να διατηρούνται δείκτες σε εκείνους τους κόμβους με τους οποίους συγκρίθηκε το στοιχείο *a* προτού η διαδικασία συνεχιστεί σε δεξί υποδέντρο. Οι κόμβοι αυτοί ονομάζονται *I*-κόμβοι και είναι τα στοιχεία της δομής *I*-list. Ο πρώτος *I*-κόμβος είναι η ρίζα του *P*-tree ενώ ο τελευταίος είναι ο κόμβος που περιέχει τη μικρότερη πληροφορία που υπάρχει στο *P*-tree, ο τελευταίος δηλαδή κόμβος του πιο αριστερού μονοπατιού του δέντρου.

Στο παράδειγμα του Σχήματος 6.1, έστω ότι έχει εισαχθεί ο κόμβος *a* με τιμή 14. Οι *I*-κόμβοι επομένως που καθορίζονται είναι οι κόμβοι με τιμές 18 και 17. Κατά την εισαγωγή ενός νέου κόμβου *b* με τιμή 13, θα γίνουν συγκρίσεις μόνο με τις τιμές των αριστερών παιδιών των *I*-κόμβων μέχρι να εντοπισθεί ο κόμβος *c* με τη μεγαλύτερη τιμή που είναι όμως μικρότερη από την τιμή του κόμβου *b*, δηλαδή το 13. Στη συνέχεια πραγματοποιείται η γνωστή διαδικασία της εισαγωγής στο υποδέντρο *P*-tree με ρίζα τον κόμβο *c*. Σημειώνεται τέλος ότι η ακολουθία των τιμών των *I*-κόμβων είναι επίσης ταξινομημένη και έτσι η διαδικασία γίνεται αποδοτικότερη αν πραγματοποιείται δυαδική αναζήτηση για τον εντοπισμό του κόμβου *c*.

6.3 Η Δομή *Multiple P*-tree (*MP*-tree)

Η δομή Multiple *P*-tree, ή *MP*-tree, αποτελείται από ένα σταθερό πλήθος *M* από *P*-trees και έναν πίνακα *index* δύο διαστάσεων ο οποίος διατηρεί δείκτες προς τη ρίζα και προς τον κόμβο με τη μικρότερη τιμή που υπάρχει σε κάθε δομή *P*-tree. Επιπλέον, περιλαμβάνει και μια δομή

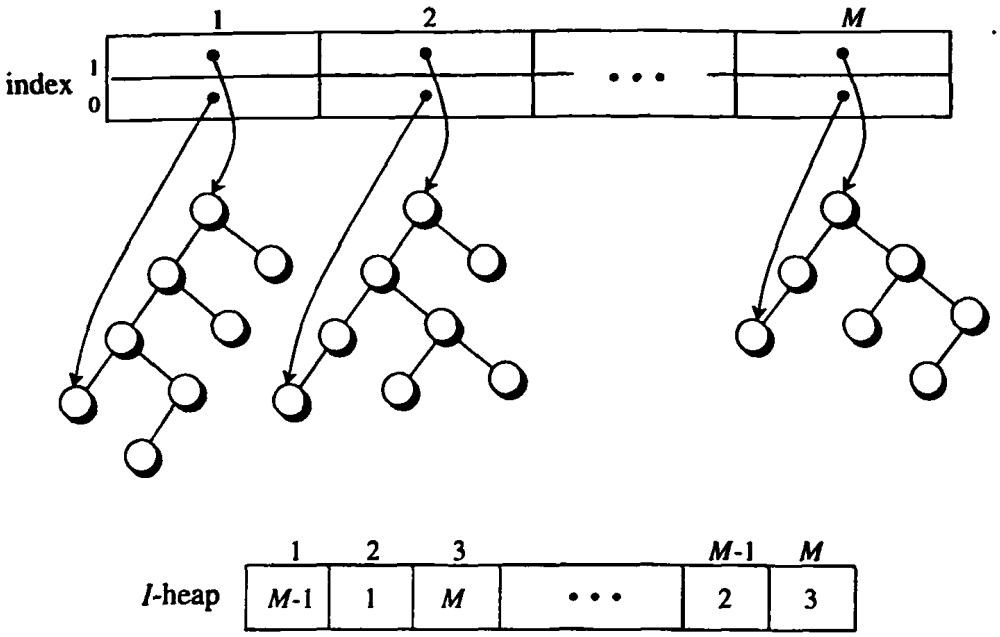
I-heap, μια στατική δηλαδή αναπαράσταση της δομής του σωρού. Όπως θα γίνει φανερό και στη συνέχεια, οι δομές *multiple P*-tree και *M*-heap παρουσιάζουν πολλές ομοιότητες; αφού ουσιαστικά η μόνη τους διαφορά είναι η χρήση της δομής *P*-tree στην πρώτη και της δομής του σωρού στη δεύτερη. Λεπτομέρειες για τις ομοιότητες και τις διαφορές τους καθώς και για την αποτελεσματικότητά τους θα παρουσιαστούν σε επόμενα κεφάλαια.

Προφανώς, τόσο ο πίνακας *index* όσο και ο πίνακας *I*-heap έχουν σταθερό μήκος *M*, όπου *M* είναι ο αριθμός των *P*-trees που αποτελούν τη δομή *MP*-tree, όπως φαίνεται και στο Σχήμα 6.2. Έτσι, θα μπορούσε κανείς να αριθμήσει τα *P*-trees σύμφωνα με τη θέση (index) των δεικτών που αντιστοιχούν σε κάθε ένα από αυτά στη δομή *index*. Η δομή *I*-heap περιέχει ακεραίους που αντιστοιχούν στη θέση-αριθμό κάθε ενός από τα δέντρα. Η θέση κάθε αριθμού, δηλαδή η θέση κάθε δέντρου, εξαρτάται από το μικρότερο στοιχείο που υπάρχει στο *P*-tree στο οποίο αντιστοιχεί ο συγκεκριμένος αριθμός καθώς και την κατάταξη αυτού του στοιχείου σε σχέση με τα υπόλοιπα αντίστοιχα στοιχεία των υπολοίπων *P*-trees. Με άλλα λόγια, ικανοποιούνται οι ιδιότητες του σωρού για τα μικρότερα στοιχεία που είναι αποθηκευμένα σε κάθε δομή. Έτσι, το μικρότερο στοιχείο που υπάρχει στη δομή *MP*-tree θα βρίσκεται σε εκείνο το *P*-tree που υποδεικνύεται από την πρώτη θέση του πίνακα *I*-heap, δηλαδή από τη ρίζα του σωρού *I*-heap.

Σημειώνεται ότι η δομή *MP*-tree δημιουργείται κατά τέτοιο τρόπο ώστε ο αριθμός, έστω *n*, των στοιχείων κάθε *P*-tree να είναι σχεδόν ίδιος για όλους. Συγκεκριμένα, αν *N* είναι το πλήθος των στοιχείων, θα ισχύει $n \approx N / M$. Η σχέση αυτή, όπως θα καταστεί φανερό και στη συνέχεια, ισχύει και κατά τη διάρκεια και μετά το τέλος της εκτέλεσης οσωνδήποτε διαδικασιών hold, επομένως, με άλλα λόγια το πλήθος των στοιχείων σε κάθε δέντρο είναι σχεδόν σταθερό.

Έστω ότι στο παράδειγμα του Σχήματος 6.2 ισχύει $M = 10$ και $N = 80$. Η διαδικασία της διαγραφής του μικρότερου στοιχείου αρχικοποιείται με τον εντοπισμό του *P*-tree που το περιέχει. Το ζητούμενο δέντρο είναι αυτό που υποδεικνύεται από την πρώτη θέση του πίνακα *I*-heap και στη συγκεκριμένη περίπτωση το 9° *P*-tree. Η θέση του στοιχείου υποδεικνύεται από έναν από τους δείκτες του πίνακα *index* στη θέση 9 και έτσι το στοιχείο εντοπίζεται σε σταθερό χρόνο. Μετά τη διαγραφή, η μικρότερη τιμή που περιέχει το συγκεκριμένο δέντρο αλλάζει. Έτσι, ενημερώνεται η δομή *I*-heap ώστε η πρώτη θέση (ρίζα) να περιέχει και πάλι το *P*-tree που αποθηκεύει τη μικρότερη τιμή της δομής *MP*-tree. Αυτό πραγματοποιείται με όμοιο τρόπο με αυτόν που πραγματοποιείται η διαδικασία της εισαγωγής σε μία δομή σωρού. Η τιμή που θεωρείται ότι εισάγεται είναι η νέα μικρότερη τιμή που υπάρχει στον 9° *P*-tree, παρόλο που τελικά αποθηκεύεται ο αριθμός 9.

Κατά τη διαδικασία της εισαγωγής ενός νέου στοιχείου στη δομή *MP*-tree πρέπει αρχικά να καθοριστεί το *P*-tree στο οποίο αυτό θα τοποθετηθεί. Έτσι, η εισαγωγή γίνεται στο δέντρο που υποδεικνύεται από την πρώτη θέση της δομής *I*-heap, δηλαδή στο προηγούμενο παράδειγμα στο 9° δέντρο. Ο πίνακας *I*-heap δεν χρειάζεται ενημέρωση αφού το 9° *P*-tree θα εξακολουθεί να περιέχει τη μικρότερη τιμή της δομής *MP*-tree.



Σχήμα 6.2

Τέλος, σημειώνεται ότι υπάρχει περίπτωση η διαδικασία της διαγραφής να έχει σαν αποτέλεσμα κάποιο από τα *P*-tree να καταστεί κενό. Προκειμένου να αποφευχθεί μια τέτοια κατάσταση, αν το στοιχείο που διαγράφεται είναι το τελευταίο που υπάρχει σε κάποιο *P*-tree, η εισαγωγή γίνεται στο ίδιο *P*-tree. Ακόμη, όπως αναφέρθηκε παραπάνω, η διαδικασία της εισαγωγής δεν επηρεάζει τη δομή *I*-heap. Έτσι, αν t είναι ο αριθμός (index) που αντιστοιχεί στο *P*-tree στο οποίο πραγματοποιήθηκε η τελευταία εισαγωγή, σύμφωνα με τα παραπάνω και με τη διαδικασία hold, το στοιχείο που θα διαγραφεί, αυτό δηλαδή με τη μικρότερη τιμή, θα βρίσκεται στο ίδιο *P*-tree, στο t^{th}. Για το λόγο αυτό, το πλήθος των στοιχείων των *P*-trees παραμένει σχεδόν σταθερό.

6.4 Θεωρητική Μελέτη των Αλγορίθμων

Για τον υπολογισμό της πολυπλοκότητας χρόνου που απαιτείται για την πραγματοποίηση των διαδικασιών της εισαγωγής και της διαγραφής για τις δομές *IP*-tree και *MP*-tree, αρκεί να λάβει κανείς υπόψη του τη χρονική πολυπλοκότητα των αντίστοιχων διαδικασιών της απλής δομής του *P*-tree. Στη συνέχεια παρουσιάζεται η θεωρητική μελέτη της αποτελεσματικότητας των δύο αλγορίθμων.

➢ Η ΔΟΜΗ *IP*-TREE

Η διαδικασία του εντοπισμού και της διαγραφής του μικρότερου στοιχείου από τη δομή πραγματοποιείται σε σταθερό χρόνο, όταν αυτό υπάρχει στην δομή του *P*-tree. Η διαδικασία της εισαγωγής ενός νέου στοιχείου στη δομή πραγματοποιείται σε σταθερό χρόνο αφού αυτό εισάγεται στην δευτερεύουσα δομή, στη δομή της λίστας. Είναι γνωστό ακόμη ότι η

διαδικασία της ταξινόμησης m γεγονότων απαιτεί χρόνο της τάξης $O(m\log m)$. Έτσι, αφού η συγχώνευση πραγματοποιείται για κάθε m γεγονότα, η σχέση που ακολουθεί αντιστοιχεί στο χρόνο που απαιτείται για την εισαγωγή ενός γεγονότος:

$$Ti = C_1 + C_2 \log_2(m) + C_3 n/m \qquad (1)$$

όπου εδώ το C_1 αντιστοιχεί στο σταθερό κόστος χρόνου για την εισαγωγή στη δευτερεύουσα δομή ενός γεγονότος, το C_2 αντιστοιχεί στο κόστος χρόνου για την ταξινόμηση m γεγονότων, και τέλος, το C_3 αναπαριστά το κόστος από τη συγχώνευση (εισαγωγή) m γεγονότων της δευτερεύουσας λίστας με n γεγονότα της κύριας δομής.

Παραγωγίζοντας την σχέση (1) ως προς m και μηδενίζοντας την παράγωγο, υπολογίζεται εύκολα ότι ο καλύτερος χρόνος για την διαδικασία της εισαγωγής επιτυγχάνεται όταν το m είναι ίσο με:

$$m_{optimal} = (C_3 / C_2)\, n \log_e(2)$$

➢ Η ΔΟΜΗ *MP*-TREE

Έστω μια δομή *MP*-tree που αποτελείται από M το πλήθος *P*-trees. Επιπλέον, περιλαμβάνει και μια δομή *I*-heap, μια στατική δηλαδή αναπαράσταση της δομής του σωρού, η οποία, προφανώς, είναι ένας πίνακας μήκους M. Αν n είναι το πλήθος των στοιχείων σε κάθε *P*-tree και N είναι το πλήθος των στοιχείων που υπάρχουν στη δομή, θα ισχύει $n \approx N / M$. Αφού το μικρότερο στοιχείο που υπάρχει στη δομή *MP*-tree θα βρίσκεται σε εκείνο το *P*-tree που υποδεικνύεται από την πρώτη θέση του πίνακα *I*-heap, δηλαδή από τη ρίζα του σωρού *I*-heap, το δέντρο από τον οποίο θα πραγματοποιηθεί η διαδικασία της διαγραφής εντοπίζεται σε σταθερό χρόνο. Η διαγραφή του μικρότερου στοιχείου ακολουθείται από τη διαδικασία της αναδιοργάνωσης του σωρού *I*-heap, η οποία προφανώς απαιτεί χρόνο $O(\log M)$. Έτσι, η διαδικασία της διαγραφής για τη δομή *MP*-tree πραγματοποιείται συνολικά σε χρόνο τάξης $O(\log M)$.

Η εισαγωγή ενός νέου στοιχείου στη δομή γίνεται στο δέντρο που υποδεικνύεται από την πρώτη θέση της δομής *I*-heap. Ο πίνακας *I*-heap δεν χρειάζεται ενημέρωση συνεπώς η διαδικασία απαιτεί στη χειρίστη περίπτωση πολυπλοκότητα χρόνου $O(n)$.

ΚΕΦΑΛΑΙΟ 7

ΠΑΡΑΛΛΗΛΗ ΠΡΟΣΟΜΟΙΩΣΗ ΔΙΑΚΡΙΤΩΝ ΓΕΓΟΝΟΤΩΝ

Ένας παράλληλος αλγόριθμος χρησιμοποιείται στην προσομοίωση διακριτών γεγονότων με σκοπό να καταστούν όσο το δυνατό πιο αποδοτικές οι διαδικασίες της εισαγωγής και της διαγραφής. Είναι δυνατόν να χρησιμοποιηθούν περισσότεροι από ένας επεξεργαστές με σκοπό να επιταχυνθούν οι διαδικασίες της εισαγωγής ενός νέου στοιχείου στο σύνολο γεγονότων και της διαγραφής του μικρότερου στοιχείου. Ακόμη είναι δυνατόν να επιτευχθεί, με τη χρήση πολλών επεξεργαστών, η ταυτόχρονη εισαγωγή ή και διαγραφή περισσότερων του ενός στοιχείων. Τότε πρόκειται για παράλληλη προσομοίωση διακριτών γεγονότων. Στη συνέχεια μελετώνται αναλυτικά και οι δύο περιπτώσεις χρήσης του παραλληλισμού, ενώ προηγείται μια εισαγωγή σε βασικές έννοιες της περιοχής των παράλληλων αλγορίθμων και της παράλληλης επεξεργασίας.

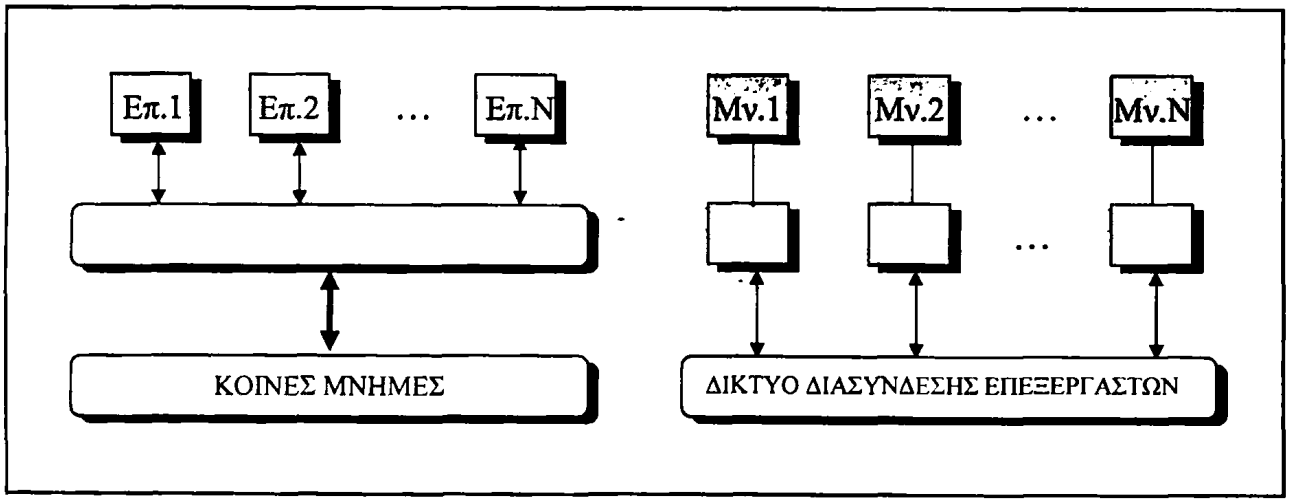
7.1 Εισαγωγή

Μερικά από τα προβλήματα που εμφανίζονται κατά τη χρήση παράλληλων επεξεργαστών είναι ο τρόπος επικοινωνίας, συντονισμού και συνεργασίας των επεξεργαστών για την εκτέλεση μιας εργασίας, αλλά και ο καθορισμός των προβλημάτων που επιδέχονται επίλυση σε

παράλληλο περιβάλλον. Έτσι, ο προγραμματισμός παράλληλων συστημάτων έχει αρκετές και σημαντικές διαφοροποιήσεις από τον κλασικό σειριακό προγραμματισμό και γενικά είναι αρκετά δυσκολότερος.

Όπως είναι γνωστό, η επικοινωνία των επεξεργαστών πραγματοποιείται με δύο τρόπους: (α) μέσω κοινής μνήμης, και (β) μέσω απευθείας συνδέσεων μεταξύ των επεξεργαστών. Ο τρόπος επικοινωνίας και συνεργασίας των επεξεργαστών καθορίζει σε ποια κατηγορία ανήκει το σύστημα. Γενικά, τα υπολογιστικά συστήματα ανήκουν σε μία από τις παρακάτω τρεις κατηγορίες:

- SISD (single-instruction single-data). Είναι οι κλασσικοί σειριακοί υπολογιστές, οι οποίοι εκτελούν μια εντολή τη φορά πάνω σε ένα δεδομένο.
- SIMD (single-instruction multiple-data). Είναι οι υπολογιστές οι οποίοι εκτελούν μια εντολή τη φορά αλλά μπορούν να την εφαρμόσουν ταυτόχρονα σε πολλαπλά δεδομένα.
- MIMD (multiple-instruction multiple-data). Πρόκειται για συστήματα που διαθέτουν N ανεξάρτητους επεξεργαστές που ο καθένας μπορεί να εκτελεί διαφορετικές εντολές σε διαφορετικά δεδομένα. Διακρίνονται σε συστήματα κοινής και κατανεμημένης μνήμης, όπως φαίνεται και στο Σχήμα 7.1.



Σχήμα 7.1: Υπολογιστές MIMD

Μοντελα Παραλληλου Υπολογισμου: Το Μοντελο Παραλληλου Υπολογισμου PRAM.

Ένα PRAM (Parallel Random Access Machine) είναι ένα υπολογιστικό πρότυπο με τις ακόλουθες ιδιότητες:

Αποτελείται από N επεξεργαστές P_1, P_2, ..., P_N οι οποίοι εργάζονται ταυτόχρονα εκτελώντας την ίδια πράξη σε διαφορετικό σύνολο δεδομένων (SIMD). Οι επεξεργαστές κατονομάζονται με διακριτούς ακεραίους i, με $1 \leq i \leq N$, οι οποίοι ονομάζονται επιγραφές. Κάθε επεξεργαστής

είναι εφοδιασμένος με την δική του τοπική μνήμη και με μια μονάδα επεξεργασίας που μπορεί να διαβάζει, να γράφει δεδομένα και να εκτελεί αριθμητικές και λογικές πράξεις. Όλοι οι επεξεργαστές έχουν πρόσβαση σε μια κοινή μνήμη, στην οποία μπορούν να γράφουν και από την οποία μπορούν να διαβάζουν και να μεταφέρουν δεδομένα στη δική τους μνήμη. Οποιαδήποτε θέση μνήμης μπορεί να αποθηκεύσει έναν οποιοδήποτε ακέραιο ή μια λογική τιμή (σταθερά). Η εκτέλεση των αριθμητικών και λογικών πράξεων καθώς και ο χρόνος κύκλου μνήμης για ανάγνωση και εγγραφή απαιτούν μια μονάδα χρόνου.

Υπενθυμίζεται ότι, με βάση τις ιδιότητες που χαρακτηρίζουν τις πράξεις ανάγνωσης και εγγραφής στην κοινή μνήμη, τα υπολογιστικά μοντέλα PRAM διακρίνονται σε τέσσερις τύπους:

- EREW-PRAM (Exclusive Read - Exclusive Write PRAM). Δύο ή περισσότεροι επεξεργαστές δεν μπορούν να διαβάσουν ταυτόχρονα το περιεχόμενο της ίδιας θέσης μνήμης και δεν μπορούν να γράψουν ταυτόχρονα στην ίδια θέση μνήμης.
- CREW-PRAM (Concurrent Read - Exclusive Write PRAM). Δύο ή περισσότεροι επεξεργαστές μπορούν να διαβάσουν ταυτόχρονα το περιεχόμενο οποιασδήποτε θέσης μνήμης αλλά δεν μπορούν να γράψουν ταυτόχρονα στην ίδια θέση μνήμης.
- CRCW-PRAM (Concurrent Read - Concurrent Write PRAM). Δύο ή περισσότεροι επεξεργαστές μπορούν να διαβάσουν ταυτόχρονα το περιεχόμενο οποιασδήποτε θέσης μνήμης και να γράψουν ταυτόχρονα σε οποιαδήποτε, ακόμη και την ίδια, θέση μνήμης.
- ERCW-PRAM (Exclusive Read - Concurrent Write PRAM). Δύο ή περισσότεροι επεξεργαστές δεν μπορούν να διαβάσουν ταυτόχρονα το περιεχόμενο της ίδιας θέσης μνήμης αλλά μπορούν να γράψουν ταυτόχρονα σε οποιαδήποτε θέση μνήμης.

Μετρα Εκτιμησης της Αποτελεσματικοτητας των Αλγοριθμων

Η αποτελεσματικότητα ενός αλγόριθμου εκτιμάται με βάση τα μέτρα πολυπλοκότητας και σύγκρισής του. Αναλυτικότερα, πρόκειται για τις πληροφορίες που ακολουθούν:

- Χρόνος $T(n)$. Είναι ο παράλληλος χρόνος εκτέλεσης του αλγόριθμου.
- Επεξεργαστές $P(n)$. Είναι το πλήθος των επεξεργαστών που χρησιμοποιούνται για την υλοποίηση του αλγόριθμου.
- Κόστος $C(n)$. Είναι η ποσότητα που ορίζεται από τη σχέση:
  $C(n) = T(n)\,P(n)$.
- Επιτάχυνση $S(n)$. Είναι η ποσότητα που ορίζεται από τη σχέση:
  $S(n) = T_s(n) / T(n)$,
  όπου $T_s(n)$ είναι ο ακολουθιακός χρόνος επίλυσης του συγκεκριμένου προβλήματος.
- Αποτελεσματικότητα $E(n)$. Είναι η ποσότητα που ορίζεται από τη σχέση:
  $E(n) = S(n) / P(n)$.
- Εργασία $W(n)$. Είναι ο συνολικός αριθμός λειτουργιών-εντολών (operations) που πραγματοποιεί ο αλγόριθμος.

7.2 Παραδείγματα Προσομοίωσης του Συνόλου Γεγονότων σε Παράλληλο Περιβάλλον

Όπως έχει ήδη αναφερθεί, είναι δυνατόν να χρησιμοποιηθούν παράλληλοι αλγόριθμοι προκειμένου να επιταχυνθούν οι διαδικασίες της εισαγωγής και της διαγραφής, για τις περιπτώσεις που σειριακά δεν πραγματοποιούνται αποδοτικά.

Η χρήση περισσότερων του ενός επεξεργαστών είναι δυνατόν να συμβάλλει σημαντικά στην προσπάθεια να γίνει όσον το δυνατόν ταχύτερα μια λειτουργία. Για παράδειγμα, η διαδικασία της διαγραφής του γεγονότος με το μικρότερο χρόνο δρομολόγησης από το σύνολο γεγονότων, όταν χρησιμοποιείται η δομή δεδομένων της ταξινομημένης, διπλά συνδεδεμένης λίστας πραγματοποιείται εύκολα σε σταθερό χρόνο. Αντίθετα, η διαδικασία της εισαγωγής ολοκληρώνεται σε χρόνο $O(N)$, όπου N είναι το πλήθος των γεγονότων της λίστας. Αν χρησιμοποιηθεί όμως CREW PRAM μοντέλο και N επεξεργαστές, τότε η αναζήτηση της κατάλληλης θέσης εισαγωγής του νέου γεγονότος πραγματοποιείται εύκολα αν κάθε ένας από τους επεξεργαστές αναλάβει να συγκρίνει το περιεχόμενο της θέσης μνήμης που αντιστοιχεί σε έναν μόνο κόμβο με την νέα τιμή που πρόκειται να εισαχθεί. Έτσι, κάθε επεξεργαστής θα εκτελέσει μία μόνο σύγκριση, δεδομένου ότι όλοι οι επεξεργαστές διαβάζουν από την κοινόχρηστη μνήμη, σε σταθερό χρόνο, τη νέα πληροφορία που πρόκειται να εισαχθεί έτσι ώστε να είναι δυνατόν να γίνει σύγκριση με την τιμή του γεγονότος της λίστας που έχουν διαβάσει.

Αφού κάθε ένας από τους επεξεργαστές πραγματοποιήσει τη σύγκριση γράφει σε δεδομένη θέση ενός διανύσματος μήκους N στην κοινόχρηστη μνήμη, την τιμή 0 ή 1, ανάλογα με το αποτέλεσμα της σύγκρισης. Αφού η λίστα είναι ταξινομημένη, το διάνυσμα θα περιέχει ακολουθία από 1 την οποία διαδέχεται ακολουθία από 0 ή αντίστροφα. Το σημείο στο οποίο γίνεται η εναλλαγή, το οποίο εντοπίζεται εύκολα και σε σταθερό χρόνο είναι αυτό στο οποίο πρέπει να πραγματοποιηθεί η εισαγωγή του νέου γεγονότος.

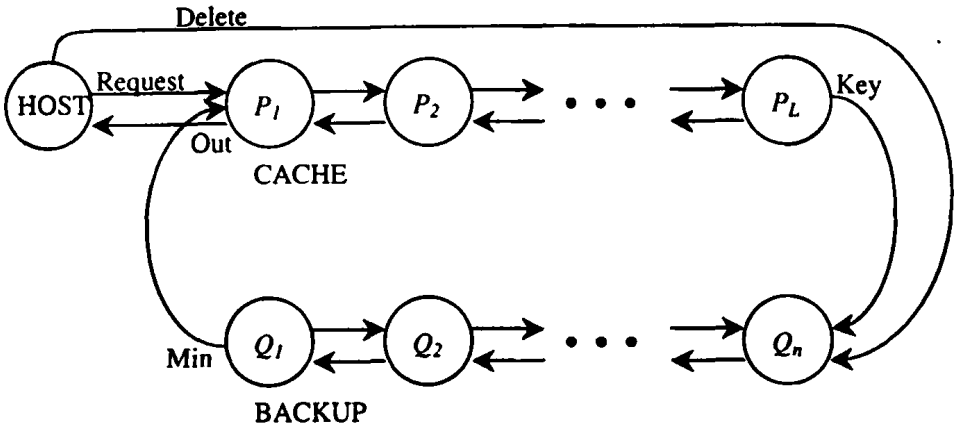
Η παραπάνω διαδικασία είναι δυνατόν να πραγματοποιηθεί σε σταθερό χρόνο, με την προϋπόθεση ότι το υπολογιστικό μοντέλο που θα χρησιμοποιηθεί θα είναι CREW PRAM.

Προσομοίωση του Συνόλου Γεγονότων με Χρήση Παράλληλου Σωρού

Στην παράγραφο αυτή αναφέρεται ένας τρόπος προσομοίωσης του συνόλου γεγονότων με χρήση παράλληλου σωρού [30]. Το Σχήμα 7.2 παρουσιάζει έναν τρόπο με τον οποίο $O(\log N)$ επεξεργαστές, γραμμικά συνδεδεμένοι, μπορούν να πραγματοποιήσουν τις διαδικασίες της εισαγωγής και της διαγραφής από δομή N στοιχείων σε σταθερό χρόνο. Η δομή που παρουσιάζεται [30] αποτελείται από δύο μέρη:

1. **Cache**. Πρόκειται για δομή προσομοίωσης του συνόλου γεγονότων η οποία χρησιμοποιεί $L = \Theta(1) + \log N$ επεξεργαστές.

2. **Backup**. Πρόκειται για $n = \log N$ επεξεργαστές σε γραμμική διάταξη. Οι αιτήσεις φτάνουν στον επεξεργαστή n και μετακινούνται μέσω της δομής από τον ένα επεξεργαστή στον άλλο.



Σχήμα 7.2

Λεπτομέρειες για τη λειτουργία της δομής δεν αναφέρονται αφού θεωρείται ότι η χρήση παράλληλου σωρού που υλοποιείται με τον τρόπο που περιγράφηκε, δεν βρίσκεται στα πλαίσια αυτής της μελέτης.

7.3 Αλγόριθμοι για την Παράλληλη Εκτέλεση των Λειτουργιών του Συνόλου Γεγονότων

Έχει ήδη αναφερθεί ότι υπάρχει η δυνατότητα, με τη χρήση ενός παράλληλου συστήματος, να επιταχυνθούν σημαντικά λειτουργίες που εφαρμόζονται σε κάποια δομή δεδομένων. Δηλαδή οι διαδικασίες της εισαγωγής και της διαγραφής ενός γεγονότος στο σύνολο γεγονότων (event set) γίνονται πιο αποδοτικές.

Υπάρχει όμως και η δυνατότητα να γίνει ταυτόχρονη εισαγωγή k γεγονότων στο σύνολο γεγονότων καθώς επίσης και διαγραφή k γεγονότων, τα οποία προφανώς θα έχουν τους k μικρότερους χρόνους δρομολόγησης. Για την πραγματοποίηση αυτής της διαδικασίας χρησιμοποιούνται περισσότεροι του ενός επεξεργαστές. Σε μια δομή Q που επιτρέπει τα παραπάνω, μπορούν να οριστούν οι εξής λειτουργίες:

* *Insert* ($<i_1, i_2, ..., i_N>$, Q), για την εισαγωγή των στοιχείων $i_1, i_2, ..., i_N$ στη δομή Q.
* *Delete_min* (Q, k), για την διαγραφή και επιστροφή των k στοιχείων με τη μικρότερο τιμή.
* *Make_Queue* (S, Q), για την σύνθεση της δομής Q από τα στοιχεία του συνόλου S.

Στη συνέχεια μελετάται η βελτίωση της αποτελεσματικότητας αλγορίθμων που χρησιμοποιούν τις δομές δεδομένων leftist heap, binomial heap, n-heap, M-heap, P-tree, IP-tree και MP-tree όπου πραγματοποιείται ταυτόχρονη εισαγωγή k στοιχείων αλλά και διαγραφή ισάριθμων στοιχείων από το σύνολο γεγονότων.

Παράλληλος Αλγόριθμος για τη Δομή Leftist Heap

Προκειμένου να επιτευχθούν αποτελεσματικά οι διαδικασίες k-insert και k-delete_minimum σε μια δομή leftist heap, δηλαδή η εισαγωγή k στοιχείων και η διαγραφή των k μικρότερων στοιχείων που υπάρχουν στη δομή, είναι απαραίτητο να γίνουν μερικές τροποποιήσεις στη δομή, όπως αυτή περιγράφηκε στο Κεφάλαιο 3. Έτσι, η νέα δομή, k-leftist heap, αποτελείται από κόμβους οι οποίοι περιέχουν k ταξινομημένα στοιχεία. Η διάταξη των κόμβων και των στοιχείων τους είναι τέτοια ώστε το στοιχείο που έχει τη μεγαλύτερη πληροφορία σε κάθε κόμβο να είναι μικρότερο ή ίσο από όλα τα στοιχεία που βρίσκονται στους απογόνους του κόμβου. Η οργάνωση αυτή έχει σαν αποτέλεσμα τα k μικρότερα στοιχεία να εντοπίζονται στον κόμβο ρίζα. Αν N είναι το πλήθος των στοιχείων που είναι αποθηκευμένα στη δομή τότε για το ύψος h του πιο δεξιού μονοπατιού της δομής θα ισχύει: $h \in O(\log(N/k))$.

Στη συνέχεια αποδεικνύεται ότι, για τη δομή k-leftist heap, με k επεξεργαστές και σε CREW-PRAM μοντέλο ισχύουν τα ακόλουθα:

- k στοιχεία εισάγονται σε χρόνο $O(h + \log k)$
- Τα k στοιχεία με τη μικρότερη πληροφορία διαγράφονται σε χρόνο $O(h + \log\log k)$
- Η δομή leftist heap μπορεί να δημιουργηθεί με στοιχεία που προέρχονται από ένα σύνολο S που περιέχει N στοιχεία σε χρόνο $O((N/k) \log k)$
- Δύο leftist heaps που περιέχουν N_1 και N_2 στοιχεία συγχωνεύονται σε χρόνο $O(h_1 + h_2 + \log\log k)$ όπου $h_1 = \log(N_1/k)$ και $h_2 = \log(N_2/k)$.

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΔΟΜΗΣ K-LEFTIST HEAP

Η διαδικασία της συγχώνευσης αποτελεί τη βασική διαδικασία για την πραγματοποίηση των διαδικασιών της εισαγωγής και της διαγραφής του μικρότερου στοιχείου. Για τη συγχώνευση δύο δομών leftist heap Q_1 και Q_2 σε μια δομή Q συνδυάζονται τα μονοπάτια ρ_1 και ρ_2 από τις ρίζες μέχρι τα πιο δεξιά φύλλα σε ένα μονοπάτι ρ μήκους $h = h_1 + h_2$, όπου h_1 και h_2 το πλήθος των κόμβων του μονοπατιού ρ_1 και ρ_2 αντίστοιχα, με $h_1 \leq h_2$. Οι υπόλοιποι κόμβοι των Q_1 και Q_2 ενώνονται κατάλληλα με αυτό το μονοπάτι και τέλος τροποποιούνται οι τιμές των μεταβλητών *rank* των κόμβων ώστε να ισχύουν οι ιδιότητες της δομής leftist heap.
Ένας κόμβος P της δομής αναπαρίσταται από ένα record $P = (E, left, right, rank)$ όπου $P.E$ είναι η ταξινομημένη ακολουθία k στοιχείων που είναι αποθηκευμένη στον κόμβο P, $P.left$ και $P.right$ είναι οι δείκτες προς το αριστερό και το δεξί παιδί αντίστοιχα του κόμβου P ενώ $P.rank$ είναι η τιμή της μεταβλητής *rank* που αντιστοιχεί στον κόμβο P. Επιπλέον, για τον κόμβο P ορίζεται $L(P)$ να είναι το ζεύγος $(P.left, \max(P.E))$ και $E_{\rho i}$ με $i = 1, 2$, είναι η συνένωση των λιστών $P.E$ των κόμβων που ανήκουν στο μονοπάτι ρ_i. Όμοια, $L_{\rho i}$ ορίζεται να είναι η ακολουθία των ζευγών που αντιστοιχούν στους κόμβους του ρ_i. Η ακολουθία $L_{\rho i}$ είναι ταξινομημένη αφού το δεύτερο μέρος κάθε ζεύγους εμφανίζεται κατά μη φθίνουσα σειρά. Η διαδικασία της συγχώνευσης περιγράφεται αλγοριθμικά στη συνέχεια.

