ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΙΤΛΟΣ:

«ΑΝΘΕΚΤΙΚΕΣ ΣΕ ΛΑΘΗ ΔΟΜΕΣ ΛΕΞΙΚΟΥ»

Αχιλλέας Α. Παπακωνσταντίνου
ΑΕΜ 609

Επιβλέποντες: Παναγιώτης Μποζάνης, Αναπληρωτής Καθηγητής
Τσομπανοπούλου Παναγιώτα, Επίκουρος Καθηγήτρια

Βόλος
Σεπτέμβριος 2012

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Η συγκεκριμένη διπλωματική εργασία με τίτλο ανθεκτικές σε λάθη δομές λεξικού (“resilient dictionaries”), αναφέρεται στις μελέτες που έχουν γίνει πάνω σε αυτόν τον τομέα από τα τέλη της δεκαετίας του 50 και τον πρωτοπόρο τότε von Neumann μέχρι και σήμερα. Ακόμη, θα δούμε αλγορίθμους που μπορούν να αντιμετωπίσουν σωστά, λανθασμένα δεδομένα και να λειτουργούν χωρίς προβλήματα. Θα μελετήσουμε εργαλεία («ελεγκτές») και αλγορίθμους που θα μπορούν να μας επιβεβαιώσουν για την ορθότητα του αποτελέσματος.

Παράλληλα θα αναλύσουμε διάφορα είδη αναζήτησης σε δομές δεδομένων καθώς και τρόπους εισαγωγής και διαγραφής στοιχείων πάνω σε αυτές. Αξίζει να σημειωθεί ότι κατά τη δημιουργία ενός αλγορίθμου όποια λειτουργία και να έχει αυτός, τα χαρακτηριστικά που μας απασχολούν και έχουμε πάντα στο μυαλό μας είναι η πολυπλοκότητα σε χώρο και χρόνο. Παράλληλα με τους αλγορίθμους που θα συναντήσουμε στην πορεία θα αναφέρουμε και τις δύο αυτές πολυπλοκότητες. Όπως είναι φυσικό οι όλο και πιο πρόσφατοι αλγόριθμοι που θα συναντήσουμε θα έχουν όλο και πιο καλή συμπεριφορά ως προς αυτά τα χαρακτηριστικά αυτά καταλήγοντας στους βέλτιστους μέχρι τις μέρες μας. Είναι δυνατόν κάποιος αλγόριθμος να υστερεί σε ένα από τα κριτήρια και να είναι βέλτιστος σε άλλο. Στους συγκεκριμένους αλγορίθμους από τα πιο σημαντικά κριτήρια είναι η ποσότητα των λανθασμένων δεδομένων στα οποία ο αλγόριθμος είναι ανεκτικός χωρίς ωστόσο να επιβαρύνει τα άλλα χαρακτηριστικά του.

Είναι αυτονόητο ότι το κάθε υπολογιστικό σύστημα έχει τις δικές του προδιαγραφές και δυνατότητες. Έτσι ανάλογα και με τις απαιτήσεις του χρήστη χρησιμοποιούνται διαφορετικοί αλγόριθμοι με διαφορετική αντιμετώπιση των καταστάσεων που μπορεί να προκύψουν από υπολογιστή σε υπολογιστή. Κλείνοντας με την πάροδο των χρόνων αναμένεται να βελτιωθούν τα κόστη στα οποία έχουμε φτάσει μέχρι σήμερα καθώς άλλωστε βελτιώνεται και η ταχύτητα των υπολογισμών.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Δομές δεδομένων και υπολογιστική ανάλυση αλγορίθμων

ΛΕΞΕΙΣ-ΕΝΝΟΙΕΣ ΚΛΕΙΔΙΑ:

-Αλγόριθμο ονομάζουμε μία σειρά από εντολές που έχουν αρχή και τέλος, είναι σαφείς και εκτελέσιμες που σκοπό έχουν την επίλυση κάποιου προβλήματος

-«αντίπαλοςF» : έτσι ονομάζουμε στη πτυχιακή αυτή εργασία το γεγονός-αιτία που καταστρέφει ή εισάγει εσφαλμένα δεδομένα της δομής δεδομένων μας κατά την εκτέλεση ενός αλγορίθμου

-Χαρακτηρισμός πολυπλοκοτήτων:

Ο(x): ασυμπτωτικό άνω όριο, χειρότερη περίπτωση

Ω(x): ασυμπτωτικό κάτω όριο, βέλτιστη περίπτωση

Θ(x): ασυμπτωτικό σφικτό όριο

-Προσωρινός αποθηκευτικός χώρος (buffer): κομμάτι μνήμης στο οποίο κρατούνται τα δεδομένα μέχρι να υποστούν κάποια επεξεργασία

-Παιχνίδι αποκομμάτων (chip game) : Ο αλγόριθμος που η κάθε απάντηση γράφει-μετακινεί τη δομή δεδομένων κατά μία θέση δημιουργώντας έτσι ένα απόκομμα.

ΚΕΦΑΛΑΙΟ 1^{ο} ΕΙΣΑΓΩΓΙΚΑ ΘΕΜΑΤΑ

1.1 ΕΙΣΑΓΩΓΗ

Οι μνήμες στις σύγχρονες υπολογιστικές πλατφόρμες δεν είναι πάντα απόλυτα αξιόπιστες και έτσι πολλές φορές τα περιεχόμενα μιας θέσης μνήμης καταστρέφονται και χάνονται προσωρινά η ακόμα και για πάντα .Αυτό προκαλείται από πολλούς λόγους. Αυτά τα φαινόμενα μπορούν να επηρεάσουν σοβαρά τους υπολογισμούς. Όσο μεγαλύτερη είναι η δομή δεδομένων ,τόσο μεγαλύτερη μπορεί να είναι η βλάβη που προκαλείται.

Σε πολλές εφαρμογές που περιέχουν μεγάλες και φτηνές μνήμες επιτυγχάνουν την ανθεκτικότητα ,με την έννοια της συνηθισμένης προσέγγισης, με τη δημιουργία αντίγραφων ή με λανθασμένη ένωση τμημάτων κυκλώματος. Αυτές οι εφαρμογές πούμωσα μπορούσαν να είναι πολύ δαπανηρές ως προς το χρονιστώ χώρο και το χρήμα που απαιτούνται για την υλοποίησή τους έτσι, προσπαθούμε να σχεδιάσουμε αλγορίθμους και δομές δεδομένων οι οποίες είναι ανθεκτικές σε σφάλματα μνήμης, τα όποια είναι ικανά να υπολογίσουν το σωστό αποτέλεσμα με βάση τα μη κατεστραμμένα δεδομένα , χωρίς δηλαδή να επηρεάζονται οι υπολογισμοί από τα δεδομένα που χάθηκαν στόχος μας είναι να ατυχούμε αυτό το αποτέλεσμα χωρίς ωστόσο να επιβαρυνθούμε επιπλέον σε χώρο και χρόνιζαν θέλουμε να ψάξουμε ένα κλειδί σε μια ταξινομημένη ακολουθία αντικειμένων στην όποια υπάρχουν κατεστραμμένα κλειδιά δυαδική κλασική αναζήτηση μπορεί να οδηγήσει σε λανθασμένη κατεύθυνση γι αυτό το λόγο μπορεί ακόμα και η αναζήτηση ενός κλειδιού, το όποιο υπάρχει στο δέντρο, να αποτύχει.

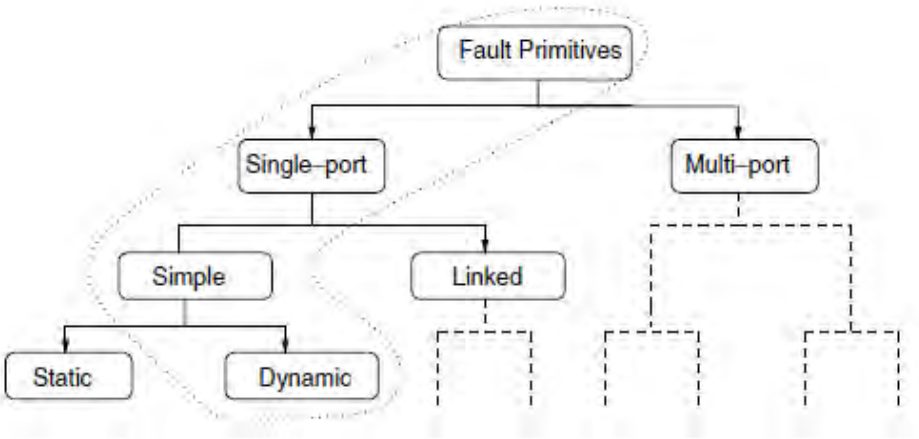
1.2 ΕΙΔΗ ΣΦΑΛΜΑΤΩΝ ΚΑΙ ΛΟΓΟΙ ΕΜΦΑΝΙΣΗΣ ΤΟΥΣ

Στην ενότητα αυτή θα μελετήσουμε τα είδη σφαλμάτων που υπάρχουν και μπορούν δηλαδή να έρθουν αντιμέτωποι οι αλγόριθμοι που θα αναπτύξουμε καθώς και τους λόγους εμφάνισής τους[16,17,18,21].

Ένας πρώτος διαχωρισμός των σφαλμάτων που παρουσιάζονται είναι τα προσωρινά (SOFT) και τα μόνιμα (HARD).Τα προσωρινά σφάλματα καταστρέφουν τα περιεχόμενα του κελιού μνήμης αλλά δεν επηρεάζουν το ίδιο το κελί. Ένας από τους πιο συχνούς λόγους που προκαλεί ένα σφάλμα τέτοιου τύπου είναι η αλλαγή ψηφίου (“bit-flip”).Ακόμη προσωρινά σφάλματα μπορεί να παρουσιαστούν από την επίδραση σωματιδίων άλφα και από κοσμική ακτινοβολία. Προσωρινά λάθη μπορεί να προκληθούν και σε SRAM και σε DRAM.

Μία ευρέως διαδεδομένη κατηγοριοποίηση των σφαλμάτων που εμφανίζονται στα υπολογιστικά συστήματα είναι τα στατικά και τα δυναμικά λάθη. Ξεκινάμε όμως από μία μεγαλύτερη και πιο αφηρημένη ομάδα σφαλμάτων τα πρωτογενή σφάλματα. Αυτά διαχωρίζονται σε μονής θύρας (Single-port), που είναι αυτά που αναφέρονται μία φορά μόνο στην μνήμη, όπως μία απλή ανάγνωση και σε πολλαπλών θυρών (Multi-port) που είναι αυτά που αναφέρονται

στη μνήμη περισσότερες από μία φορές. Τα σφάλματα μονής θύρας χωρίζονται σε απλά (Simple) και συνδεόμενα (Linked). Τα απλά είναι λάθη που δεν μπορούν να επηρεαστούν από άλλο σφάλμα δηλαδή ένα τέτοιο σφάλμα δε μπορεί να αλλάξει την συμπεριφορά ενός άλλου σφάλματος. Επίσης δε μπορεί να πραγματοποιηθεί συγκάλυψη με αυτά τα σφάλματα. Όσο αφορά τώρα τα συνδεόμενα σφάλματα μνήμης, αυτά είναι λάθη που επηρεάζονται μεταξύ τους. Έτσι μπορεί η αλλαγή συμπεριφοράς του ενός να αποκρύπτει την ύπαρξη του άλλου. Όπως γίνεται κατανοητό ένα συνδεόμενο σφάλμα αποτελείται από τουλάχιστον δύο απλά. Συνεχίζοντας τώρα τον διαχωρισμό των σφαλμάτων χωρίζουμε τα απλά σφάλματα σε στατικά (Static) και δυναμικά (Dynamic).Στατικά είναι αυτά που αναφέρονται σε μία σταθερή λειτουργία όπως για παράδειγμα ένα ψηφίο (bit) το οποίο πρέπει να είναι σταθερό στην τιμή μηδέν για παράδειγμα και αλλάζει σε ένα. Αντίθετα δυναμικά αποκαλούμε τα σφάλματα τα οποία για να προκληθεί ένα από αυτά πρέπει να γίνουν περισσότερες από μία συνεχόμενες λανθασμένες λειτουργίες. Ο διαχωρισμός αυτός φαίνεται στην εικόνα 1.Τα δυναμικά λάθη με τη σειρά τους χωρίζονται με ενός κελιού και πολλαπλών κελιών μνήμης. Τα πρώτα προκαλούνται από περισσότερες από μία ακολουθίες λανθασμένων ενεργειών οι οποίες όμως ασκούνται πάνω σε ένα μόνο κελί μνήμης σε αντίθεση με τα δεύτερα.



Εικόνα 1:Δέντρο διαχωρισμού των διάφορων ειδών σφαλμάτων που μπορούν να παρατηρηθούν[16].

ΚΕΦΑΛΑΙΟ 2^{ο} ΠΡΟΓΕΝΝΕΣΤΕΡΕΣ ΜΕΛΕΤΕΣ ΣΕ ΜΝΗΜΕΣ ΜΕ ΛΑΘΗ

2.1 ΜΟΝΤΕΛΑ ΜΝΗΜΩΝ RAM ΜΕ ΛΑΘΗ

Τα μοντέλα μνήμων τα όποια είναι επιρρεπή σε λάθη είναι ένα αντικείμενο που απασχολεί μεγάλο αριθμό μελετών από παλιά έως και σήμερα. Σε αυτά τα μοντέλα υποθέτουμε ότι υπάρχει ένας εχθρικός μηχανισμός ο όποιος μπορεί να καταστρέψει περίπου δ λέξεις μνήμης σε κάθε χώρο σε κάθε στιγμή. Λέγοντας καταστροφή μιας λέξης μνήμης, εννοούμε ότι αντικαθίσταται το περιεχόμενο της με μια διαφορετική τυχαία τιμή .Δεν υπάρχει μηχανισμός εντοπισμού λαθών των κατεστραμμένων θέσεων μνήμης από τις μη κατεστραμμένες. Τονίζουμε ότι το δ μπορεί να είναι μια συνάρτηση του μεγέθους της εισόδου. Το απαισιόδοξο μοντέλο έρχεται σε καταστάσεις όπως επιρροή από κοσμική ακτινοβολία και μη ομοιόμορφες πιθανότητες λάθους, οι οποίες είναι δύσκολο να μοντελοποιηθούν. Επίσης υποθέτουμε ότι υπάρχουν O(1) ασφαλείς θέσεις μνήμης στις οποίες δεν έχει πρόσβαση ο “αντίπαλος” .Σημειώνουμε ότι χωρίς αυτήν την υπόθεση δεν είναι δυνατόν να γίνουν αξιόπιστοι υπολογισμοί. Επίσης, η O(1) ασφαλής μνήμη μπορεί να αποθηκεύσει τον κώδικα του ιδίου του αλγορίθμου ,γιατί αλλιώς θα μπορούσε να καταστραφεί από τον “αντίπαλο”. Ακόμη, υποθέτουμε ότι η ανάγνωση μιας λέξης μνήμης στην αναξιόπιστη μνήμη είναι

μια λειτουργία, στην όποια ο “αντίπαλος” δεν μπορεί να καταστρέψει μια λέξη μνήμης από τη στιγμή που η λειτουργιά της ανάγνωσης έχει ξεκινήσει .Με τις δυο τελευταίες υποθέσεις μας η δύναμη της χυδαιότητας θα χαθεί.

Μια φυσική προσέγγιση στο σχεδιασμό των αλγορίθμων και των δομές δεδομένων με την παρουσία λαθών μνήμης είναι η δημιουργία αντίγραφων των δεδομένων. Πληροφοριακά, μια ανθεκτική σε λάθη μεταβλητή αποτελείται από (2δ+1) αντίγραφα μιας κανονικής μεταβλητής. Θα αναφερθούμε και στη συνεχεία στον ορισμό αυτό. Η τιμή της μεταβλητής δίνεται από την πλειοψηφία των αντιγράφων. Παρατηρούμε ότι η τιμή του χ είναι αξιόπιστη από τη στιγμή που ο αντίπαλος δε μπορεί να καταστρέψει την πλειοψηφία των αντίγραφων. Χαρακτηρίζουμε έναν αλγόριθμο /δομή δεδομένων σαν επιπόλαια-ανθεκτική αν υλοποιεί έναν μη ανθεκτικό αλγόριθμο/δομή δεδομένων με την αντικατάσταση κανονικών μεταβλητών με ανθεκτικές. Οι επιπόλαια-ανθεκτικές υλοποιήσεις υποφέρουν από Θ(δ) πολλαπλασιαστική επιβάρυνση σε χώρο και χρόνο εκτέλεσης. Για παράδειγμα, μια επιπόλαια-ανθεκτική υλοποίηση μιας σταθερής δομής λεξικού βασίζεται στα AVL-δέντρα θα απαιτεί O(δn) χώρο και O(δlogn) χρόνο για μια αναζήτηση, εισαγωγή και διαγραφή. Αυτό μπορεί να αντιμετωπίσει μόνο O(1) σφάλματα μνήμης ενώ παράλληλα διατηρεί τα βέλτιστα όρια σε χρόνο και χώρο.

Αυτού του τύπου οι επιβαρύνσεις μοιάζουν να είναι αναπόφευκτες εάν κάποιος επιθυμεί να διαχειριστεί σωστά αυτό το μοντέλο με επιρρεπή σε λάθη μνήμη RAM.Για παράδειγμα ,με λιγότερα από (2δ+1) αντίγραφα ενός κλειδιού δε θα μπορούσαμε να βρούμε τη σωστή τιμή αν χανόταν. Από τη στιγμή που το πολλαπλασιαστικό κόστος θα είναι αναπόφευκτο σε πολλές εφαρμογές ακόμα και σε μικρές τιμές του δ, το μόνο που μπορούμε να κάνουμε είναι να

χαλαρώσουμε την έννοια της ορθότητας του αλγορίθμου .Είπαμε ότι ένας αλγόριθμος ή μια δομή δεδομένων είναι ανθεκτικός σε σφάλματα μνήμης αν εκτός από την καταστροφή μερικών θέσεων μνήμης κατά τη διάρκεια ζωής του, είναι πάντα εφικτό να λειτουργούν σωστά τουλάχιστον οι μη κατεστραμμένες τιμές .Εννοείται πως κάθε επιπόλαιος-ανθεκτικός αλγόριθμος είναι και ανθεκτικός .Παρ όλα αυτά όπως δείχνουν τα παραδείγματα που ακολούθου υπάρχουν ανθεκτικοί αλγόριθμοι για κάποια φυσικά προβλήματα που εκτελούνται πολύ καλυτέρα από τους αντίστοιχους επιπόλαιους-ανθεκτικούς.

2.2 ΑΝΘΕΚΤΙΚΗ ΣΕ ΛΑΘΗ ΤΑΞΙΝΟΜΗΣΗ

Σ αυτήν την ενότητα θα περιγράψουμε λεπτομερώς το πρόβλημα της ανθεκτικής σε λάθη ταξινόμησης. Στο πρόβλημα αυτό αρχικά μας δίνεται ένα σύνολο K από n κλειδιά. Ένα κλειδί είναι έμπιστο αν δεν έχει καταστραφεί ποτέ. Στην αντίθετη περιπτώσει το αποκαλούμε λανθασμένο. Το πρόβλημα είναι να υπολογίσουμε μια σωστά ταξινομημένη μετάθεση του συνόλου K. Μία μετάθεση του K είναι σωστή όταν η υπακολουθία αποτελείται από τα κλειδιά που είναι σωστά ταξινομημένα .Αυτό είναι το καλύτερο δυνατό για το όποιο μπορούμε να ελπίζουμε, όταν ο “αντίπαλος” μπορεί να καταστρέψει ένα κλειδί στο τέλος της εκτέλεσης του αλγορίθμου. Τα εσφαλμένα κλειδιά λαμβάνουν λάθος θέσεις στη μνήμης. Αυτό το πρόβλημα μπορεί να λυθεί επιπόλαια σε O(δnlogn) χρόνο. Ένας αλγόριθμος ταξινόμησης που απαιτεί O(nlogn+ δ^2) χρόνο περιγράφεται στο [12]Το αποτέλεσμα δείχνει ότι μπορεί να ταξινομήσει σε O(nlogn) χρόνο στη βέλτιστη περίπτωση [14]. Σε ειδική περίπτωση τα πολυονιμικά όρια των ακέραιων κλειδιών βελτιώνουν τον χρόνο εκτέλεσης σε O(n+δ^2)[12].

ΠΡΟΣΟΧΗ! : Οι αλγόριθμοι και οι πολυπλοκότητες της ενότητας αναφέρονται σε στατική περίπτωση δομής δεδομένων, δηλαδή δεν αλλάζουν τα περιεχόμενα της με εισαγωγές ή διαγραφές κλειδιών.

2.3 ΑΝΘΕΚΤΙΚΗ ΣΕ ΛΑΘΗ ΑΝΑΖΗΤΗΣΗ

Το πρόβλημα της ανθεκτικής σε λάθη αναζήτησης έχει μελετηθεί πολύ και υπάρχουν πολλοί αξιόλογοι αλγόριθμοι [12, 14].Έχοντας μία ταξινομημένη ακολουθία, έστω K από n κλειδιά και ψάχνουμε το κλειδί κ. Το πρόβλημά μας είναι πώς να επιστρέψουμε ένα κλειδί, το οποίο να είναι είτε λανθασμένο είτε αξιόπιστο, με τιμή κ δεδομένου ότι η ακολουθία K περιέχει ένα αξιόπιστο κλειδί κ. Σε περίπτωση που η K δεν περιέχει αξιόπιστο κλειδί με τιμή κ, η αναζήτηση μπορεί να μας επιστρέφει είτε ΟΧΙ, είτε ένα εσφαλμένο κλειδί με τιμή κ. Εδώ αξίζει να αναφερθεί ότι στην παραπάνω περίπτωση η καλύτερη δυνατή περίπτωση είναι ο αντίπαλος όντως να έχει καταστρέψει ένα κλειδί με τιμή κ στην αρχή της εκτέλεσης του αλγορίθμου, ή να παράγει ένα εσφαλμένο κλειδί με την τιμή αυτή στο τέλος του αλγορίθμου. Υπάρχει αλγόριθμος που επιλύει αυτό το πρόβλημα σε O(δlogn) χρόνο και ακόμη ότι το κατώτερο όριο προσδοκώμενου χρόνου εκτέλεσης είναι Ω(logn+δ). Επιπρόσθετα, υπάρχει ένας βέλτιστος αλγόριθμος με τυχαιότητα που έχει πολυπλοκότητα O(logn+δ). Ο αντίστοιχος βέλτιστος ντετερμινιστικός αλγόριθμος έχει O(logn+$\delta^{1+\varepsilon}$) χρόνο εκτέλεσης για κάθε σταθερό ε>0.