Meld(Q_1, Q_2, Q):

1. E = Parallel-Merge($E_{\rho 1}$, $E_{\rho 2}$);
2. L = Parallel-Merge($L_{\rho 1}$, $L_{\rho 2}$) (με βάση το δεύτερο πεδίο των ζευγών);
   Έστω E^1, ..., $E^{h1 + h2}$ οι $h_1 + h_2$ το πλήθος υποακολουθίες μήκους k που αποτελούν την ακολουθία E;
   Έστω L^1, ..., $L^{h1 + h2}$ τα $h_1 + h_2$ το πλήθος πρώτα μέρη των ζευγών του L;
3. $h = h_1 + h_2$;
4. new(P);
   4.1 $P.E = E^h$;
   4.2 $P.left = L^h$;
   4.3 $P.right$ = NULL;
   4.4 $P.rank = 1$;
5. Για $i = h$-1 μέχρι 1 εκτέλεσε:
   5.1 $M = (E^i, L^i, P, 1)$;
   5.2 Αν $M.right.rank > M.left.rank$ τότε swap($M.left$, $M.right$);
   5.3 $M.rank = M.right.rank + 1$;
   5.4 new(P);
   5.5 $P = M$;
6. $Q = P$;

Σημειώνεται ότι το βήμα 5 της παραπάνω διαδικασίας πραγματοποιεί τη δημιουργία του μονοπατιού καθώς και την ενημέρωση των μεταβλητών *rank*. Για την ορθότητα του παραπάνω αλγόριθμου αρκεί να παρατηρήσει κανείς ότι η σύνθεση του πιο δεξιού μονοπατιού ρ εγγυάται την ισχύ των ιδιοτήτων της δομής του leftist heap, αφού για κάθε υποδέντρο T με ρίζα έναν κόμβο P του μονοπατιού ρ, τα στοιχεία του $P.E$ μπορεί να είναι μικρότερα ή ίσα με εκείνα του κόμβου του ρ_1 ή του ρ_2 που αρχικά ήταν ρίζα του T. Η χρονική πολυπλοκότητα της διαδικασίας καθορίζεται από την συνάρτηση Parallel-Merge. Όλα τα υπόλοιπα βήματα απαιτούν χρόνο τάξης $O(h_1 + h_2)$. Έτσι, για την χρονική πολυπλοκότητα της συνολικής διαδικασίας θα ισχύει $O(h_1 + h_2 + \text{loglog } k) = O(h + \text{loglog } k)$.

Όλες οι υπόλοιπες διαδικασίες, δηλαδή η εισαγωγή k στοιχείων, η διαγραφή των k μικρότερων στοιχείων και η δημιουργία της δομής βασίζονται στη διαδικασία της συγχώνευσης. Συγκεκριμένα, προκειμένου να εισαχθούν k νέα στοιχεία στη δομή, έστω Q, αυτά εισάγονται ταξινομημένα σε έναν κόμβο ο οποίος αποτελεί δομή k-leftist heap η οποία συγχωνεύεται στη συνέχεια με τη δομή Q. Για τη διαγραφή των k μικρότερων στοιχείων, τα οποία βρίσκονται στη ρίζα, διαγράφεται η ρίζα της δομής και συγχωνεύονται τα δύο υποδέντρα της, τα οποία αποτελούν δομές k-leftist heap. Για τη δημιουργία της δομής N στοιχείων δημιουργούνται N/k δομές, οι οποίες αποτελούνται από έναν κόμβο, που στη συνέχεια συγχωνεύονται. Υπολογίζεται ότι η χρονική πολυπλοκότητα των διαδικασιών της εισαγωγής, της διαγραφής και της δημιουργίας της δομής είναι $O(h + \log k)$, $O(h + \text{loglog} k)$, $O((N/k)\log k)$, αντίστοιχα [29].

Παράλληλος Αλγόριθμος για τη Δομή Binomial Heap

Σύμφωνα με όσα έχουν αναφερθεί στο Κεφάλαιο 3, ένας διωνυμικός σωρός n κόμβων αποτελείται από το πολύ $\lfloor \log n \rfloor + 1$ διωνυμικά δέντρα. Έτσι, κατά τη διαδικασία της συνένωσης, μετά τη συγχώνευση των λιστών των ριζών των δύο δομών, συνδέονται τα διωνυμικά δέντρα των οποίων οι ρίζες έχουν τον ίδιο βαθμό. Η χρονική πολυπλοκότητα της διαδικασίας είναι $O(\log n)$, όπου n είναι ο συνολικός αριθμός των κόμβων που περιέχουν οι δύο

σωροί που συγχωνεύονται. Για την εισαγωγή ενός νέου στοιχείου σε μια δομή διωνυμικού σωρού, θεωρείται ότι το νέο αυτό στοιχείο αποτελεί σωρό ενός κόμβου οπότε συγχωνεύεται με τη δομή n κόμβων σε χρόνο O(log$n$). Η διαδικασία της διαγραφής του στοιχείου με τη μικρότερη τιμή, η οποία ολοκληρώνεται σε χρόνο O(log$n$), αρχικοποιείται με τον εντοπισμό και τη διαγραφή από τη λίστα των ριζών της ρίζας με τη μικρότερη τιμή. Στη συνέχεια δημιουργείται ο σωρός που αποτελείται από τα παιδιά του κόμβου που διαγράφηκε ο οποίος συγχωνεύεται στη συνέχεια με το υπόλοιπο της δομής.

Η δομή k-binomial heap προκύπτει από τη δομή binomial heap. Η διαφορά είναι ότι οι διαδικασίες της εισαγωγής και της διαγραφής που περιγράφηκαν παραπάνω τροποποιούνται ώστε η δομή k-Binomial heap να επιτρέπει την εισαγωγή k στοιχείων και τη διαγραφή των k μικρότερων στοιχείων με την προϋπόθεση ότι ισχύει $k = 2^d$. Προφανώς και για τη δομή αυτή ισχύουν όσα αναφέρθηκαν παραπάνω.

Η διαδικασία της εισαγωγής k νέων στοιχείων στη δομή k-binomial heap, έστω H, που περιέχει N στοιχεία, περιλαμβάνει δύο στάδια. Κατά το πρώτο στάδιο, τα k νέα στοιχεία συγχωνεύονται και δημιουργούν ένα νέο σωρό ο οποίος, με βάση τις ιδιότητες που αναφέρθηκαν παραπάνω, αποτελείται από ένα δέντρο B_d (αφού $k = 2^d$). Έτσι, στο δεύτερο στάδιο της διαδικασίας ο σωρός αυτός συγχωνεύεται με τη δομή H. Η δημιουργία του σωρού του πρώτου σταδίου ολοκληρώνεται σε χρόνο τάξης $O(\log^2 k)$ με $O(k)$ επεξεργαστές και σε EREW PRAM. Αρχικά, κάθε ένας από τους επεξεργαστές πραγματοποιεί τη συγχώνευση 2 στοιχείων με αποτέλεσμα να προκύψουν $k/2$ δέντρα που το κάθε ένα περιέχει 2 στοιχεία. Στη συνέχεια τα δέντρα συγχωνεύονται ανά δύο και προκύπτουν $k/4$ δέντρα των 4 στοιχείων. Έτσι, κατά την i-οστή επανάληψη προκύπτουν $k/2^i$ δέντρα που το κάθε ένα περιέχει 2^i στοιχεία. Μετά την ολοκλήρωση των $h = O(\log k)$ επαναλήψεων δημιουργείται ένα B_d δέντρο που περιέχει k στοιχεία. Επομένως, αφού η συγχώνευση δύο δομών με n συνολικά στοιχεία απαιτεί χρόνο τάξης O(log$n$), ο χρόνος που απαιτείται για τη δημιουργία του B_d δέντρου είναι:

$$O\left(\sum_{i=1}^{h} \log 2^i\right) = O\left(\sum_{i=1}^{h} i \log 2\right) = O(h^2) = O(\log^2 k)$$

Η συγχώνευση του B_d δέντρου με τη δομή H πραγματοποιείται σε χρόνο O(logN') όπου $N' = N+k$. Επομένως, η διαδικασία της εισαγωγής συνολικά πραγματοποιείται σε χρόνο $O(\log N' + \log^2 k)$.

Η διαδικασία της διαγραφής των k μικρότερων στοιχείων από μια δομή N στοιχείων ολοκληρώνεται σε χρόνο $O(k \log N)$ όταν θα έχουν πραγματοποιηθεί k διαδοχικές διαγραφές. Δηλαδή η αλγοριθμική περιγραφή της διαδικασίας είναι η εξής:

Delete_minimum(H):

1. Για i = 1 μέχρι k πραγματοποιούνται τα εξής:
   1.1 Εντοπίζεται και αφαιρείται από τη λίστα των ριζών η ρίζα $x[i]$ με τη μικρότερη τιμή;
   1.2 Αντιστρέφεται η σειρά των κόμβων που αποτελούν τη λίστα των παιδιών του κόμβου $x[i]$ και δημιουργείται έτσι ο σωρός H';
   1.3 H = Union(H, H');
2. return X;

Παράλληλος Αλγόριθμος για τη Δομή n-Heap

Η δομή δεδομένων που περιγράφεται στη συνέχεια και ονομάζεται *k*-heap, βασίζεται στη δομή *n*-heap. Αποτελεί ουσιαστικά επέκταση της δομής του δυαδικού σωρού (δυναμική αναπαράσταση), αφού η διαφορά είναι ότι κάθε κόμβος της δομής περιλαμβάνει *k* ταξινομημένα στοιχεία. Έτσι, αν *N* είναι ο συνολικός αριθμός των στοιχείων που βρίσκονται αποθηκευμένα στη δομή, τότε το ύψος του δέντρου θα είναι $h = \log(N/k)$. Στη συνέχεια αποδεικνύεται ότι με *k* επεξεργαστές και σε CREW-PRAM μοντέλο ισχύουν τα ακόλουθα:

- *k* στοιχεία εισάγονται σε χρόνο $O(h + \log k)$
- Τα *k* στοιχεία με τη μικρότερη πληροφορία διαγράφονται σε χρόνο $O(h + \log\log k)$
- Η δομή μπορεί να δημιουργηθεί με στοιχεία που προέρχονται από ένα σύνολο *S* που περιέχει *N* στοιχεία σε χρόνο $O((N/k)\log k)$

Η δομή *k*-heap είναι ουσιαστικά ένας δυαδικός σωρός που κάθε ένας από τους κόμβους του περιέχει *k* στοιχεία τα οποία είναι διατεταγμένα με έναν τρόπο ο οποίος στηρίζεται στις ιδιότητες της διάταξης των στοιχείων της δομής του σωρού. Δηλαδή, προκειμένου να εντοπίζονται τα *k* μικρότερα στοιχεία σε O(1) παράλληλο χρόνο, τα στοιχεία αποθηκεύονται κατά *k*-άδες στους κόμβους, κατά τέτοιο τρόπο ώστε το στοιχείο που έχει τη μεγαλύτερη πληροφορία να είναι μικρότερο ή ίσο από όλα τα στοιχεία που βρίσκονται στους απογόνους του κόμβου. Με αυτή την οργάνωση τα *k* μικρότερα στοιχεία εντοπίζονται στον κόμβο ρίζα. Όπως προαναφέρθηκε, τα στοιχεία που περιέχει κάθε κόμβος είναι ταξινομημένα.

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΔΟΜΗΣ *K*-HEAP

Έστω E_p η ταξινομημένη ακολουθία των *k* στοιχείων που βρίσκονται αποθηκευμένα σε έναν κόμβο *P* και $\max(E_p)$ το μέγιστο στοιχείο της ακολουθίας E_p. Επιπλέον, ένα μονοπάτι $\pi = P_1,...,P_n$ είναι μια ακολουθία από κόμβους της δομής από τη ρίζα P_1 μέχρι ένα φύλλο P_n και για δεδομένο μονοπάτι π, το E_π αντιστοιχεί στη συνένωση $E_{p1}E_{p2}...E_{pn}$ η οποία είναι μια ταξινομημένη ακολουθία στοιχείων. Όμως, κατά τη διάρκεια εισαγωγών ή διαγραφών το σύνολο που είναι αποθηκευμένο στον κόμβο P_1 ή P_n αντικαθίσταται από άλλο σύνολο, κάτι το οποίο μπορεί να διαταράξει τη σωστή διάταξη του E_π. Προκειμένου αυτό να ταξινομηθεί ξανά και να επανακτηθούν οι ιδιότητες της δομής, χρησιμοποιείται η συνάρτηση Rearrange, η οποία αποδεικνύεται ότι απαιτεί χρόνο της τάξης $O(h + \log\log k)$ με *k* επεξεργαστές σε CREW-PRAM μοντέλο, όπου *h* είναι το πλήθος των κόμβων που αποτελούν το μονοπάτι π.

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΕΙΣΑΓΩΓΗΣ

Προκειμένου να εισαχθούν *k* νέα στοιχεία στη δομή, πρώτα ταξινομούνται και εισάγονται αρχικά στο πιο αριστερό, μη κενό φύλλο της δομής, έστω V_L. Στη συνέχεια αναδιοργανώνεται το μονοπάτι π από τη ρίζα μέχρι το φύλλο V_L με κλήση της συνάρτησης Rearrange. Ακολουθεί η περιγραφή της συνάρτησης Rearrange και ο αλγόριθμος της διαδικασίας της εισαγωγής.

Rearrange(π, $P \in \{P_1, P_n\}$):

1. E = Parallel-Merge(E_p, $E_{\pi\text{-}P}$);

2. Έστω E^1, ..., E^h είναι οι υποακολουθίες (h το πλήθος) που αποτελούν την ακολουθία E και έχουν μήκος k. Τότε:
   Για $i = 1$ μέχρι h: $E_{Pi} = E^i$;

Αφού η αντιγραφή των υποακολουθιών E^i στις υποακολουθίες E_{Pi} απαιτεί παράλληλο χρόνο $O(h)$, η χρονική πολυπλοκότητα της συνάρτησης Rearrange καθορίζεται από το χρόνο που απαιτεί η συνάρτηση Parallel-Merge για τη συγχώνευση δύο ακολουθιών που έχουν μήκος k και k (h - 1) με k επεξεργαστές. Είναι όμως γνωστό ότι η συγχώνευση δύο ακολουθιών με μήκος έστω k_1 και k_2 πραγματοποιείται σε χρόνο τάξης:

$$\Theta\left(\frac{k_1 + k_2}{n} + \log\log n\right)$$

με n επεξεργαστές και CREW-PRAM μοντέλο. Έτσι, προκύπτει ότι η χρονική πολυπλοκότητα της διαδικασίας Rearrange είναι $O(h + \log\log k)$, όπως έχει ήδη αναφερθεί παραπάνω.

Insert(x_1, x_2, ..., x_k):

1. Τα k νέα στοιχεία ταξινομούνται (Parallel-Sort) και τοποθετούνται στο πιο αριστερό κενό φύλλο V_L της δομής;
2. Καθορίζεται το μονοπάτι π από τη ρίζα στο φύλλο V_L;
3. Rearrange(π, V_L);

Αφού h είναι το πλήθος των κόμβων που αποτελούν το μονοπάτι π, το τελευταίο μπορεί να προσδιοριστεί σε $O(h)$ χρόνο. Έτσι, ο χρόνος που απαιτεί η διαδικασία της εισαγωγής καθορίζεται από τις συναρτήσεις Parallel-Sort και Rearrange. Όπως είναι γνωστό όμως, σε CREW PRAM μοντέλο n επεξεργαστών, n στοιχεία ταξινομούνται σε χρόνο $\Theta(\log n)$. Συμπερασματικά, η διαδικασία της εισαγωγής πραγματοποιείται επομένως σε χρόνο της τάξης $O(h + \log k)$ με χρήση k επεξεργαστών και CREW-PRAM μοντέλου.

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

Για την περιγραφή της διαδικασίας της διαγραφής είναι απαραίτητο να διευκρινιστεί ο όρος ελάχιστο μονοπάτι μ της δομής, ο οποίος ορίζεται αναδρομικά. Έτσι:

1. η ρίζα της δομής ανήκει στο μονοπάτι μ.
2. έστω ότι ένας εσωτερικός κόμβος P ανήκει επίσης στο μονοπάτι μ. Τότε, αν X και Y είναι τα παιδιά του P και το Y είτε δεν υπάρχει είτε ισχύει $\max(E_X) \leq \max(E_Y)$, ο κόμβος X ανήκει στο μονοπάτι μ.

Η διαδικασία της διαγραφής χρησιμοποιεί τις διαδικασίες Adjust και Rearrange. Ακολουθεί η περιγραφή της διαδικασίας Adjust.

Adjust(μ):

1. Για κάθε κόμβο $P \in \mu$ πραγματοποιούνται τα εξής:
   Αν υπάρχει αδερφός του κόμβου P τότε:
   1.1 E = Parallel-Merge(E_p, $E_{Sibling(P)}$);
   1.2 Έστω E^1 και E^2 το αριστερό και το δεξί μισό τμήμα του E, αντίστοιχα. Τότε:

$E_p = E^1$;
$E_{Sibling(P)} = E^2$;

Η παραπάνω διαδικασία αναδιοργανώνει τα στοιχεία των κόμβων που ανήκουν στο μονοπάτι μ καθώς στους αδερφούς αυτών των κόμβων προκειμένου να εξακολουθούν να ισχύουν οι ιδιότητες της δομής. Έστω ότι ανατίθενται $\max\{\lfloor k/h \rfloor, 1\}$ επεξεργαστές σε κάθε κόμβο του μ προκειμένου να πραγματοποιήσουν τη διαδικασία Parallel-Merge μεταξύ του συγκεκριμένου κόμβου και του αδερφού του. Ο παραπάνω αλγόριθμος αποτελείται τότε από $\lceil h/k \rceil$ φάσεις κάθε μια από τις οποίες πραγματοποιεί $\min\{h, k\}$ στιγμιότυπα της Parallel-Merge σε διαφορετικά ζεύγη κόμβων. Αφού κάθε φάση απαιτεί χρόνο τάξης $O(\min\{h, k\} + \log\log k)$, η διαδικασία Adjust τελικά έχει χρονική πολυπλοκότητα ίση με $O(h + \log\log k)$.

Έστω R και F_R οι κόμβοι που αντιστοιχούν στη ρίζα και στο πιο δεξί μη κενό φύλλο της δομής σε ύψος h. Αφού η ρίζα περιέχει τα k μικρότερα στοιχεία που υπάρχουν στη δομή, η διαδικασία της διαγραφής επιστρέφει την ακολουθία E_R και αντικαθιστά τα στοιχεία της ρίζας με τα στοιχεία του κόμβου F_R. Καλούνται στη συνέχεια οι συναρτήσεις Adjust και Rearrange προκειμένου να μην παραβιαστούν οι ιδιότητες της δομής. Έτσι, η διαδικασία της διαγραφής περιγράφεται αλγοριθμικά ως εξής:

Delete(k):

1. Επιστρέφονται τα στοιχεία της ακολουθίας E_R;
2. $E_R = E_{FR}$;
3. Προσδιορίζεται το ελάχιστο μονοπάτι μ;
4. Adjust(μ);
5. Rearrange(μ, R);

Η πολυπλοκότητα χρόνου της διαδικασίας της διαγραφής καθορίζεται από εκείνη των συναρτήσεων Adjust και Rearrange, αφού τα υπόλοιπα βήματα του αλγόριθμου πραγματοποιούνται σε χρόνο $O(h)$. Έτσι, απαιτεί χρόνο της τάξης $O(h + \log\log k)$.

➢ Η ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΔΟΜΗΣ

Η δημιουργία της δομής πραγματοποιείται σε δύο ουσιαστικά φάσεις. Έστω $S[1, ..., N]$ το σύνολο των N στοιχείων που πρέπει να αποθηκευθούν στη δομή. Αρχικά τα στοιχεία τοποθετούνται στο δέντρο και τα στοιχεία που υπάρχουν σε κάθε κόμβο ταξινομούνται. Στη συνέχεια η δομή τροποποιείται προκειμένου να αποκτήσει και τις ιδιότητες του σωρού. Αυτό επιτυγχάνεται με μια διαδικασία που ελέγχει το δέντρο κατά επίπεδα ξεκινώντας από τα φύλλα. Συγκεκριμένα, το ελάχιστο μονοπάτι μ_T που αντιστοιχεί σε κάθε υποδέντρο T στο επίπεδο i αναδιοργανώνεται με τη βοήθεια των συναρτήσεων Adjust και Rearrange. Η συνολική διαδικασία της δημιουργίας της δομής k-heap παρουσιάζεται στη συνέχεια:

Make_Heap(S, k-heap Q)

1. Για $i = 1$ μέχρι $N/k - 1$:
   $Q[i\,k + 1, ..., (i + 1)\,k] =$ Parallel-Sort($S[i\,k + 1, ..., (i + 1)\,k]$);
2. Για $i = h$ μέχρι 1:
   Για κάθε κόμβο P του επιπέδου i:
   Έστω T το υποδέντρο με ρίζα τον κόμβο P.

2.1 Καθορίζεται το ελάχιστο μονοπάτι μ_T;
2.2 Adjust(μ_T);
2.3 Rearrange(μ_T, P);

Η ορθότητα του παραπάνω αλγόριθμου μπορεί εύκολα να αποδειχθεί με επαγωγή στο ύψος h. Σύμφωνα με τα γνωστά για τη χρονική πολυπλοκότητα της ταξινόμησης, το πρώτο βήμα του παραπάνω αλγόριθμου πραγματοποιείται σε χρόνο $O((N/k)\log k)$. Κατά τη διάρκεια του δεύτερου βήματος όλοι οι επεξεργαστές δουλεύουν παράλληλα προκειμένου να αναδιοργανωθεί κάθε υποδέντρο με ρίζα κάθε κόμβο P του επιπέδου i. Έτσι, κάθε επιμέρους βήμα του βήματος 2 απαιτεί χρόνο τάξης $O(h - i + \log\log k)$, ενώ συνολικά το δεύτερο βήμα απαιτεί χρόνο:

$$O\left(\sum_{j=1}^{h}\frac{N}{k2^j}(j+\log\log k)\right)=O\left(\frac{N}{k}\log\log k\right)$$

Επομένως, συνολικά η διαδικασία Make_Heap εκτελείται με πολυπλοκότητα χρόνου τάξης:

$$O\left(\frac{N}{k}\log k\right)$$

Παράλληλος Αλγόριθμος για τη Δομή M-Heap

Η δομή δεδομένων που περιγράφεται στη συνέχεια, και ονομάζεται k-M-heap, βασίζεται στη δομή M-heap. Η διαφορά είναι ότι η δομή k-M-heap επιτρέπει την εισαγωγή k στοιχείων καθώς και τη διαγραφή των k μικρότερων στοιχείων που υπάρχουν στη δομή. Η τελευταία πραγματοποιείται εύκολα αν τα k μικρότερα στοιχεία βρίσκονται αποθηκευμένα σε k διακεκριμένους σωρούς (στατικής αναπαράστασης), κάτι που επιτυγχάνεται από τον τρόπο δημιουργίας της δομής και αποτελεί τη βασική της ιδιότητα. Έτσι, αν N είναι ο συνολικός αριθμός των στοιχείων που βρίσκονται αποθηκευμένα στη δομή και k το πλήθος των σωρών που την αποτελούν, τότε κάθε σωρός περιέχει N/k στοιχεία. Η διαδικασία της εισαγωγής των k νέων στοιχείων πραγματοποιείται με τέτοιο τρόπο ώστε οι σωροί να εξακολουθούν να αποτελούνται από ισάριθμα στοιχεία, όπως ισχύει κατά τη δημιουργία της δομής, αλλά και χωρίς να διαταράσσεται η ιδιότητα που περιγράφηκε παραπάνω. Σημειώνεται ακόμη ότι η δομή I-heap που χρησιμοποιείται από τη δομή M-heap, όπως περιγράφηκε σε προηγούμενο κεφάλαιο, δεν χρησιμοποιείται από τη δομή k-M-heap, αφού δεν χρειάζεται να είναι γνωστός ο σωρός που περιέχει τη μικρότερη πληροφορία που υπάρχει στη δομή. Στη συνέχεια αποδεικνύεται ότι με n επεξεργαστές και σε CREW-PRAM μοντέλο ισχύουν τα ακόλουθα:

- k στοιχεία εισάγονται σε χρόνο $O((n/k)\log\log k)$
- Τα k στοιχεία με τη μικρότερη πληροφορία διαγράφονται σε χρόνο $O(1)$

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΔΟΜΗΣ *K-M*-HEAP

Έστω ότι η δομή αποτελείται από τα N στοιχεία της ακολουθίας X, τα οποία, κατά τη δημιουργία της, μοιράζονται σε k σωρούς. Κατά τη διαδικασία της διαγραφής επιστρέφεται ένας μονοδιάστατος πίνακας $MIN[k]$ που περιέχει τα k μικρότερα στοιχεία της δομής, ενώ κατά

την εισαγωγή χρησιμοποιείται ένας επίσης μονοδιάστατος πίνακας *NEW*[*k*] ο οποίος περιέχει τα *k* νέα στοιχεία που πρόκειται να εισαχθούν.

➢ Η ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΔΟΜΗΣ

Για τη δημιουργία της δομής χρησιμοποιείται η διαδικασία Build_*k*-*M*-Heap, η οποία περιγράφεται αλγοριθμικά στη συνέχεια. Η ακολουθία *X* ταξινομείται κατά το βήμα 1 με χρήση της συνάρτησης Parallel-Sort σε O(log*k*) χρόνο σε CREW PRAM μοντέλο *k* επεξεργαστών. Αφού πραγματοποιηθεί η ταξινόμηση, κάθε ένας από τους επεξεργαστές εισάγει ακολουθιακά μια ακολουθία *N*/*k* στοιχείων στον αντίστοιχο σωρό ως εξής: ο πρώτος επεξεργαστής εισάγει στον πρώτο από τους σωρούς τα πρώτα *N*/*k* στοιχεία της ταξινομημένης ακολουθίας *X*, ο δεύτερος επεξεργαστής εισάγει ακολουθιακά στον δεύτερο σωρό το δεύτερο τμήμα *N*/*k* στοιχείων της ακολουθίας *X* κτλ. Για την ακολουθιακή εισαγωγή των στοιχείων, που πραγματοποιεί κάθε επεξεργαστής, εκτελείται η γνωστή διαδικασία της εισαγωγής ενός στοιχείου σε μια δομή σωρού.

Build_*k*-*MP*-tree(*X*):

1. Parallel-Sort(*X*);
2. For $i = 1$ to k do in parallel:
   2.1 $n = i$;
   2.2 $m = 1$;
   2.3 while $n \leq N$ do:
      2.3.1 $MH[i][m] = X[n]$;
      2.3.2 $n = n + k$;
      2.3.3 $m = m + 1$;
3. For $i = 1$ to k do in parallel:
   3.1 For $n = 1$ to N / k do:
      Insert($MH[i][n]$ at heap #i);

Σε κάθε γραμμή του πίνακα δύο διαστάσεων *MH*[*k*][*N*/*k*] που χρησιμοποιεί η παραπάνω συνάρτηση, αποθηκεύονται τα *N*/*k* στοιχεία που αντιστοιχούν σε κάθε έναν από τους σωρούς.

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

Για την πραγματοποίηση της διαδικασίας της διαγραφής των *k* μικρότερων στοιχείων (Delete_*k*-*M*-Heap) χρησιμοποιείται η συνάρτηση Delete_minimum, η οποία διαγράφει και επιστρέφει το μικρότερο στοιχείο από μια δομή σωρού, το οποίο βρίσκεται στην πρώτη θέση του πίνακα που αντιστοιχεί στο σωρό και στη συνέχεια τα στοιχεία του πίνακα μετακινούνται μια θέση αριστερά. Κάθε ένας από τους επεξεργαστές εκτελεί τη διαδικασία Delete_minimum σε κάθε έναν από τους σωρούς, με αποτέλεσμα η Delete_*k*-*M*-Heap να επιστρέφει ένα διάνυσμα *MIN*[*k*] που περιέχει τα *k* μικρότερα στοιχεία που υπάρχουν στη δομή *k*-*M*-heap. Υπενθυμίζεται ότι τα *k* μικρότερα στοιχεία βρίσκονται σε *k* διακεκριμένους σωρούς. Έτσι, η διαδικασία της διαγραφής πραγματοποιείται σε σταθερό χρόνο με χρήση *n*/*k* επεξεργαστών για κάθε έναν από τους σωρούς. Ακολουθεί η αλγοριθμική περιγραφή της διαδικασίας της διαγραφής.

Delete_*k*-*M*-Heap:

1. for $i = 1; i \leq k; i = i + 1;$ do in parallel:
   MIN[i] = Delete_minimum(from heap #i);
3. return(*MIN*);

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΕΙΣΑΓΩΓΗΣ

Έστω *NEW* η ακολουθία των k νέων στοιχείων που πρόκειται να εισαχθούν. Όπως έχει ήδη αναφερθεί, είναι απαραίτητο η εισαγωγή των νέων στοιχείων να μην διαταράσσει την βασική ιδιότητα της δομής, σύμφωνα με την οποία τα k μικρότερα στοιχεία ανήκουν σε k διαφορετικούς σωρούς. Προκειμένου να επιτευχθεί αυτό, πραγματοποιείται συγχώνευση των στοιχείων που ήδη υπάρχουν στη δομή με τα k νέα. Έτσι, συγκρίνεται το μικρότερο στοιχείο της ακολουθίας *NEW*, έστω a, με τα στοιχεία του πρώτου σωρού, ξεκινώντας από το μικρότερο, μέχρι να βρεθεί το i-στό το οποίο είναι το πρώτο μεγαλύτερο στοιχείο από το a. Με αυτό τον τρόπο εντοπίζεται το σημείο από το οποίο ξεκινά η συγχώνευση, προκειμένου να αποφευχθούν άσκοπες συγκρίσεις. Συγκεκριμένα, αφού εντοπιστούν τα i-στά στοιχεία σε κάθε έναν από τους σωρούς, σχηματίζεται από αυτά η ακολουθία R, η οποία συγχωνεύεται με την αρχική ακολουθία *NEW*. Προκύπτει έτσι μια νέα ακολουθία M που έχει μήκος $2k$. Στη συνέχεια, τα k στοιχεία της R, τα οποία ανήκουν σε διαφορετικούς σωρούς, αντικαθίστανται από τα k μικρότερα στοιχεία της M. Τα υπόλοιπα k στοιχεία της M αποτελούν στη συνέχεια τα στοιχεία της νέας ακολουθίας *NEW* και έτσι η διαδικασία επαναλαμβάνεται μέχρι να έχουν εισαχθεί όλα τα νέα στοιχεία, δηλαδή τα στοιχεία της αρχικής ακολουθίας *NEW*.

Σημειώνεται ότι ο εντοπισμός του i-στού μικρότερου στοιχείου που υπάρχει σε κάθε έναν από τους σωρούς γίνεται σε σταθερό χρόνο αφού πρόκειται για το στοιχείο που είναι αποθηκευμένο στη θέση i του πίνακα που χρησιμοποιείται από κάθε σωρό για την αποθήκευση των στοιχείων του. Η ιδιότητα αυτή ισχύει λόγω του τρόπου δημιουργίας της δομής.

Η παραπάνω διαδικασία περιγράφεται αλγοριθμικά ως εξής:

Insert_k-M-Heap(*NEW*):

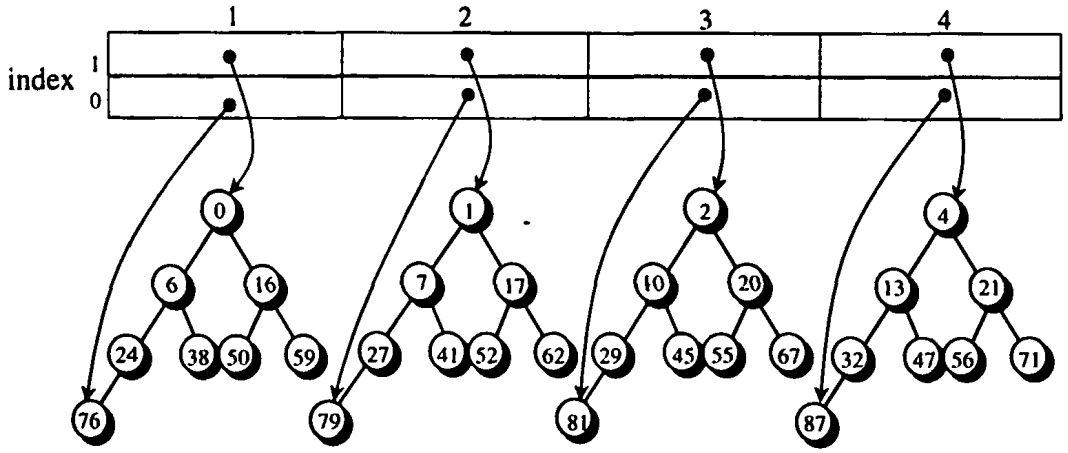
1. Parallel-Sort(*NEW*);
2. $i = 1$;
3. *NEXT* = *NEW*;
4. Εφόσον το i-οστό μικρότερο στοιχείο του σωρού #1 < *NEXT*[1] επανέλαβε: $i = i + 1$;
5. $i = i - 1$;
6. Εφόσον υπάρχει στοιχείο της ακολουθίας *NEW* που δεν έχει εισαχθεί επανέλαβε:
   6.1 Για $n = 1$ μέχρι $n \leq k$ εκτέλεσε παράλληλα:
   R[n] = *MH*[n][i]; (το i-οστό μικρότερο στοιχείο του σωρού #n)
   6.2 *M* = Parallel-Merge(*R*, *NEXT*);
   6.3 Για $n = 1$ μέχρι $n \leq k$ εκτέλεσε παράλληλα:
   Replace(το στοιχείο *R*[n] του σωρού #n με το στοιχείο *M*[n]);
   6.4 Για $n = 1$ μέχρι $n \leq k$ εκτέλεσε: *NEXT*[n] = *M*[k+n];
   6.5 $i = i + 1$;
7. Για $n = 1$ μέχρι $n \leq k$ εκτέλεσε παράλληλα:
   Insert(*NEXT*[n] στο σωρό #n στη θέση i μετακινώντας μια θέση δεξιά τα υπόλοιπα στοιχεία);

Από τη μελέτη της διαδικασίας της εισαγωγής, όπως περιγράφηκε παραπάνω, προκύπτει ότι ο χρόνος που απαιτείται για την ολοκλήρωσή της είναι της τάξης $O((n/k)\log\log k)$. Σημειώνεται ότι το βήμα 6 του αλγορίθμου επαναλαμβάνεται στη χειρίστη περίπτωση n/k φορές, όσο δηλαδή είναι και το πλήθος των στοιχείων του κάθε σωρού.

Έστω ότι στη δομή k-M-heap του Σχήματος 7.3, όπου $k = 4$ πρόκειται να πραγματοποιηθεί η διαδικασία της εισαγωγής και ότι η ακολουθία των 4 στοιχείων που πρόκειται να εισαχθούν είναι η εξής:

$$NEW = \{8, 19, 31, 46\}$$

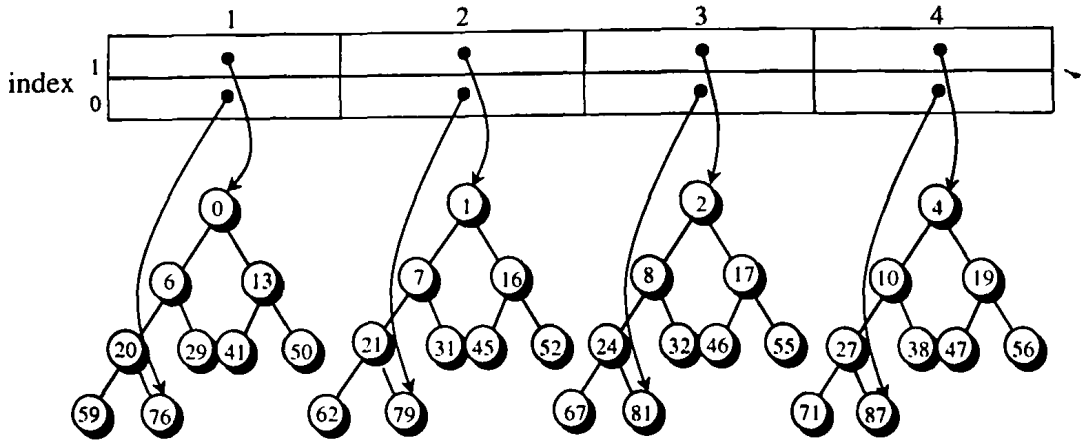
Σύμφωνα με τον αλγόριθμο, αρχικά γίνεται η συγχώνευση των ακολουθιών {6, 7, 10, 13} και {8, 19, 31, 46} οπότε στη θέση των στοιχείων της πρώτης ακολουθίας τοποθετούνται τα στοιχεία {6, 7, 8, 10}. Στη συνέχεια συγχωνεύονται οι ακολουθίες {16, 17, 20, 21} και {13, 19, 31, 46} οπότε στη θέση των στοιχείων της πρώτης ακολουθίας τοποθετούνται τα 13, 16, 17 και 19 αντίστοιχα. Ακολουθεί η συγχώνευση των ακολουθιών {24, 27, 29, 32} και {20, 21, 31, 46} οπότε στη θέση των στοιχείων 24, 27, 29 και 32 τοποθετούνται τα 20, 21, 24 και 27 αντίστοιχα. Η συγχώνευση των {38, 41, 45, 47} και {29, 31, 32, 46} οδηγεί στην εισαγωγή των 29, 31, 32 και 38 στη θέση των στοιχείων της πρώτης ακολουθίας. Τέλος, συγχωνεύονται οι ακολουθίες {50, 52, 55, 56} και {41, 45, 46, 47} και έτσι τα στοιχεία 41, 45, 46 και 47 εισάγονται στη θέση των στοιχείων της πρώτης ακολουθίας.



Σχήμα 7.3

Αφού έχουν εισαχθεί όλα τα στοιχεία της ακολουθίας *NEW*, εισάγονται τα 50, 52, 55 και 56 και η διαδικασία της εισαγωγής ολοκληρώνεται. Το αποτέλεσμα της διαδικασίας που περιγράφηκε είναι η δομή που παρουσιάζεται στο Σχήμα 7.4.

Όπως ίσως θα έχει ήδη παρατηρηθεί, οι ιδιότητες της δομής του σωρού δεν χρησιμοποιούνται στους παραπάνω αλγορίθμους. Έτσι, προκειμένου να απαλλαγεί κανείς από μετακινήσεις στοιχείων στους πίνακες, θα μπορούσε να χρησιμοποιήσει συνδεδεμένες λίστες για την αναπαράσταση των σωρών. Όμως σε μια τέτοια περίπτωση, κατά τη διαδικασία εντοπισμού του i-οστού στοιχείου που υπάρχει σε κάθε έναν από τους σωρούς, απαιτείται διάσχιση της λίστας που αναπαριστά το σωρό.



Σχήμα 7.4

Παράλληλος Αλγόριθμος για τη Δομή P-tree

Η δομή δεδομένων που περιγράφεται στη συνέχεια και ονομάζεται *k-P*-tree, βασίζεται, προφανώς, στη δομή *P*-tree. Αποτελεί ουσιαστικά επέκταση της δομής του *P*-tree, αφού η διαφορά είναι ότι κάθε κόμβος της δομής περιλαμβάνει *k* ταξινομημένα στοιχεία. Έτσι, αν *N* είναι ο συνολικός αριθμός των στοιχείων που βρίσκονται αποθηκευμένα στη δομή, τότε το *P*-tree αποτελείται από *N/k* κόμβους. Στη συνέχεια αποδεικνύεται ότι με $4k$ επεξεργαστές και σε CREW-PRAM μοντέλο ισχύουν τα ακόλουθα:

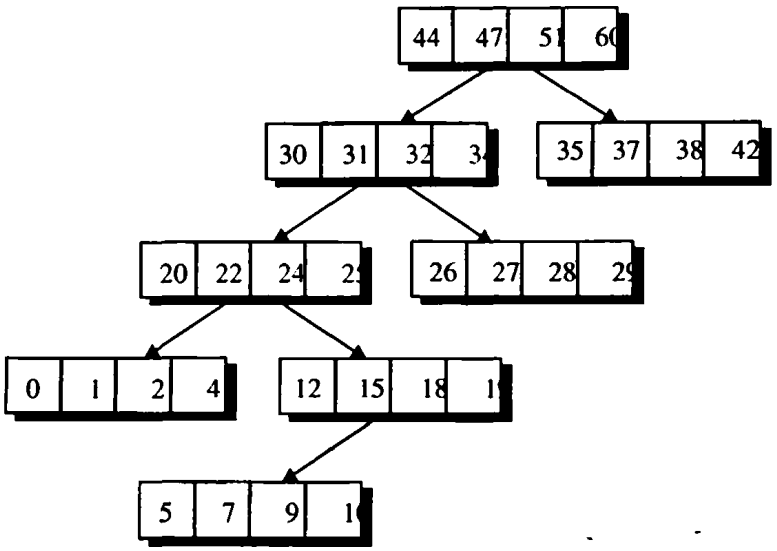
- *k* στοιχεία εισάγονται σε χρόνο $O((n/k)\log\log k)$
- Τα *k* στοιχεία με τη μικρότερη πληροφορία διαγράφονται σε χρόνο $O(1)$

Η δομή *k-P*-tree είναι ουσιαστικά ένα *P*-tree που κάθε ένας από τους κόμβους του περιέχει *k* ταξινομημένα στοιχεία, όπως προαναφέρθηκε. Επιπλέον, οι κόμβοι είναι συνδεδεμένοι με έναν τρόπο ο οποίος στηρίζεται στις ιδιότητες της διάταξης των στοιχείων της δομής του *P*-tree. Δηλαδή, προκειμένου να εντοπίζονται τα *k* μικρότερα στοιχεία σε $O(1)$ παράλληλο χρόνο, τα στοιχεία αποθηκεύονται κατά *k*-άδες στους κόμβους, κατά τέτοιο τρόπο ώστε να ισχύουν οι ιδιότητες του *P*-tree, δηλαδή, τα στοιχεία κάθε κόμβου είναι μικρότερα από όλα τα στοιχεία που βρίσκονται στο υποδέντρο με ρίζα τον συγκεκριμένο κόμβο. Με αυτή την οργάνωση τα *k* μικρότερα στοιχεία εντοπίζονται στον τελευταίο κόμβο του πιο αριστερού μονοπατιού.

Πιο συγκεκριμένα, αν E_{Pi} είναι η ακολουθία των στοιχείων που περιέχει ο κόμβος P_i και E_{P1} είναι η ακολουθία των στοιχείων της ρίζας του δέντρου, η δομή *k-P*-tree είναι ένα δυαδικό δέντρο το οποίο είναι είτε κενό είτε αποτελείται από ένα αριστερό μονοπάτι, μήκους έστω *M*, από κόμβους οι οποίοι περιέχουν ακολουθίες *k* στοιχείων που αν συνενωθούν, $(E_{P1}E_{P2}E_{P3}...E_{PM}$, όπου $P_1, P_2, ..., P_M$ είναι οι κόμβοι του πιο αριστερού μονοπατιού με τη σειρά που διασχίζονται αν ξεκινήσει κανείς από τη ρίζα) ξεκινώντας από τη ρίζα, δίνουν φθίνουσα ακολουθία στοιχείων. Επιπλέον, κάθε ένας από τους κόμβους του πιο αριστερού μονοπατιού, εκτός από τον τελευταίο, έχει σαν δεξί υποδέντρο ένα *k-P*-tree. Η ακολουθία με τα *k* μεγαλύτερα στοιχεία της δομής βρίσκεται πάντα στον κόμβο-ρίζα. Οι διάδοχοι κάθε κόμβου έχουν ακολουθίες από στοιχεία μικρότερα από τα στοιχεία του κόμβου αυτού και οι ακολουθίες του δεξιού υποδέντρου του περιέχουν στοιχεία που είναι μεγαλύτερα από όλα τα στοιχεία όλων των ακολουθιών του αριστερού υποδέντρου. Έτσι, δεν υπάρχει δεξί υποδέντρο σε κάποιον κόμβο,

αν δεν υπάρχει αριστερό, όπως ισχύει και για τη δομή του *P*-tree, και επιπλέον, όπως αναφέρθηκε παραπάνω, ο τερματικός κόμβος στο πιο αριστερό μονοπάτι περιέχει τα *k* μικρότερα στοιχεία που υπάρχουν στη δομή.

Το Σχήμα 7.5 παρουσιάζει ένα παράδειγμα της δομής *k-P*-tree που αποτελείται από 8 κόμβους και για την οποία ισχύει $k = 4$.



Σχήμα 7.5

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΔΟΜΗΣ *K-P*-TREE

Έστω E_{pi} η ταξινομημένη ακολουθία των *k* στοιχείων που βρίσκονται αποθηκευμένα σε έναν κόμβο *Pi* και έστω $P_{i.left}$ και $P_{i.right}$ το αριστερό και το δεξί παιδί αντίστοιχα του κόμβου P_i. Έτσι, αν P_1 είναι ο κόμβος-ρίζα και P_2 και P_3 είναι το αριστερό και το δεξί παιδί της ρίζας αντίστοιχα (έστω ότι υπάρχει δεξί παιδί) τότε η ακολουθία $E_{p1}E_{p3}E_{p2}$ είναι μια ταξινομημένη (φθίνουσα) ακολουθία στοιχείων. Η διαδικασία της εισαγωγής των *k* νέων στοιχείων πρέπει να γίνει με τέτοιο τρόπο ώστε να εξακολουθούν να ισχύουν οι ιδιότητες της δομής *k-P*-tree. Επομένως δεν είναι δυνατόν να χρησιμοποιηθεί η γνωστή διαδικασία της εισαγωγής που εφαρμόζεται σε μια δομή *P*-tree όπως αυτή έχει περιγραφεί σε προηγούμενο κεφάλαιο. Για τη διαδικασία της διαγραφής των *k* μικρότερων στοιχείων όμως μπορεί να χρησιμοποιηθεί η γνωστή διαδικασία της διαγραφής του μικρότερου στοιχείου, αφού τα *k* μικρότερα στοιχεία βρίσκονται, όπως και στη δομή του *P*-tree στον τελευταίο κόμβο του πιο αριστερού μονοπατιού. Προφανώς, για τη δημιουργία της δομής χρησιμοποιείται η διαδικασία της εισαγωγής.

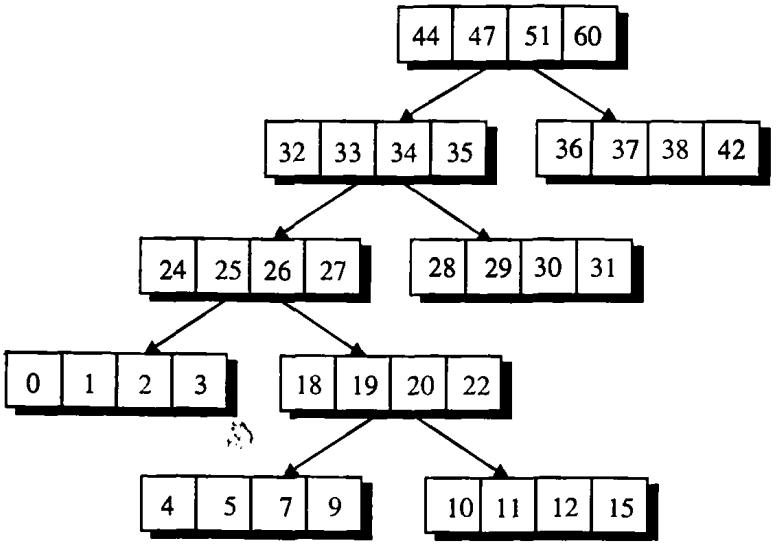
➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΕΙΣΑΓΩΓΗΣ

Προκειμένου να εισαχθούν *k* νέα στοιχεία στη δομή, πρώτα ταξινομούνται και εισάγονται αρχικά σε ένα νέο κόμβο P_N και αποτελούν την ακολουθία E_{PN}. Σημειώνεται αρχικά ότι ακόμη και αν βρεθεί η κατάλληλη θέση εισαγωγής του νέου κόμβου P_N και συνδεθεί με τους κόμβους της δομής, η διαδικασία της εισαγωγής μπορεί να μην έχει ακόμη ολοκληρωθεί, αφού ταυτόχρονα πραγματοποιείται συγχώνευση μερικών ακολουθιών και αυτή η διαδικασία των συγχωνεύσεων υπάρχει περίπτωση να μην έχει ολοκληρωθεί.

Η ακολουθία E_{PN} των στοιχείων του κόμβου P_N συγχωνεύεται με τις ακολουθίες E_{PR}, $E_{PR.left}$ και $E_{NPR.right}$ των κόμβων P_R, $P_{R.left}$ και $P_{R.right}$, δηλαδή της ρίζας, του αριστερού και του δεξιού της παιδιού αντίστοιχα (αν το τελευταίο υπάρχει). Έστω ότι προκύπτει η αύξουσα ακολουθία $M[1...4k]$, η οποία αποτελείται από $4k$ στοιχεία. Τα k μεγαλύτερα στοιχεία της (τελευταίο τμήμα της ακολουθίας M) τοποθετούνται στον κόμβο-ρίζα, τα k μικρότερα (πρώτο τμήμα της ακολουθίας M) στο αριστερό παιδί της ρίζας και τα στοιχεία $M[4k-7]$, $M[4k-6]$, $M[4k-5]$ και $M[4k-4]$ (το προτελευταίο, τρίτο τμήμα της ακολουθίας M) τοποθετούνται στο δεξί παιδί της ρίζας. Αν δεν υπήρχε δεξί παιδί τότε δεξί παιδί γίνεται ο νέος κόμβος και περιέχει τα στοιχεία που προαναφέρθηκαν. Η διαδικασία επαναλαμβάνεται αναδρομικά για το αριστερό υποδέντρο της ρίζας θεωρώντας ότι τα στοιχεία που θα εισαχθούν και που αποτελούν την ακολουθία E_{PN}, είναι τα στοιχεία $M[k+1]$, $M[k+2]$, $M[k+3]$ και $M[k+4]$ (το δεύτερο τμήμα της ακολουθίας M).

Αν κατά τη διάρκεια της διαδικασίας βρεθεί κόμβος του οποίου τα στοιχεία είναι όλα μικρότερα από τα στοιχεία που πρέπει να εισαχθούν (ακολουθία E_{PN}) η διαδικασία συνεχίζεται για το δεξί υποδέντρο του πατέρα του συγκεκριμένου κόμβου. Αν αυτό δεν υπάρχει τότε η εισαγωγή γίνεται σε αυτό το σημείο και η διαδικασία τερματίζεται. Ακόμη, η διαδικασία της εισαγωγής τερματίζεται όταν γίνει η συγχώνευση της νέας ακολουθίας με κάποιον κόμβο-φύλλο. Σε αυτή την περίπτωση η ακολουθία M αποτελείται από $2k$ στοιχεία. Τα k μεγαλύτερα από αυτά τοποθετούνται στο συγκεκριμένο κόμβο, ενώ τα k μικρότερα γίνονται αριστερό παιδί του κόμβου.

Στο Σχήμα 7.6 παρουσιάζεται το αποτέλεσμα της εισαγωγής των $k = 4$ στοιχείων της ακολουθίας $E_{PN} = \{3, 11, 33, 36\}$στη δομή του Σχήματος 7.5. Με βάση τον αλγόριθμο της εισαγωγής, τα νέα στοιχεία συγχωνεύονται αρχικά με τις ακολουθίες $E_{PR} = \{44, 47, 51, 60\}$, $E_{PR.left} = \{30, 31, 32, 34\}$ και $E_{PR.right} = \{35, 37, 38, 42\}$. Προκύπτει η ακολουθία $M = \{3, 11, 30, 31, 32, 33, 34, 35, 36, 37, 38, 42, 44, 47, 51, 60\}$, οπότε στη συνέχεια θα ισχύει $E_{PR} = \{44, 47, 51, 60\}$, $E_{PR.left} = \{3, 11, 30, 31\}$, $E_{PR.right} = \{36, 37, 38, 42\}$ και $E_{PN} = \{32, 33, 34, 35\}$. Η διαδικασία επαναλαμβάνεται θεωρώντας σαν κόμβο P_R το αριστερό παιδί της ρίζας και $P_{R.left}$ και $P_{R.right}$ το αριστερό και το δεξί παιδί του, αντίστοιχα. Όταν τεθεί P_R ο τελευταίος κόμβος του πιο αριστερού μονοπατιού θα περιέχει τα στοιχεία 0, 1, 2 και 3 τα οποία είναι όλα μικρότερα από τα στοιχεία της ακολουθίας $E_{PN} = \{4, 11, 12, 15\}$ οπότε η διαδικασία συνεχίζεται στο δεξί υποδέντρο του πατέρα του κόμβου P_R, ο οποίος περιέχει την ακολουθία $\{18, 19, 20, 22\}$.

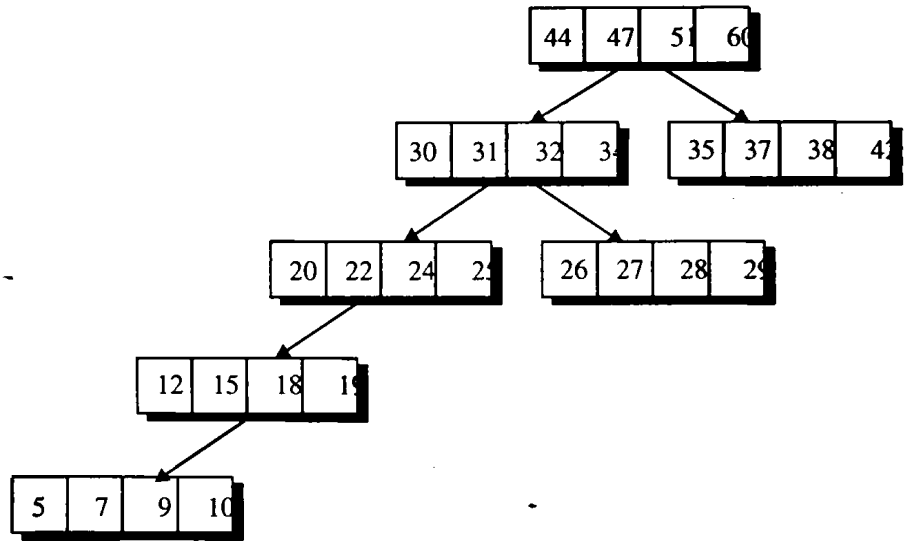


Σχήμα 7.6

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

Ο εντοπισμός του κόμβου με τις k μικρότερες πληροφορίες μπορεί να πραγματοποιηθεί σε σταθερό χρόνο, εφόσον υπάρχει μία επιπλέον μεταβλητή τύπου δείκτη που αντιστοιχεί στον τερματικό κόμβο του πιο αριστερού μονοπατιού. Μετά την αφαίρεση αυτού του κόμβου, το τελευταίο δεξί υποδέντρο, δηλαδή αυτό που αντιστοιχεί στον πατέρα του κόμβου που αφαιρέθηκε, αν δεν είναι κενό, παίρνει τη θέση του κόμβου που αφαιρέθηκε, σαν αριστερό υποδέντρο. Δηλαδή η διαδικασία της διαγραφής των k μικρότερων στοιχείων από τη δομή k-P-tree είναι ίδια με η διαδικασία της διαγραφής του μικρότερου στοιχείου από μια δομή P-tree.

Αν πραγματοποιηθεί η διαδικασία της διαγραφής των $k = 4$ μικρότερων στοιχείων στη δομή του Σχήματος 7.5, θα διαγραφούν τα στοιχεία {0, 1, 2, 4} και θα προκύψει η δομή του Σχήματος 7.7.



Σχήμα 7.7

Παράλληλος Αλγόριθμος για τη Δομή IP-tree

Η δομή *IP*-tree αποτελείται, όπως περιγράφηκε σε προηγούμενο κεφάλαιο, από μια δομή *P*-tree και μια δομή που ονομάζεται *I*-list. Υπενθυμίζεται ότι τα στοιχεία της δομής *I*-list είναι δείκτες σε κόμβους της δομής του *P*-tree.

Χρησιμοποιείται η τεχνική Event Horizon και επομένως κάθε φορά που η μικρότερη τιμή που υπάρχει στη δομή *IP*-tree βρίσκεται στη δευτερεύουσα δομή, η τελευταία ταξινομείται και τα στοιχεία της επανατοποθετούνται στο *P*-tree. Στη δομή *IP*-tree μπορεί κανείς να εκμεταλλευτεί το γεγονός ότι τα στοιχεία της δευτερεύουσας δομής που εισάγονται διαδοχικά είναι ταξινομημένα κατά φθίνουσα τάξη καθώς και ότι η δομή *I*-list καθορίζει τμήματα του δέντρου (υποδέντρα). Έτσι, για την εισαγωγή ενός στοιχείου δεν είναι απαραίτητο η αναζήτηση για την κατάλληλη θέση να ξεκινήσει από τη ρίζα, αλλά από ένα συγκεκριμένο υποδέντρο που εντοπίζεται με τη βοήθεια της δομής *I*-list.
Η δομή δεδομένων που περιγράφεται στη συνέχεια και ονομάζεται *k-IP*-tree, βασίζεται, προφανώς, στη δομή *IP*-tree. Αποτελεί ουσιαστικά επέκταση της δομής του *IP*-tree κατά τον τρόπο με τον οποίο πραγματοποιήθηκε και η επέκταση της δομής του *P*-tree και

δημιουργήθηκε η δομή *k-P*-tree. Έτσι, η διαφορά της δομής *k-IP*-tree σε σχέση με τη δομή *IP*-tree είναι ότι κάθε κόμβος της δομής περιλαμβάνει *k* ταξινομημένα στοιχεία. Έτσι, αν *N* είναι ο συνολικός αριθμός των στοιχείων που βρίσκονται αποθηκευμένα στη δομή, τότε το *P*-tree αποτελείται από *N/k* κόμβους. Στη συνέχεια αποδεικνύεται ότι με $4k$ επεξεργαστές και σε CREW-PRAM μοντέλο ισχύουν τα ακόλουθα:

- *k* στοιχεία εισάγονται σε χρόνο $O(n \log\log k / k\, m)$
- Τα *k* στοιχεία με τη μικρότερη πληροφορία διαγράφονται σε χρόνο O(1)

όπου ο αριθμός *m* είναι πολλαπλάσιος του *k*. Η δομή *k-IP*-tree είναι ουσιαστικά μια δομή *IP*-tree που κάθε ένας από τους κόμβους της περιέχει *k* ταξινομημένα στοιχεία, όπως προαναφέρθηκε. Επιπλέον, οι κόμβοι είναι συνδεδεμένοι με έναν τρόπο ο οποίος στηρίζεται στις ιδιότητες της διάταξης των στοιχείων της δομής του *P*-tree. Συγκεκριμένα, το *P*-tree της δομής είναι ουσιαστικά μια δομή *k-P*-tree, η οποία περιγράφηκε παραπάνω. Έτσι, προκειμένου να εντοπίζονται τα *k* μικρότερα στοιχεία σε O(1) παράλληλο χρόνο, τα στοιχεία αποθηκεύονται κατά *k*-άδες στους κόμβους, κατά τέτοιο τρόπο ώστε τα στοιχεία κάθε κόμβου να είναι μικρότερα από όλα τα στοιχεία που βρίσκονται στο υποδέντρο με ρίζα τον συγκεκριμένο κόμβο. Με αυτή την οργάνωση τα *k* μικρότερα στοιχεία εντοπίζονται στον τελευταίο κόμβο του πιο αριστερού μονοπατιού.

Με τη δομή *k-IP*-tree προφανώς χρησιμοποιείται η τεχνική Event Horizon και επομένως κάθε φορά που οι *k* μικρότερες τιμές που υπάρχουν στη δομή βρίσκονται στη δευτερεύουσα δομή, η τελευταία ταξινομείται και τα στοιχεία της επανατοποθετούνται στο *P*-tree. Η διαδικασία της συγχώνευσης πραγματοποιείται με τον τρόπο που γίνεται η συγχώνευση σε μια δομή *IP*-tree.

Συγκεκριμένα, έστω ότι πρέπει να συγχωνευθεί με τη δομή του *P*-tree ένα σύνολο στοιχείων, το οποίο είναι πολλαπλάσιο του *k*. Η συγχώνευση γίνεται εισάγοντας διαδοχικά τμήματα της ακολουθίας μήκους *k* κάθε φορά. Η διαδικασία της συγχώνευσης-εισαγωγής περιγράφεται στη συνέχεια.

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΔΟΜΗΣ *K-IP*-TREE

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΣΥΓΧΩΝΕΥΣΗΣ-ΕΙΣΑΓΩΓΗΣ

Κατά τη διαδικασία της εισαγωγής μιας ακολουθίας *k* στοιχείων στη δομή του *P*-tree αναζητείται η κατάλληλη θέση διασχίζοντας το δέντρο ξεκινώντας από τη ρίζα. Χρησιμοποιώντας τη δομή *I*-list, για την εισαγωγή της ακολουθίας δεν είναι απαραίτητο η αναζήτηση για την κατάλληλη θέση να ξεκινήσει από τη ρίζα, αλλά από ένα συγκεκριμένο υποδέντρο. Αυτό οφείλεται στο γεγονός ότι είναι γνωστό ότι τα *k* προηγούμενα στοιχεία που τοποθετήθηκαν στο *P*-tree είχαν όλα μεγαλύτερες τιμές από τα νέα *k* στοιχεία (αφού η δευτερεύουσα δομή είχε ταξινομηθεί) και παρέχεται έτσι η δυνατότητα να τοποθετούνται δείκτες σε συγκεκριμένους κόμβους στο μονοπάτι που ακολούθησε η προηγούμενη ακολουθία προκειμένου να διευκολυνθεί η εισαγωγή της νέας. Υπενθυμίζεται ότι η διαδικασία που περιγράφεται πραγματοποιείται όταν διαπιστωθεί ότι δεν περιέχονται όλα τα *k* μικρότερα στοιχεία στο *P*-tree αλλά υπάρχει τουλάχιστον ένα από αυτά στη δευτερεύουσα δομή.

Αφού εντοπισθεί το κατάλληλο υποδέντρο, η διαδικασία της εισαγωγής ολοκληρώνεται στο συγκεκριμένο *P*-tree σύμφωνα με τον τρόπο που περιγράφηκε παραπάνω για τη δομή *k-P*-tree.

Για παράδειγμα, έστω E_a η ακολουθία των k στοιχείων της οποίας η εισαγωγή μόλις ολοκληρώθηκε και E_r η τελευταία ακολουθία ενός κόμβου r του πιο αριστερού μονοπατιού με την οποία συγκρίθηκε και συγχωνεύθηκε η ακολουθία E_a προτού η αναζήτηση της κατάλληλης θέσης προχωρήσει σε δεξί υποδέντρο. Αν η νέα ακολουθία που πρέπει να εισαχθεί, έστω E_b, έχει όλα της τα στοιχεία μικρότερα, αρκεί η διαδικασία της εισαγωγής να ξεκινήσει από το υποδέντρο (P-tree) με ρίζα τον κόμβο r. Έτσι, είναι απαραίτητο να διατηρούνται δείκτες σε εκείνους τους κόμβους με τις ακολουθίες των οποίων συγκρίθηκε η ακολουθία E_a προτού η διαδικασία συνεχιστεί σε δεξί υποδέντρο. Οι κόμβοι αυτοί είναι οι γνωστοί I-κόμβοι. Ο πρώτος I-κόμβος είναι η ρίζα του P-tree ενώ ο τελευταίος είναι ο κόμβος που περιέχει τις k μικρότερες πληροφορίες που υπάρχουν στο P-tree, ο τελευταίος δηλαδή κόμβος του πιο αριστερού μονοπατιού του δέντρου.

Αν n είναι το πλήθος των στοιχείων που υπάρχουν στη δομή, τότε το P-tree αποτελείται από n/k κόμβους. Έτσι, για τη διαδικασία της εισαγωγής k στοιχείων στη δομή του P-tree, απαιτείται πολυπλοκότητα χρόνου τάξης ίση με αυτή που απαιτεί η αντίστοιχη διαδικασία για τη δομή του k-P-tree, δηλαδή $O((n/k)\log\log k)$ με $4k$ επεξεργαστές. Η διαδικασία της εισαγωγής k νέων στοιχείων στη δομή πραγματοποιείται σε σταθερό χρόνο αφού αυτά εισάγονται στη δευτερεύουσα δομή, στη δομή της λίστας. Είναι γνωστό ακόμη ότι η διαδικασία της ταξινόμησης m γεγονότων απαιτεί χρόνο της τάξης $O(m\log m)$. Έτσι, αφού η συγχώνευση πραγματοποιείται για κάθε m, έστω, γεγονότα, η σχέση που ακολουθεί αντιστοιχεί στο χρόνο που απαιτείται για την εισαγωγή k γεγονότων στη δομή k-IP-tree:

$$Ti = C_1 + C_2 \log_2(m) + C_3\, n \log\log k / (k\, m) \qquad (1)$$

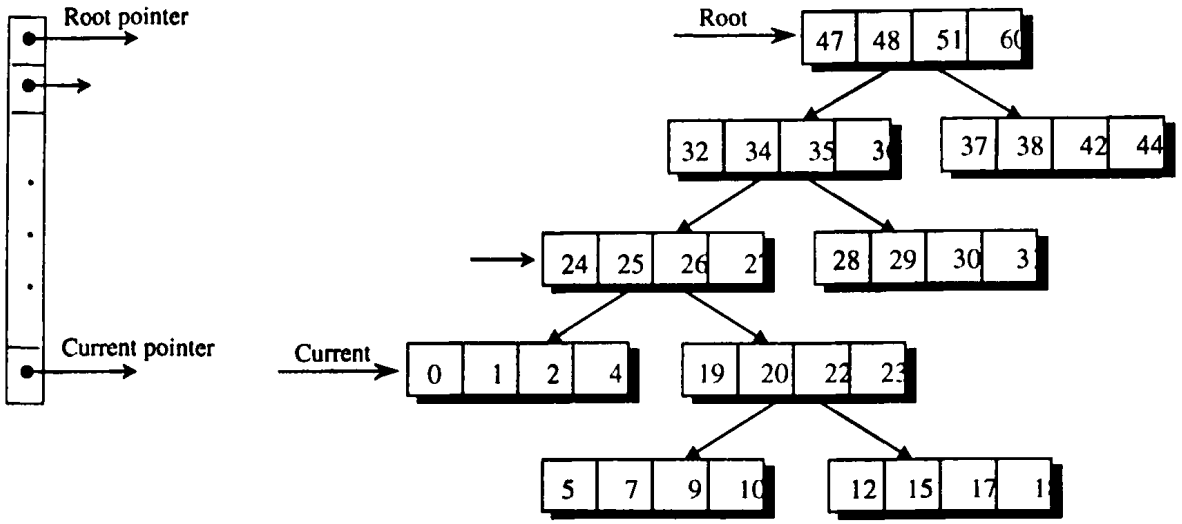
όπου εδώ το C_1 αντιστοιχεί στο σταθερό χρονικό κόστος για την εισαγωγή στη δευτερεύουσα δομή k γεγονότων, το C_2 αντιστοιχεί στο χρονικό κόστος για την ταξινόμηση m γεγονότων, και τέλος, το C_3 αναπαριστά το κόστος από τη συγχώνευση (εισαγωγή) m γεγονότων της δευτερεύουσας λίστας με n γεγονότα της κύριας δομής. Προφανώς το m είναι πολλαπλάσιο του k.

Έστω ότι το P-tree της δομής είναι αυτό του Σχήματος 7.5, όπου $k = 4$, και ότι η δευτερεύουσα δομή περιέχει τα στοιχεία {48, 36, 23, 17, 16, 13, 8, 3}. Προφανώς τα 4 μικρότερα στοιχεία δεν βρίσκονται όλα στη δομή του P-tree. Έτσι η διαδικασία της συγχώνευσης αρχικοποιείται με την εισαγωγή των στοιχείων {17, 23, 36, 48} σύμφωνα με τον τρόπο που περιγράφηκε παραπάνω κατά την περιγραφή της διαδικασίας της εισαγωγής στη δομή k-P-tree. Το αποτέλεσμα είναι η δομή που παρουσιάζεται στο Σχήμα 7.8. Όπως φαίνεται στο Σχήμα, υπάρχει ένας νέος I-κόμβος. Έτσι, η εισαγωγή των στοιχείων {3, 8, 13, 16} πραγματοποιείται με τη βοήθεια αυτού του κόμβου και η διαδικασία της εισαγωγής δεν ξεκινά από τη ρίζα αλλά από αυτό το συγκεκριμένο κόμβο. Τελικά το αποτέλεσμα της διαδικασίας της συγχώνευσης είναι το P-tree που παρουσιάζεται στο Σχήμα 7.9.

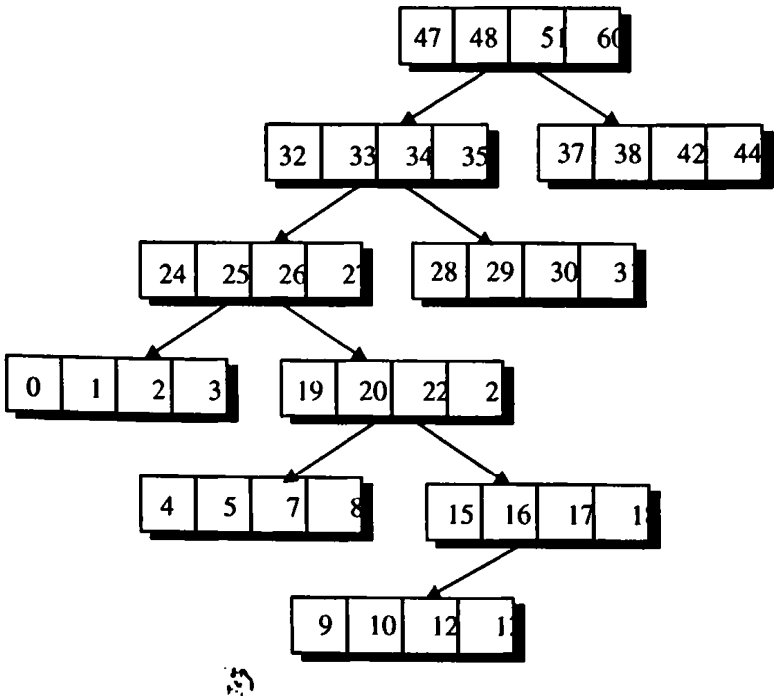
➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

Ο εντοπισμός του κόμβου με τις k μικρότερες πληροφορίες μπορεί να πραγματοποιηθεί σε σταθερό χρόνο, εφόσον υπάρχει μία επιπλέον μεταβλητή τύπου δείκτη που αντιστοιχεί στον τερματικό κόμβο του πιο αριστερού μονοπατιού. Μετά την αφαίρεση αυτού του κόμβου, πραγματοποιείται η ίδια διαδικασία που ισχύει και για τη δομή k-P-tree. Έτσι, το τελευταίο δεξί υποδέντρο, δηλαδή αυτό που αντιστοιχεί στον πατέρα του κόμβου που αφαιρέθηκε, αν δεν είναι κενό, παίρνει τη θέση του κόμβου που αφαιρέθηκε, σαν αριστερό υποδέντρο. Δηλαδή η

διαδικασία της διαγραφής των k μικρότερων στοιχείων από τη δομή k-IP-tree είναι ίδια με η διαδικασία της διαγραφής του μικρότερου στοιχείου από μια απλή δομή P-tree και συνεπώς ολοκληρώνεται σε σταθερό χρόνο.



Σχήμα 7.8



Σχήμα 7.9

Παράλληλος Αλγόριθμος για τη Δομή MP-tree

Η δομή δεδομένων που περιγράφεται στη συνέχεια και ονομάζεται *k-MP*-tree, βασίζεται στη δομή *MP*-tree. Η διαφορά είναι ότι η δομή *k-MP*-tree επιτρέπει την εισαγωγή *k* στοιχείων καθώς και τη διαγραφή των *k* μικρότερων στοιχείων που υπάρχουν στη δομή. Η τελευταία πραγματοποιείται εύκολα αν τα *k* μικρότερα στοιχεία βρίσκονται αποθηκευμένα σε *k* διακεκριμένα *P*-trees, κάτι που επιτυγχάνεται από τον τρόπο δημιουργίας της δομής και αποτελεί τη βασική της ιδιότητα. Έτσι, αν *N* είναι ο συνολικός αριθμός των στοιχείων που βρίσκονται αποθηκευμένα στη δομή και *k* το πλήθος των *P*-trees που την αποτελούν, τότε κάθε *P*-tree περιέχει *N/k* στοιχεία. Η διαδικασία της εισαγωγής των k νέων στοιχείων πραγματοποιείται με τέτοιο τρόπο ώστε τα *P*-trees να εξακολουθούν να αποτελούνται από ισάριθμα στοιχεία, όπως ισχύει κατά τη δημιουργία της δομής, αλλά και χωρίς να διαταράσσεται η ιδιότητα που περιγράφηκε παραπάνω. Προκειμένου να είναι αποδοτική η διαδικασία τροποποιείται η δομή MP-tree με την πρόσθεση μιας ακόμη μεταβλητής δείκτη *left_leaf* σε κάθε κόμβο. Συγκεκριμένα, η μεταβλητή αυτή δείχνει στο πιο αριστερό φύλλο του δεξιού υποδέντρου του κόμβου, αν ο κόμβος έχει δεξί υποδέντρο. Σημειώνεται ακόμη ότι η δομή *I*-heap που χρησιμοποιείται από τη δομή *MP*-tree, όπως περιγράφηκε σε προηγούμενο κεφάλαιο, δεν χρησιμοποιείται από τη δομή *k-MP*-tree, αφού δεν χρειάζεται να είναι γνωστό το *P*-tree που περιέχει τη μικρότερη πληροφορία που υπάρχει στη δομή. Ένα παράδειγμα μιας δομής *k-MP*-tree, όπου $k = 4$, παρουσιάζεται στο Σχήμα 7.10. Στη συνέχεια αποδεικνύεται ότι με *k* επεξεργαστές και σε CREW-PRAM μοντέλο ισχύουν τα ακόλουθα:

- *k* στοιχεία εισάγονται σε χρόνο $O((N/k)\log\log k)$
- Τα *k* στοιχεία με τη μικρότερη πληροφορία διαγράφονται σε χρόνο O(1)

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΔΟΜΗΣ *K-MP*-TREE

Έστω ότι η δομή αποτελείται από τα *N* στοιχεία της ακολουθίας *X*, τα οποία, κατά τη δημιουργία της, μοιράζονται σε *k* *P*-trees. Κατά τη διαδικασία της διαγραφής επιστρέφεται ένας μονοδιάστατος πίνακας *MIN*[*k*] που περιέχει τα *k* μικρότερα στοιχεία της δομής, ενώ κατά την εισαγωγή χρησιμοποιείται ένας επίσης μονοδιάστατος πίνακας *NEW*[*k*] ο οποίος περιέχει τα *k* νέα στοιχεία που πρόκειται να εισαχθούν.