ΠΡΟΣΟΧΗ! : Οι αλγόριθμοι και οι πολυπλοκότητες της ενότητας αναφέρονται σε στατική περίπτωση δομής δεδομένων, δηλαδή δεν αλλάζουν τα περιεχόμενα της με εισαγωγές ή διαγραφές κλειδιών.

2.4 ΜΕΛΕΤΗ VON NEUMANN

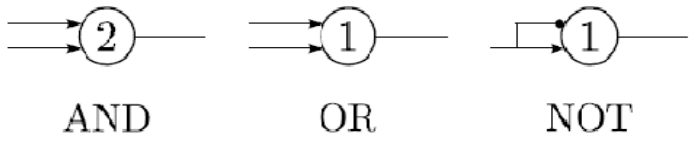
Ο von Neumann έκανε μία πρωτόγνωρη για την εποχή του μελέτη και βοήθησε τα μέγιστα στην εξέλιξη του αντικείμενου και αποτέλεσε βάση για πολλούς επιστήμονες. Ασχολήθηκε με λογικές πράξεις και αυτόματα. Η παρουσίαση της μελέτης του [25] είναι μεγάλη και στην ενότητα αυτή θα σας παρουσιάσουμε τους πιο βασικούς ορισμούς, μερικά όργανα που χρησιμοποίησε καθώς και την στάση που κράτησε απέναντι στην αντιμετώπιση των λαθών.

Μερικοί ορισμοί:

i)Ένα αυτόματο μονής εξόδου με καθυστέρηση χρόνου δ, με δ>0, έχει ένα πεπερασμένο σύνολο εισόδων, ακριβώς μία έξοδο και ένα αριθμήσιμο αριθμό υποσυνόλων. Το αυτόματο αυτό υπολογίζει την έξοδό του σε t+δ χρόνο αν και μόνον αν το υπολογίσιμο υποσύνολο ανήκει στα προσδοκώμενα υποσύνολα του συγκεκριμένου αυτόματου.

ii)Τα αυτόματα με δύο μονές εξόδους είναι ισοδύναμα με την ευρεία έννοια, αν διαφέρουν μόνο στο χρόνο καθυστέρησής τους, αλλά παίρνοντας την ίδια είσοδο παράγουν την ίδια έξοδο.

Τα αυτόματα διαφέρουν από τις λογικές πράξεις στο ότι υπάρχει μία χρονική καθυστέρηση. Δε θα ασχοληθούμε με το εσωτερικό των αυτομάτων απλά θα ξέρουμε ότι η είσοδός τους και η έξοδός του παίρνουν τις τιμές 0 (απενεργοποιημένη) και 1 (ενεργοποιημένη). Για n dδιαφορετικές εισόδους μπορούν να δημιουργηθούν 2^{2n} αυτόματα. Σε κάθε αυτόματο υπάρχουν ενεργοποιημένες και ανενεργές είσοδοι. Τα τρία πιο απλά και ευρέως διαδεδομένα παραδείγματα είναι οι λογικές πύλες AND OR και NOT που φαίνονται αμέσως μετά:



Η ενότητα της μελέτης του von Neumann που μας ενδιαφέρει κυρίως στην συγκεκριμένη εργασία είναι η παρουσία λαθών στα μοντέλα αυτά. Τα λάθη μπορεί να εμφανιστούν στις ηλεκτρονικές μονάδες. Για κάθε λειτουργία μπορεί να προκληθεί σφάλμα με πιθανότητα ε. Η πιθανότητα αυτή είναι θεωρητικά ανεξάρτητη από το χρόνο και την κατάσταση του δικτύου. Γενικότερα, οι αποτυχίες μπορεί να είναι εξαρτημένες αλλά φράζονται από πάνω με το ε. Ο στόχος είναι να βρούμε τα όρια της μικρότητας του ε, έτσι ώστε η εκτέλεση των αυτομάτων να είναι αξιόπιστη και να ισχύει $Pr(\varepsilon)<\delta$.

Τα απλά αυτόματα που αποτελούν τη βάση για τη δημιουργία πιο σύνθετων αυτομάτων αποκαλούνται βασικά όργανα. Αυτά μπορεί να έχουν μία

μοναδική έξοδο και ένα πεπερασμένο αριθμό εισόδων. Αυτές χωρίζονται σε δύο ειδών τις διεγερμένες και τις απενεργοποιημένες. Οι πρώτες διακρίνονται από τις δεύτερες από τον τρόπο που καταλήγουν στον κύκλο του σχήματος, καθώς οι μεν είναι σαν βέλος και οι δε καταλήγουν σε ένα μικρό κύκλο ο οποίος είναι κολλημένος με τον κύκλο του οργάνου.

Ας αναφερθούμε στην αντιμετώπιση των λαθών σε πολλαπλές μηχανές. Τρέχουμε m εκδόσεις του δικτύου Ο παράλληλα χρησιμοποιώντας το όργανο πλειοψηφίας που προσδιορίζει την τιμή της εξόδου. Έστω η είναι το άνω όριο των λαθών του Ο. Τα σφάλματα στην έξοδο του οργάνου πλειοψηφίας είναι πάνω φραγμένο από το $\eta^*=\varepsilon+(1-2\varepsilon)(3\eta^2-2\eta^3)$.Η αρχικοποίησή τους είναι :

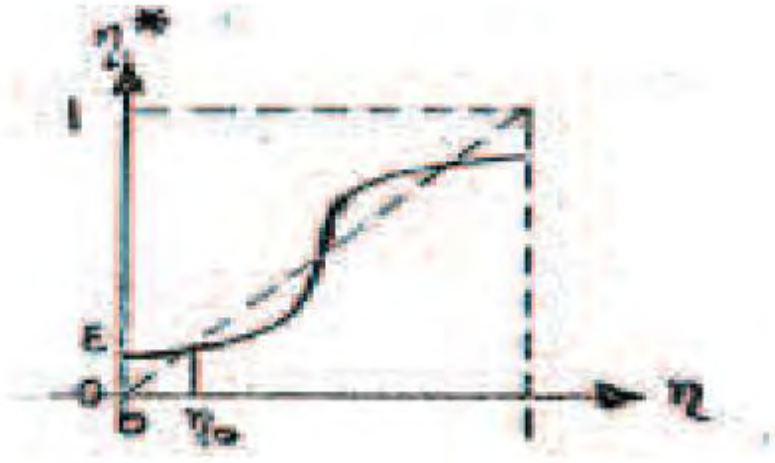
½,$\eta_0=1/2(1-\sqrt{(1-6^{\varepsilon})/(1-2^{\varepsilon})}),1-\eta_0$

Αργότερα μόνο δύο θα είναι η πραγματικές τιμές αν ε<1/6.

Θεωρούμε διαδοχικά συμβάντα στο δίκτυο.

Αν ε>1/6, το η τείνει στο 1/2

Αν ε<1/6, το η τείνει στο η_0



Με βάση το σχήμα για να γίνουν αξιόπιστοι οι υπολογισμοί πρέπει το επίπεδο των λαθών να ικανοποιεί τη σχέση :$\eta_0=\varepsilon+3\varepsilon^2$.

2.5 ΜΕΛΕΤΗ BORGSTORM-KOSARAJU (ΠΑΙΧΝΙΔΙΑ ΔΥΟ ΠΡΟΣΩΠΩΝ)

Παιχνίδια δύο προσώπων[2, 6, 10, 11, 20, 22, 23, 24] είναι αυτά που εξελίσσονται μεταξύ δύο πρωταγωνιστών. Ο ένας κάνει ερωτήσεις και ο άλλος απαντάει. Καθένας από αυτούς έχει μία στρατηγική που ακολουθεί ως προς τις ερωτήσεις που θα κάνει και ως προς το πότε θα απαντήσει σωστά ή λάθος αντίστοιχα. Ο πρώτος που κάνει τις ερωτήσεις προσπαθεί να εκμαιεύσει πληροφορίες και να οδηγηθεί έτσι σε ένα σωστό συμπέρασμα. Ο αντίπαλος έχει ως σκοπό να τον μπερδέψει και να τον οδηγήσει σε λάθος αποτέλεσμα. Όπως

άλλωστε είναι λογικό και οι δύο υπόκεινται σε κάποιους αυστηρούς περιορισμούς. Με βάση τις παραλλαγές των περιορισμών αυτών έχουν γίνει πολλές μελέτες και έχουμε οδηγηθεί σε διαφορετικούς αλγορίθμους-στρατηγικές. Ας δούμε τώρα μερικούς από τους διαφορετικούς περιορισμούς που αντιμετωπίζει ο πρωταγωνιστής που απαντά στις ερωτήσεις τους οποίους συναντάμε σε όλες τις μελέτες που έχουν αναπτυχθεί γύρω από το θέμα:

-ένα σταθερό άνω όριο στον αριθμό των ψεμάτων (λαθών): μπορεί δηλαδή να απαντήσει λανθασμένα σε το πολύ e φορές κατά τη διάρκεια του παιχνιδιού, όπου e είναι ένας σταθερός θετικός ακέραιος

-ένα άνω όριο στον περιορισμό των λαθών: αν όλο το παιχνίδι διαρκεί n ερωτήσεις, μπορεί να απαντήσει λανθασμένα το πολύ pn φορές, όπου p<1 είναι σταθερό

-ένα άνω όριο στον περιορισμό των λαθών σε κάθε αρχικό τμήμα: σε κάθε αρχικό τμήμα των πρώτων m απαντήσεων, μπορεί να απαντήσει λανθασμένα το πολύ σε pm φορές, όπου p<1 είναι σταθερό

-ένα άνω όριο στον αριθμό των ψεμάτων σε κάθε τμήμα του δοσμένου σταθερού μήκους: σε κάθε τμήμα από m διαδοχικές απαντήσεις, όπου m>1 είναι σταθερό, μπορεί να απαντήσει εσφαλμένα το πολύ σε e φορές, όπου e είναι σταθερός θετικός ακέραιος

-τυχαία λάθη: πριν από την κάθε απάντηση ο πρωταγωνιστής που απαντά πετάει ένα κέρμα και κάθε φορά που το αποτέλεσμα είναι κεφαλή, απαντά εσφαλμένα με πιθανότητα δηλαδή ½

-αυθαίρετα ορισμένα πρότυπα ψεμάτων: σε ένα παιχνίδι από n ερωτήσεις, ένα πρότυπο ψεμάτων είναι μία δυαδική ακολουθία με μήκος n.Ο πρωταγωνιστής που απαντά ψεύδεται σύμφωνα με ένα πρότυπο ψεμάτων s. Όταν αυτός ψεύδεται στην απάντηση i, θα ισχύει s(i)=1 και το αντίστροφο. Πριν το παιχνίδι και οι δύο πρωταγωνιστές συμφωνούν σε ένα σύνολο S από πρότυπα ψεμάτων και έτσι αυτός που απαντά επιλέγει ένα από τα S χωρίς να το γνωρίζει αυτός που κάνει τις ερωτήσεις, με sεS και ακολουθεί αυτό σε όλη τη διάρκεια του παιχνιδιού.

-μισά - ψέματα: οι λανθασμένες απαντήσεις είναι το πολύ e κατά τη διάρκεια του παιχνιδιού αλλά μόνο οι απαντήσεις «ΟΧΙ» μπορούν να είναι εσφαλμένες και έτσι όλες οι απαντήσεις «ΝΑΙ» είναι εγγυημένα σωστές.