➢ Η ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΔΟΜΗΣ

Για τη δημιουργία της δομής χρησιμοποιείται η διαδικασία Build_*k-MP*-tree, η οποία περιγράφεται αλγοριθμικά στη συνέχεια. Η ακολουθία *X* ταξινομείται κατά το βήμα 1 με χρήση της συνάρτησης Parallel-Sort σε O(log*k*) χρόνο σε CREW PRAM μοντέλο *k* επεξεργαστών. Αφού πραγματοποιηθεί η ταξινόμηση, κάθε ένας από τους επεξεργαστές εισάγει ακολουθιακά μια ακολουθία *N/k* στοιχείων στο αντίστοιχο *P*-tree ως εξής: ο πρώτος επεξεργαστής εισάγει στο πρώτο από τα *P*-trees τα πρώτα *N/k* στοιχεία της ταξινομημένης ακολουθίας *X*, ο δεύτερος επεξεργαστής εισάγει ακολουθιακά στο δεύτερο από τα *P*-trees το δεύτερο τμήμα *N/k* στοιχείων της ακολουθίας *X* κτλ. Για την ακολουθιακή εισαγωγή των στοιχείων, που πραγματοποιεί κάθε επεξεργαστής, εκτελείται η γνωστή διαδικασία της εισαγωγής ενός στοιχείου σε μια δομή *P*-tree. Σημειώνεται ακόμη ότι κάθε υποακολουθία μήκους *N/k* που πρέπει να εισαχθεί σε κάθε ένα από τα *P*-trees, τροποποιείται κατάλληλα πριν

την εισαγωγή των στοιχείων, κατά το βήμα 3.1 του αλγορίθμου, αφού η διαδοχική εισαγωγή ταξινομημένων στοιχείων οδηγεί σε δομή *P*-tree η οποία έχει τη μορφή συνδεδεμένης λίστας.

Build_*k*-*MP*-tree(*X*):

4. Parallel-Sort(*X*);
5. for $i = 1$ to k do in parallel:
   2.1 $n = i$;
   2.2 $m = 1$;
   2.3 while $n \leq N$
      2.3.1 $MP[i][m] = X[n]$;
      2.3.2 $n = n + k$;
      2.3.3 $m = m + 1$;
3. for $i = 1$ to k do in parallel:
   3.1 Permute(*MP*[*i*]);
   3.2 for $n = 1$ to N / k Insert(*MP*[*i*][*n*] at *P*-tree #*i*);

Σε κάθε γραμμή του πίνακα δύο διαστάσεων $MP[k][N/k]$ που χρησιμοποιεί η παραπάνω συνάρτηση, αποθηκεύονται τα N/k στοιχεία που αντιστοιχούν σε κάθε ένα από τα k *P*-trees.

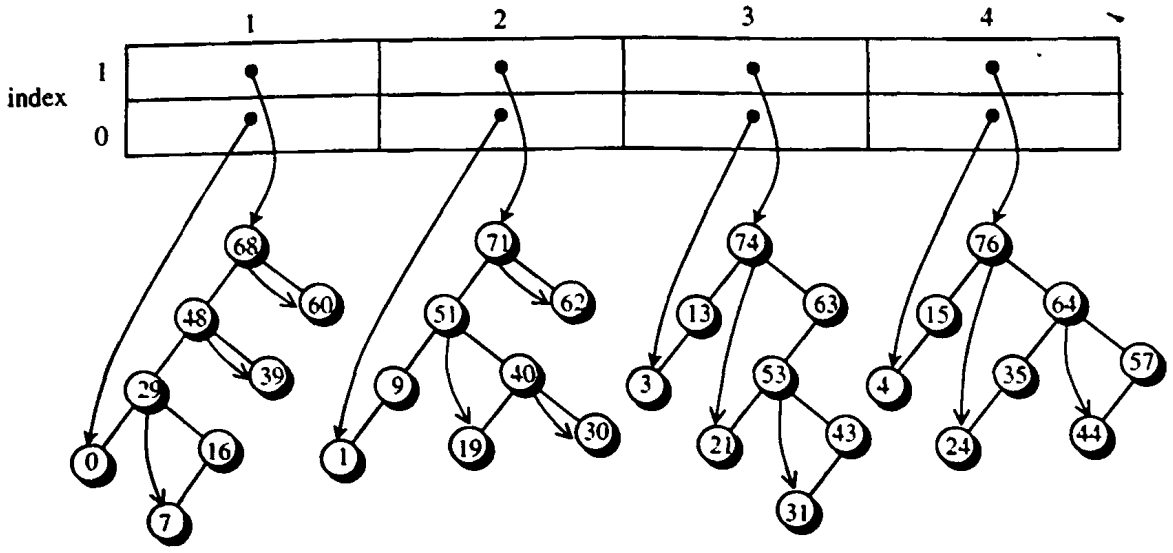
➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

Για την πραγματοποίηση της διαδικασίας της διαγραφής των k μικρότερων στοιχείων (Delete_*k*-*MP*-tree) χρησιμοποιείται η συνάρτηση Delete_minimum, δηλαδή η γνωστή διαδικασία της διαγραφής του μικρότερου στοιχείου από μια δομή *P*-tree. Κάθε ένας από τους επεξεργαστές εκτελεί τη διαδικασία Delete_minimum σε κάθε ένα από τα *P*-trees, με αποτέλεσμα η Delete_*k*-*MP*-tree να επιστρέφει ένα διάνυσμα $MIN[k]$ που περιέχει τα k μικρότερα στοιχεία που υπάρχουν στη δομή *k*-*MP*-tree. Υπενθυμίζεται ότι τα k μικρότερα στοιχεία βρίσκονται σε k διακεκριμένα *P*-trees. Έτσι, η διαδικασία της διαγραφής αυτών των στοιχείων είναι δυνατόν να πραγματοποιηθεί σε σταθερό χρόνο. Ακολουθεί η αλγοριθμική περιγραφή της διαδικασίας της διαγραφής.

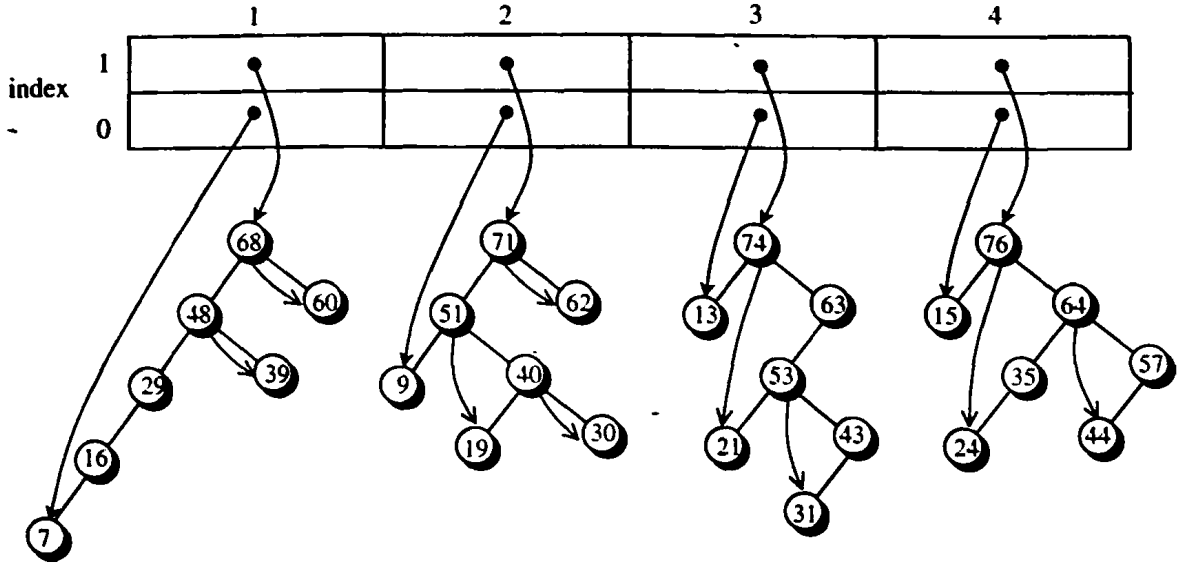
Delete_*k*-*MP*-tree:

1. for $i = 1; i \leq k; i = i + 1;$ do in parallel:
   $MIN[i]$ = Delete_minimum(από τη δομή *P*-tree #*i*);
6. return(*MIN*);

Στο Σχήμα 7.11 παρουσιάζεται το αποτέλεσμα της διαγραφής των $k = 4$ μικρότερων στοιχείων από τη δομή του Σχήματος 7.10.



Σχήμα 7.10



Σχήμα 7.11

➢ Η ΔΙΑΔΙΚΑΣΙΑ ΤΗΣ ΕΙΣΑΓΩΓΗΣ

Έστω *NEW* η ακολουθία των *k* νέων στοιχείων που πρόκειται να εισαχθούν. Όπως έχει ήδη αναφερθεί, είναι απαραίτητο η εισαγωγή των νέων στοιχείων να μην διαταράσσει την βασική ιδιότητα της δομής, σύμφωνα με την οποία τα *k* μικρότερα στοιχεία ανήκουν σε *k* διαφορετικά *P*-trees. Προκειμένου να επιτευχθεί αυτό, πραγματοποιείται συγχώνευση των στοιχείων που ήδη υπάρχουν στη δομή με τα *k* νέα. Έτσι, συγκρίνεται το μικρότερο στοιχείο της ακολουθίας *NEW*, έστω *a*, με τα στοιχεία του πρώτου *P*-tree, ξεκινώντας από το μικρότερο, μέχρι να βρεθεί το *i*-στό το οποίο είναι το πρώτο μεγαλύτερο στοιχείο από το *a*. Με αυτό τον τρόπο εντοπίζεται το σημείο από το οποίο ξεκινά η συγχώνευση, προκειμένου να αποφευχθούν άσκοπες συγκρίσεις. Συγκεκριμένα, αφού εντοπιστούν τα *i*-στά στοιχεία σε κάθε ένα από τα *P*-trees κατά το βήμα 6.1 του αλγόριθμου που παρουσιάζεται στη συνέχεια, σχηματίζεται από αυτά η ακολουθία *R*, η οποία συγχωνεύεται με την αρχική ακολουθία *NEW*. Προκύπτει έτσι, κατά το βήμα 6.2, μια νέα ακολουθία *M* που έχει μήκος 2*k*. Στη συνέχεια, τα *k* στοιχεία της *R*,

τα οποία ανήκουν σε διαφορετικά *P*-trees, αντικαθίστανται από τα *k* μικρότερα στοιχεία της *M*. Τα υπόλοιπα *k* στοιχεία της *M* αποτελούν στη συνέχεια τα στοιχεία της νέας ακολουθίας *NEW* και έτσι η διαδικασία επαναλαμβάνεται μέχρι να έχουν εισαχθεί όλα τα νέα στοιχεία, δηλαδή τα στοιχεία της αρχικής ακολουθίας *NEW*.

Σημειώνεται ότι η διαδικασία Find που χρησιμοποιείται στο βήμα 6.1, η οποία εκτελείται σε κάθε έναν από τους επεξεργαστές, είναι αυτή που εντοπίζει το *i*-στό μικρότερο στοιχείο που υπάρχει σε κάθε ένα από τα *P*-trees. Προκειμένου να πραγματοποιηθεί αυτό, η διαδικασία χρησιμοποιεί τις μεταβλητές (δείκτες) *left_leaf* των κόμβων.

Η παραπάνω διαδικασία περιγράφεται αλγοριθμικά ως εξής:

Insert_*k*-*MP*-tree(*NEW*):

1. Parallel-Sort(*NEW*);
2. $i = 1$;
3. *NEXT* = *NEW*;
4. Εφόσον το *i*-οστό μικρότερο στοιχείο του *P*-tree #1 < *NEXT*[1] επανέλαβε: $i = i + 1$;
5. $i = i - 1$;
6. Εφόσον υπάρχει στοιχείο της ακολουθίας *NEW* που δεν έχει εισαχθεί επανέλαβε:
   - 6.1 Για $n = 1$ μέχρι k εκτέλεσε παράλληλα:
     R[*n*] = Find (το *i*-οστό μικρότερο στοιχείο της δομής *P*-tree #*n*);
   - 6.1 *M* = Parallel-Merge(*R*, *NEXT*);
   - 6.2 Για $n = 1$ μέχρι k εκτέλεσε παράλληλα:
     Replace(το στοιχείο *R*[*n*] του *P*-tree #*n* με το στοιχείο *M*[*n*]);
   - 6.3 Για $n = 1$ μέχρι k εκτέλεσε: *NEXT*[*n*] = *M*[*k*+*n*];
   - 6.4 $i = i + 1$;
7. Για $n = 1$ μέχρι k εκτέλεσε παράλληλα: Insert(*NEXT*[*n*] στο *P*-tree #*n*);

Έστω *N* το πλήθος των στοιχείων που υπάρχουν στη δομή *k*-*MP*-tree οπότε το πλήθος των κόμβων που θα περιλαμβάνει κάθε ένα από τα *k* *P*-trees θα είναι *N*/*k*. Αφού ο αλγόριθμος της διαδικασίας της εισαγωγής απαιτεί χρόνο O(log*k*) για την ταξινόμηση, O(*N*/*k*) χρόνο για την πραγματοποίηση των βημάτων 4 και 7 και O((*N*/*k*)loglog*k*) για το βήμα 6, συνεπάγεται ότι η διαδικασία συνολικά ολοκληρώνεται με χρονική πολυπλοκότητα O((*N*/*k*)loglog*k*).

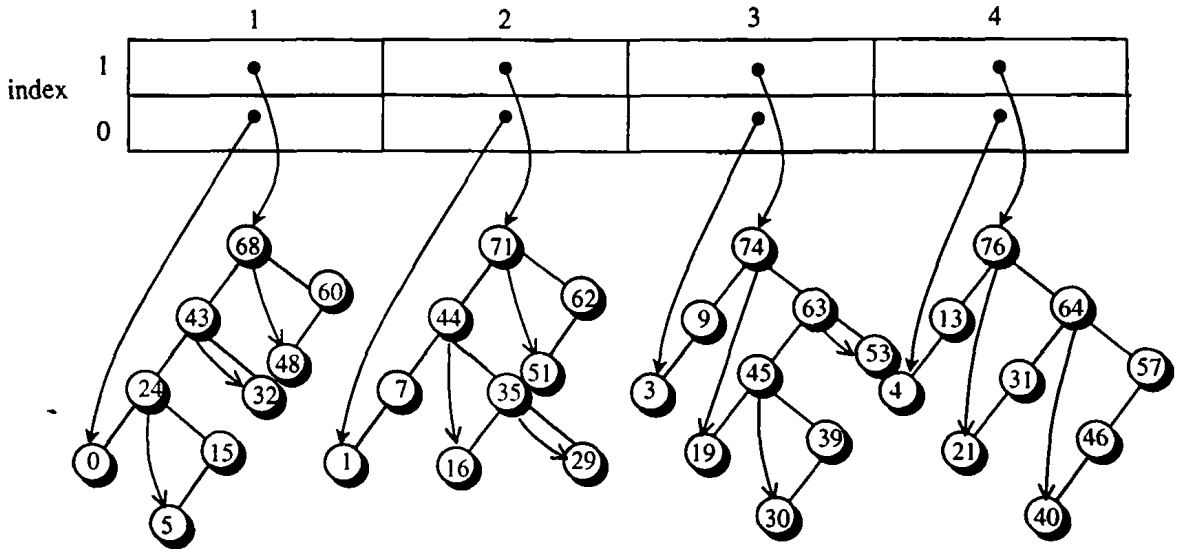
Έστω ότι στη δομή *k*-*MP*-tree του Σχήματος 7.10 πρόκειται να πραγματοποιηθεί η διαδικασία της εισαγωγής και ότι η ακολουθία των 4 ($k = 4$) στοιχείων που πρόκειται να εισαχθούν είναι η εξής:

$$NEW = \{5, 32, 45, 46\}$$

Σύμφωνα με τον αλγόριθμο, αρχικά γίνεται η συγχώνευση των ακολουθιών {0, 1, 3, 4} και {5, 32, 45, 46} οπότε τα στοιχεία της δομής δεν αλλάζουν αφού τα 4 μικρότερα είναι τα στοιχεία 0, 1, 3 και 4, που ήδη υπάρχουν. Στη συνέχεια συγχωνεύονται οι ακολουθίες {7, 9, 13, 15} και {5, 32, 45, 46} οπότε στη θέση των στοιχείων 7, 9, 13 και 15 τοποθετούνται τα 5, 7, 9 και 13 αντίστοιχα. Ακολουθεί η συγχώνευση των ακολουθιών {16, 19, 21, 24} και {15, 32, 45, 46} οπότε στη θέση των στοιχείων 16, 19, 21 και 24 τοποθετούνται τα 15, 16, 19 και 21 αντίχτοιχα. Μετά τη συγχώνευση των ακολουθιών {29, 30, 31, 35} και {24, 32, 45, 46} τα στοιχεία 24, 29, 30 και 31 τοποθετούνται στη δομή. Η συγχώνευση των {39, 40, 43, 44} και {32, 35, 45, 46}

έχει σαν αποτέλεσμα την εισαγωγή των στοιχείων 32, 35, 39 και 40. Τέλος, αφού συγχωνευθούν οι ακολουθίες {48, 51, 53, 57} και {43, 44, 45, 46} θα εισαχθούν στη δομή τα στοιχεία 43, 44, 45 και 46.

Αφού έχουν εισαχθεί όλα τα στοιχεία της ακολουθίας *NEW*, εισάγονται τα 48, 51, 53 και 57 και η διαδικασία της εισαγωγής ολοκληρώνεται. Το αποτέλεσμα της διαδικασίας που περιγράφηκε είναι η δομή που παρουσιάζεται στο Σχήμα 7.12.



Σχήμα 7.12

7.4 Συμπεράσματα

Η εισαγωγή του παραλληλισμού στην προσομοίωση διακριτών γεγονότων, σύμφωνα με όσα προηγήθηκαν μπορεί να επιφέρει αξιόλογη βελτίωση στην αποτελεσματικότητα ενός αλγόριθμου. Η χρήση περισσοτέρων του ενός επεξεργαστή είναι δυνατόν να επιταχύνει την ολοκλήρωση των διαδικασιών της εισαγωγής και της διαγραφής. Επιπλέον, η επεξεργασία περισσοτέρων του ενός γεγονότος κάθε φορά, δηλαδή η εισαγωγή ή η διαγραφή k γεγονότων ταυτόχρονα, καθιστά την προσομοίωση ενός συστήματος διακριτών γεγονότων πιο ρεαλιστική.

Στον Πίνακα 7.1 που ακολουθεί παρουσιάζεται συνοπτικά η πολυπλοκότητα χρόνου για τις διαδικασίες της εισαγωγής και της διαγραφής, καθώς επίσης και το πλήθος των επεξεργαστών που απαιτεί κάθε ένας από τους παράλληλους αλγορίθμους. Οι μεταβλητές N και M που εμφανίζονται στον πίνακα συμβολίζουν το πλήθος των στοιχείων, το πλήθος των σωρών ή των P-trees που υπάρχουν στη δομή M-heap ή MP-tree, αντίστοιχα. Η μεταβλητή m αντιστοιχεί στο πλήθος των στοιχείων που υπάρχουν στη δευτερεύουσα δομή ακριβώς πριν πραγματοποιηθεί η συγχώνευσή της με την κύρια. Τέλος, η μεταβλητή k προφανώς αντιστοιχεί στο πλήθος των στοιχείων που εισάγονται ή διαγράφονται ταυτόχρονα.

Αξίζει να σημειωθεί ότι οι αλγόριθμοι *M*-heap, *P*-tree, *MP*-tree και *IP*-tree πραγματοποιούν τη διαδικασία της διαγραφής των k μικρότερων στοιχείων σε σταθερό χρόνο. Στο τελευταίο κεφάλαιο πραγματοποιείται σύγκριση ανάμεσα στους ακολουθιακούς και τους παράλληλους αλγορίθμους που μελετήθηκαν.

Π. ΑΛΓΟΡΙΘΜΟΣ	k-ΕΙΣΑΓΩΓΗ	k-ΔΙΑΓΡΑΦΗ	ΕΠΕΞΕΡΓΑΣΤΕΣ
Leftist heap	$O(\log N)$	$O(\log(N/k)+\log\log k)$	k
Binomial heap	$O(\log(N+k)+\log^2 k)$	$O(k\log N)$	k
***n*-heap**	$O(\log N)$	$O(\log(N/k)+\log\log k)$	k
***M*-heap**	$O((N/k)\log\log k)$	$O(1)$	N
***P*-tree**	$O((N/k)\log\log k)$	$O(1)$	$4k$
***MP*-tree**	$O((N/k)\log\log k)$	$O(1)$	k
***IP*-tree**	$O((N/km)\log\log k+\log m)$	$O(1)$	$4k$

Πίνακας 7.1.

ΚΕΦΑΛΑΙΟ 8

ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ

8.1 Εισαγωγή

Η μελέτη που παρουσιάζεται στη συνέχεια έχει στόχο την εκτίμηση της αποτελεσματικότητας των αλγορίθμων με βάση πειραματικά αποτελέσματα. Η αποτελεσματικότητα ενός αλγόριθμου εκτιμάται με βάση τη μέση πολυπλοκότητα χρόνου που απαιτείται για την εκτέλεση των λειτουργιών της δημιουργίας ενός νέου γεγονότος με κατάλληλο χρόνο δρομολόγησης και της εισαγωγής του στο σύνολο γεγονότων καθώς επίσης και της διαγραφής του γεγονότος με τον μικρότερο χρόνο δρομολόγησης.

Για κάθε μία δομή προσομοίωσης του συνόλου γεγονότων και με δεδομένο το μέγεθος του και την κατανομή από την οποία προέρχονται οι χρόνοι δρομολόγησης των γεγονότων, η διαδικασία hold εκτελείται 16000 φορές. Άρα πραγματοποιούνται συνολικά 16000 εισαγωγές και διαγραφές ενώ ταυτόχρονα υπολογίζεται ο χρόνος που απαιτήθηκε, όπως ακριβώς περιγράφεται στη συνάρτηση main στο Κεφάλαιο 2. Μια παράμετρος που προφανώς καθορίζει την αποτελεσματικότητα ενός αλγόριθμου είναι το πλήθος των κόμβων γεγονότων, N, που αποτελούν το σύνολο γεγονότων. Οι τιμές του N που έχουν επιλεγεί είναι N = 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 και 16384 κόμβοι. Με την επιλογή της κατανομής που

χρησιμοποιείται κάθε φορά επιδιώκεται να προσδιοριστούν τα πλεονεκτήματα αλλά και μειονεκτήματα της κάθε δομής δεδομένων που χρησιμοποιήθηκε.

Στη συνέχεια παρουσιάζονται τα αποτελέσματα καθενός από τους αλγορίθμους που μελετήθηκαν. Ο χρόνος επεξεργασίας που απαιτεί κάθε ένας από τους αλγορίθμους σε συνάρτηση με το μέγεθος N του συνόλου γεγονότων παρουσιάζεται σε μορφή γραφικών παραστάσεων. Τα αποτελέσματα παρατίθενται επίσης και σε μορφή πίνακα για μεγαλύτερη ακρίβεια.

Κρίνεται όμως απαραίτητο να μελετηθεί και το πλήθος των συγκρίσεων μεταξύ των στοιχείων που πραγματοποιεί ο κάθε αλγόριθμος προκειμένου να ολοκληρωθούν οι διαδικασίες της εισαγωγής και της διαγραφής. Παρέχονται κατά αυτόν τον τρόπο ακριβέστερα αποτελέσματα αφού η αποτελεσματικότητα μερικών αλγορίθμων σχεδόν ταυτίζεται όταν αυτή κρίνεται με βάση το χρόνο που απαιτείται από τον επεξεργαστή προκειμένου να πραγματοποιήσει τις διαδικασίες που προαναφέρθηκαν.

8.2 Βασικές Δομές Δεδομένων

Η Δομή του Σωρού

Μελετώντας τα αποτελέσματα που προκύπτουν από την πειραματική μελέτη της δομής του σωρού, συμπεραίνει κανείς ότι η αποτελεσματικότητα της δομής είναι σχεδόν η ίδια είτε κατά τη στατική είτε κατά τη δυναμική αναπαράστασή της. Όπως ήταν αναμενόμενο, οι χρόνοι επεξεργασίας που απαιτεί η δομή του σωρού κατά τη δυναμική αναπαράσταση είναι ελάχιστα μεγαλύτεροι από εκείνους της στατικής αναπαράστασης, αφού στη δεύτερη περίπτωση δεν περιλαμβάνεται η χρήση δεικτών. Αυτό όμως δεν ισχύει και για το πλήθος των συγκρίσεων που πραγματοποιεί κάθε ένας από τους δυο αλγορίθμους, οι οποίοι είναι το ίδιο αποτελεσματικοί αν συγκριθούν με βάση αυτό το κριτήριο.

Για τους λόγους που αναφέρθηκαν παραπάνω, παρουσιάζονται στη συνέχεια τα αποτελέσματα από τη μελέτη της αποτελεσματικότητας της δομής του σωρού μόνο κατά τη στατική αναπαράστασή του. Έτσι, στα Σχήματα 8.1 και 8.2 παρουσιάζονται σε μορφή πινάκων οι χρόνοι επεξεργασίας και το πλήθος των συγκρίσεων, αντίστοιχα που απαιτεί ο αλγόριθμος για την πραγματοποίηση των διαδικασιών της εισαγωγής και της διαγραφής, ενώ στα Σχήματα 8.3 και 8.4 παρουσιάζονται τα ίδια αποτελέσματα σε μορφή γραφικών παραστάσεων. Αυτό που παρατηρεί κανείς είναι ότι η δομή είναι εξαιρετικά αποτελεσματική και μάλιστα ανεξάρτητα από την εκάστοτε κατανομή.

Όπως προέκυψε πειραματικά, η χρήση της τεχνικής Event Horizon, με οποιοδήποτε τρόπο και να εφαρμοστεί στη δομή του σωρού, δεν βελτιώνει την αποτελεσματικότητα της δομής. Παρατηρείται σχεδόν ίδια συμπεριφορά είτε χρησιμοποιείται η τεχνική Event Horizon είτε όχι, τόσο ως προς το συνολικό χρόνο επεξεργασίας όσο και ως προς τις συγκρίσεις και τις κατανομές. Ο τρόπος με τον οποίο εφαρμόζεται η τεχνική Event Horizon δεν είναι ακριβώς αυτός που παρουσιάστηκε κατά την περιγραφή της τεχνικής. Έτσι, η δευτερεύουσα δομή είναι επίσης μια δομή σωρού, η οποία δεν συγχωνεύεται με την κύρια όταν βρεθεί ότι περιέχει το μικρότερο στοιχείο, αλλά γίνεται κύρια δομή, δηλαδή η διαδικασία της διαγραφής

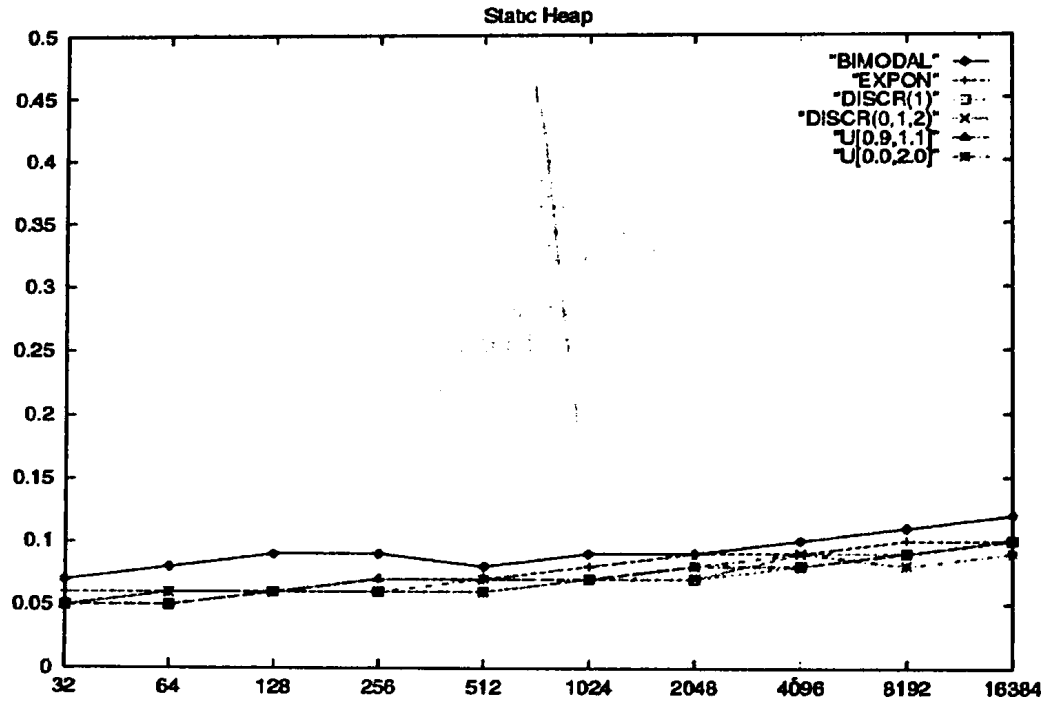
πραγματοποιείται στη συνέχεια από αυτή. Η αλλαγή αυτή πραγματοποιείται κάθε φορά που το μικρότερο στοιχείο που πρέπει να διαγραφεί βρίσκεται στην δευτερεύουσα δομή. Τα πειραματικά αποτελέσματα της μελέτης της δομής παρουσιάζονται στα Σχήματα 8.5-8.8 όπου με τη μορφή γραφικών παραστάσεων και πινάκων παρουσιάζονται οι χρόνοι επεξεργασίας και οι συγκρίσεις που πραγματοποιεί ο αλγόριθμος.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.05	0.06	0.05	0.08	0.05	0.05
64	0.06	0.06	0.05	0.08	0.06	0.05
128	0.08	0.06	0.06	0.08	0.06	0.06
256	0.08	0.07	0.06	0.08	0.07	0.05
512	0.08	0.07	0.07	0.09	0.07	0.07
1024	0.08	0.07	0.06	0.09	0.07	0.07
2048	0.08	0.08	0.07	0.09	0.08	0.08
4096	0.09	0.09	0.08	0.10	0.08	0.09
8192	0.10	0.09	0.09	0.11	0.09	0.09
16384	0.10	0.09	0.10	0.11	0.09	0.09

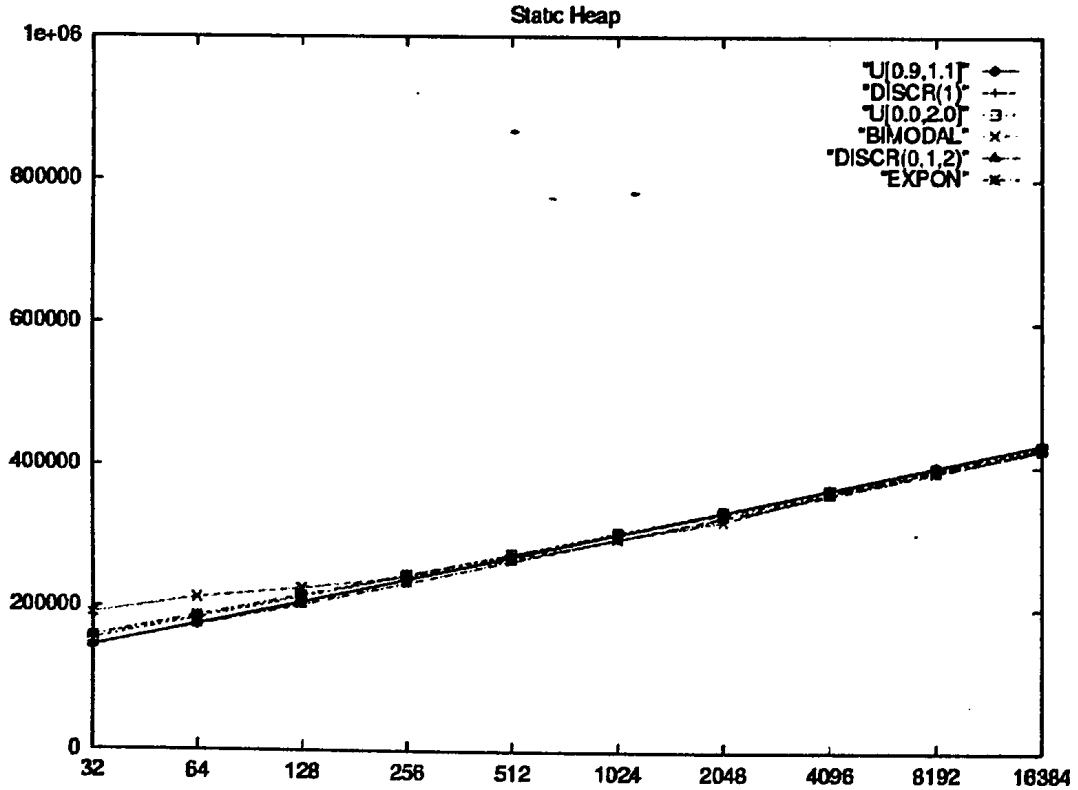
Σχήμα 8.1. Οι χρόνοι επεξεργασίας για τη δομή Static heap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	158405	154261	144702	190486	144000	145375
64	188406	185113	176529	214188	176000	174924
128	227949	215449	208371	227949	208000	204419
256	246512	245614	240252	243188	240000	234921
512	275304	275911	272147	269011	272000	266240
1024	304422	306228	304106	297773	304000	297046
2048	333738	336455	336057	323336	336000	328771
4096	363397	366847	368023	363532	368000	360101
8192	393004	397796	400008	395788	400000	392042
16384	423351	428683	432006	427044	432000	424375

Σχήμα 8.2. Οι συγκρίσεις για τη δομή Static heap.



Σχήμα 8.3. Οι χρόνοι επεξεργασίας για τη δομή Static heap.



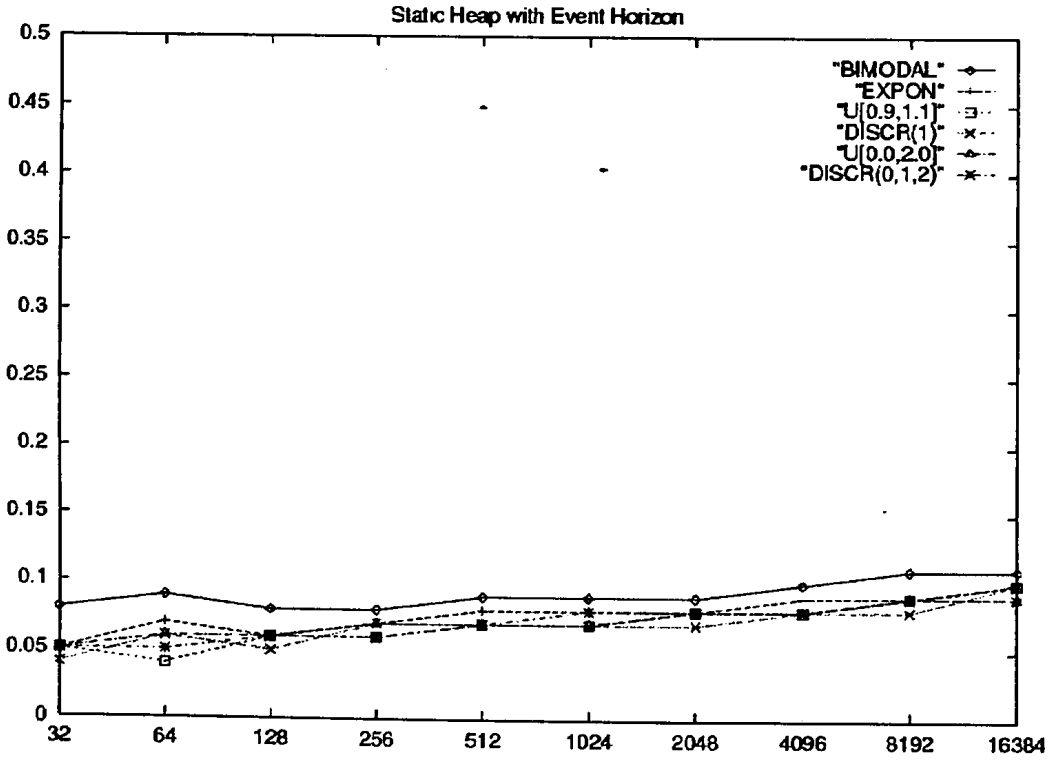
Σχήμα 8.4. Οι συγκρίσεις για τη δομή Static heap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.05	0.05	0.04	0.07	0.04	0.04
64	0.06	0.05	0.05	0.09	0.05	0.05
128	0.08	0.05	0.06	0.08	0.06	0.06
256	0.06	0.06	0.06	0.08	0.06	0.06
512	0.07	0.07	0.06	0.09	0.07	0.07
1024	0.08	0.07	0.07	0.09	0.07	0.07
2048	0.08	0.07	0.07	0.09	0.08	0.08
4096	0.09	0.08	0.08	0.10	0.08	0.08
8192	0.09	0.09	0.09	0.10	0.09	0.08
16384	0.10	0.10	0.09	0.12	0.09	0.09

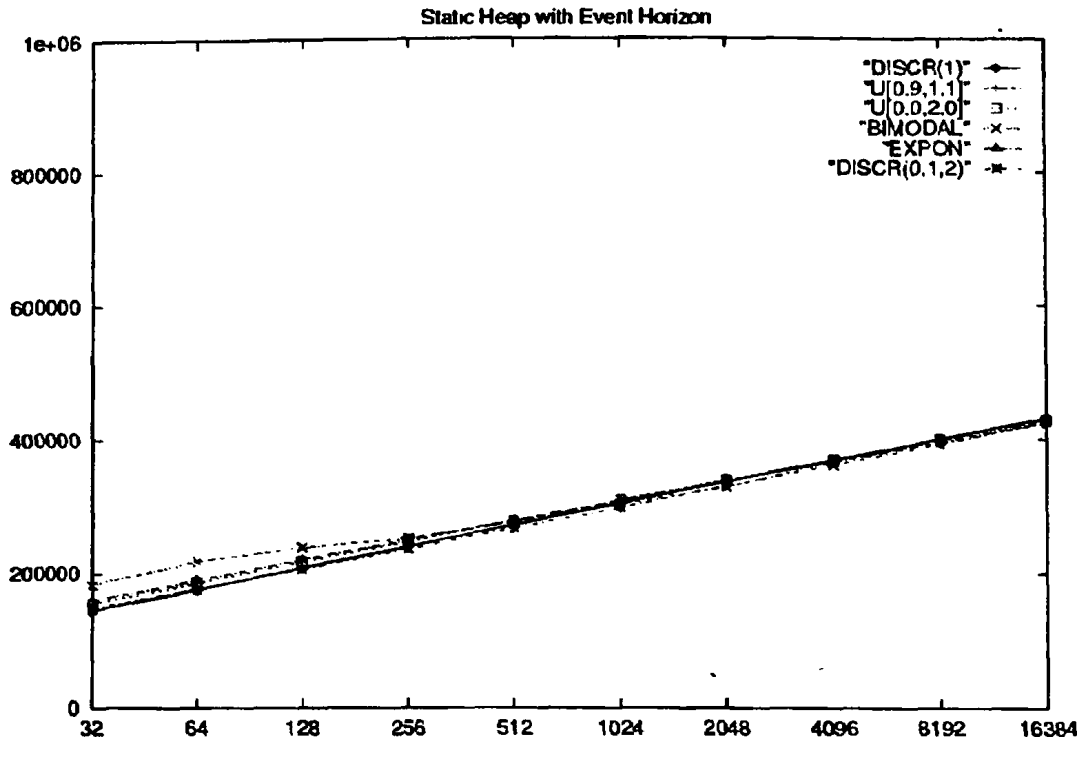
Σχήμα 8.5. Οι χρόνοι επεξεργασίας για τη δομή Static heap με την τεχνική Event Horizon.

N \ D	EXP	U02	U09	BIM	D1	D012
32	159255	154698	144817	183239	144000	147883
64	190314	186552	176742	218423	176000	177362
128	238560	217051	208504	238560	208000	206680
256	249704	247466	240397	252090	240000	236529
512	278469	277742	272268	276150	272000	267017
1024	307403	308080	304197	302631	304000	297695
2048	336207	338180	336090	327881	336000	328998
4096	365740	368665	368058	365249	368000	360556
8192	394971	398970	400042	397253	400000	392202
16384	425269	429725	431991	428239	432000	424109

Σχήμα 8.6. Οι συγκρίσεις για τη δομή Static heap με την τεχνική Event Horizon.



Σχήμα 8.7. Οι χρόνοι επεξεργασίας για τη δομή Static heap με την τεχνική Event Horizon.



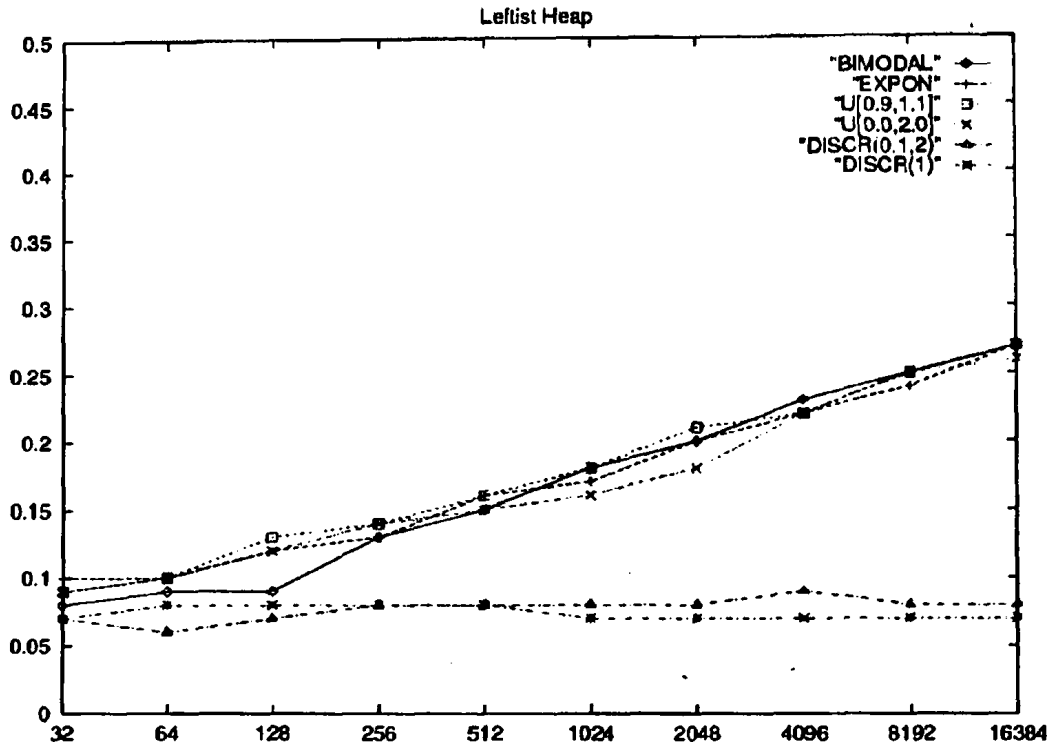
Σχήμα 8.8. Οι συγκρίσεις για τη δομή Static heap
με την τεχνική Event Horizon.

Η Δομή Leftist Heap

Στα Σχήματα 8.9 και 8.10 παρουσιάζονται οι χρόνοι επεξεργασίας που προέκυψαν από την πειραματική μελέτη της δομής Leftist heap σε μορφή πίνακα και γραφικής παράστασης αντίστοιχα. Αυτό που παρατηρεί κανείς μελετώντας τα αποτελέσματα είναι ότι η δομή παρουσιάζει την καλύτερη συμπεριφορά με τις διακριτές κατανομές, DISRETE(1) και DISCRETE(0,1,2), ενώ με τις υπόλοιπες συνεχείς κατανομές δεν είναι αποδοτική. Αυτό συμβαίνει διότι, όταν τα στοιχεία που αποτελούν τη δομή προέρχονται από τις διακριτές κατανομές, κάθε ένα από τα δεξιά μονοπάτια της δομής έχει μικρό μήκος, γεγονός που καθιστά τις διαδικασίες της εισαγωγής και της διαγραφής πολύ αποτελεσματικές.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.09	0.09	0.09	0.09	0.07	0.07
64	0.11	0.10	0.10	0.09	0.08	0.07
128	0.11	0.11	0.13	0.11	0.08	0.08
256	0.11	0.14	0.14	0.13	0.08	0.08
512	0.15	0.15	0.16	0.16	0.09	0.08
1024	0.18	0.18	0.17	0.19	0.06	0.08
2048	0.21	0.20	0.21	0.20	0.06	0.09
4096	0.22	0.22	0.22	0.23	0.06	0.08
8192	0.24	0.26	0.26	0.26	0.07	0.08
16384	0.29	0.27	0.28	0.29	0.06	0.08

Σχήμα 8.9. Οι χρόνοι επεξεργασίας για τη δομή Leftist heap.



Σχήμα 8.10. Οι χρόνοι επεξεργασίας για τη δομή Leftist heap.

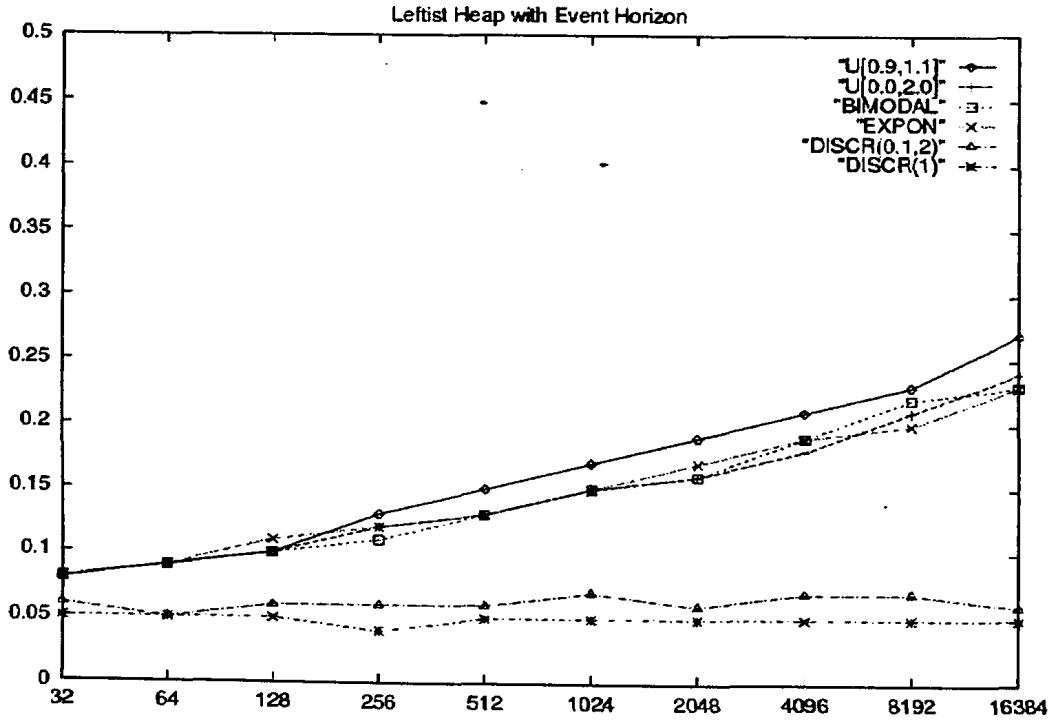
Η χρήση της τεχνικής Event Horizon, όπως φαίνεται και στα Σχήματα 8.11–8.14, βελτιώνει αρκετά την απόδοση της δομής Leftist heap. Η δευτερεύουσα δομή είναι επίσης μια δομή Leftist heap, η οποία, κάθε φορά που το μικρότερο στοιχείο βρίσκεται σε αυτή, συγχωνεύεται με την κύρια και δημιουργείται μια δομή Leftist heap. Εξάλλου η διαδικασία της συγχώνευσης δυο δομών Leftist heap είναι πολύ αποδοτική. Αξίζει να σημειωθεί ακόμη ότι, σύμφωνα με την μελέτη που πραγματοποιήθηκε, αν η δευτερεύουσα δομή είναι λίστα και η διαδικασία της συγχώνευσης πραγματοποιείται με διαδοχικές εισαγωγές των στοιχείων της στην κύρια δομή, η βελτίωση που παρατηρείται στην απόδοση της δομής δεν είναι το ίδιο σημαντική με αυτή που πραγματοποιείται από την τεχνική που περιγράφηκε παραπάνω. Το ίδιο ισχύει και για την περίπτωση που δεν πραγματοποιείται συγχώνευση των δυο δομών Leftist heap, όταν το μικρότερο στοιχείο εντοπιστεί στην δευτερεύουσα δομή, αλλά εναλλαγή μεταξύ κύριας και δευτερεύουσας δομής. Τα πειραματικά αποτελέσματα που προκύπτουν δείχνουν ότι ένας τέτοιος αλγόριθμος δεν είναι το ίδιο αποδοτικός με εκείνον που περιγράφηκε αρχικά, παρά το γεγονός ότι είναι αποδοτικότερος από τον αλγόριθμο της δομής Leftist heap χωρίς τη χρήση της τεχνικής Event Horizon.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.08	0.08	0.07	0.07	0.04	0.06
64	0.10	0.09	0.09	0.08	0.05	0.06
128	0.10	0.10	0.11	0.10	0.05	0.06
256	0.12	0.12	0.12	0.11	0.04	0.06
512	0.14	0.11	0.14	0.13	0.05	0.07
1024	0.16	0.14	0.16	0.16	0.05	0.06
2048	0.19	0.17	0.21	0.17	0.05	0.05
4096	0.19	0.18	0.20	0.19	0.05	0.06
8192	0.21	0.19	0.23	0.22	0.05	0.07
16384	0.22	0.22	0.27	0.24	0.05	0.07

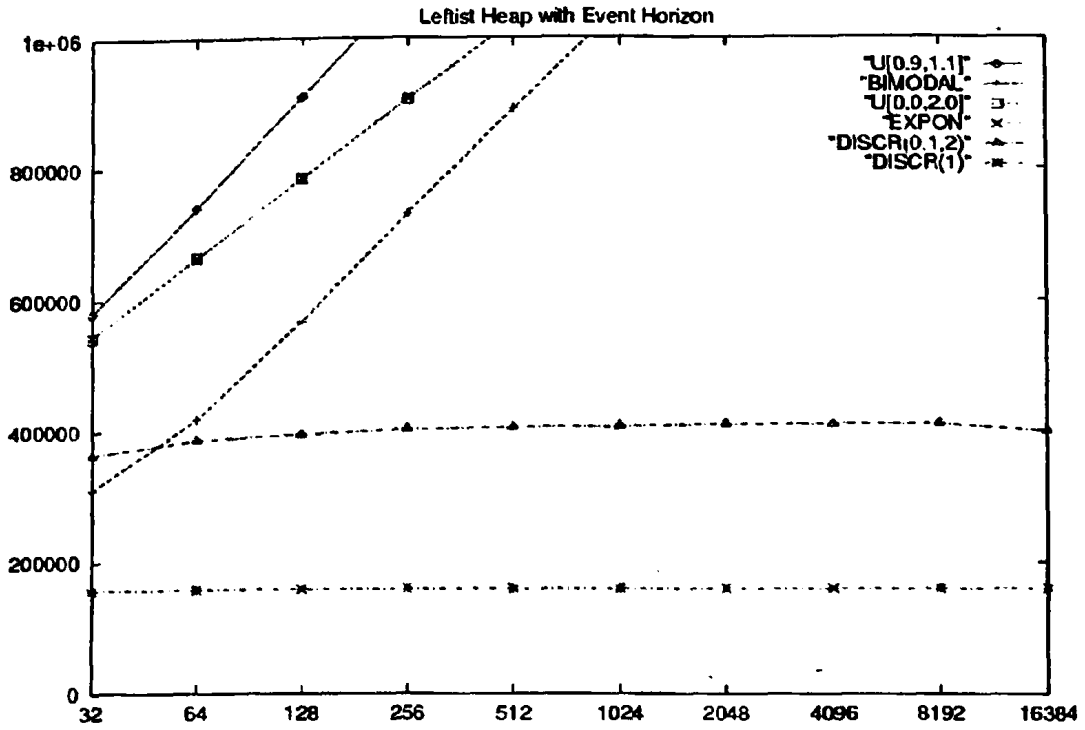
Σχήμα 8.11. Οι χρόνοι επεξεργασίας για τη δομή Leftist heap με την τεχνική Event Horizon.

N \ D	EXP	U02	U09	BIM	D1	D012
32	545719	543049	580181	311204	157000	364447
64	664142	665230	740103	418744	158500	387050
128	567082	784779	909967	567082	159250	395380
256	906581	905139	1080397	731353	159624	402655
512	1025402	1024952	1256997	890706	159810	405191
1024	1147543	1142380	1429202	1046332	159906	406460
2048	1267434	1265539	1609876	1176665	159954	410221
4096	1393732	1387231	1786481	1380242	159978	410146
8192	1510501	1512546	1960143	1512399	159990	410630
16384	1630152	1634880	2137740	1659277	159996	399110

Σχήμα 8.12. Οι συγκρίσεις για τη δομή Leftist heap με την τεχνική Event Horizon.



Σχήμα 8.13. Οι χρόνοι επεξεργασίας για τη δομή Leftist heap με την τεχνική Event Horizon.



Σχήμα 8.14. Οι συγκρίσεις για τη δομή Leftist heap
με την τεχνική Event Horizon.

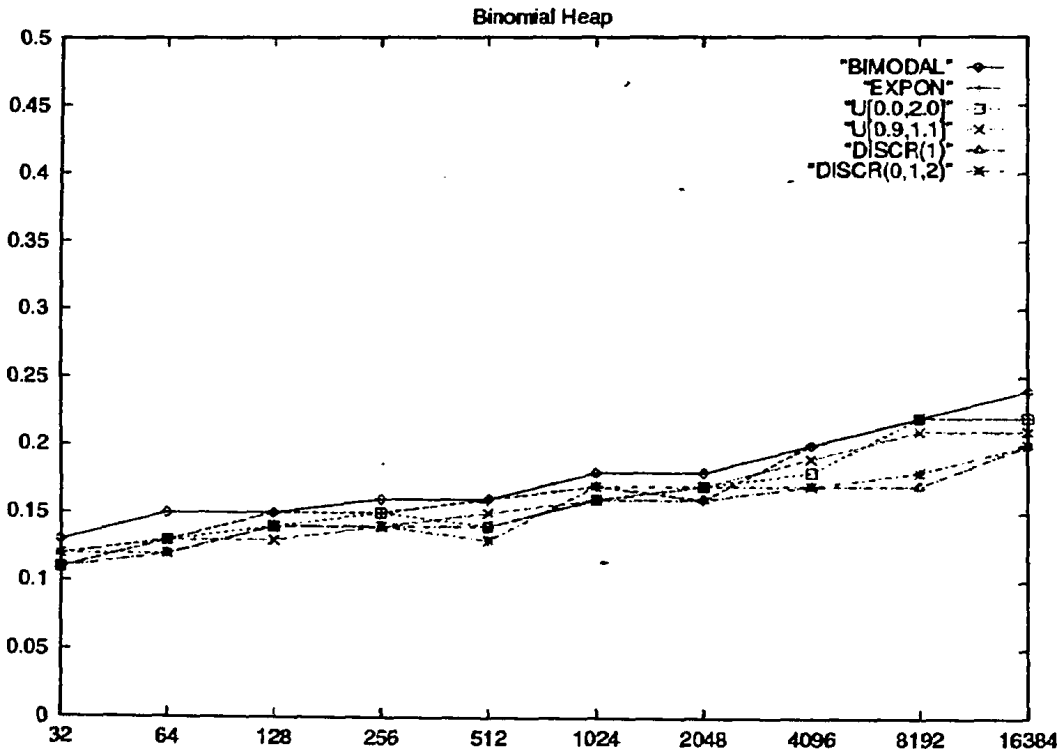
Η Δομή Binomial Heap

Στα Σχήματα 8.15 και 8.16 παρουσιάζονται οι χρόνοι επεξεργασίας που προέκυψαν από την πειραματική μελέτη της δομής Binomial heap σε μορφή πίνακα και γραφικής παράστασης αντίστοιχα. Αυτό που παρατηρεί κανείς μελετώντας τα αποτελέσματα είναι ότι η δομή παρουσιάζει την ίδια συμπεριφορά με όλες τις κατανομές. Τα ίδια συμπεράσματα προκύπτουν από τη μελέτη του πλήθους των συγκρίσεων που απαιτούνται από τον αλγόριθμο, όπως φαίνεται και στα Σχήματα 8.17 και 8.18.

Η χρήση της τεχνικής Event Horizon, όπως φαίνεται στα Σχήματα 8.19–8.22, βελτιώνει την απόδοση της δομής Binomial heap μόνο όταν χρησιμοποιούνται οι διακριτές κατανομές. Η δευτερεύουσα δομή είναι επίσης μια δομή Binomial heap, η οποία, κάθε φορά που το μικρότερο στοιχείο βρίσκεται σε αυτή, συγχωνεύεται με την κύρια και δημιουργείται μια δομή Binomial heap. Εξάλλου η διαδικασία της συγχώνευσης δυο δομών Binomial heap είναι πολύ αποδοτική. Όταν χρησιμοποιούνται οι διακριτές κατανομές, η κύρια δομή γίνεται κενή κατά τη διάρκεια της εκτέλεσης με αποτέλεσμα η διαδικασία της συγχώνευσης να πραγματοποιείται σε σταθερό χρόνο.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.12	0.11	0.11	0.13	0.12	0.12
64	0.13	0.12	0.12	0.14	0.13	0.13
128	0.15	0.13	0.14	0.15	0.12	0.13
256	0.15	0.14	0.13	0.16	0.14	0.14
512	0.15	0.15	0.15	0.17	0.15	0.15
1024	0.18	0.16	0.16	0.19	0.17	0.16
2048	0.19	0.18	0.18	0.20	0.17	0.17
4096	0.20	0.18	0.19	0.22	0.19	0.18
8192	0.22	0.21	0.19	0.20	0.19	0.18
16384	0.25	0.22	0.22	0.26	0.20	0.20

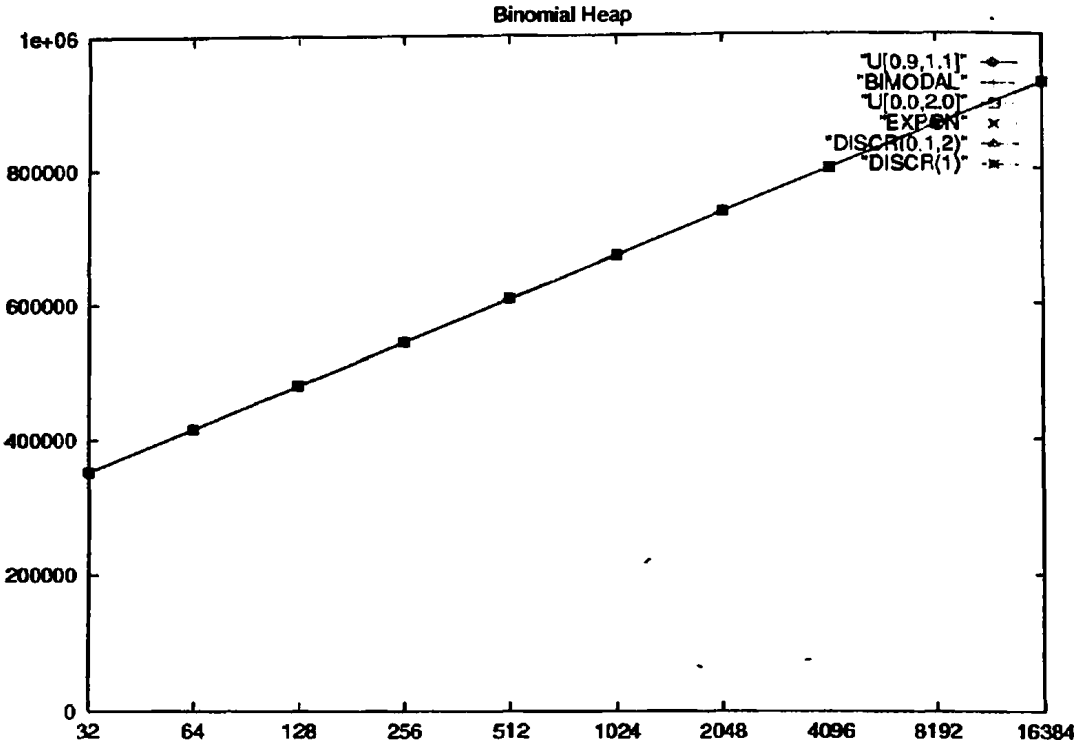
Σχήμα 8.15. Οι χρόνοι επεξεργασίας για τη δομή Binomial heap.



Σχήμα 8.16. Οι χρόνοι επεξεργασίας για τη δομή Binomial heap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	352000	352000	352000	352000	352000	352000
64	416000	416000	416000	416000	416000	416000
128	480000	480000	480000	480000	480000	480000
256	544000	544000	544000	544000	544000	544000
512	608000	608000	608000	608000	608000	608000
1024	672000	672000	672000	672000	672000	672000
2048	736000	736000	736000	736000	736000	736000
4096	800000	800000	800000	800000	800000	800000
8192	864000	864000	864000	864000	864000	864000
16384	928000	928000	928000	928000	928000	928000

Σχήμα 8.17. Οι συγκρίσεις για τη δομή Binomial heap.



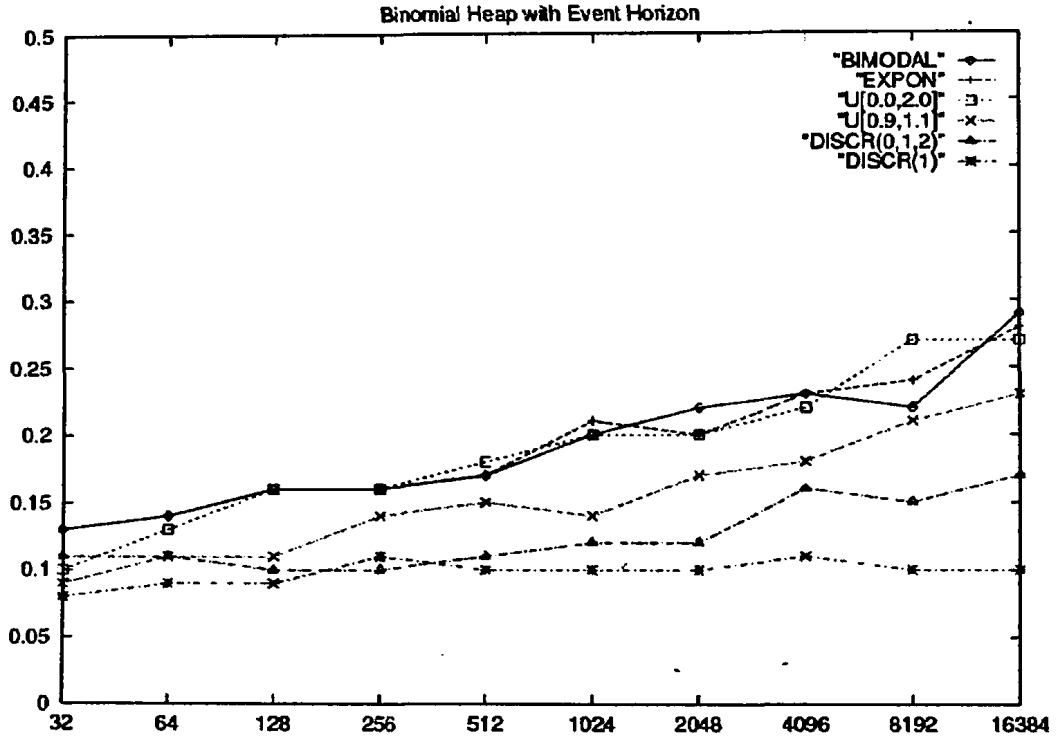
Σχήμα 8.18. Οι συγκρίσεις για τη δομή Binomial heap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.13	0.11	0.10	0.13	0.09	0.11
64	0.14	0.13	0.11	0.14	0.10	0.11
128	0.15	0.14	0.12	0.15	0.09	0.11
256	0.17	0.17	0.12	0.18	0.10	0.10
512	0.16	0.16	0.14	0.18	0.09	0.12
1024	0.19	0.17	0.15	0.20	0.10	0.13
2048	0.22	0.22	0.18	0.22	0.10	0.12
4096	0.24	0.23	0.19	0.24	0.11	0.14
8192	0.27	0.25	0.21	0.26	0.13	0.12
16384	0.25	0.27	0.27	0.27	0.11	0.13

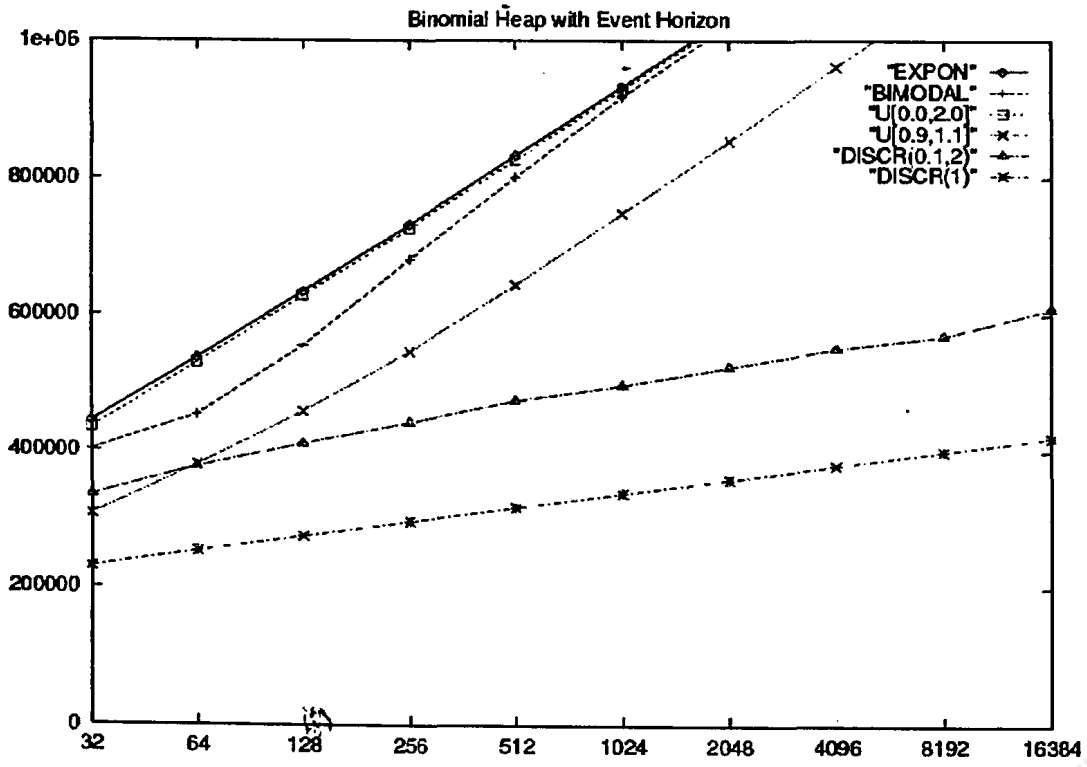
Σχήμα 8.19. Οι χρόνοι επεξεργασίας για τη δομή Binomial heap με την τεχνική Event Horizon.

N \ D	EXP	U02	U09	BIM	D1	D012
32	443558	433366	306778	401371	230000	334858
64	537183	530043	379776	453759	253500	377416
128	555238	628186	458689	555238	275875	411671
256	732016	726597	546368	680794	297480	442175
512	832987	826689	644075	801346	318505	474794
1024	932802	928249	748736	917937	339241	497468
2048	1033886	1030529	854228	1022625	359729	524773
4096	1136662	1130475	961575	1134102	380061	551691
8192	1239724	1232451	1069020	1233780	400297	569774
16384	1343245	433366	1171744	1335569	420475	609592

Σχήμα 8.20. Οι συγκρίσεις για τη δομή Binomial heap με την τεχνική Event Horizon.



Σχήμα 8.21. Οι χρόνοι επεξεργασίας για τη δομή Binomial heap με την τεχνική Event Horizon.



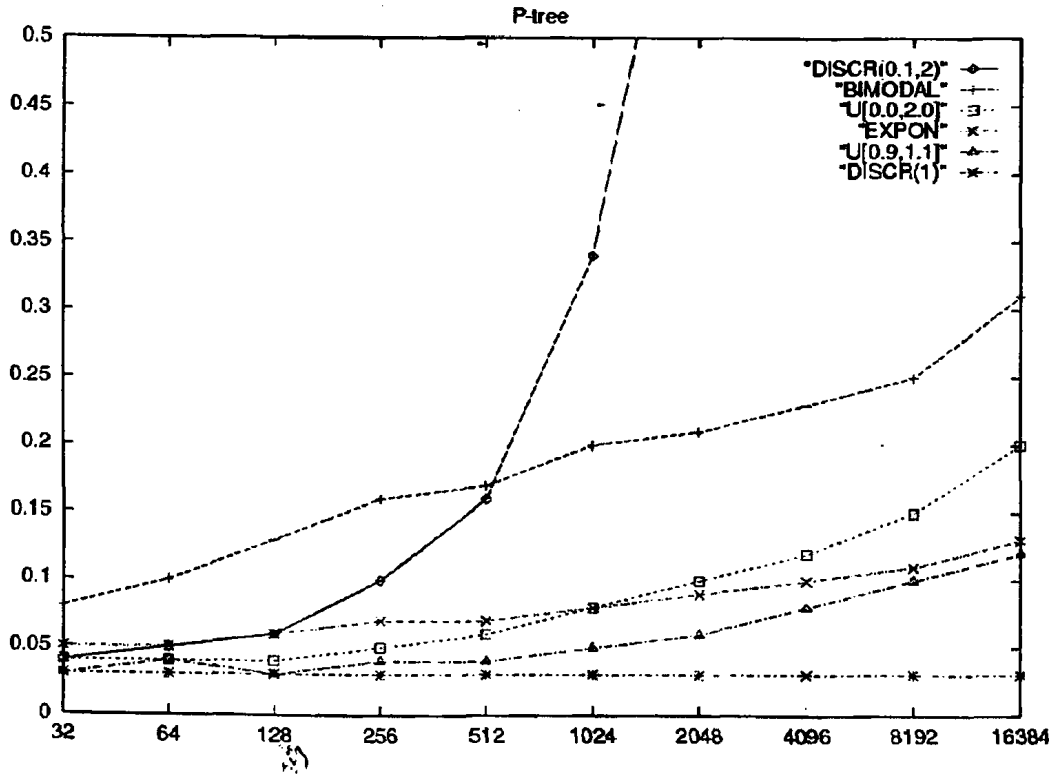
Σχήμα 8.22. Οι συγκρίσεις για τη δομή Binomial heap με την τεχνική Event Horizon.

Η Δομή P-tree

Η δομή του *P*-tree συνδυάζει τα πλεονεκτήματα της ταξινομημένης λίστας σε συνδυασμό με τα πλεονεκτήματα τα οποία παρουσιάζουν οι δενδροειδείς δομές. Τα αποτελέσματα της πειραματικής μελέτης και συγκεκριμένα οι χρόνοι επεξεργασίας για τη δομή του *P*-tree, παρουσιάζονται στα Σχήματα 8.23 και 8.24, σε μορφή πίνακα και γραφικής παράστασης αντίστοιχα. Οι συγκρίσεις που απαιτεί ο αλγόριθμος παρουσιάζονται στα Σχήματα 8.25 και 8.26, με τη μορφή πίνακα και γραφικής παράστασης, αντίστοιχα. Αξιοσημείωτο είναι το γεγονός ότι η κατανομή DISCRETE(1) παρουσιάζει εξαιρετικά καλή συμπεριφορά ενώ η DISCRETE(0,1,2) όταν το σύνολο γεγονότων αποκτά περισσότερα από 512 γεγονότα είναι εξαιρετικά αναποτελεσματική σε σχέση με τις υπόλοιπες κατανομές.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.05	0.04	0.03	0.08	0.03	0.04
64	0.05	0.04	0.04	0.10	0.03	0.05
128	0.13	0.04	0.03	0.13	0.03	0.06
256	0.07	0.05	0.04	0.16	0.03	0.10
512	0.07	0.06	0.04	0.17	0.03	0.16
1024	0.08	0.08	0.05	0.20	0.03	0.34
2048	0.09	0.10	0.06	0.21	0.03	0.73
4096	0.10	0.12	0.08	0.23	0.03	1.49
8192	0.11	0.15	0.10	0.25	0.03	2.89
16384	0.13	0.20	0.12	0.31	0.03	5.41

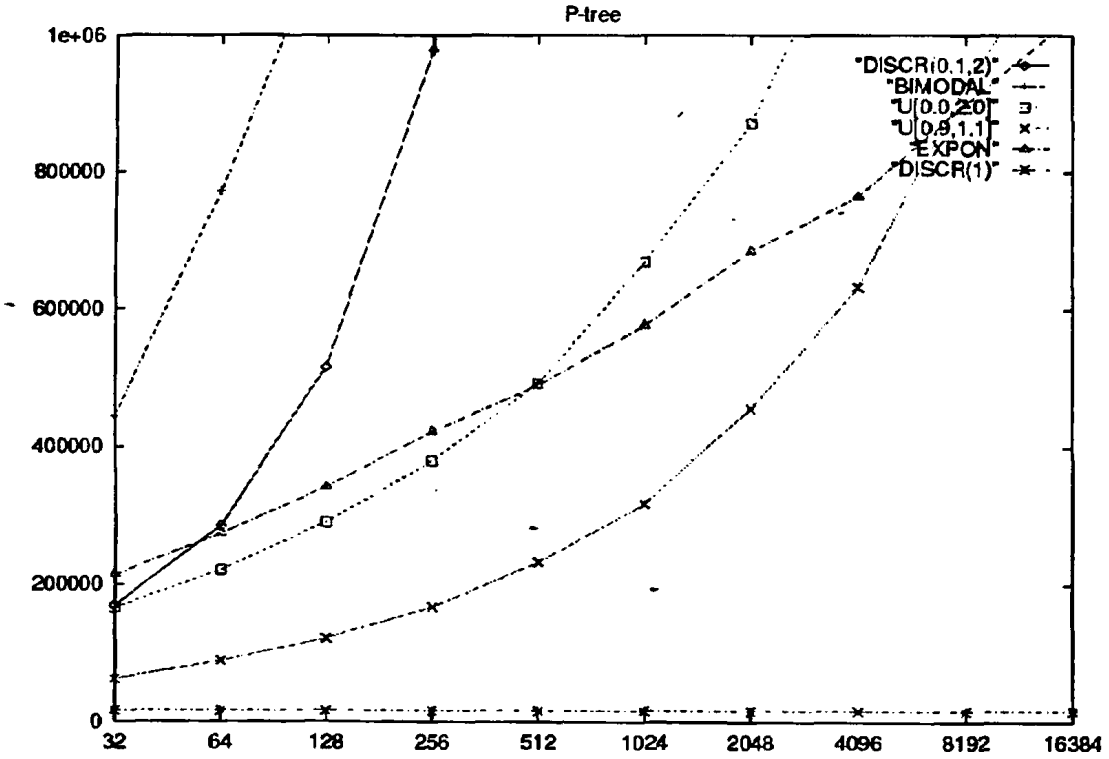
Σχήμα 8.23. Οι χρόνοι επεξεργασίας για τη δομή *P*-tree.