Θα δούμε σ' αυτό το σημείο μερικούς από τους διάφορους τύπους ερωτήσεων που μπορεί να αντιμετωπίσουμε στην έως τώρα βιβλιογραφία:

-v-αδικές ερωτήσεις για σταθερό v>1: οι ερωτήσεις σε αυτήν την περίπτωση είναι του τύπου «Σε ποιο από τα σύνολα $A_1,\ldots,A_v$ ανήκει το άγνωστο στοιχείο; »,όπου $A_1,\ldots,A_v$ είναι κάθε τμήμα του χώρου αναζήτησης. Για v=2 έχουμε τις ερωτήσεις με δύο διαφορετικές πιθανές απαντήσεις, όπως για παράδειγμα «ΝΑΙ- ΟΧΙ»

-ερωτήσεις συγκρίσεων: εδώ ανήκουν ερωτήσεις του τύπου «Είναι το x<a ;» ,όπου a είναι ένα στοιχείο του χώρου αναζήτησης { 1,….,M }

-ερωτήσεις διαστημάτων: ερωτήσεις δηλαδή της μορφής «Είναι το x στο διάστημα [a,b];» ,όπου a<b και c<d.

-ερωτήσεις περιορισμού: ερωτήσεις δηλαδή της μορφής «xεA;» ,όπου το μέγεθος του A δε μπορεί να υπερβαίνει το δοσμένο άνω όριο.

-ερωτήσεις προθέματος: ερωτήσεις δηλαδή της μορφής «Μήπως η δυαδική αναπαράσταση του x έχει το πρόθεμα s; »,όπου s είναι μία δυαδική ακολουθία.

-ερωτήσεις μεταβλητού κόστους: ο πρωταγωνιστής που ρωτά χρεώνεται ανάλογα με την ερώτηση και έχει περιορισμένο συνολικό κόστος.

-μη-επαναλαμβανόμενες ερωτήσεις: οι απαντήσεις «ΝΑΙ-ΟΧΙ» δε μπορούν να επαναληφθούν (σε αυτήν την περίπτωση το τελευταίο πράγμα που πρέπει να συμφωνήσουν οι παίχτες πριν αρχίσουν να παίζουν είναι το πόση διαδραστικότητα θα υπάρχει μεταξύ τους).

-πλήρως προσαρμοζόμενο παιχνίδι: ο πρωταγωνιστής που κάνει τις ερωτήσεις μαθαίνει την απάντηση σε κάθε ερώτηση προτού κάνει την επόμενη ερώτηση.

-παιχνίδι n κομματιών: οι ερωτήσεις γίνονται σε n κομμάτια(ακολουθίες ερωτήσεων), όπου n είναι ένας σταθερός θετικός ακέραιος. Μετά από κάθε κομμάτι ερωτήσεων λαμβάνει τις απαντήσεις σε όλες τις ερωτήσεις του κομματιού και στη συνέχεια ετοιμάζει τις ερωτήσεις του επόμενου κομματιού των ερωτήσεων.

-μη προσαρμοζόμενο παιχνίδι: έχουμε μόνο ένα κομμάτι ερωτήσεων

-καθυστερίσεις και διακοπές του χρόνου: οι δύο παίχτες συμφωνούν σε δύο ακέραιους αριθμούς c,d≥0. Μέχρι c απαντήσεις μπορεί να χαθούν, δηλαδή να μην τις απαντήσει ο άλλος παίχτης και ο παίχτης που ρωτάει μπορεί να ρωτά μία ερώτηση για κάθε μονάδα χρόνου αλλά λαμβάνει απάντηση μετά από d μονάδες χρόνου από τη στιγμή που ρώτησε.

Οι Brogstrom και Kosaraju [6] ασχολήθηκαν με την αναζήτηση σε δομή δεδομένων η οποία όμως είναι βασισμένη σε συγκρίσεις. Ακόμη, θεώρησαν ότι η πιθανότητα λάθους έχει σταθερό ανώτατο φράγμα και έδειξαν πως προκύπτουν τα κάτω όρια στην ταξινόμηση και σε άλλα σχετικά προβλήματα δεδομένης αυτής της υπόθεσης.

Αναλυτικότερα οι δύο αυτοί ερευνητές ασχολήθηκαν με το θέμα της αναζήτησης σε ταξινομημένη λίστα στην οποία τα δεδομένα δεν είναι αξιόπιστα και επομένως η αναζήτηση γίνεται αρκετά πιο πολύπλοκη. Στο μοντέλο που ανέπτυξαν ο αριθμός των λανθασμένων ανταποκρίσεων του συστήματος δε θα υπερβαίνει σε κανένα σημείο τις r φορές τον αριθμό των ερωτήσεων. Όπου r είναι το ανώτατο φράγμα λανθασμένων απαντήσεων, το οποίο στη συγκεκριμένη περίπτωση είναι το ½. Η αναζήτηση εκτελείται αποκλειστικά με συγκρίσεις. Ειδικότερα έχουμε την ταξινομημένη λίστα L[1…n] που περιέχει διακριτούς αριθμούς, έναν αριθμό x και το θετικό άνω όριο r<1/2.Οι ερωτήσεις που γίνονται έχουν τη μορφή: είναι το x<L[j] ; Γνωρίζουμε πως αν μέχρι στιγμής έχουν γίνει κ ερωτήσεις αποκλείεται να έχουμε περισσότερες από ⌞rk⌟ λανθασμένες απαντήσεις. Στόχος του αλγορίθμου είναι να εντοπίσει το i έτσι ώστε το x να είναι στη λίστα αν και μόνον αν x=L[i].

Ας προχωρήσουμε τώρα στην περιγραφή του αλγορίθμου του «chip game» (παιχνίδι αποκομμάτων). Αρχικά για κάθε απάντηση x=L[i], τοποθετούμε ένα απόκομμα i πάνω στον πίνακα μιάς διάστασης. Η θέση l του πίνακα θα περιέχει το σύνολο των αποκομμάτων που είναι πιθανές απαντήσεις αν ακριβώς l από τις αποκρίσεις ήταν λανθασμένες. Θα τοποθετήσουμε μία πύλη για κάθε χρονική στιγμή t στην τοποθεσία rt .Η απόσταση της πύλης θα καταμετρά τον αριθμό των ανεκτών λαθών τη κάθε χρονική στιγμή. Κάθε απόκομμα στα δεξιά της πύλης αντιστοιχεί σε μία απάντηση η οποία δεν είναι πλέον δυνατή. Ακολουθούν δύο χαρακτηριστικά που διατηρούνται συνεχώς:

Α) Όλα τα αποκόμματα αρχικοποιούνται στην θέση μηδέν, δεδομένου ότι κάθε στοιχείο είναι μία πιθανή απάντηση και δεν έχει προκληθεί κανένα σφάλμα. Η πύλη βρίσκεται επίσης στη θέση μηδέν.

Β) Κάθε ερώτηση «x<L(j);» ,μπορεί να απαντηθεί με ΝΑΙ ή ΟΧΙ. Αν η απάντηση είναι ΝΑΙ, μετακινούμε τα αποκόμματα από τη θέση j έως τη n μπροστά κατά μία μοναδιαία απόσταση. Αν η απάντηση είναι ΟΧΙ, μετακινούμε τα αποκόμματα 1 έως j-1 μπροστά κατά μία μοναδιαία απόσταση. Σε κάθε βήμα η πύλη μετακινείται μπροστά κατά μία απόσταση r.

Η καταμέτρηση πραγματοποιείται με βάση ένα ειδικό βάρος της κάθε απάντησης. Η απόσταση θεωρείται θετική σε περίπτωση που το απόκομμα i βρίσκεται στα αριστερά της πύλης και αρνητική στην αντίθετη περίπτωση. Η αναζήτηση στην περίπτωσή μας που τα λάθη είναι οριοθετημένα υλοποιείται σε δύο στάδια. Το πρώτο είναι η μείωση του αριθμού των αποκομμάτων στον πίνακα σε Ο(1). Το δεύτερο στάδιο είναι η μείωση του αριθμού των αποκομμάτων στον πίνακα σε ένα. Αρχικά πραγματοποιήθηκε μία υλοποίηση του αλγορίθμου με $O(\log^2 n)$ βήματα. Υλοποιούμε το πρώτο στάδιο σε $O(\log^2 n)$ χρόνο και το δεύτερο σε Ο(logn) χρόνο. Έπειτα ακολουθεί μία βελτίωση σε Ο(logn*log(logn)).Στο τέλος βρέθηκε μία λύση-υλοποίηση με Ο(logn) βήματα.

Στη συνέχεια παρουσιάζουν έναν αλγόριθμο ο οποίος αποτελείται από Ο(logn) βήματα για κάθε r<1/2.Αξίζει να σημειωθεί για να συνειδητοποιήσουμε τη σημαντικότητα του αλγορίθμου ότι ο προηγούμενος βέλτιστος αλγόριθμος ήταν για κάθε r ανάμεσα στο 1/3 και στο ½ και είχε πολυπλοκότητα $O(^{n\log 1/(1-r)})$.Έπειτα αναφέρονται σε έναν ακόμη αλγόριθμο στον οποίο αφαιρούμε τον περιορισμό και υποθέτουμε ότι ο συνολικός αριθμός των λανθασμένων αποκρίσεων δεν είναι περισσότερες από r φορές το συνολικό αριθμό των ερωτήσεων. Για κάθε r<1/2 αναπτύσσουμε έναν αλγόριθμο με Ο(logn) βήματα, που υπολογίζει ένα σύνολο Ο(1) μεγέθους τέτοιο ώστε το x να είναι στη λίστα αν και μόνον αν το x είναι στο υπολογίσιμο σύνολο. Κλείνοντας αποδεικνύουν το εκθετικά κάτω όρια ταξινόμησης, μελετώντας n ψηφία ή βρίσκοντας το μέγιστο.

2.6 ΟΙ ΕΛΕΓΚΤΕΣ ΜΕΣΑ ΑΠΟ ΤΗ ΜΕΛΕΤΗ ΤΟΥ BLUM

Σε αντίθεση με όλα τα υπόλοιπα κεφάλαια, σε αυτήν την ενότητα θα μελετήσουμε τους ελεγκτές οι οποίοι χρησιμοποιούνται για να αντιλαμβάνονται και να αντιμετωπίζουν λανθασμένες καταστάσεις σε μη ανθεκτικές σε λάθη δομές δεδομένων [5]. Ένας ελεγκτής σε προγράμματα που αλληλεπιδρούν με τη μνήμη πρέπει να ελέγχει ότι η έξοδος, όχι μόνο ακολουθεί την προδιαγραφή του προβλήματος αλλά να είναι και συνεπής με την ακολουθία εισόδου. Μια δομή δεδομένων ορίζεται προσδιορίζοντας την έξοδο κάθε λειτουργιάς της δομής δεδομένων σε κάθε ακολουθία από λειτουργιές που εκτελούνται στην δομή δεδομένων ξεκινώντας από κάποια αρχική μορφή. Έστω ότι η δομή δεδομένων βρίσκεται σε μια μεγάλη αναξιόπιστη μνήμη και ελέγχεται από έναν “αντίπαλο” .Ο χρήστης αλληλεπιδρά με τη δομή δεδομένων παρουσιάζοντας μια ακολουθία από ενέργειες. Η δουλεία του ελεγκτή είναι να εντοπίζει κάθε λανθασμένη συμπεριφορά της δομής δεδομένων ενώ εκτελούνται οι λειτουργιές του χρήστη. Ο ελεγκτής έχει στη διάθεση του για να πετύχει τον στόχο του μόνο μια μικρή ποσότητα από αξιόπιστες θέσεις μνήμης. Ως λάθος σε μια δομή δεδομένων ορίζουμε το γεγονός να υπάρχει τιμή που επιστρέφεται από τη δομή δεδομένων και δεν ταιριάζει με την αντίστοιχη τιμή που είχαμε βάλει στη συγκεκριμένη θέση της δομής. Στην περίπτωση της μνήμης RAM, μια ανάγνωση σε συγκεκριμένη

διεύθυνση θα πρέπει να επιστρέφει την τελευταία τιμή που είχε γραφτεί σ αυτή τη θέση. Αξίζει εδώ να σημειώσουμε ότι όταν εισάγουμε μια τιμή στη δομή δεδομένων δεν επιστρέφεται κάποιο αποτέλεσμα και ως εκ τούτου δεν υπάρχει κάποια αποδεκτή έννοια που να μας δείχνει αν έχει εκτελεστεί σωστά η αποθήκευση του στοιχείου.

Ορισμός: Ένας ελεγκτής μνήμης για μια δομή δεδομένων, έστω D είναι μια πιθανοτική μηχανή που έχει πέντε διαφορετικούς τύπους.

1) διαβάζει μόνο είσοδο από την όποια ο ελεγκτής διαβάζει τις ορισμένες από τον χρηστή λειτουργιές στην D.

2) γραφεί μόνο έξοδο στον οποίο ο ελεγκτής γραφεί την έξοδο κάθε λειτουργιάς ή αγνοεί την υλοποίηση της D σαν “ΛΑΘΟΣ”

3) διαβάζει/γράφει στην ταινία εργασίας καλεί την αξιόπιστη ή ιδιωτική μνήμη του ελεγκτή

4) γράφει μόνο είσοδο στην οποία ο ελεγκτής προσδιορίζει τις λειτουργιές στη D και

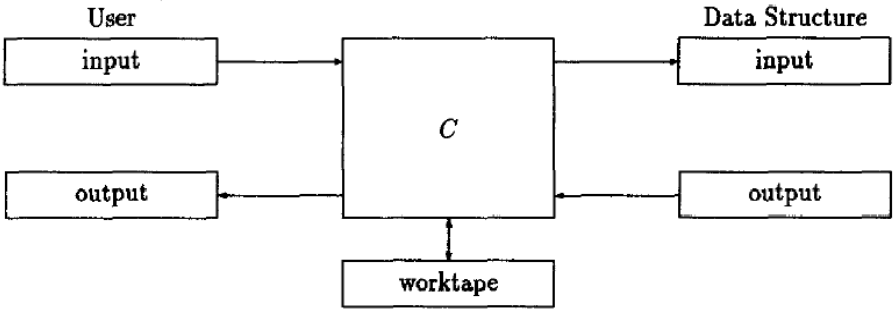
5) διαβάζει μόνο έξοδο από τον οποίο ο ελεγκτής διαβάζει την έξοδο από κάθε λειτουργιά

(όπως προσδιορίζονται από κάποιες υλοποιήσεις της D)

Ο ελεγκτής C εμφανίζεται με διάφορες λειτουργιές στην ταινία εισόδου της D. Απαιτείται να γράφει την έξοδο από κάθε λειτουργιά ή τη λέξη “ΛΑΘΟΣ” στην ταινία εξόδου ,πριν παρουσιαστεί η επομένη λειτουργία. Για όλες τις υλοποιήσεις της δομής δεδομένων D και για όλες τις ακολουθίες λειτουργιών του χρηστή το κόστος είναι πολυονιμικό στο μήκος n,όπου μήκος n είναι το μέγεθος της δομής δεδομένων:

-Αν η έξοδος της D είναι σωστή για όλες τις λειτουργιές στην ακολουθία, τότε η έξοδος του C είναι σωστή με πιθανότητα >3/4

-Αν η έξοδος της D είναι λανθασμένη για μερικές λειτουργιές, τότε η έξοδος του C είναι “ΛΑΘΟΣ” με πιθανότητα >3/4.



Εικόνα 2:Τρόπος αλληλεπίδρασης ελεγκτή με τον χρήστη και τη δομή δεδομένων[5].

Αν η αξιόπιστη μνήμη του C είναι αρκετά μεγάλη, τότε ο C μπορεί να ελέγχει την λειτουργιά της D απλά εκτελώντας τις λειτουργιές της δομής δεδομένων μονός του χρησιμοποιώντας την ταινία εργασίας του για να κρατά τα δεδομένα και να ελέγχει αν η D είναι πάντα σύμφωνη. Μας ενδιαφέρει ακόμη η απόκτηση ελεγκτών οι όποιοι έχουν μικρές ταινίες λειτουργιάς, που να έχουν περίπου λογαριθμικό μέγεθος στο μέγεθος της δομής δεδομένων. Σημειώνουμε ότι περιορίζοντας το μέγεθος της ταινίας λειτουργιάς των ελεγκτών οδηγούμε τον ελεγκτή να έχει διαφορετική υλοποίηση από την δομή δεδομένων. Ο ορισμός ενός ελεγκτή μνήμης δεν προσδιορίζεται όταν ο ελεγκτής έχει σαν αποτέλεσμα το "ΛΑΘΟΣ" αν εντοπίσει κάποιο σφάλμα. Ιδανικά, θα θέλαμε ένας ελεγκτής να βγάλει σαν έξοδο "ΛΑΘΟΣ" αμέσως μετά από μια λανθασμένη λειτουργιά. Τέτοιου τύπου ελεγκτές είναι οι on-line ελεγκτές. Διαφορετικά, αν επιτρέπουμε τον ελεγκτή να περιμένει μέχρι το τέλος της ακολουθίας των λειτουργιών για να βγάλει ως αποτέλεσμα το "ΛΑΘΟΣ", ονομάζεται off-line ελεγκτής. Στην πρώτη περίπτωση όμως ο ελεγκτής θα πρέπει να βγάζει ένα αποτέλεσμα για κάθε λειτουργιά πρώτου ο χρήστης προχωρήσει στην επομένη. Ακόμη, διακρίνουμε τους ελεγκτές οι όποιοι χρησιμοποιούν την δομή δεδομένων για να αποθηκεύσουν πληροφορίες διαφορετικές από αυτές που ζητεί ο χρήστης. Αυτοί οι ελεγκτές ονομάζονται «εισβολείς». Οι ελεγκτές που δεν παράγουν τις δίκες τους λειτουργιές ονομάζονται «μη-εισβολείς» .Για παράδειγμα όταν ένας χρήστης θέλει να προσθέσει μια τιμή ένας ελεγκτής "εισβολέας" ίσως γράψει μια σφραγίδα χρόνου και την τιμή στην μνήμη ή μια κρυπτογραφημένη μορφή της τιμής. Αντίθετα ένας ελεγκτής "μη-εισβολέας" πρέπει να γράψει μόνο την τιμή του χρηστή στη μνήμη. Ακόμη, θα θέλαμε να σχεδιάσουμε ελεγκτές που παράγουν πολύ μικρή καθυστέρηση ανά λειτουργιά. Ιδανικά ο ελεγκτής θα εκτελεί έναν σταθερό αριθμό από λειτουργιές δομής δεδομένων και μια σταθερά που εξαρτάται από τη δουλεία ανά λειτουργιά χρηστή. Μερικοί από τους ελεγκτές που θα δούμε είναι τέτοιου τύπου και κάθε λειτουργιά είναι αναλογική με το μέγεθος της δομής δεδομένων.

Ας μελετήσουμε αρχικά τους off-line ελεγκτές. Για το σχεδιασμό τους ακολουθούμε την ιδία βασική στρατηγική άσχετα από το αν αναφέρονται σε μνήμη RAM,στοίβα ή ουρά. Στην ιδιωτική του μνήμη ο ελεγκτής κρατά τις παρακάτω πληροφορίες:

-Την περιγραφή μιας συνάρτησης κατακερματισμού

-Την τιμή κατακερματισμού h(W) ενός αλφαριθμητικού W το οποίο κωδικοποιεί την πληροφορία σε όλες τις συνάρτησης εγράφης στην δομή δεδομένων.

-Την τιμή κατακερματισμού h(R) ενός αλφαριθμητικού R το οποίο κωδικοποιεί την πληροφορία σε όλες τις συναρτήσεις ανάγνωσης στην δομή δεδομένων. Επιλέγουμε κωδικοποίηση έτσι ώστε W=R αν και μόνο αν η D λειτουργεί σωστά. Η επιλογή της κωδικοποίησης εξαρτάται από την δομή δεδομένων που ελέγχεται. Έχουμε μερικούς περιορισμούς στην συνάρτηση κατακερματισμού h:

1)Οι τιμές περιγραφής του h,h(R) και h(W) πρέπει να είναι αποθηκευμένες στην μνήμη του ελεγκτή.

2)Πρεπει να μπορούμε να ανανεώνουμε τα h(W) και h(R) γρήγορα

3)Αν W διαφορετικό από το R τότε h(W) πρέπει να διαφέρει από το h(R) με μεγάλη πιθανότητα. Στις ενότητες που ακλουθούν περιγράφουμε τον τρόπο που επιτυχαίνουμε τους παραπάνω στόχους για την κωδικοποίηση και την συνάρτηση κατακερματισμού σε κάθε περίπτωση από τις τρεις δομές δεδομένων.

2.6.1 ΕΛΕΓΧΟΝΤΑΣ ΜΝΗΜΕΣ RAM.

Για να ελέγξουμε αν μια μνήμη RAM λειτουργεί σωστά πρέπει να ελέγξουμε ότι η τιμή που χρησιμοποιούμε για να διαβάσουμε μια διεύθυνση είναι η τελευταία τιμή που γράφτηκε σε αυτήν την διεύθυνση. Για να εκτελέσουμε αυτόν τον έλεγχο αποθηκεύουμε σε κάθε διεύθυνση μνήμης όχι μόνο την τιμή αλλά και τον χρόνο που αυτή η τιμή γράφτηκε. Έτσι η τιμή της ώρας εγράφης θα πρέπει να ανανεώνεται οποτεδήποτε μια λειτουργιά εγγραφής εκτελείται στην δομή δεδομένων. Οι τρεις αυτές τιμές, τιμή, διεύθυνση, ώρα, που γράφονται κάθε φορά αποτελούν και τις τιμές που διαβάζονται. Τα αλφαριθμητικά W και R σχεδιάζονται έτσι ώστε να αντιπροσωπεύουν αυτά τα σύνολα τιμών. Μια πιθανή κωδικοποίηση των συνόλων τιμών από τα αλφαριθμητικά W και R είναι το (u,a,t).Αυτό μεταφράζεται ότι βάζουμε 1 στην θέση u+an+tn^2 του W,όπου n είναι το μέγεθος της μνήμης RAM που είναι διαθέσιμη στον χρήστη. Θεωρούμε ότι το μέγεθος των u,a,t είναι λέξεις logn ψηφίων. Τα αλφαριθμητικά W και R έχουν πολυονιμικό μήκος .Αυτά τα αλφαριθμητικά είναι πολύ μεγάλα για να αποθηκευτούν ολόκληρα στην μνήμη του ελεγκτή. Έτσι αποθηκεύουμε τα W και R χρησιμοποιώντας ε-biased συναρτήσεις. Η περιγραφή μιας τέτοιας συνάρτησης απαιτεί O(logn+k) ψηφία. Οι h(W) και h(R), έχουν k ψηφία μήκος η καθεμία. Η διαδικασία που ακολουθεί ο ελεγκτής για να γράψει και να διαβάσει από τις συναρτήσεις του χρηστή είναι η παρακάτω.