Σχήμα 8.24. Οι χρόνοι επεξεργασίας για τη δομή *P*-tree.

N \ D	EXP	U02	U09	BIM	D1	D012
32	213844	165829	62315	444667	16000	169300
64	274188	221039	88736	771129	16000	285346
128	1159446	291351	121460	1159446	16000	516035
256	422100	379917	167403	1522347	16000	977758
512	489887	492181	232777	1781631	16000	1892975
1024	577076	669015	318054	2053770	16000	3730739
2048	684742	870776	456022	2147027	16000	7321325
4096	764949	1196768	632870	2177813	16000	14628685
8192	899180	1543863	929672	2433977	16000	28248822
16384	1026140	2060237	1151740	2897753	16000	52969971

Σχήμα 8.25. Οι συγκρίσεις για τη δομή *P*-tree.



Σχήμα 8.28. Οι συγκρίσεις για τη δομή *P*-tree.

Όταν χρησιμοποιείται η κατανομή DISCRETE(1) τότε η δομή του *P*-tree εκφυλλίζεται σε λίστα, αφού κάθε νέος κόμβος που θα εισάγεται θα περιέχει την μεγαλύτερη τιμή από όλα τα στοιχεία της δομής, οπότε θα έχει ως αριστερό παιδί τη ρίζα του δένδρου και θα αποτελεί τη νέα ρίζα του *P*-tree. Ο κάθε νέος κόμβος επομένως θα εισάγεται σε σταθερό χρόνο στη δομή. Η διαγραφή πραγματοποιείται σε σταθερό χρόνο, ανεξάρτητα από την κατανομή που χρησιμοποιείται, αφού ο κόμβος με την μικρότερη πληροφορία εντοπίζεται αμέσως. Επομένως, όταν πρόκειται για την κατανομή DISCRETE(1), εισαγωγή και διαγραφή πραγματοποιούνται ταχύτατα, σε σταθερό χρόνο.

Όπως προαναφέρθηκε, ο αλγόριθμος δεν είναι αποτελεσματικός όταν η δομή περιέχει στοιχεία που ακολουθούν την κατανομή DISCRETE(0,1,2). Αυτό οφείλεται στο γεγονός ότι η δομή του *P*-tree, λόγω των στοιχείων που περιέχει, ουσιαστικά εκφυλλίζεται σε τρεις λίστες (προφανώς

συνδεδεμένες μεταξύ τους) με αποτέλεσμα η διαδικασία της εισαγωγής να μην πραγματοποιείται αποτελεσματικά.

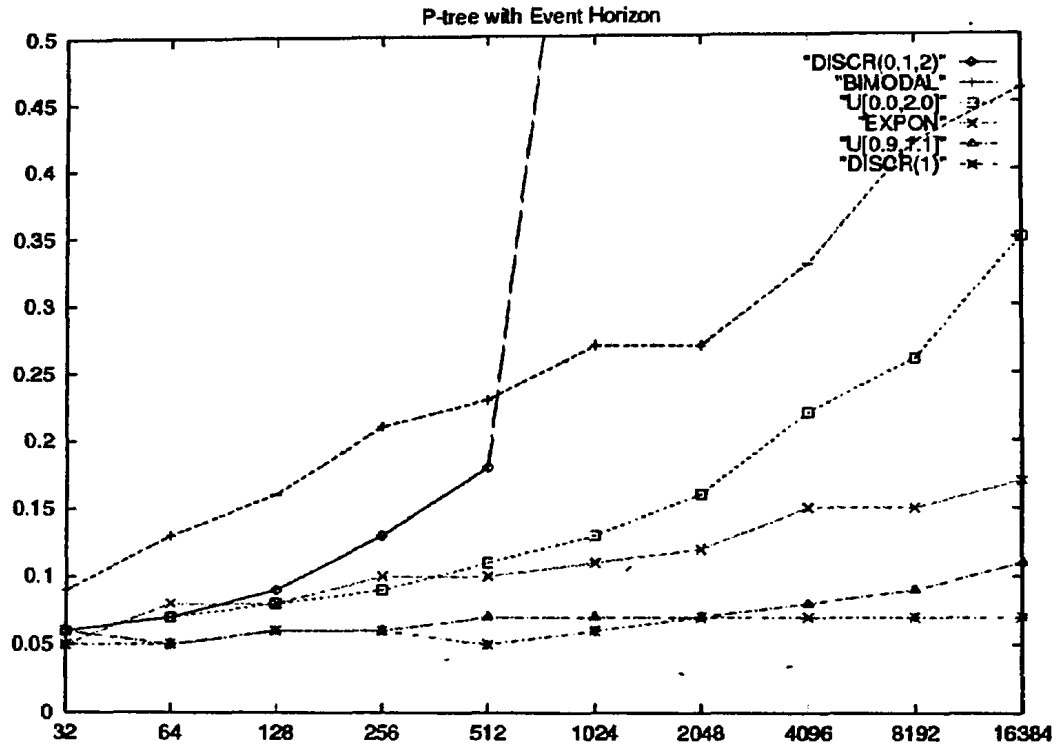
Από τη χρήση της τεχνικής Event Horizon στη δομή του P-tree προκύπτουν τα ίδια αποτελέσματα όσον αφορά στη συμπεριφορά των κατανομών αλλά το πλήθος των συγκρίσεων που απαιτούνται για τις διαδικασίες εισαγωγής και διαγραφής είναι μικρότερο. Ένα αξιοσημείωτο συμπέρασμα που προκύπτει από την πειραματική μελέτη είναι ότι η ταξινόμηση της δευτερεύουσας δομής δεν φαίνεται να διευκολύνει τη διαδικασία της εισαγωγής των γεγονότων στην κύρια, αφού αν δεν πραγματοποιηθεί ταξινόμηση η απόδοση της δομής βελτιώνεται αλλά όχι σε μεγάλο βαθμό. Επιπλέον, αν μελετηθεί το πλήθος των συγκρίσεων που πραγματοποιούνται προκειμένου τα στοιχεία της δευτερεύουσας δομής να επανενταχθούν στην κύρια, προκύπτει ότι είναι μεγαλύτερος αν προηγηθεί ταξινόμηση της πρώτης. Το γεγονός αυτό είναι εξάλλου αναμενόμενο, αφού και ο χρόνος επεξεργασίας είναι μεγαλύτερος σε αυτήν την περίπτωση. Ο χρόνος επεξεργασίας που απαιτείται καθώς και το πλήθος των συγκρίσεων που πραγματοποιούνται παρουσιάζονται στα Σχήματα 8.27 και 8.28 με τη μορφή πινάκων, ενώ οι αντίστοιχες γραφικές παραστάσεις παρουσιάζονται στα Σχήματα 8.29 και 8.30.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.05	0.04	0.05	0.08	0.04	0.04
64	0.05	0.05	0.06	0.11	0.03	0.05
128	0.13	0.05	0.06	0.13	0.04	0.07
256	0.07	0.06	0.06	0.16	0.03	0.12
512	0.08	0.07	0.08	0.19	0.03	0.18
1024	0.09	0.07	0.07	0.21	0.04	0.37
2048	0.10	0.10	0.10	0.25	0.02	0.77
4096	0.12	0.12	0.10	0.24	0.04	1.73
8192	0.12	0.15	0.11	0.25	0.04	3.00
16384	0.15	0.19	0.15	0.31	0.04	7.34

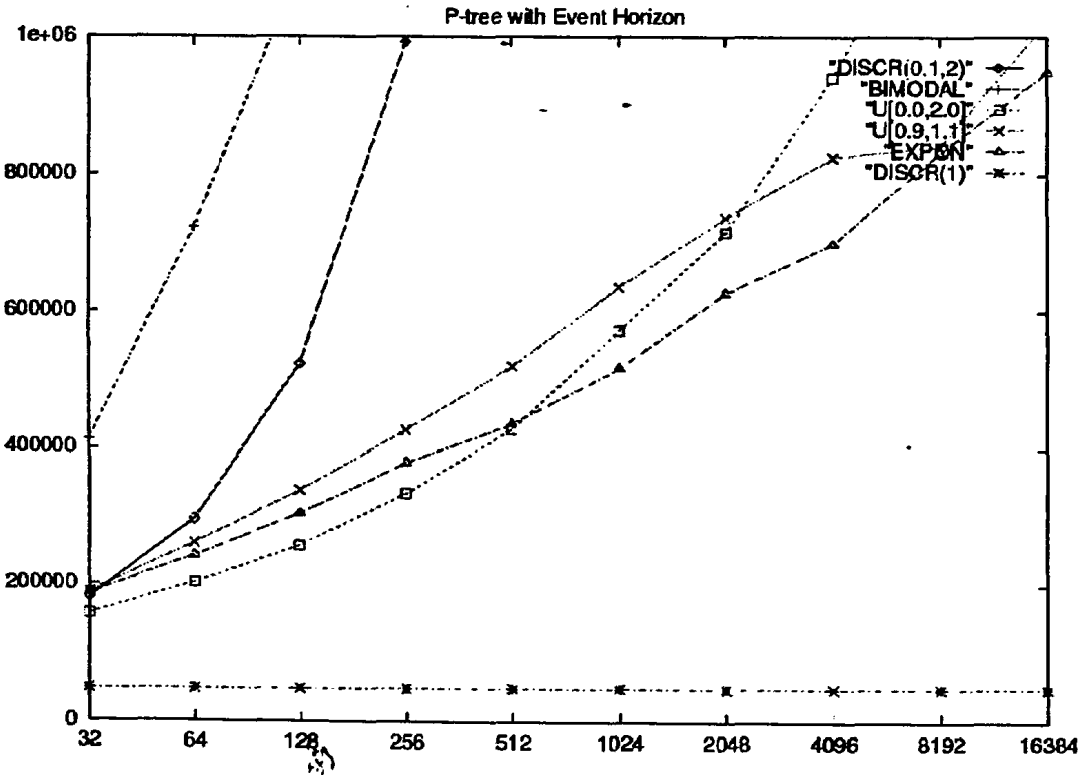
Σχήμα 8.27. Οι χρόνοι επεξεργασίας για τη δομή *P*-tree με την τεχνική Event Horizon.

N \ D	EXP	U02	U09	BIM	D1	D012
32	186414	156465	188715	412455	47000	180481
64	242966	203342	261662	722556	47500	296470
128	1100410	258846	339256	1100410	47750	523896
256	379756	335175	427851	1434903	48003	992601
512	436894	429254	520734	1677148	48321	1924623
1024	518017	573100	636466	1903608	48353	3834181
2048	627611	716420	737550	1943069	48369	7541296
4096	700135	940066	824930	1996283	48377	12978957
8192	831374	1227481	840763	2201651	48381	28335580
16384	948376	1590807	1021979	2492718	48383	50245060

Σχήμα 8.28. Οι συγκρίσεις για τη δομή *P*-tree με την τεχνική Event Horizon.



Σχήμα 8.29. Οι χρόνοι επεξεργασίας για τη δομή *P*-tree με την τεχνική Event Horizon.



Σχήμα 8.30. Οι συγκρίσεις για τη δομή *P*-tree με την τεχνική Event Horizon.

8.3 Σύνθετες Δομές Δεδομένων

Η Δομή SPEEDES Qheap

Η δομή SPEEDES Qheap χρησιμοποιεί την τεχνική Event Horizon, με αποτέλεσμα η διαδικασία της εισαγωγής να πραγματοποιείται σε σταθερό χρόνο. Όπως έχει ήδη αναφερθεί σε προηγούμενο Κεφάλαιο, όταν το μικρότερο στοιχείο το οποίο θα πρέπει να διαγραφεί βρίσκεται στη δευτερεύουσα δομή, αυτή ταξινομείται και αναδιοργανώνεται ώστε να αποτελέσει ένα μετα-κόμβο. Έτσι, η συγχώνευση των δύο δομών, κύριας και δευτερεύουσας, είναι ουσιαστικά η διαδικασία της εισαγωγής ενός στοιχείου σε μια ταξινομημένη λίστα μήκους *S*. Για τη διαδικασία της διαγραφής του μικρότερου στοιχείου, όπως είναι γνωστό, απαιτείται αναδιοργάνωση του πρώτου μετα-κόμβου της δομής. Η διαδικασία αυτή όμως δεν είναι χρονοβόρα αφού οι μετα-κόμβοι της δομής θα περιέχουν κυρίως απλά στοιχεία, γεγονός που οφείλεται στον τρόπο με τον οποίο γίνεται η συγχώνευση της δευτερεύουσας με την κύρια δομή.

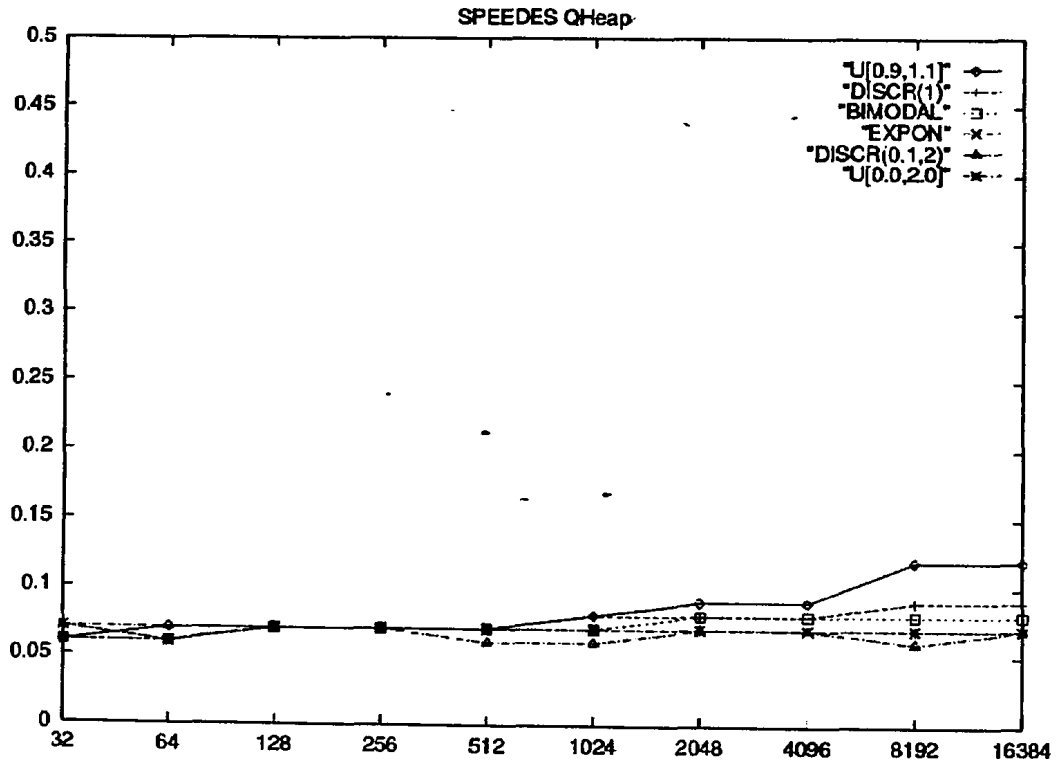
Τα παραπάνω οδηγούν στο συμπέρασμα ότι ο αλγόριθμος είναι πολύ αποτελεσματικός, γεγονός που επιβεβαιώνεται από τα πειραματικά αποτελέσματα που προέκυψαν. Τα Σχήματα 8.31-8.34 παρουσιάζουν τα αποτελέσματα της πειραματικής μελέτης της δομής, όπου S = 40, συγκεκριμένα το χρόνο επεξεργασίας που απαιτείται καθώς και τις συγκρίσεις που πραγματοποιούνται, με τη μορφή πινάκων και γραφικών παραστάσεων. Η τιμή του S επιλέχθηκε μετά από πειράματα από τα οποία διαπιστώθηκε ότι για τιμή S = 40 η δομή είναι περισσότερο αποτελεσματική.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.07	0.07	0.06	0.06	0.06	0.07
64	0.07	0.06	0.06	0.06	0.07	0.06
128	0.06	0.07	0.06	0.06	0.07	0.07
256	0.07	0.07	0.08	0.07	0.06	0.06
512	0.07	0.06	0.07	0.07	0.07	0.07
1024	0.07	0.07	0.08	0.07	0.07	0.07
2048	0.05	0.07	0.08	0.07	0.08	0.07
4096	0.07	0.07	0.09	0.07	0.08	0.07
8192	0.07	0.06	0.10	0.07	0.08	0.07
16384	0.07	0.07	0.11	0.07	0.09	0.07

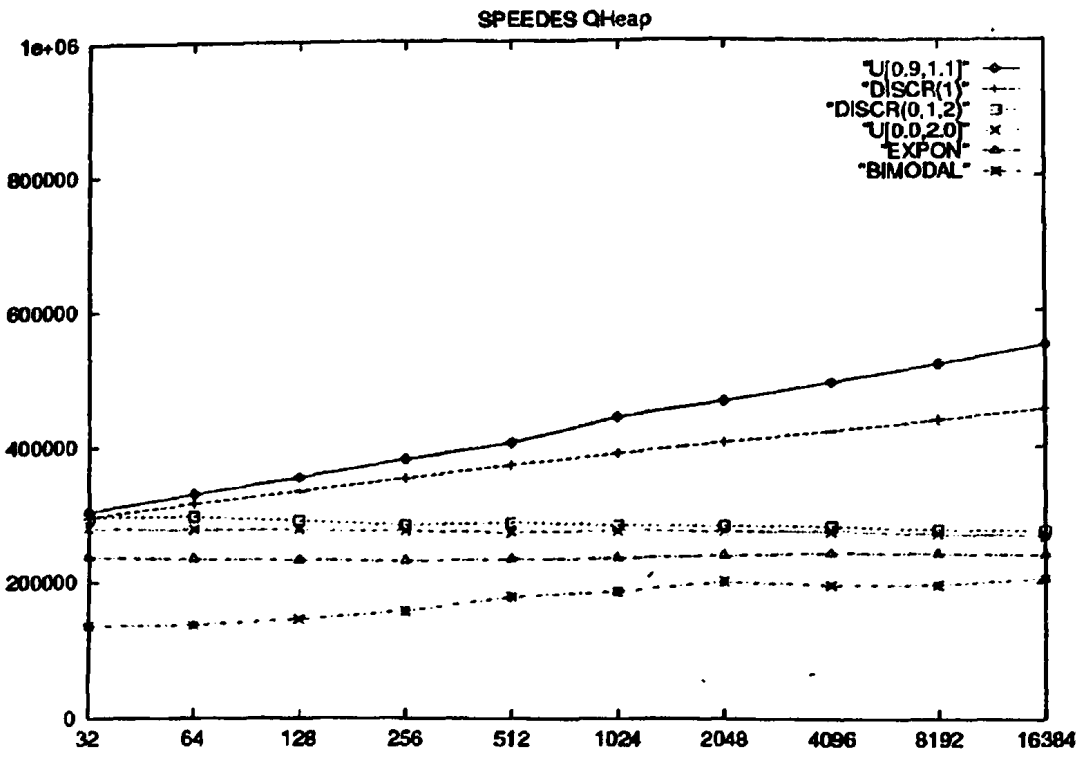
Σχήμα 8.31. Οι χρόνοι επεξεργασίας για τη δομή SPEEDES Qheap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	237291	279892	304728	135859	296500	296568
64	234091	277690	330035	137325	316250	297094
128	145471	277617	354337	145471	334125	290900
256	230769	275122	380877	157106	352724	284126
512	233816	273274	405527	178883	372901	288016
1024	234904	275556	443014	186045	389525	283725
2048	238774	274538	466423	200733	406029	282820
4096	242840	274311	494059	195760	422473	282031
8192	241699	270680	521203	196271	438887	277118
16384	239980	268296	549397	205396	455286	275156

Σχήμα 8.32. Οι συγκρίσεις για τη δομή SPEEDES Qheap.



Σχήμα 8.33. Οι χρόνοι επεξεργασίας για τη δομή SPEEDES Qheap.



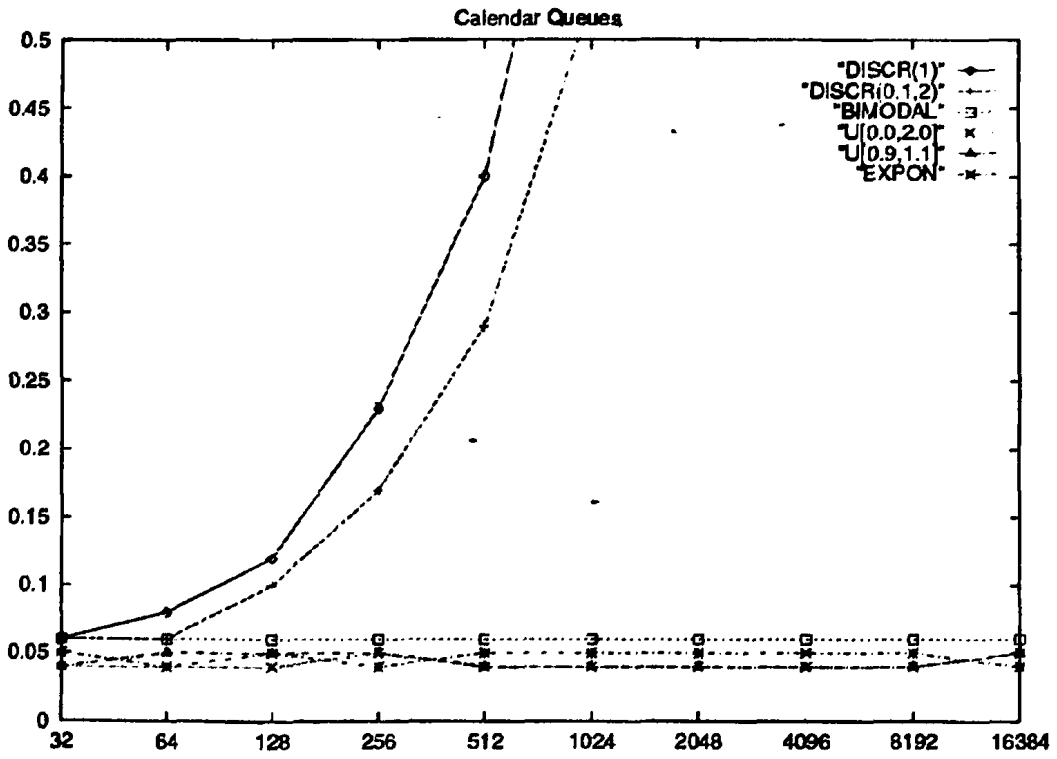
Σχήμα 8.34. Οι συγκρίσεις για τη δομή SPEEDES Qheap.

Η Δομή Calendar Queues

Η δομή Calendar Queues είναι πολύ αποτελεσματική όταν το πλήθος των buckets, το οποίο όπως έχει περιγραφεί σε προηγούμενο κεφάλαιο αλλάζει ανάλογα με το πλήθος των στοιχείων, είναι ικανοποιητικά μεγάλο, όπως έχει ήδη αναφερθεί σε προηγούμενο κεφάλαιο και τα στοιχεία που περιέχει κατανέμονται ομοιόμορφα σε κάθε ένα από αυτά. Σε μια τέτοια περίπτωση οι λίστες που θα αντιστοιχούν σε κάθε bucket δεν θα αποτελούνται από μεγάλο αριθμό στοιχείων οπότε οι διαδικασίες της εισαγωγής και της διαγραφής θα είναι αποδοτικές. Έτσι, όπως είναι αναμενόμενο, ο αλγόριθμος είναι εξαιρετικά αναποτελεσματικός όταν τα στοιχεία που υπάρχουν στη δομή προέρχονται από τις κατανομές DISCRETE(1) και DISCRETE(0,1,2), αφού τα στοιχεία μοιράζονται σε ελάχιστα μόνο buckets. Τα αποτελέσματα της πειραματικής μελέτης παρουσιάζονται στα Σχήματα 8.35-8.38.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.04	0.04	0.05	0.06	0.06	0.06
64	0.04	0.05	0.04	0.06	0.08	0.06
128	0.04	0.05	0.06	0.06	0.12	0.10
256	0.05	0.05	0.04	0.06	0.23	0.17
512	0.04	0.04	0.05	0.06	0.40	0.29
1024	0.04	0.04	0.05	0.06	0.76	0.53
2048	0.04	0.04	0.05	0.06	1.54	1.06
4096	0.04	0.04	0.05	0.06	3.09	2.10
8192	0.04	0.04	0.05	0.06	6.27	4.36
16384	0.05	0.05	0.04	0.06	15.94	11.07

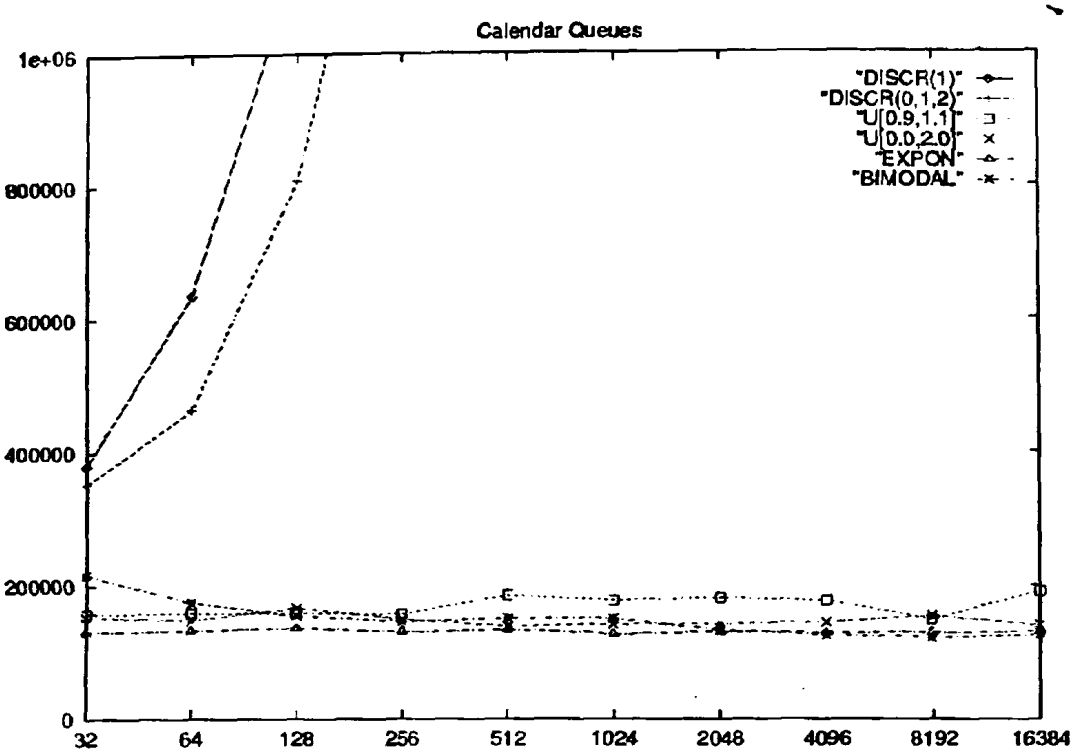
Σχήμα 8.35. Οι χρόνοι επεξεργασίας για τη δομή Calendar Queues.



Σχήμα 8.36. Οι χρόνοι επεξεργασίας για τη δομή Calendar Queues.

N \ D	EXP	U02	U09	BIM	D1	D012
32	153844	157780	130739	216462	380076	352131
64	148642	159284	133230	175088	636054	465000
128	167067	159809	155092	155092	1148071	808324
256	149874	157126	131076	146587	2164202	1488875
512	138728	184379	134142	150426	4196360	2863442
1024	142485	177520	128019	150893	8194404	5569613
2048	143098	180922	132168	134203	16176507	11007685
4096	144357	176644	128641	124726	32155295	21770743
8192	154319	148893	129966	122688	64130862	43781965
16384	140965	191041	131153	126738	128090999	87390715

Σχήμα 8.37. Οι συγκρίσεις για τη δομή Calendar Queues.



Σχήμα 8.38. Οι συγκρίσεις για τη δομή Calendar Queues.

8.4 Επεκτάσεις της Δομής του Σωρού

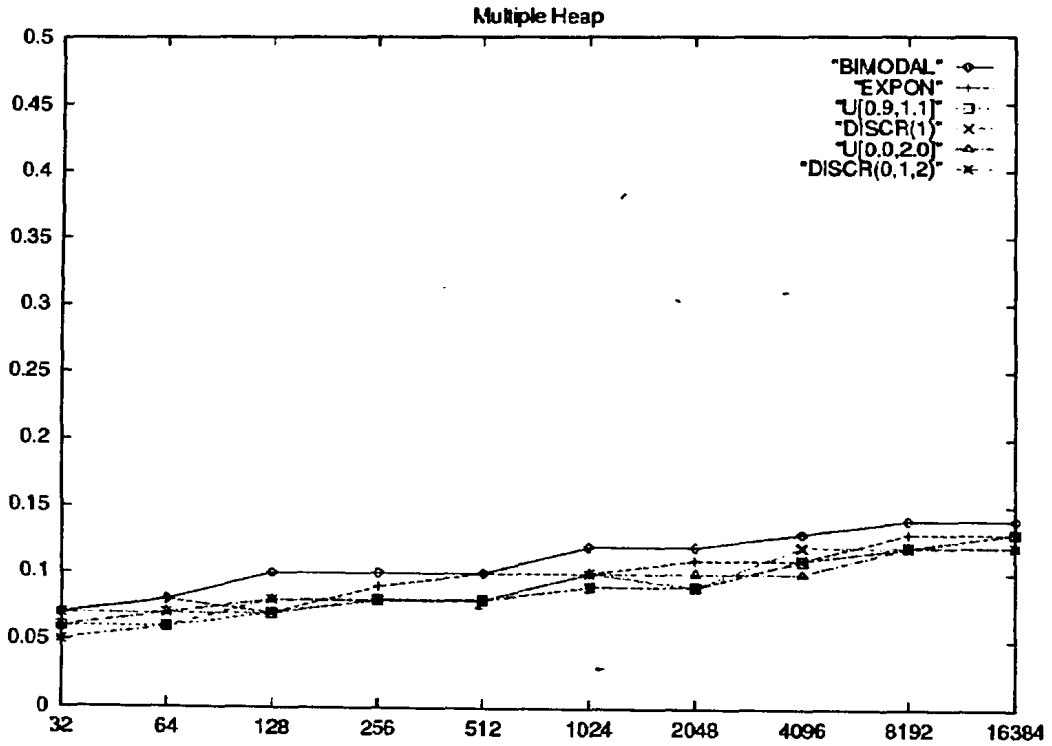
Η Δομή Multiple Heap (M-heap)

Έστω M το πλήθος των σωρών που αποτελούν τη δομή. Διαπιστώθηκε πειραματικά ότι η απόδοση του αλγόριθμου είναι μέγιστη όταν η τιμή του M είναι ικανοποιητικά μικρή και συγκεκριμένα όταν είναι κοντά στην τιμή 60 ή και μικρότερη. Υπενθυμίζεται ότι η τιμή του M καθορίζει το μέγεθος της δομής του σωρού I-heap. Τα αποτελέσματα που παρουσιάζονται στη συνέχεια, στα Σχήματα 8.39-8.42, προκύπτουν για τιμή $M = 16$. Όπως ισχύει και για την απλή δομή του σωρού, η δομή M-heap είναι εξίσου αποτελεσματική ανεξάρτητα από την κατανομή που χρησιμοποιείται κάθε φορά.

Μελετήθηκε ακόμη το αποτέλεσμα της χρήσης της τεχνικής Event Horizon σε μια τροποποιημένη δομή M-heap. Έτσι, αν η δομή I-heap είναι λίστα και όχι σωρός και η δευτερεύουσα δομή είναι μια δομή σωρού η οποία εισάγεται στην κύρια αυξάνοντας την τιμή του M κατά 1 κάθε φορά που το μικρότερο στοιχείο εντοπίζεται σε αυτή, σύμφωνα με τα αποτελέσματα της πειραματικής μελέτης, η χρήση της τεχνικής Event Horizon δεν βελτιώνει την απόδοση της δομής, είτε πρόκειται για τη στατική είτε για τη δυναμική της αναπαράσταση. Σημειώνεται ακόμη ότι το πλήθος M των σωρών δεν παραμένει σταθερό και μάλιστα μπορεί να ισχύει $M = 1$, όπως για παράδειγμα στην περίπτωση της κατανομής DISCRETE(1).

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.06	0.06	0.07	0.07	0.04	0.07
64	0.08	0.07	0.06	0.08	0.07	0.07
128	0.10	0.07	0.07	0.10	0.07	0.08
256	0.09	0.08	0.07	0.11	0.07	0.08
512	0.09	0.09	0.09	0.11	0.09	0.09
1024	0.10	0.10	0.09	0.11	0.09	0.09
2048	0.11	0.10	0.09	0.12	0.10	0.10
4096	0.12	0.11	0.11	0.13	0.11	0.10
8192	0.12	0.12	0.11	0.14	0.12	0.12
16384	0.12	0.12	0.11	0.13	0.13	0.12

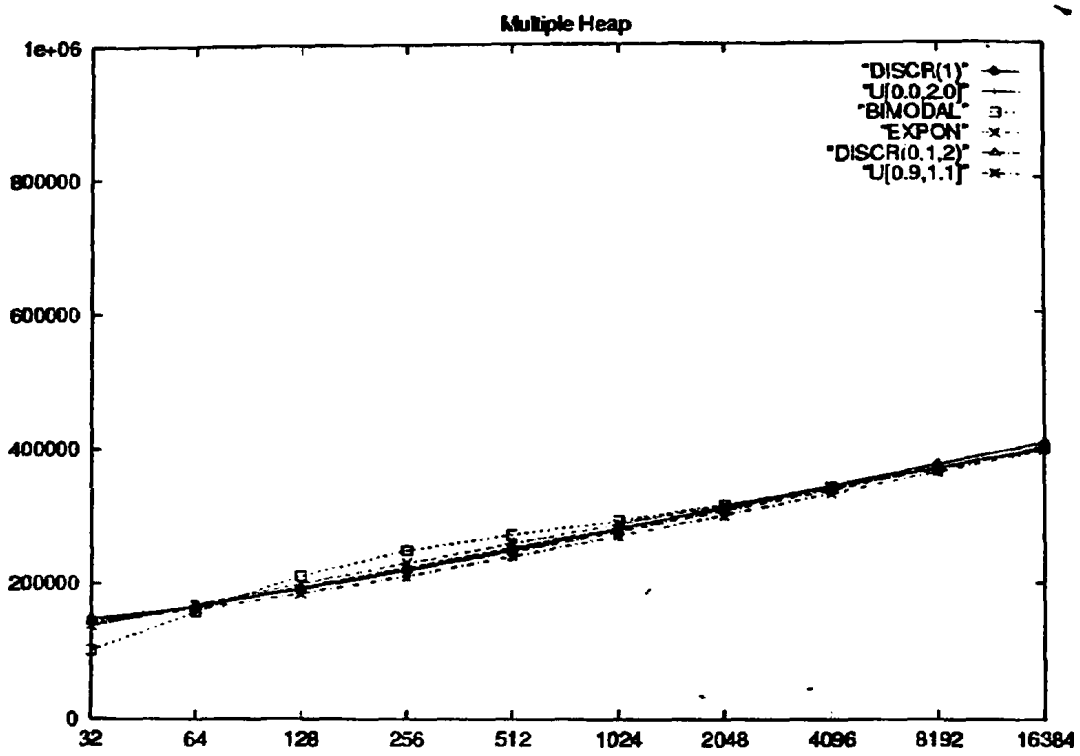
Σχήμα 8.39. Οι χρόνοι επεξεργασίας για τη δομή *M*-heap.



Σχήμα 8.40. Οι χρόνοι επεξεργασίας για τη δομή *M*-heap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	137254	140182	143862	101084	148000	143878
64	168447	166439	162059	158994	166000	167493
128	211018	194086	184915	211018	191000	192758
256	229712	222680	210594	248590	219436	219640
512	259046	252214	239654	272056	249684	247080
1024	287500	281722	269918	292212	280548	276571
2048	315116	311236	300944	315761	311862	307418
4096	343114	340801	332411	340862	343638	337500
8192	370718	369848	364213	369744	375526	368450
16384	399194	399842	396046	399225	407470	398145

Σχήμα 8.41. Οι συγκρίσεις για τη δομή *M*-heap.



Σχήμα 8.42. Οι συγκρίσεις για τη δομή *M*-heap.

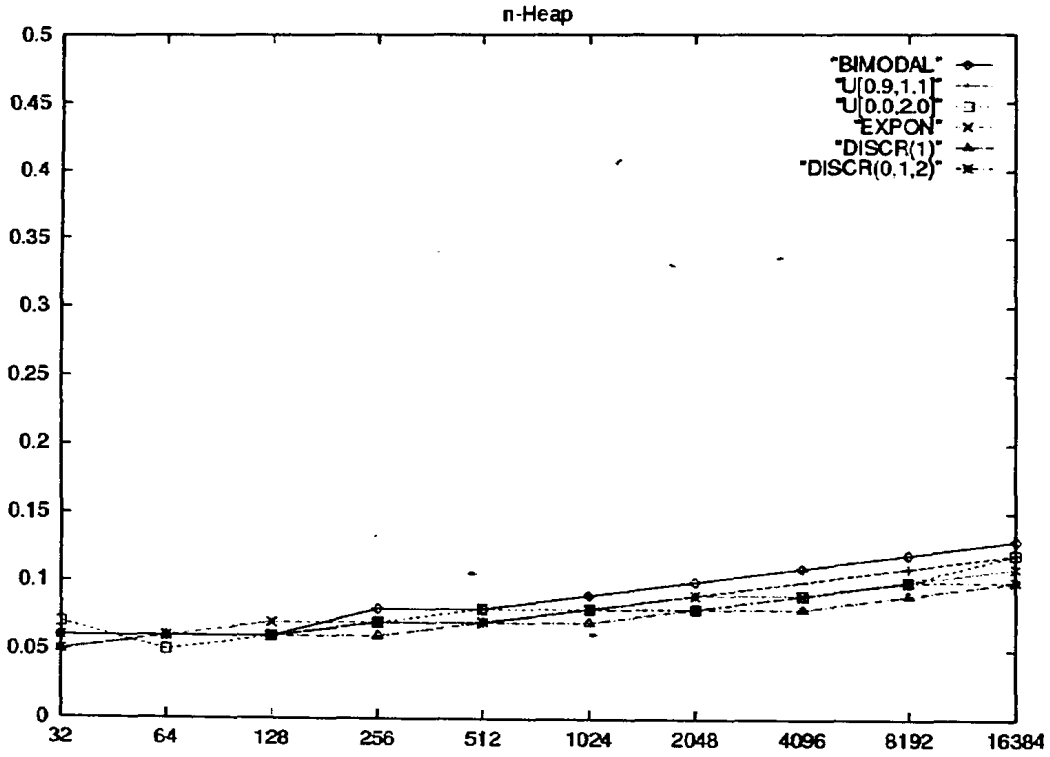
Η Δομή n-heap

Έστω *n* το πλήθος των στοιχείων που υπάρχουν σε κάθε κόμβο της δομής. Τα αποτελέσματα που παρουσιάζονται στη συνέχεια προέρχονται από μια δομή *n*-heap για την οποία ισχύει *n* = 8. Σύμφωνα με την πειραματική μελέτη που πραγματοποιήθηκε, αν η τιμή του *n* επιλεγεί να είναι είτε μικρότερη είτε μεγαλύτερη, η δομή είναι λιγότερο αποδοτική για τις διαδικασίες της εισαγωγής και της διαγραφής.

Όπως φαίνεται και στα Σχήματα 8.43-8.46, όπου παρουσιάζονται οι χρόνοι επεξεργασίας και οι συγκρίσεις που πραγματοποιούνται σε πίνακες και γραφικές παραστάσεις, ο αλγόριθμος είναι εξίσου αποτελεσματικός ανεξάρτητα από την κατανομή από την οποία προέρχονται κάθε φορά τα στοιχεία, κάτι που ισχύει εξάλλου και για την απλή δομή του σωρού.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.05	0.07	0.05	0.06	0.05	0.06
64	0.06	0.05	0.06	0.06	0.06	0.06
128	0.07	0.06	0.06	0.07	0.06	0.05
256	0.07	0.07	0.07	0.08	0.06	0.06
512	0.07	0.07	0.07	0.08	0.07	0.08
1024	0.08	0.07	0.08	0.08	0.07	0.08
2048	0.09	0.09	0.09	0.10	0.09	0.09
4096	0.10	0.09	0.09	0.11	0.09	0.09
8192	0.11	0.10	0.10	0.12	0.09	0.10
16384	0.12	0.11	0.12	0.14	0.11	0.10

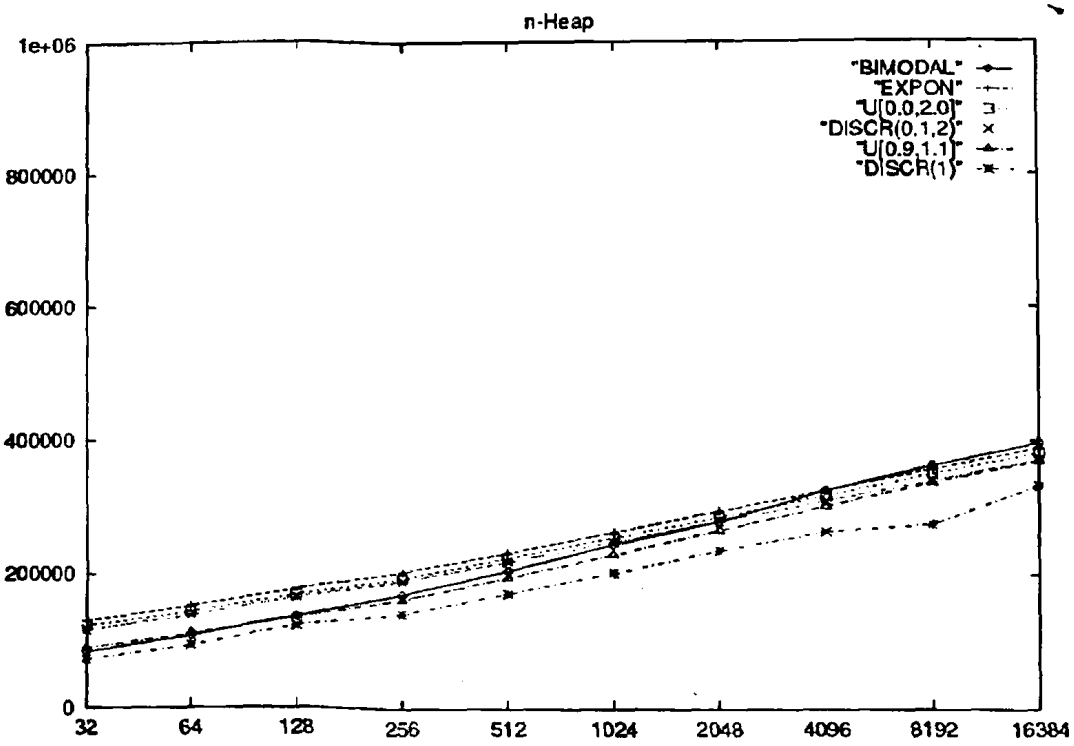
Σχήμα 8.43. Οι χρόνοι επεξεργασίας για τη δομή *n*-heap.



Σχήμα 8.44. Οι χρόνοι επεξεργασίας για τη δομή *n*-heap.

N \ D	EXP	U02	U09	BIM	D1	D012
32	130390	122056	88520	82954	72000	115396
64	151995	143571	109738	106993	92500	138097
128	135917	168168	134426	135917	122000	164305
256	205153	195349	163679	171511	142200	191176
512	233053	225638	196111	206392	172578	219986
1024	263797	255490	230288	245305	203102	249153
2048	295525	286337	266524	279999	235942	280216
4096	326137	318087	302431	325906	264308	310439
8192	357943	350764	337586	363243	275406	340731
16384	389470	382117	370153	396800	332942	372735

Σχήμα 8.45. Οι συγκρίσεις για τη δομή *n*-heap.



Σχήμα 8.46. Οι συγκρίσεις για τη δομή *n*-heap.

8.5 Επεκτάσεις της Δομής του *P*-tree

Η Δομή Multiple P-tree (MP-tree)

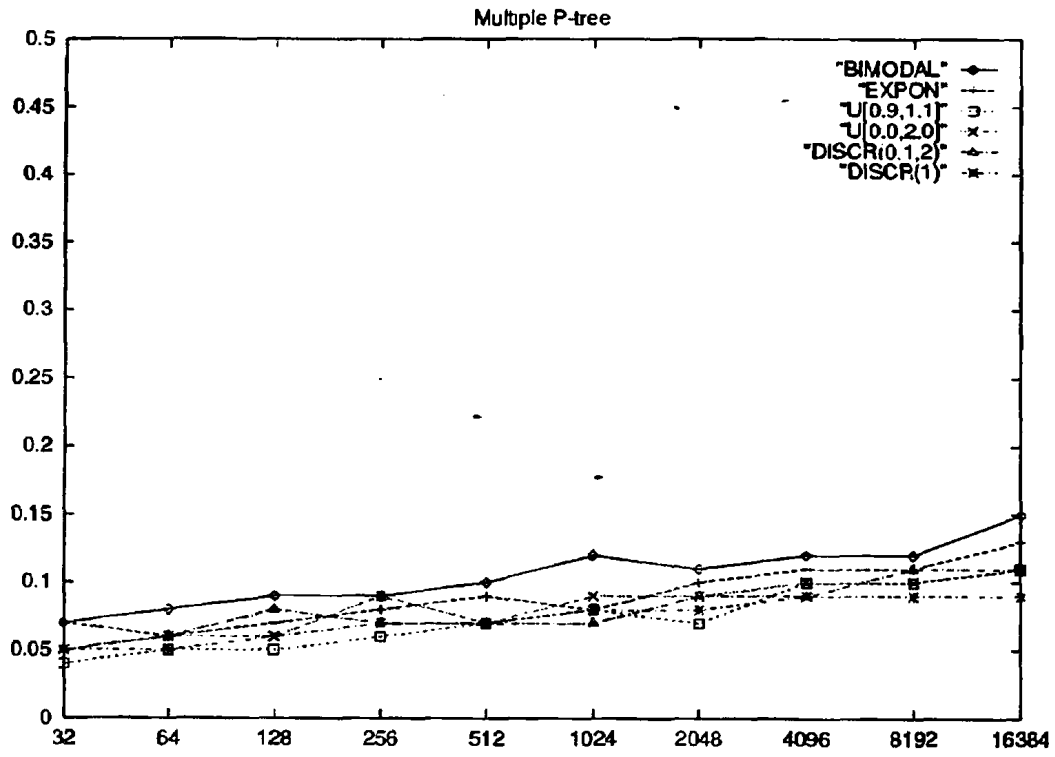
Έστω M το πλήθος των *P*-trees που αποτελούν τη δομή. Διαπιστώθηκε πειραματικά ότι η απόδοση του αλγόριθμου είναι μέγιστη όταν η τιμή του M είναι ικανοποιητικά μεγάλη. Συγκεκριμένα, αν N είναι το πλήθος των στοιχείων που υπάρχουν στη δομή, όταν ισχύει $M \approx N/16$ η δομή είναι πολύ αποτελεσματική και μάλιστα ανεξάρτητα από την κατανομή από την οποία προέρχονται τα στοιχεία της. Υπενθυμίζεται ότι η τιμή του M καθορίζει το μέγεθος της δομής του σωρού *I*-heap αλλά και το πλήθος των στοιχείων που θα περιέχει κάθε ένα από τα *P*-trees. Τα αποτελέσματα που παρουσιάζονται στη συνέχεια, στα Σχήματα 8.47-8.50, προκύπτουν για τιμή $M = N/16$. Ο αλγόριθμος παρουσιάζει σημαντική βελτίωση στην απόδοσή του σε σύγκριση με την απλή δομή του *P*-tree κυρίως όταν χρησιμοποιείται η κατανομή DISCRETE(0,1,2).

Μελετήθηκε ακόμη το αποτέλεσμα της χρήσης της τεχνικής Event Horizon σε μια τροποποιημένη δομή *MP*-tree. Έτσι, αν η δομή *I*-heap είναι λίστα και όχι σωρός και η δευτερεύουσα δομή είναι μια δομή *P*-tree η οποία εισάγεται στην κύρια αυξάνοντας την τιμή του M κατά 1 κάθε φορά που το μικρότερο στοιχείο εντοπίζεται σε αυτή, σύμφωνα με τα αποτελέσματα της πειραματικής μελέτης, η χρήση της τεχνικής Event Horizon όχι μόνο δεν βελτιώνει την απόδοση της δομής, αλλά την καθιστά λιγότερο αποτελεσματική. Σημειώνεται

ακόμη ότι το πλήθος M των P-trees δεν παραμένει σταθερό και μάλιστα μπορεί να ισχύει $M = 1$, όπως για παράδειγμα στην περίπτωση της κατανομής DISCRETE(1).

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.06	0.05	0.04	0.08	0.05	0.05
64	0.06	0.06	0.05	0.09	0.03	0.05
128	0.09	0.06	0.06	0.09	0.05	0.06
256	0.07	0.06	0.06	0.09	0.06	0.07
512	0.08	0.07	0.07	0.09	0.06	0.07
1024	0.09	0.07	0.07	0.11	0.07	0.06
2048	0.10	0.10	0.08	0.11	0.07	0.09
4096	0.10	0.11	0.10	0.12	0.09	0.09
8192	0.11	0.11	0.11	0.13	0.09	0.10
16384	0.14	0.12	0.11	0.15	0.09	0.11

Σχήμα 8.47. Οι χρόνοι επεξεργασίας για τη δομή *MP*-tree.



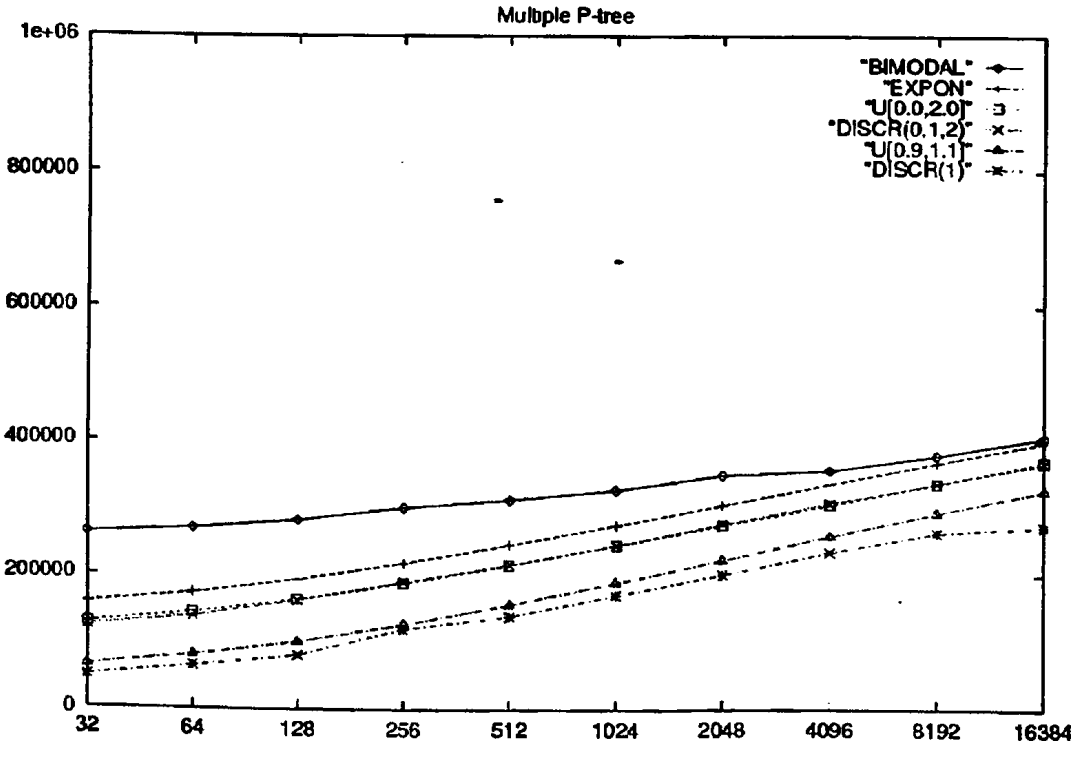
Σχήμα 8.48. Οι χρόνοι επεξεργασίας για τη δομή *MP*-tree.

Αξίζει ακόμη να σημειωθεί ότι έχουν μελετηθεί και μερικές άλλες δομές που βασίζονται στη δομή *MP*-tree, οι οποίες όμως δεν είναι το ίδιο αποτελεσματικές όσο η τελευταία. Συγκεκριμένα, αν η δομή *I*-heap δεν είναι σωρός αλλά ταξινομημένη, διπλά συνδεδεμένη λίστα ή ακόμη και η γνωστή δομή Two Level, η δομή *MP*-tree γίνεται λιγότερο αποδοτική. Ανάλογα αποτελέσματα προέκυψαν για την περίπτωση κατά την οποία το αρχικό πλήθος των P-trees είναι ένα και αυξάνεται σταδιακά μέχρι την τιμή M. Ακόμη, στην περίπτωση κατά την οποία η εισαγωγή ενός νέου στοιχείου δεν πραγματοποιείται στο P-tree που περιέχει το μικρότερο στοιχείο αλλά σε κάποιο άλλο που επιλέγεται με κάποια συγκεκριμένα κριτήρια, τα πειραματικά αποτελέσματα που προκύπτουν δεν είναι το ίδιο ικανοποιητικά. Το P-tree στο

οποίο γίνεται η εισαγωγή ενός νέου στοιχείου μπορεί να είναι ένα από τα γειτονικά δέντρα του *P*-tree από το οποίο πραγματοποιήθηκε η τελευταία διαδικασία διαγραφής. Επίσης, για την διαδικασία της εισαγωγής μπορεί να επιλέγεται ένα διαφορετικό δέντρο κάθε φορά έτσι ώστε να επιλέγονται διαδοχικά και με τη σειρά όλα τα *P*-trees της δομής. Μετά την πειραματική μελέτη των παραπάνω δομών, δεν κρίνεται απαραίτητη η παρουσίαση των αποτελεσμάτων που προέκυψαν, αφού η δομή *MP*-tree όπως έχει περιγραφεί σε προηγούμενο Κεφάλαιο είναι αποτελεσματικότερη.

N \ D	EXP	U02	U09	BIN	D1	D012
32	157225	127113	63658	261629	48000	122175
64	173163	144069	80263	269948	64000	138540
128	282081	164002	100611	282081	80500	162298
256	218004	188009	126514	300513	119688	190071
512	245093	215082	156135	312288	138544	215399
1024	274257	244601	188756	327243	170420	244799
2048	304524	274635	222737	349261	201218	276376
4096	336228	304925	258245	356213	234612	307628
8192	367434	336639	292862	378322	263394	337005
16384	398588	369331	326837	405808	273160	367327

Σχήμα 8.49. Οι συγκρίσεις για τη δομή *MP*-tree.



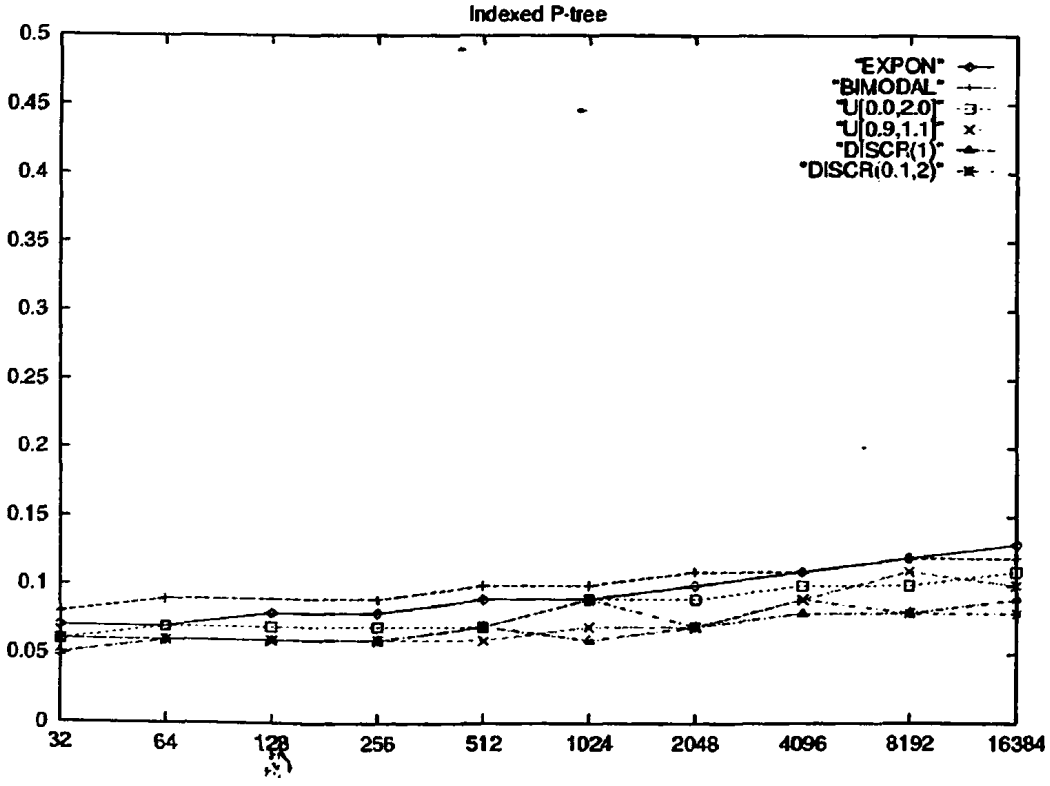
Σχήμα 8.50. Οι συγκρίσεις για τη δομή *MP*-tree.

Η Δομή Indexed P-tree (IP-tree)

Η αποτελεσματικότητα της δομής *IP*-tree σε σχέση με τη δομή *P*-tree είναι προφανής αν μελετήσει κανείς τα πειραματικά αποτελέσματα που προκύπτουν για τη δομή *IP*-tree. Η βελτίωση της αποτελεσματικότητας της δομής του *P*-tree είναι μεγαλύτερη για την περίπτωση κατά την οποία χρησιμοποιείται η κατανομή DISCRETE(0,1,2). Έτσι, ο αλγόριθμος είναι το ίδιο αποτελεσματικός είτε χρησιμοποιείται η κατανομή DISCRETE(1) είτε η DISCRETE(01,2) είτε τελικά οποιαδήποτε από τις κατανομές που μελετήθηκαν. Τα αποτελέσματα της πειραματικής μελέτης παρουσιάζονται αναλυτικά στα Σχήματα 8.51-8.54 που ακολουθούν.

N \ D	EXP	U02	U09	BIM	D1	D012
32	0.07	0.06	0.05	0.08	0.05	0.06
64	0.07	0.07	0.06	0.08	0.06	0.06
128	0.09	0.06	0.06	0.09	0.07	0.06
256	0.08	0.08	0.06	0.09	0.06	0.07
512	0.08	0.08	0.07	0.09	0.06	0.07
1024	0.09	0.08	0.08	0.10	0.07	0.08
2048	0.10	0.09	0.08	0.10	0.07	0.08
4096	0.10	0.09	0.08	0.11	0.07	0.07
8192	0.11	0.10	0.09	0.12	0.08	0.08
16384	0.12	0.10	0.10	0.13	0.09	0.07

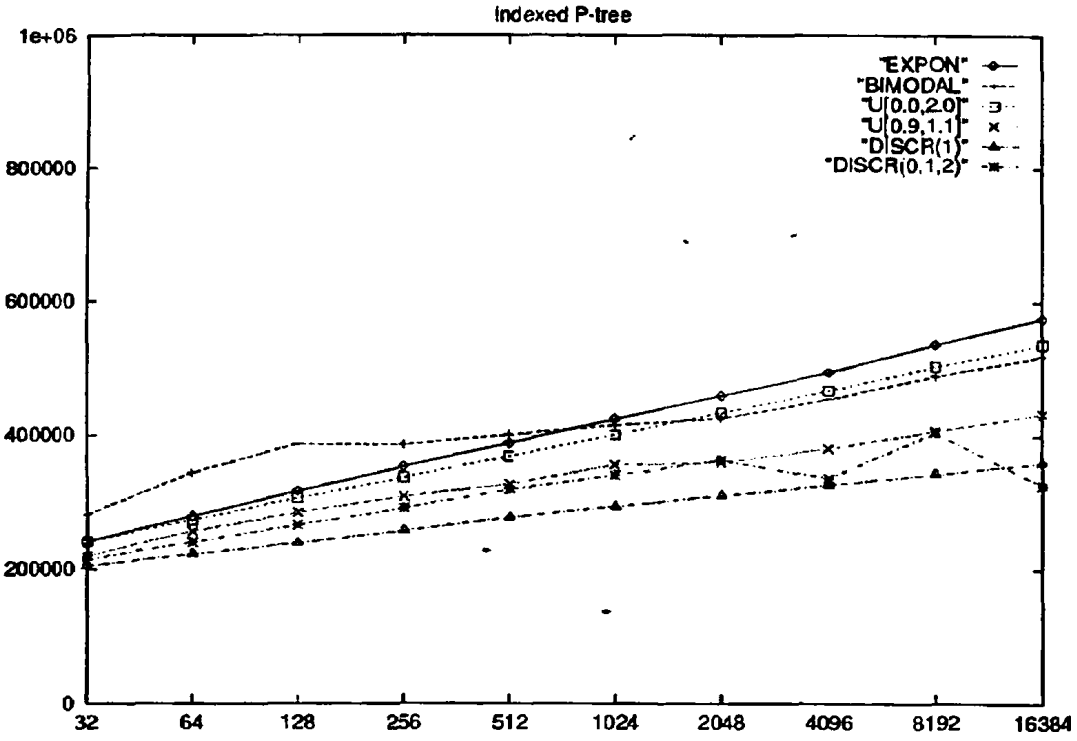
Σχήμα 8.51. Οι χρόνοι επεξεργασίας για τη δομή *IP*-tree.



Σχήμα 8.52. Οι χρόνοι επεξεργασίας για τη δομή *IP*-tree.

N \ D	EXP	U02	U09	BIM	D1	D012
32	241276	241188	220062	281737	205000	215122
64	279427	273993	256997	344143	222500	240623
128	386925	306373	284909	386925	239250	266140
256	352950	337246	309303	386091	257542	291512
512	388385	368456	327311	401095	277952	319541
1024	424866	401192	356410	415793	294432	341152
2048	459925	434714	360363	426456	310864	363874
4096	494970	467508	381187	454670	327272	336841
8192	537300	504202	407519	490109	343668	405073
16384	575889	537010	433329	519295	360058	326381

Σχήμα 8.53. Οι συγκρίσεις για τη δομή *IP*-tree.



Σχήμα 8.54. Οι συγκρίσεις για τη δομή *IP*-tree.

8.6 Συμπεράσματα

Έχοντας ολοκληρώσει την πειραματική μελέτη όλων των αλγορίθμων, είναι κανείς σε θέση να καταλήξει σε κάποια συμπεράσματα σχετικά με την αποτελεσματικότητά τους. Έτσι, η δομή του σωρού είναι πολύ απλή και εξαιρετικά αποδοτική. Η στατική αναπαράστασή του μειονεκτεί σε σχέση με τη δυναμική στο ότι είναι απαιτητική σε μνήμη, αφού πρέπει να ορισθεί ένα διάνυσμα μέγιστου μήκους για την αποθήκευση των γεγονότων. Όμως, η διάσχιση του δένδρου που απαιτείται κατά τη δυναμική αναπαράσταση για την ολοκλήρωση των διεργασιών εισαγωγής και διαγραφής, αποτελεί μειονέκτημα έναντι της στατικής. Επιπλέον, με

βάση τα αποτελέσματα της πειραματικής μελέτης, η τεχνική Event Horizon δεν επηρεάζει την αποτελεσματικότητα της δομής.

Οι υπόλοιπες από τις βασικές δομές που μελετήθηκαν, Leftist heap, Binomial heap και *P*-tree δεν είναι ιδιαίτερα αποτελεσματικές αν και οι δύο πρώτες έχουν πολύ αποδοτική διαδικασία συγχώνευσης. Επίσης, η εφαρμογή της τεχνικής Event Horizon δεν προκαλεί σημαντική βελτίωση στην απόδοση των αλγορίθμων που χρησιμοποιούν της δομές που προαναφέρθηκαν. Σημειώνεται ακόμη ότι η δομή του *P*-tree δεν είναι καθόλου αποτελεσματική όταν τα στοιχεία προέρχονται από την κατανομή DISCRETE(0,1,2).

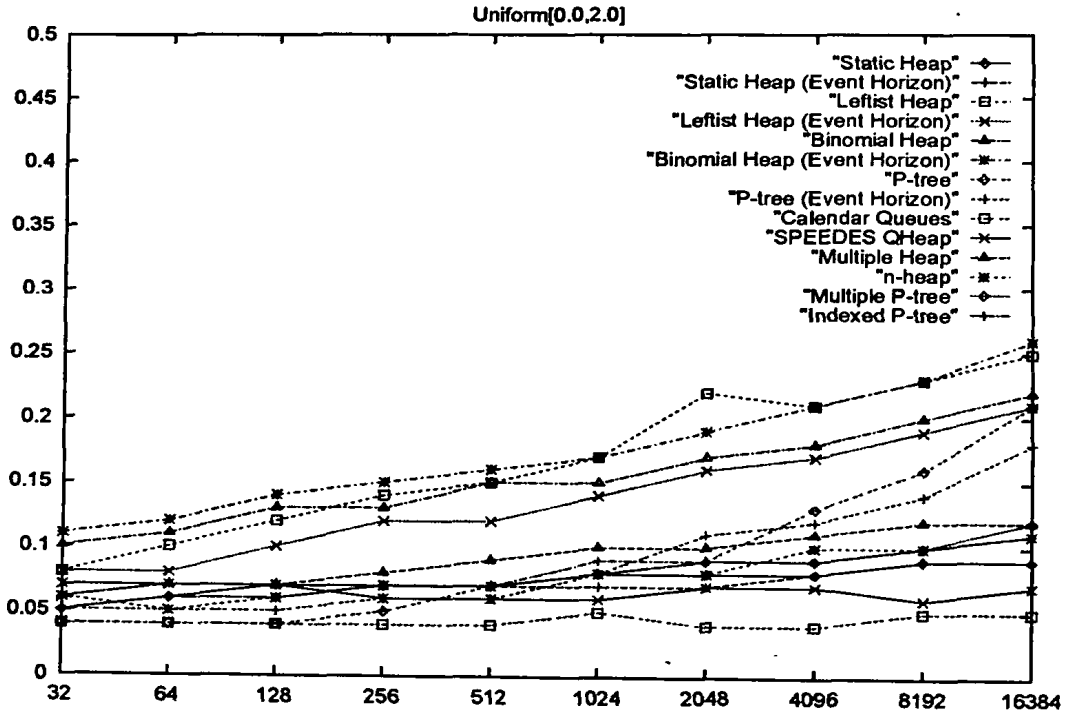
Η δομή Calendar Queues είναι εξαιρετικά αποδοτική όταν τα στοιχεία της προέρχονται από οποιαδήποτε από τις συνεχείς κατανομές που μελετήθηκαν. Η χρήση της όμως είναι απαγορευτική σε περίπτωση που τα στοιχεία προέρχονται από τις κατανομές DISCRETE(1) και DISCRETE(0,1,2), αφού, όπως έχει ήδη αναφερθεί, η δομή τότε ουσιαστικά εκφυλλίζεται σε λίστα. Αντίθετα, η δεύτερη από τις σύνθετες δομές που μελετήθηκαν, η δομή SPEEDES Qheap είναι εξαιρετικά αποτελεσματική ανεξάρτητα από την εκάστοτε κατανομή. Συγκεκριμένα, παρουσιάζει απόδοση καλύτερη από εκείνη της δομής του σωρού, δεν είναι όμως το ίδιο απλή.

Οι επεκτάσεις της δομής του σωρού που προτάθηκαν και μελετήθηκαν, δηλαδή οι δομές *n*-heap και *M*-heap απαιτούν χρόνους επεξεργασίας ανάλογους με εκείνους της δομής του σωρού. Το πλήθος των συγκρίσεων όμως που πραγματοποιούνται είναι μικρότερο από το αντίστοιχο της απλής δομής του σωρού. Έτσι, οι επεκτάσεις της δομής του σωρού βελτιώνουν την απόδοση της απλής δομής του σωρού.

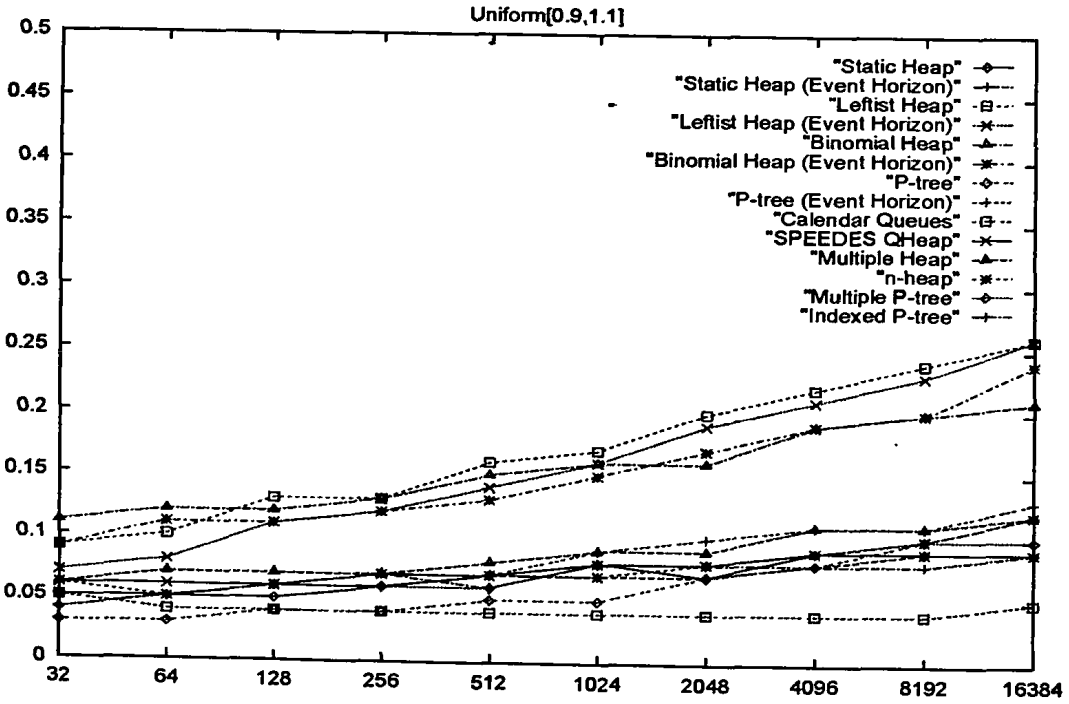
Οι δομές *MP*-tree και *IP*-tree αποτελούν εξαιρετικές επεκτάσεις της δομής του *P*-tree, αφού βελτιώνεται κατά πολύ μεγάλο βαθμό η αποτελεσματικότητα της τελευταίας. Η δομή *IP*-tree είναι περισσότερο αποτελεσματική από τη δομή *MP*-tree, αλλά και οι δυο δομές είναι σχεδόν εξίσου αποτελεσματικές, με κριτήριο το χρόνο επεξεργασίας, ακόμη και από τη δομή του σωρού. Αν συγκρίνει κανείς τις δομές *MP*-tree και *IP*-tree με τη δομή του σωρού με κριτήριο το πλήθος των συγκρίσεων, διαπιστώνει ότι είναι περισσότερο αποτελεσματικές. Στο επόμενο κεφάλαιο πραγματοποιείται πιο αναλυτικά σύγκριση μεταξύ όλων των αλγορίθμων που μελετήθηκαν.

Τέλος, στα Σχήματα 8.55-8.60 παρουσιάζεται η αποτελεσματικότητα όλων των αλγορίθμων για κάθε μια από τις κατανομές. Αν μελετήσει κανείς τα σχήματα, συμπεραίνει ότι για τις συνεχείς κατανομές οι δομές SPEEDES Qheap, Calendar queues, Static heap καθώς και οι νέες δομές που προτείνονται (επεκτάσεις της δομής του σωρού και του *P*-tree) είναι πολύ αποτελεσματικές.