Για να γράψει μια τιμή v στην διεύθυνση a:

-Διαβάζει την τιμή u' και τον χρόνο t' που αποθηκεύτηκαν στην διεύθυνση a

-Ελέγχει αν το t' είναι μικρότερο από την τρέχουσα τιμή του χρόνου,t

-Ενημερώνει την h(R) του αλφαριθμητικού R με τα u',a,t'

-γράφει την καινούρια u και την τρέχουσα τιμή του χρόνου στην διεύθυνση a

-Ενημερώνει την h(W) του αλφαριθμητικού W με τα u,a,t.

Ο ελεγκτής για την ανάγνωση από το χρήστη της διεύθυνσης a:

-Διαβάζει την τιμή u' και τον χρόνο t' από τη διεύθυνση a

-Ελέγχει αν το t' είναι μικρότερο από την τρέχουσα τιμή του t.

-Ενημερώνει την h(R) του αλφαριθμητικού R με u',a,t'

-Γράφει u' και t στην διεύθυνση a

- Ενημερώνει την h(W) από το αλφαριθμητικό W με u',a,t

Ενημερώνοντας την h(W) σε μια τριάδα εγγραφής (u,a,t) (για μία ε-biased συνάρτηση κατακερματισμού) συνεπάγεται το συμπληρωματικό ψηφίο i της

h(W) αν το ψηφίο $u+an+tn^2$ του ι-οστού διακριτού είναι 1.Προσδιορίζοντας κάθε ψηφίο για κάθε διακριτό αλφαριθμητικό απαιτεί O(1) λειτουργίες σε λέξεις των logn ψηφίων. Ενημερώνοντας την h(W) που αποτελείται από το εσωτερικό γινόμενο του W με κ διακριτά αλφαριθμητικά απαιτεί O(κ) λειτουργίες. Το ίδιο ισχύει και για την ενημέρωση του h(R).

Για να ελέγξει τη λειτουργία της μνήμης RAM στο τέλος μιας ακολουθίας λειτουργιών. ο ελεγκτής διαβάζει όλα τα κελιά μνήμης και της αντίστοιχες ενημερώσεις της h(R). Υποθέτοντας στην αρχή ότι W=R=0 και η μνήμη RAM είναι κενή, η h(W) θα πρέπει να ισούται με την h(R) αν η μνήμη λειτουργεί σωστά και θα πρέπει να είναι διαφορετική με μεγάλη πιθανότητα σε περίπτωση που η μνήμη λειτουργεί λανθασμένα. Μπορούμε να εξασφαλίσουμε ότι το t είναι μικρότερο από το n ελέγχοντας την μνήμη μετά από κάθε n λειτουργίες και μηδενίζοντας τον χρόνο.

Η ορθότητα του σχήματος ελέγχου αποδεικνύεται από τη παρακάτω πρόταση:

Αν η μνήμη RAM δυσλειτουργεί, τότε W διαφορετικό από το R.

Απόδειξη: Μία λειτουργία εγγραφής εκτελείται από τον ελεγκτή και μία λειτουργία ανάγνωσης εκτελείται από τον ελεγκτή αν η ανάγνωση αυτή είναι η πρώτη ανάγνωση μετά την εγγραφή, που περιέχει την ίδια διεύθυνση με την εγγραφή. Το γεγονός ότι και οι δύο λειτουργίες εγγραφή και ανάγνωση που εκτελούνται για κάθε λειτουργία χρήστη εξασφαλίζει ότι κάθε λειτουργία ανάγνωσης/εγγραφής που εκτελείται από τον ελεγκτή έχει μία αντίστοιχη λειτουργία ανάγνωσης/εγγραφής. Μία δυσλειτουργία προκαλείται αν η τιμή και ο χρόνος που ο ελεγκτής διαβάζει από μία διεύθυνση είναι διαφορετικές από την τιμή και το χρόνο της αντίστοιχης εγγραφής.Το συμπέρασμα από το παραπάνω καταλήγει στο ακόλουθο θεώρημα :

Για μία μνήμη RAM με 2n θέσεις μνήμης αποθηκεύει λέξεις των logn ψηφίων υπάρχει ένας off-line ελεγκτής ο οποίος χρησιμοποιεί O(logn+k) ιδιωτική μνήμη και εντοπίζει λάθη με πιθανότητα $>=1-(1/2)^{\kappa}$

2.6.2 ΕΛΕΓΧΟΝΤΑΣ ΣΤΟΙΒΕΣ

Με τον ίδιο τρόπο που ελέγχουμε τις μνήμες RAM μπορούμε να ελέγχουμε και τις στοίβες. Η «διεύθυνση» μιας λειτουργίας της στοίβας είναι το επίπεδο της στοίβας που διατηρείται στον ελεγκτή μνήμης. Σε μία λειτουργία ένθεσης ,βάζουμε την τιμή και τον χρόνο μέσα στη στοίβα και ανανεώνουμε την h(W).Αν το επίπεδο της στοίβας στο οποίο το αντικείμενο τοποθετήθηκε είναι κενό πριν τη λειτουργία δεν χρειάζεται να ενημερώσουμε την h(R).Επίσης δεν ξαναενημερώνουμε την h(W) μέχρι το επίπεδο να αδειάσει από μία λειτουργία αφαίρεσης στοιχείου. Με αυτές της τροποποιήσεις το παραπάνω θεώρημα για τις μνήμες RAM εφαρμόζεται και στις στοίβες. Μπορούμε να μειώσουμε την ικανότητα εισβολής του ελεγκτή εκμεταλλευόμενοι το μοντέλο περιορισμένης πρόσβασης δεδομένων της στοίβας. Αντί να διατηρήσει την τρέχουσα ώρα ο ελεγκτής διατηρεί τον αριθμό των φορών που το επίπεδο της στοίβας επιτυγχάνει ένα τοπικό ελάχιστο. Με άλλα λόγια ο ελεγκτής μετράει της φορές που η στοίβα

«περιστρέφεται» ύστερα από μία ακολουθία αφαιρέσεων και αρχίζει μία ακολουθία από ενθέσεις. Ο ελεγκτής χρησιμοποιεί αυτό σα μετρητή από τοπικά ελάχιστα στη θέση της σφραγίδας χρόνου. Ο μετρητής .όπως και ο χρόνος είναι γνησίως αύξοντες για κάθε επίπεδο. Έτσι η απόδειξη ότι το W διαφέρει από το R αν προκληθεί κάποιο σφάλμα όταν ο χρόνος αντικαθίσταται από τον μετρητή .Ο μετρητής παραμένει αμετάβλητος κατά τη διάρκεια μίας ακολουθίας ενθέσεων. Μειώνουμε την εισβολή με την αύξηση του μετρητή μόνο την πρώτη φορά που βάζουμε μετά από μία ακολουθία από αφαιρέσεις. Το ποσό κατά το οποίο θα μειωθεί η επιθετικότητα εξαρτάται από την ακολουθία εισόδου. Όμως το σχήμα αυτό είναι πάντα λιγότερο επιθετικό από το σχήμα που βασίζεται άμεσα στον ελεγκτή μνήμης RAM.

2.6.3 ΕΛΕΓΧΟΝΤΑΣ ΟΥΡΕΣ

Ο ελεγκτής μνήμης RAM μπορεί επίσης να χρησιμοποιηθεί για να ελέγχει ουρές. Σ' αυτήν την περίπτωση η «διεύθυνση» μια λειτουργίας εγγραφής είναι ο αριθμός της προηγούμενης εγγραφής. Έτσι μία διεύθυνση δεν ξαναχρησιμοποιείται ποτέ και μία χρονοσφραγίδα θα ήταν περιττή. Έστω w ο αριθμός των τιμών που προστέθηκαν στην ουρά και r ο αριθμός των τιμών που αφαιρέθηκαν από την ουρά. Ο ελεγκτής διατηρεί τις δύο αυτές τιμές σε ιδιωτική μνήμη. Σε μία ένθεση της τιμής u, ο ελεγκτής ενημερώνει την h(W) με u και w.,βάζει στην ουρά την τιμή u και αυξάνει το w κατά ένα. Σημειώνουμε ότι ο ελεγκτής αυτός δεν είναι επεκτατικός και γράφει μόνο την τιμή, u, που εισάγουμε στην ουρά. Σε μία αφαίρεση, ο ελεγκτής αφαιρεί μία τιμή u', ενημερώνει την h(R) χρησιμοποιώντας τις u' και r, και αυξάνει το r.Η ουρά δυσλειτουργεί αν και μόνο αν η i-οστή τιμή που αφαιρείται δεν ταιριάζει νε την i-οστή τιμή που εισάγαμε στην ουρά για οποιοδήποτε i. Έτσι, όπως και στην περίπτωση της μνήμης RAM αφού η ουρά αδειάσει το W διαφέρει από το R αν προκληθεί κάποια δυσλειτουργία.

2.6.4 ΕΛΕΓΧΟΝΤΑΣ ΜΝΗΜΕΣ RAM ΜΕ ΕΛΕΓΚΤΕΣ ΣΥΝΔΕΔΕΜΕΝΟΥΣ ΣΤΟ ΔΙΑΔΙΚΤΥΟ (ONLINE)

Οι ελεγκτές που είναι συνδεδεμένοι στο διαδίκτυο έχουν την δυνατότητα να ελέγχουν την ορθότητα της κάθε λειτουργίας αμέσως μετά την υλοποίησή της σε αντίθεση με τους ελεγκτές που δεν έχουν πρόσβαση στο διαδίκτυο που ελέγχουν την ορθότητα μιας ακολουθίας λειτουργιών.

Είναι δυνατό να ελέγχουμε μία μνήμη RAM χρησιμοποιώντας είτε συναρτήσεις ψευδοτυχαίες είτε UOWHF και το μέγεθος της μνήμης είναι O(n) .Και οι δύο λύσεις δημιουργούν ένα δυαδικό δέντρο στην κορυφή της μνήμης. Τα φύλλα του δέντρου αντιστοιχούν στις n τοποθεσίες στη μνήμη που είναι διαθέσιμες στο χρήστη. Ο ελεγκτής χρησιμοποιεί την μνήμη που περισσεύει για να αποθηκεύσει τους εσωτερικούς κόμβους του δέντρου. Έτσι ο ελεγκτής και στις δύο περιπτώσεις είναι επεκτατικός.

Στην περίπτωση του ελεγκτή που βασίζεται σε ψευδοτυχαία συνάρτηση, η αξιόπιστη μνήμη του πρέπει να είναι να είναι κρυφή. Ενώ αντίθετα στους ελεγκτές που είναι βασισμένοι στο UOWHF η αξιόπιστη μνήμη του ελεγκτή πρέπει να είναι γνωστή στον αντίπαλο. Στην περίπτωση του UOWHF αν το μέγεθος της λέξης

στη RAM είναι O(logn),δεδομένου ότι ο αντίπαλος γνωρίζει την συνάρτηση κατακερματισμού του ελεγκτή, ένας αντίπαλος πολυονιμικού χρόνου θα μπορούσε να κατακερματίσει όλες τις O(logn) μεγέθους λέξεις και να βρει μία τιμή η οποία θα αντικρούεται με μερικές τιμές στην ακολουθία εισόδου. Με σκοπό να αντιμετωπίσουμε έναν τέτοιο αντίπαλο, το μέγεθος της λέξης στη RAM θα πρέπει να είναι πολυονιμικό. Αυτές είναι οι βασικές διαφορές ανάμεσα στους δύο τύπους ελεγκτών που βασίζονται στην κρυπτογραφία.