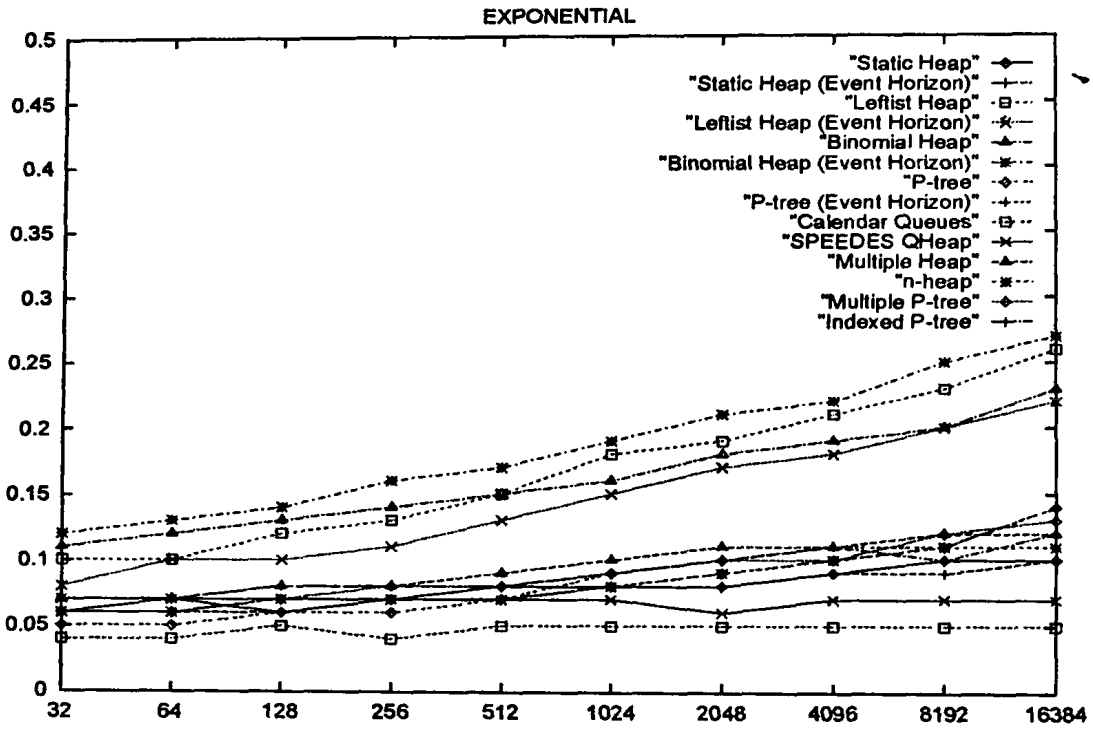
Όταν χρησιμοποιείται η κατανομή Discrete(1), πιο αποτελεσματική δομή είναι η δομή του *P*-tree, αφού σε αυτή την περίπτωση η δομή γίνεται λίστα και μάλιστα οι διαδικασίες της εισαγωγής και της διαγραφής πραγματοποιούνται σε σταθερό χρόνο. Αντίθετα, η δομή Calendar queues είναι πολύ αναποτελεσματική, τόσο με την κατανομή Discrete(1) όσο και με την Discrete(0,1,2). Το γεγονός αυτό ισχύει διότι τα buckets της δομής περιέχουν πολύ μεγάλες λίστες μια και τα στοιχεία συγκεντρώνονται σε μερικά μόνο από αυτά. Κατά την χρήση της κατανομής Discrete(0,1,2) η δομή του *P*-tree είναι επίσης αναποτελεσματική, αφού σε αυτή την περίπτωση η δομή εκφυλλίζεται σε τρεις λίστες.



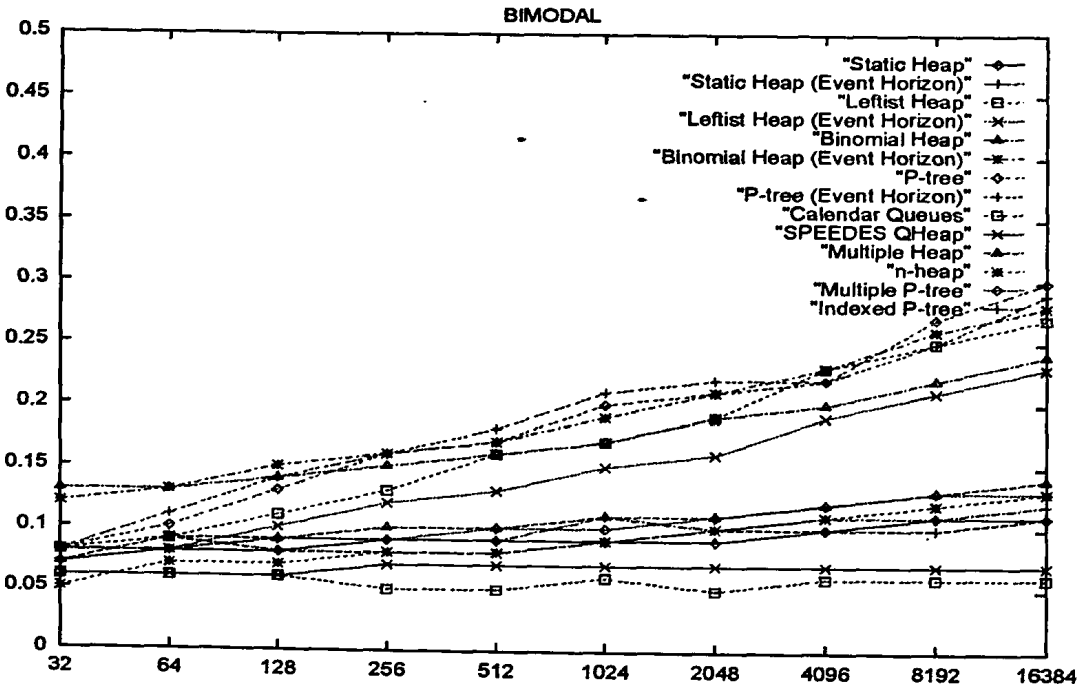
Σχήμα 8.55. Η κατανομή Uniform[0.0,2.0].



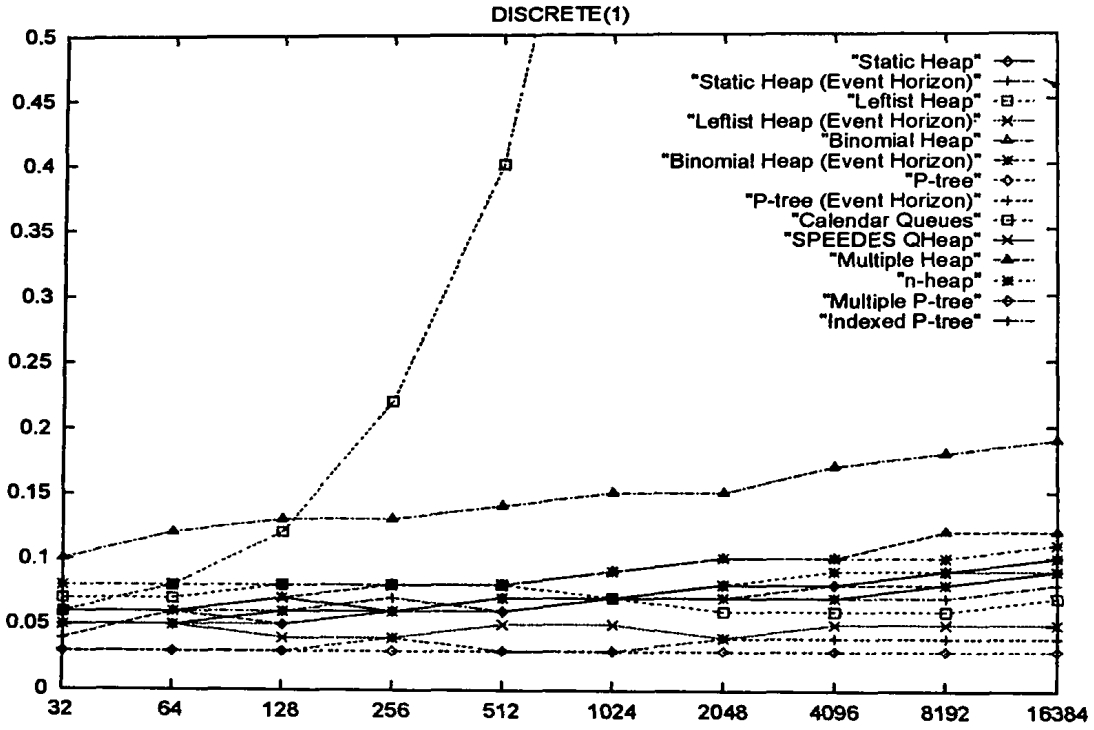
Σχήμα 8.56. Η κατανομή Uniform[0.9,1.1].



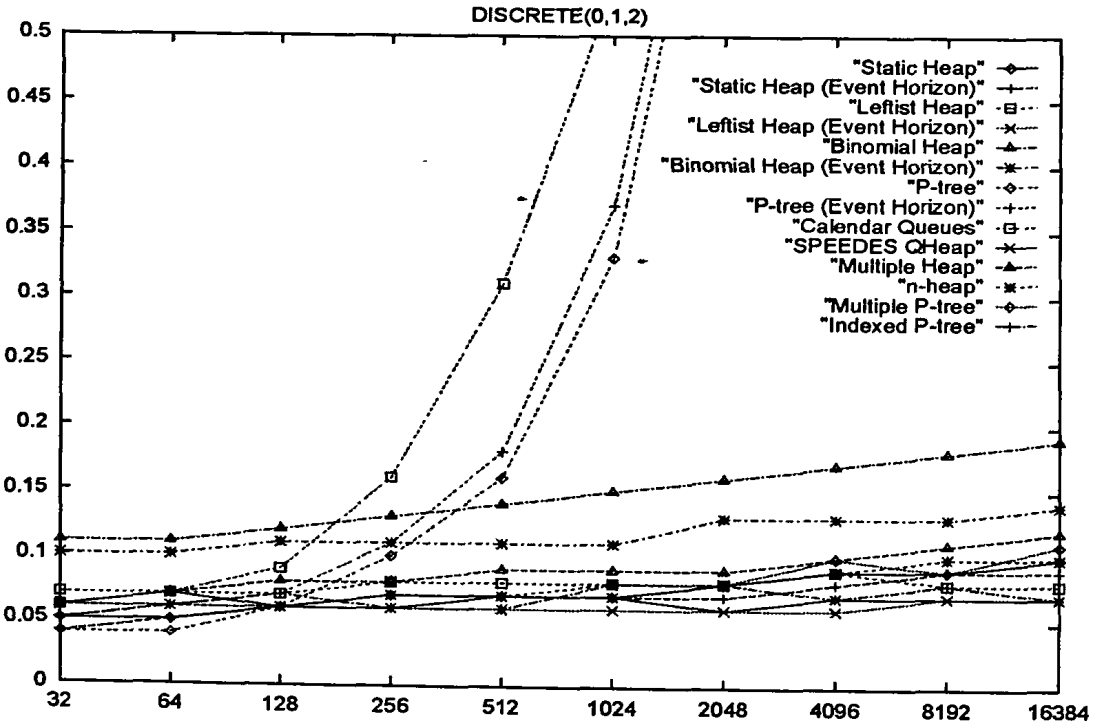
Σχήμα 8.57. Η κατανομή Exponential.



Σχήμα 8.58. Η κατανομή Bimodal.



Σχήμα 8.59. Η κατανομή Discrete(1).



Σχήμα 8.60. Η κατανομή Discrete(0,1,2).

ΚΕΦΑΛΑΙΟ 9

ΣΥΓΚΡΙΣΗ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ - ΣΥΜΠΕΡΑΣΜΑΤΑ - ΕΠΕΚΤΑΣΕΙΣ

9.1 Ακολουθιακοί Αλγόριθμοι

Με βάση την πειραματική μελέτη που πραγματοποιήθηκε θα μπορούσε κανείς να κατατάξει τους αλγορίθμους σε τρεις κατηγορίες, ανάλογα με την αποτελεσματικότητά τους. Έτσι, μη αποδοτικοί, σε σύγκριση πάντα με τους υπόλοιπους, θα μπορούσαν να χαρακτηριστούν οι αλγόριθμοι που χρησιμοποιούν τις δομές Calendar Queues και *P*-tree, αφού απαιτούν μεγάλο χρόνο επεξεργασίας κυρίως για τις διακριτές κατανομές (Κατηγορία Ι). Στην Κατηγορία ΙΙ θα μπορούσαν να ομαδοποιηθούν οι αλγόριθμοι που χρησιμοποιούν τις δομές Leftist heap και Binomial heap με ή χωρίς την τεχνική Event Horizon καθώς και ο αλγόριθμος που χρησιμοποιεί τη δομή *P*-tree με χρήση της τεχνικής Event Horizon. Τέλος, πολύ αποτελεσματικοί θα μπορούσαν να χαρακτηριστούν οι αλγόριθμοι που χρησιμοποιούν τις δομές SPEEDES Qheap, Static heap, *n*-heap, *M*-heap, *MP*-tree και *IP*-tree, οι οποίοι και σχηματίζουν την Κατηγορία ΙΙΙ. Η ομαδοποίηση αυτή των αλγορίθμων παρουσιάζεται στον Πίνακα 9.1.

Αφού πολλοί από τους αλγορίθμους έχουν διαφορετική απόδοση ανάλογα με την κατανομή που χρησιμοποιείται, πραγματοποιήθηκε η μελέτη τους χρησιμοποιώντας όχι μόνο μια κατανομή κάθε φορά αλλά όλες μαζί με βάση κάποιον αλγόριθμο επιλογής. Συγκεκριμένα,

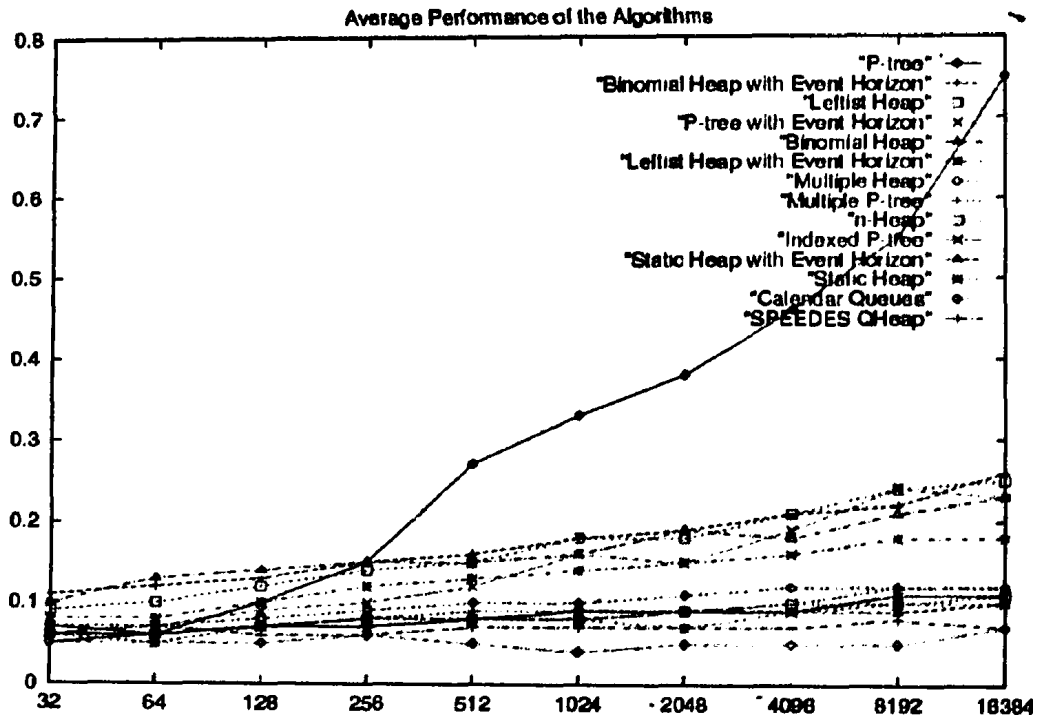
επιλέγεται τυχαία κάποια κατανομή και χρησιμοποιείται κατά ένα χρονικό διάστημα κατά την εκτέλεση του αλγόριθμου. Στη συνέχεια του αλγορίθμου επιλέγεται ξανά τυχαία κάποια από τις κατανομές και η διαδικασία συνεχίζεται μέχρι να τερματιστεί ο αλγόριθμος. Έτσι, είναι δυνατή η εκτίμηση της αποτελεσματικότητας κάθε ενός από τους αλγορίθμους με βάση ένα διαφορετικό κριτήριο. Με αυτό τον τρόπο επιχειρείται η καλύτερη αξιολόγηση των αλγορίθμων αφού εκτελούνται υπό διαφορετικές συνθήκες. Οι χρόνοι επεξεργασίας που προκύπτουν με τον τρόπο που περιγράφηκε παραπάνω παρουσιάζονται στον Πίνακα 9.2, ενώ το Σχήμα 9.1 παρουσιάζει τα ίδια αποτελέσματα με τη μορφή γραφικών παραστάσεων. Το πλήθος των συγκρίσεων που πραγματοποιεί κάθε αλγόριθμος παρουσιάζεται στον Πίνακα 9.3 και σε μορφή γραφικής παράστασης στο Σχήμα 9.2. Από τα σχήματα και τους πίνακες επιβεβαιώνεται η ομαδοποίηση των αλγορίθμων που περιγράφηκε παραπάνω, με εξαίρεση τον αλγόριθμο που χρησιμοποιεί τη δομή Calendar Queues η οποία είναι πολύ αποτελεσματική, αφού με τη χρήση όλων των κατανομών δεν εμφανίζονται τα προβλήματα που παρουσιάζονται όταν χρησιμοποιούνται οι διακριτές (DISCRETE(1) και DISCRETE(0,1,2)). Ο Πίνακας 9.4 παρουσιάζει μια κατάταξη των αλγορίθμων με κριτήριο τα αποτελέσματα του Πίνακα 9.2.

ΚΑΤΗΓΟΡΙΑ I	ΚΑΤΗΓΟΡΙΑ II	ΚΑΤΗΓΟΡΙΑ III
Calendar queues	***P*-tree (E.H)**	**Static heap**
***P*-tree**	**Leftist heap**	**Static heap (E.H)**
	Leftist heap (E.H)	**SPEEDES Qheap**
	Binomial heap	***n*-heap**
	Binomial heap (E.H)	***M*-heap**
		***MP*-tree**
		***IP*-tree**

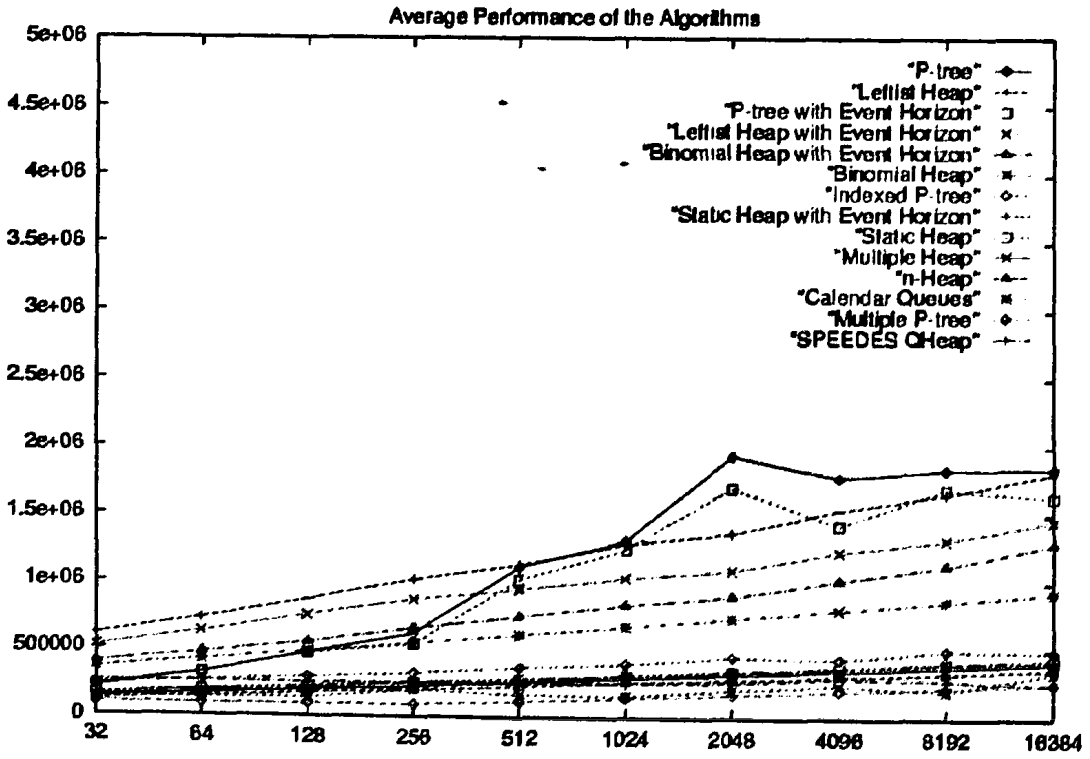
Πίνακας 9.1.

Algorithm \ *N*	32	64	128	256	512	1024	2048	4096	8192	16384
Static heap	0.06	0.05	0.07	0.07	0.08	0.08	0.07	0.09	0.10	0.10
Static heap(E.H)	0.06	0.06	0.07	0.08	0.08	0.08	0.09	0.09	0.09	0.10
Leftist heap	0.09	0.10	0.12	0.14	0.15	0.18	0.18	0.21	0.24	0.25
Leftist heap (E.H)	0.08	0.08	0.10	0.12	0.13	0.14	0.15	0.16	0.18	0.18
Binomial heap	0.10	0.13	0.14	0.15	0.15	0.16	0.19	0.18	0.21	0.23
Binomial heap (E.H)	0.11	0.12	0.13	0.15	0.16	0.18	0.19	0.21	0.22	0.26
- ***P*-tree**	0.05	0.06	0.10	0.15	0.27	0.33	0.38	0.46	0.55	0.75
***P*-tree (E.H)**	0.06	0.07	0.07	0.10	0.12	0.16	0.17	0.19	0.23	0.24
Calendar queues	0.05	0.05	0.05	0.06	0.05	0.04	0.05	0.05	0.05	0.07
SPEEDES Qheap	0.06	0.07	0.06	0.06	0.07	0.07	0.07	0.07	0.08	0.07
***n*-heap**	0.06	0.06	0.07	0.08	0.08	0.08	0.09	0.10	0.10	0.11
***M*-heap**	0.07	0.07	0.08	0.09	0.10	0.10	0.11	0.12	0.12	0.12
***MP*-tree**	0.06	0.06	0.07	0.08	0.09	0.09	0.09	0.10	0.12	0.12
***IP*-tree**	0.07	0.06	0.07	0.07	0.08	0.09	0.09	0.09	0.11	0.11

Πίνακας 9.2. Οι χρόνοι επεξεργασίας.



Σχήμα 9.1. Οι χρόνοι επεξεργασίας.



Σχήμα 9.2. Το πλήθος των συγκρίσεων.

N / *Algorithm*	32	64	128	256	512	1024	2048	4096	8192	16384
Static heap	155661	188012	217676	249717	278351	309878	340309	370043	406140	429668
Static heap(E.H)	156473	188502	219299	250786	280934	312042	342905	372435	408402	431928
Leftist heap	603878	727575	869836	1016258	1130820	1282528	1364173	1532878	1654642	1806660
Leftist heap(E.H)	516942	626591	753980	868633	946361	1034764	1091980	1231404	1317768	1455738
Binomial heap	352000	416000	480000	544000	608000	672000	736000	800000	864000	928000
Binomial heap(E.H)	391212	466686	550305	653716	745867	832540	898891	1019700	1128000	1290071
P-tree	208672	316148	469903	615455	1112267	1306006	1933576	1772335	1833560	1842175
P-tree(E.H)	219048	316156	465725	536679	1018575	1246940	1699971	1420350	1691204	1631503
Calendar queues	135835	133586	136932	196599	166678	159300	207820	245200	201316	340468
SPEEDES Qheap	250141	257388	248273	255271	247325	254958	259262	295541	280643	247609
n-heap	116177	138663	172989	199642	227466	260035	279852	308548	330950	364696
M-heap	134492	164123	196401	224177	257932	290027	318915	348142	377030	407188
MP-tree	95412	88722	92480	91464	116284	143814	172982	198982	227172	255758
IP-tree	227202	258177	290727	320019	365957	396017	450314	436059	501804	493353

Πίνακας 9.3. Το πλήθος των συγκρίσεων.

32	128	512	1024	4096	16384
Calendar queues	Calendar queues	Calendar queues	Calendar queues	Calendar queues	Calendar queues
P-tree	SPEEDES Qheap	SPEEDES Qheap	SPEEDES Qheap	SPEEDES Qheap	SPEEDES Qheap
MP-tree	*IP*-tree	*IP*-tree	*n*-heap	*IP*-tree	Static heap
n-heap	*MP*-tree	*n*-heap	Static heap	Static heap	Static heap(E.H)
SPEEDES Qheap	*n*-heap	Static heap	Static heap(E.H)	Static heap(E.H)	*IP*-tree
P-tree (E.H)	*P*-tree (E.H)	Static heap(E.H)	*IP*-tree	*MP*-tree	*n*-heap
Static heap	Static heap	*MP*-tree	*MP*-tree	*n*-heap	*MP*-tree
Static heap(E.H)	Static heap(E.H)	*M*-heap	*M*-heap	*M*-heap	*M*-heap
IP-tree	*M*-heap	*P*-tree (E.H)	Leftist heap(E.H)	Leftist heap(E.H)	Leftist heap(E.H)
M-heap	*P*-tree	Leftist heap(E.H)	*P*-tree (E.H)	Binomial heap	Binomial heap
Leftist heap(E.H)	Leftist heap(E.H)	Leftist heap	Binomial heap	*P*-tree (E.H)	*P*-tree (E.H)
Leftist heap	Leftist heap	Binomial heap	Binomial heap(E.H)	Leftist heap	Leftist heap
Binomial heap	Binomial heap(E.H)	Binomial heap(E.H)	Leftist heap	Binomial heap(E.H)	Binomial heap(E.H)
Binomial heap(E.H)	Binomial heap	*P*-tree	*P*-tree	*P*-tree	*P*-tree

Πίνακας 9.4.

9.2 Σύγκριση Ακολουθιακών και Παράλληλων Αλγορίθμων

Οι πίνακες που ακολουθούν συνοψίζουν την αποτελεσματικότητα των ακολουθιακών και των παράλληλων αλγορίθμων που έχουν μελετηθεί. Από τους πίνακες είναι εύκολο να συμπεράνει κανείς, λαμβάνοντας υπόψη και το πλήθος των επεξεργαστών που χρησιμοποιούνται, ότι οι παράλληλοι αλγόριθμοι που χρησιμοποιούν τις δομές Leftist heap, Binomial heap, *P*-tree, *MP*-tree και *IP*-tree είναι το ίδιο αποτελεσματικοί με τους αντίστοιχους ακολουθιακούς, το οποίο προφανώς είναι ένα πάρα πολύ χρήσιμο συμπέρασμα. Οι μεταβλητές *N* και *M* που εμφανίζονται στους πίνακες συμβολίζουν το πλήθος των στοιχείων, το πλήθος των σωρών ή

των *P*-trees που υπάρχουν στη δομή *M*-heap ή *MP*-tree, αντίστοιχα. Σημειώνεται όμως ότι για τον ακολουθιακό αλγόριθμο που χρησιμοποιεί τη δομή *n*-heap, η μεταβλητή *M* αντιστοιχεί στο πλήθος των στοιχείων που υπάρχουν σε κάθε κόμβο. Η μεταβλητή *m* αντιστοιχεί στο πλήθος των στοιχείων που υπάρχουν στη δευτερεύουσα δομή ακριβώς πριν πραγματοποιηθεί η συγχώνευσή της με την κύρια. Τέλος, η μεταβλητή *k* προφανώς αντιστοιχεί στο πλήθος των στοιχείων που εισάγονται ή διαγράφονται ταυτόχρονα.

ΠΑΡΑΛ. ΑΛΓΟΡ.	*k*-ΕΙΣΑΓΩΓΗ	*k*-ΔΙΑΓΡΑΦΗ	ΕΠΕΞΕΡΓΑΣΤΕΣ
Leftist heap	$O(\log N)$	$O(\log(N/k)+\log\log k)$	k
Binomial heap	$O(\log(N+k)+\log^2 k)$	$O(k\log N)$	k
***n*-heap**	$O(\log N)$	$O(\log(N/k)+\log\log k)$	k
***M*-heap**	$O((N/k)\log\log k)$	$O(1)$	N
***P*-tree**	$O((N/k)\log\log k)$	$O(1)$	$4k$
***MP*-tree**	$O((N/k)\log\log k)$	$O(1)$	k
***IP*-tree**	$O((N/km)\log\log k+\log m)$	$O(1)$	$4k$

Πίνακας 9.3. Οι παράλληλοι αλγόριθμοι.

ΑΚΟΛΟΥΘ.ΑΛΓΟΡ.	ΕΙΣΑΓΩΓΗ	ΔΙΑΓΡΑΦΗ
Leftist heap	$O(\log N)$	$O(\log N)$
Binomial heap	$O(\log N)$	$O(\log N)$
***n*-heap**	$O(\log M)$	$O(\log(N/M))$
***M*-heap**	$O(\log(N/M))$	$O(\log k+\log(N/M))$
***P*-tree**	$O(N)$	$O(1)$
***MP*-tree**	$O(N)$	$O(\log M)$
***IP*-tree**	$O((N/m)+\log m)$	$O(1)$

Πίνακας 9.4. Οι ακολουθιακοί αλγόριθμοι.

9.3 Επεκτάσεις

Όπως προέκυψε από τη μελέτη που πραγματοποιήθηκε, δομές που δεν είναι ιδιαίτερα αποτελεσματικές είναι δυνατόν να τροποποιηθούν κατάλληλα ώστε να βελτιωθεί η απόδοσή τους, όπως η δομή *P*-tree οδήγησε στη δομή *IP*-tree. Με βάση τη μελέτη προτείνονται κάποιοι αλγόριθμοι προκειμένου να χρησιμοποιηθούν κατά την προσομοίωση διακριτών γεγονότων που θεωρούνται αποτελεσματικοί.

Η αναζήτηση για ακόμη πιο αποτελεσματικούς αλγορίθμους, είτε ακολουθιακούς είτε παράλληλους, αποτελεί προφανώς ένα πεδίο που απαιτεί περαιτέρω έρευνα. Έτσι, υπάρχουν δομές των οποίων η αποτελεσματικότητα κατά την προσομοίωση του συνόλου γεγονότων δεν έχει μελετηθεί εκτενώς, όπως για παράδειγμα η δομή Soft heap [8]. Επίσης θα μπορούσε να μελετηθεί η επίδραση της τεχνικής Event Horizon σε δομές που δεν έχει εφαρμοστεί, όπως για παράδειγμα η δομή Calendar Queues. Τέλος, αλγόριθμοι που δεν είναι ιδιαίτερα αποτελεσματικοί θα μπορούσαν ίσως να τροποποιηθούν κατάλληλα ώστε να βελτιωθεί η απόδοσή τους. Για παράδειγμα, ο αλγόριθμος που χρησιμοποιεί τη δομή Calendar Queues ο οποίος είναι αναποτελεσματικός όταν οι χρόνοι δρομολόγησης των γεγονότων ακολουθούν κατανομές όπως οι DISCRETE(1) και DISCRETE(0,1,2), θα μπορούσε ίσως να βελτιωθεί με κατάλληλη τροποποίηση για αυτές τις καταστάσεις, αλλάζοντας για παράδειγμα το εύρος των buckets.

Βιβλιογραφία

[1] M. Andreou, S.D. Nikolopoulos, "An Efficient Data Structure for Maintaining Future Events in a Discrete-Event Simulation System", *2nd IMACS Intern. Conf. on Circuits, Systems and Computers (IMACS '98)*, Piraeus, 1998.
Also in: Recent Advances in Information Science and Technology (World Scientific), pp. 11-22, 1998.

[2] J.D. Bright, G. Sullivan, Checking Mergeable Priority Queues, *Digest of the 24th Symposium on Fault-Tolerant Computing*, IEEE Computer Society Press (1994) 144-153.

[3] G.S. Brodal, Priority Queues on Parallel Machines, *5th Scandinavian Workshop on Algorithm Theory (SWAT '96)*, 416-427, 1996.

[4] G.S. Brodal, Worst-Case Efficient Priority Queues, *Proc 7th ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*, 52-58,1996.

[5] G. S. Brodal and J. Katajainen, Worst Case Efficient External-Memory Priority Queues, DIKU Report 97/25, Department of Computer Science, University of Copenhagen (1997).

[6] R. Brown, Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem, *Communications of the ACM*, 31 (1998) 1220-1227.

[7] S. Carlsson, Improving Worst-Case Behaviour of Heaps, *BIT* 24 (1984) 14-18.

[8] B. Chazelle, The Soft Heap: An Approximate Priority Queue with Optimal Error Rate. JACM 6 (2000) 1012-1027.

[9] B.V. Cherkassky, A.V. Goldberg, C. Silverstein, Buckets, Heaps, Lists, and Monotone Priority Queues, *SIAM Journal on Computing* 28 (1999) 1326-1346.

[10] R. Cole, Parallel Merge Sort, *SIAM J. Comput.* 17 (1988) 130-145.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *"Introduction to Algorithms"*, MIT press (1990).

[12] S.K. Das, M.C. Pinotti, Parallel Priority Queues Based on Binomial Heaps, *Parallel Computing* 26 (2000) 1411-1428.

[13] P.F. Dietz, R. Raman, Very Fast Optimal Parallel Algorithms for Heap Construction, *Proc. 6th IEEE Symposium on Parallel and Distributed Processing (SPDP '94)* 514-521, 1994.

[14] E.E. Doberkat, Inserting a New Element into a Heap, *BIT* 21 (1981) 255-269.

[15] K.B. Erickson, R.E. Ladner, A. LaMarca, Optimizing Static Calendar Queues, *ACM Transactions on Modeling and Computer Simulation* (2000) 1-30.

[16] W.R. Franta, K. Maly, An Efficient Data Structure for the Simulation Event Set, *Comm. ACM* 20 (1977) 596-602.

[17] J.M. de Graaf, W.A. Kosters, Expected Heights in Heaps, *BIT* 32 (1992) 570-579.

[18] Peter Høyer, *A General Technique for Implementation of Efficient Priority Queues*, Proc. 3rd Israel Symp. on Theory of Computing and Systems (*ISTCS '95*), IEEE Computer Society Press, 57-66, 1995.

[19] G.C. Hunt, M.M Michael, S. Parthasarathy, M.L. Scott, An Efficient Algorithm for Concurrent Priority Queue Heaps, *Information Processing Letters* 60 (1996) 151-157.

[20] A. Jonassen, O.J. Dahl, Analysis of an Algorithm for Priority Queue Administration, *BIT* 15 (1975) 409-422.

[21] D.W. Jones, Concurrent operations on priority queues, *Comm. ACM* 32 (1989) 132-137.

[22] R.L. Kruse, *"Data Structures and Program Design"*, Third Edition (1994).

[23] C.P. Kruskal, Searching, Merging and Sorting in Parallel Computation, *IEEE Trans. Comput.* 32 (1983) 942-946.

[24] M. Marin, Binary Tournaments and Priority Queues: PRAM and BSP, Tecnical Report PRG-TR-7-97, Oxford University (1997).

[25] M. Marin, An Empirical Comparison of Priority Queue Algorithms, Technical report PRG-TR-10-97, Oxford University (1997)

[26] M.Marin, Priority Queue Operations on EREW-PRAM, Proceedings of the *Workshop on Parallel and Distributed Algorithms (Europar'97)*, 417-420, 1997.

[27] S.D. Nikolopoulos, R. MacLeod, An Experimental Analysis of Event Set Algorithms for Discrete Event Simulation, *Microprocessing and Microprogramming* 36 (1992/93) 71-81.

[28] A. Panholzer, H. Prodinger, Average Case-Analysis of Priority Trees: A Structure for Priority Queue Administration, *Algorithmica* 22 (1998) 600-630.

[29] M.C. Pinotti, G. Pucci, Parallel Priority Queues, *Information Processing Letters* 40 (1991) 33-40.

[30] A. Ranade, S. Cheng, E. Deprit, J. Jones, S. Shih, Parallelism and Locality in Priority Queues, Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing, (*SPDP '94*), 26-29, 1994.

[31] V.N. Rao, V. Kumar, Concurrent Access of Priority Queues, *IEEE Trans.Comput.* 37 (1988) 1657-1665.

[32] C.M. Reeves, Complexity Analyses of Event Set Algorithms, *The Computer Journal* 27 (1984).

[33] P. Reynolds, An Efficient Framework for Parallel Simulations, *Proc. Of the Advances in Parallel and Distributed Simulation Conference (PADS '91)*, 167-174, 1991.

[34] N. Shavit, A. Zemach, Scalable Concurrent Priority Queue Algorithms, *Proceedings of the 18th Annual ACM Symposium on Principals of Distributed Computing (PODC '99)*, 113-122, 1999.

[35] J. Steinman, SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation, *International Journal in Computer Simulation* 2 (1992) 251-286.

[36] J. Steinman, Discrete Event Simulation and the Event Horizon, Part 2: Event List Management, *Proceedings of the 10th workshop on Parallel and Distributed Simulation*, 170-178, 1996.

[37] R.E. Tarjan, *Data structures and Network Algorithms*, *SIAM*, Philadelphia, PA (1983).