2.6.5 ΕΛΕΓΧΟΝΤΑΣ ΣΤΟΙΒΕΣ ΜΕ ΕΛΕΓΚΤΕΣ ΣΥΝΔΕΔΕΜΕΝΟΥΣ ΣΤΟ ΔΙΑΔΙΚΤΥΟ (ONLINE)

Οι ελεγκτές στοίβας που περιγράφτηκαν παραπάνω ελέγχουν την ορθότητα όταν η στοίβα αδειάσει. Ένας τρόπος για να ελέγχουμε την ορθότητα μετά από κάθε λειτουργία είναι να αδειάζουμε τη στοίβα μετά από κάθε λειτουργία, αποθηκεύοντας τα περιεχόμενα σε μία βοηθητική στοίβα. Ο ελεγκτής ελέγχει αν h(W)=h(R) και ξαναγεμίζει την κύρια στοίβα από την βοηθητική. Αυτός ο τρόπος ελέγχει την βοηθητική στοίβα με δύο βοηθητικούς κατακερματισμούς. Ο τρόπος αυτός απαιτεί Ω(T) λειτουργίες ανά αφαίρεση στοιχείου και $\Omega(T^2)$ λειτουργίες συνολικά, όπου Τα είναι ο αριθμός των λειτουργιών στην εξεταζόμενη ακολουθία. Ο ελεγκτής που θα περιγράψουμε ακολουθεί την μέθοδο με την βοηθητική στοίβα. Αρχικά ξέρουμε ότι ο ελεγκτής έχει πρόσβαση σε δύο στοίβες δομών δεδομένων, οι οποίες ίσως ελέγχονται από τον ίδιο αντίπαλο. Η πρώτη από αυτές είναι βοηθητική και χρησιμοποιείται για να κρατάει τα περιεχόμενα της άλλης στοίβας την οποία εμείς περιοδικά αδειάζουμε μερικώς. Ο χρήστης μπορεί να χρησιμοποιεί και τις δύο στοίβες. Αυτό σημαίνει ότι και οι δυο στοίβες ίσως περιέχουν δεδομένα του χρήστη. Σ αυτήν την περίπτωση η κάθε στοίβα μπορεί να λειτουργεί ως βοηθητική της άλλης. Με σκοπό να αποφύγω τον τετραγωνισμό του αριθμού των λειτουργιών που εκτελούνται για να ελέγξουν την συμπεριφορά της στοίβας ,ο ελεγκτής διατηρεί ενδιάμεσα «σημάδια» στην ιδιωτική του μνήμη. Ένα σημάδι στο επίπεδο l είναι η τιμή που τα h(W) και h(R) ταιριάζουν όταν η στοίβα φτάσει στο επίπεδο l.Οι τιμές h(W) και h(R) αρχικοποιούνται μετά την τοποθέτηση του σημαδιού έτσι ώστε το σημάδι να πάνω από τα h(W) και h(R) για τις τιμές πάνω από το επίπεδο l.Όταν ο ελεγκτής ελέγχει μία λειτουργία, χρειάζεται μόνο να αδειάσει τη στοίβα κάτω από τη θέση του σημαδιού και να ελέγξει αν h(W)=h(R).Αν η στοίβα αδειάσει κάτω από έναν ελεγκτή, αυτός αρχικοποιεί τα h(W) και h(R) στις τιμές που αποθηκεύτηκαν από το σημάδι. Χρησιμοποιούμαι O(logH) σημάδια και εκτελούμε O(logH) επιπλέον λειτουργίες ανά λειτουργία χρήστη γι αυτόν τον ελεγκτή, όπου Η είναι ο μεγαλύτερος αριθμός από αντικείμενα στην στοίβα. Το κόλπο του αλγορίθμου που περιγράφουμε είναι στην τοποθέτηση των σημαδιών. Χρησιμοποιούμε την ιδέα της προσομοίωσης των μηχανών Turing χωρίς μνήμη. Για να απλοποιήσουμε την εξήγηση, υποθέτουμε ότι έχουμε h=logH στοίβες S0,S1,..,Sh-1.Κάθε στοίβα έχει τις δικές της h(W) και H(R).Η χωρητικότητα της στοίβας Si είναι $2*2^i$ λέξεις. Είναι εύκολα να σκεφτούμε ότι η στοίβα Si αποτελείται από δύο κομμάτια το καθένα με μέγεθος 2^i.Μια συνολική λειτουργία στην στοίβα Si είναι μία ακολουθία από 2^i προσθέσεις στοιχείων και 2^i αφαιρέσεις. Προτιμούμε τις ομαδικές λειτουργίες για ευκολία. Φυσικά οι πραγματικές λειτουργίες που εκτελέστηκαν στην στοίβα περιλαμβάνουν μονές λέξεις.

Οι στοίβες λειτουργούν σαν προσωρινοί αποθηκευτικοί χώροι. Εκτελούμε τις λειτουργίες ένθεσης και αφαίρεσης στοιχείου χρησιμοποιώντας την στοίβα

S_0.Εστω ότι η S_0 έχει δύο αντικείμενα και λάβει μία λειτουργία ένθεσης και υπερχειλίζει, μετακινούμε τα δύο αυτά δεδομένα από την S_0 και τα βάζουμε σαν ένα κομμάτι στην στοίβα S_1.Παρομοίως, αν μία λειτουργία αφαίρεσης γίνει στην άδεια στοίβα S_1,αφαιρούμε δύο αντικείμενα από την S_1 και τα βάζουμε στην S0,η οποία τώρα πλέον περιέχει δύο αντικείμενα και μπορεί να εκτελέσει μία διαδικασία αφαίρεσης. Η λειτουργία της στοίβας Si είναι ίδια ακριβώς με της S_0,εκτός από το ότι η S_i χρησιμοποιεί ένα κομμάτι από 2^i λέξεις σαν δεδομένα. Ένα απλό επαγωγικό επιχείρημα δείχνει ότι ακολουθώντας αυτήν την στρατηγική η στοίβα Si λαμβάνει ένα σύνολο από λειτουργίες ένθεσης και αφαίρεσης στοιχείων το πολύ κάθε 2_i λειτουργίες χρήστη. Ο χρόνος που απαιτεί η Si για να εξυπηρετήσει κομμάτι από λειτουργίες ένθεσης και διαγραφής είναι $O(2^i)$.Σ αυτόν τον χρόνο περιλαμβάνεται και ο χρόνος που χρειάζεται για να αδειάσει η στοίβα Si.Έτσι ο χρόνος για να εξυπηρετήσει n λειτουργίες χρήστη είναι O(nlogH).Δεδομένου ότι το Η είναι το μεγαλύτερο ύψος της στοίβας ,η Sh-1 δεν υπερχειλίζει ποτέ. Για να μετατρέψουμε της στοίβες $S_0,S_1,..,S_{h-1}$ σε μία μονή στοίβα, στοιβάζουμε τις στοίβες στην κορυφή της πρώτης. Δηλαδή βάζουμε τα περιεχόμενα της S_0 πάνω από τα περιεχόμενα της S_1 κτλ. Σημειώνουμε ότι ακόμη χρειαζόμαστε μία ξεχωριστή βοηθητική στοίβα για να εκτελούμε το τμηματικό άδειασμα και ξαναγέμισμα στο κάθε βήμα. Κρατάμε έναν πίνακα με O(h) ψηφία σε αξιόπιστη μνήμη, ο οποίος μας δείχνει τον αριθμό των κομματιών σε καθεμιά από τις h στοίβες. Αυτός ο πίνακας υποδεικνύει την θέση των σημαδιών. Κάθε σημάδι είναι ένα ζευγάρι από O(κ) ψηφίων τιμές κατακερματισμού. Το μέγεθος της αξιόπιστης μνήμης του ελεγκτή είναι O(klogn).

Για τις ουρές ένας παρόμοιος ελεγκτής μπορεί να υλοποιηθεί με O(logn) ουρές. Όμως σ αυτό το σημείο δεν ξέρουμε έναν απλό τρόπο να ενώσουμε αυτές τις ουρές σε μία μονή ουρά.

2.7 ΑΝΑΛΥΣΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ ΕΛΕΓΚΤΩΝ

Θα αναλύσουμε δύο κατώτερα όρια στο μέγεθος της μνήμης των ελεγκτών. Το πρώτο είναι το κατώτερο όριο το οποίο ισχύει για όλους τους τύπους ελεγκτών που μελετήσαμε. Το δεύτερο είναι πιο αυστηρό ακόμα και ισχύει ειδικά για τους μη συνδεδεμένους στο διαδίκτυο ,μη επεκτατικούς ελεγκτές. Επίσης θα δείξουμε ότι τα δύο αυτά όρια είναι σφιχτά.

Ξεκινώντας από το πρώτο, θα δείξουμε κάτω όρια στην πλειοψηφία της ιδιωτικής μνήμης που πρέπει να χρησιμοποιεί ένας ελεγκτής για να ελέγχει σωστά ακολουθίες οι οποίες αποθηκεύουν n ψηφία δεδομένων. Θα αποδείξουμε τρείς ισχυρισμούς. Προτού προχωρήσουμε όμως στην απόδειξη αυτών των ισχυρισμών πρέπει να αναφέρουμε ότι το κατώτερό μας όριο κατασκευάζεται από ένα ειδικό σενάριο στο οποίο η αλληλουχία των εργασιών που πρέπει να εκτελεστούν είναι μία ακολουθία που γράφει σε διακριτές διευθύνσεις ακολουθούμενη από μία αλληλουχία που διαβάζει από αυτές τις διευθύνσεις. Αυτό είναι ένα πιθανό σενάριο για όλες τις δομές δεδομένων που έχουμε μελετήσει. Για λόγους ευκολίας, σκεφτόμαστε η ακολουθία τον εγγραφών καθώς αποθηκεύει ένα μεγάλο αλφαριθμητικό το οποίο ο ελεγκτής πρέπει να ανακατασκευάσει κάποια στιγμή στο μέλλον μετά την ακολουθία των αναγνώσεων. Θα μελετήσουμε μία διαμάχη στην κύρια μνήμη η οποία είναι

προσβάσιμη από τον αντίπαλο. Επιτρέπουμε στον αντίπαλο να είναι πολύ ενεργητικός και του δίνουμε την δυνατότητα να τοποθετείται στην κύρια μνήμη σε κάθε μορφοποίηση που ακολουθεί την ακολουθία των εγγραφών από τον ελεγκτή. Ο κύριος περιορισμός του αντιπάλου είναι ότι δεν γνωρίζει το αλφαριθμητικό εισόδου ή την κατάσταση της μνήμης του ελεγκτή. Η είσοδος του ελεγκτή είναι ένα αλφαριθμητικό δεδομένων από n ψηφία που παρουσιάζεται σε μία ακολουθία από εγγραφές. Ο ελεγκτής κωδικοποιεί την είσοδο σε κατάσταση της μνήμης του ελεγκτή και σε μία κατάσταση κύριας μνήμης. Κατόπιν, μετά την ακολουθία των αναγνώσεων ο ελεγκτής πρέπει να ανακατασκευάσει το αλφαριθμητικό εισόδου χρησιμοποιώντας μόνο τα περιεχόμενα της μνήμης του και την κύρια μνήμη που έχει πρόσβαση ο αντίπαλος. Αν η μνήμη του ελεγκτή είναι πολύ μικρή, μία είσοδος ,έστω χ, υπάρχει έτσι ώστε ο αντίπαλος να ξεγελάσει τον ελεγκτή, όταν αυτός προσπαθεί να κωδικοποιήσει και να αποκωδικοποιήσει την είσοδο. Ξεγελώντας τον ελεγκτή, εννοούμε ότι ο αντίπαλος αλλάζει τα περιεχόμενα της κύριας μνήμης έτσι ώστε ο ελεγκτής να κωδικοποιεί την κύρια μνήμη και την δική του μνήμη με ένα διαφορετικό αλφαριθμητικό .Ο αντίπαλος δεν γνωρίζει την είσοδο του ελεγκτή. Παρ όλα αυτά ,αυτός το μόνο που χρειάζεται είναι να μπορεί να μπερδέψει τον ελεγκτή με μεγάλη πιθανότητα. Έτσι ο αντίπαλος θα υποθέτει πάντα ότι η είσοδος του ελεγκτή θα είναι η χ. Αν είναι όντως αυτή ο ελεγκτής θα μπερδεύεται. Στον πρώτο ισχυρισμό θα δείξουμε επίσης ότι αν η είσοδος δεν είναι η χ, ο αντίπαλος ίσως δεν μπερδέψει τον ελεγκτή ,που θα διαφύγει την ανίχνευση. Τότε ακόμα και ο αντίπαλος θα αλλάξει την κύρια μνήμη και ο ελεγκτής δεν θα επιστρέψει «BUGGY».Είναι εύκολο να επεκτείνουμε τους δύο επόμενους ισχυρισμούς, έτσι ώστε ο αντίπαλος να αποφεύγει πάντα τον εντοπισμό.

Ισχυρισμός 1: Ένας ελεγκτής ο οποίος αποκωδικοποιεί σωστά ένα αλφαριθμητικό που αποθηκεύεται αν η κύρια μνήμη είναι αξιόπιστη πρέπει να έχει μία ιδιωτική μνήμη μεγέθους $m>\log(n)-1$, όπου n είναι το μήκος του αλφαριθμητικού.

ΑΠΟΔΕΙΞΗ: Υποθέτουμε $m<\log(n)-1$ και προσδιορίζουμε το πρωτόκολλο του ελεγκτή. Ένα αλφαριθμητικό εισόδου χ, με μήκος n υπάρχει έτσι ώστε οποτεδήποτε ο ελεγκτής κωδικοποιήσει το x σε κάποια κατάσταση κύριας μνήμης και μετά προσπαθήσει να ανακατασκευάσει την είσοδο, ένας αντίπαλος ίσως πάντα υποκαθιστά μία διαφορετική κατάσταση κύριας μνήμης και μπερδεύει τον ελεγκτή στο να πιστεύει ότι η γνήσια είσοδος ήταν κάτι διαφορετικό από την x. Ο αντίπαλος σκέφτεται πως ο ελεγκτής κωδικοποιεί τις εισόδους και αποκωδικοποιεί τους συνδυασμούς της μνήμης του ελεγκτή και τις καταστάσεις τις κύριας μνήμης.

Ορισμός: Για κάθε είσοδο χ και κάθε κατάσταση της κύριας μνήμης M, θέτουμε σφαίρα(x,M) να είναι ένας πίνακας με μήκος 2^m, όπου το c-οστό τμήμα έχει οριστεί σε A αν το ζεύγος (M,c) είναι μία πιθανή κωδικοποίηση του x. Αν το ζευγάρι είναι μία πιθανή κωδικοποίηση από άλλες εισόδους και I αν είναι αδύνατο για τον ελεγκτή να φτάσει σ αυτό το ζευγάρι από κάποια είσοδο.

Εάν ο αντίπαλος δεν μεταβάλλει την κύρια μνήμη (δηλαδή, η κύρια μνήμη είναι αξιόπιστη) ο ελεγκτής πρέπει να αποκωδικοποιήσει την κύρια μνήμη και τη μνήμη του ελεγκτή ως είσοδο x με πιθανότητα 1. Έτσι για είσοδο x, η μνήμη του ελεγκτή πρέπει να είναι σε ορισμένες διαμορφώσεις του c και η κύρια μνήμη σε ορισμένες διαμορφώσεις του M έτσι ώστε η c-οστό τμήμα της σφαίρας (x, M) να είναι το A.

Αξίζει να σημειώσουμε ότι αν υπάρχει χ τέτοιο ώστε για κάθε Μ να υπάρχει y διαφορετικό από το x και Μ' έτσι ώστε σφαίρα (x,M)= σφαίρα(y,M'),τότε ο αντίπαλος μπορεί πάντα να εξαπατήσει τον ελεγκτή με το να πιστεύει ότι έχει αποθηκεύσει το y διαφορετικό από το χ. Σημειώνουμε ότι το y ίσως εξαρτάται από το Μ.

Ο αντίπαλος αντικαθιστά πάντα το Μ' με το Μ παρόλο που δεν γνωρίζει ποια ακριβώς ήταν η είσοδος του ελεγκτή. Αν η είσοδος ήταν η χ και η μνήμη του ελεγκτή ήταν c,τότε το c-οστό κομμάτι της σφαίρας (x,M)= σφαίρα(y,M') είναι ένα Α και ο ελεγκτής θα αποκωδικοποιήσει το (M,c) σαν y.Αν η είσοδος στον ελεγκτή δεν είναι x και η κύρια μνήμη είναι Μ τότε η μνήμη του ελεγκτή c πρέπει να αντιστοιχεί στο a κομμάτι της σφαίρας(x,M)= σφαίρα(y,M') και ο ελεγκτής θα αποκωδικοποιήσει το ζευγάρι μνήμης (Μ',c) σαν είσοδο και ο αντίπαλος θα περάσει απαρατήρητος. Για να αποτρέψουμε τον αντίπαλο, μία είσοδος πρέπει να έχει μία σφαίρα (που αντιστοιχεί σ αυτήν την είσοδο και σε κάποια κύρια μνήμη) που προκαλείται για καμία άλλη είσοδο. Υπάρχουν 2^n είσοδοι με μήκος n και υπάρχουν μόνο $(3^2)^m$ διαφορετικές σφαίρες. Αν m<logn-1 τότε $(3^2)^m<2^n$. Υπάρχει μία είσοδος που ικανοποιεί τις συνθήκη και δεν μπορεί ο ελεγκτής να αποθηκεύσει με ασφάλεια. Ας υποθέσουμε τώρα ότι ο ελεγκτής λειτουργεί σωστά με πιθανότητα p, αλλά με σφάλμα «δύο όψεων». Με αυτό εννοούμε ότι αν η κύρια μνήμη είναι ανέγγιχτη από τον αντίπαλο, ο ελεγκτής πρέπει να αποκωδικοποιήσει σωστά μόνο τη μνήμη του ελεγκτή και τις κύριες καταστάσεις με πιθανότητα ≥p, και σε περίπτωση που ο αντίπαλος αλλάξει την κύρια μνήμη, ο ελεγκτής είτε θα εντοπίσει την απάτη είτε θα αποκωδικοποιήσει σωστά με πιθανότητα ≥p.

Ισχυρισμός 2: Ένας ελεγκτής ο οποίος λειτουργεί σωστά με πιθανότητα $p\geq 1/2 +(1/2)^{l+1}$ στην ακολουθία εισόδου αποθηκεύοντας n ψηφία δεδομένων πρέπει να έχει ιδιωτική μνήμη με μέγεθος m≥log(n)-log(l) με l να ανήκει στο Z+.

Δηλαδή αν ο ελεγκτής χρησιμοποιεί λίγο λιγότερα ψηφία από logn μνήμη ελεγκτή, τότε υπάρχει είσοδος έτσι ώστε η πιθανότητα ότι ο ελεγκτής μπορεί να αποκωδικοποιήσει σωστά αυτήν την είσοδο είναι το πολύ ½.

Απόδειξη: Υποθέτουμε m<logn-logl.Ο αντίπαλος θα προσέξει στις σφαίρες τον τρόπο που ο ελεγκτής αποκωδικοποιεί την κύρια μνήμη και την μνήμη του ελεγκτή. Ορίζουμε ξανά την σφαίρα (x,M) να είναι ένας πίνακας $l*2^m$ ψηφίων όπου το c-οστό σύνολο από l ψηφία είναι η κοντινότερη δυαδική προσέγγιση στο Pr[αποκωδικοποίηση ελεγκτή (c,M) σαν x]. Σημειώνουμε ότι πλέον υπάρχουν το πολύ $2^{(l2\^m)} < 2^n$ διακριτές σφαίρες και υπάρχει x τέτοιο ώστε για κάθε Μ να υπάρχει y διαφορετικό από το x και Μ' τέτοιο ώστε σφαίρα(x,M)= σφαίρα(y,M'). Δημιουργούμε ένα τέτοιο x. Έστω $q=\sum_{M,c}$ (Pr[ο ελεγκτής να παράγει το M,c από το x]*Pr[ο ελεγκτής να αποκωδικοποιήσει το Μ', c σαν y])

Δηλαδή, q είναι η πιθανότητα αποκωδικοποιήσει και να κωδικοποιήσει σωστά ο ελεγκτής το x όταν ο αντίπαλος δεν αλλάζει την κύρια μνήμη. Έστω

$q'=\sum_{M,c}$(Pr[ο ελεγκτής να παράγει το M,c από το x]*Pr[ο ελεγκτής να αποκωδικοποιήσει το Μ',c σαν y]) όπου σφαίρα(x,M)= σφαίρα(y,M').Με άλλα λόγια q' είναι η πιθανότητα αν ο ελεγκτής κωδικοποιήσει το x και ο αντίπαλος αλλάξει την κύρια μνήμη βρίσκοντας μια σφαίρα που ταιριάζει, τότε ο αντίπαλος παγιδεύει με επιτυχία τον ελεγκτή στην αποκωδικοποίηση στην μνήμη ελεγκτή και αλλάζει την κύρια μνήμη σε κάποια άλλη είσοδο. Δεδομένου ότι η σφαίρα (x,M)=

σφαίρα(y,M') και η προσέγγιση του κάθε τμήματος του πίνακα χρησιμοποιεί l ψηφία, Pr[αποκωδικοποίηση ελεγκτή M,c ως x]<Pr[αποκωδικοποίηση ελεγκτή M',c ως y]+1/2^{l}. Ακόμη q-(1/2) ^{l}<q'.Από τον ορισμό του ελεγκτή μας p<q και q'<1-p το οποίο συνεπάγεται ότι p<1/2 + ½ $^{(l+1)}$. Σημειώνουμε πως αν ο ελεγκτής χρησιμοποιεί πολύ λίγα ψηφία στην ιδιωτική μνήμη του, τότε η πιθανότητα ένας αντίπαλος να ξεγελάσει έναν ελεγκτή είναι σχεδόν τόσο υψηλή όσο η πιθανότητα ο ελεγκτής να αποκωδικοποιήσει σωστά από αναλλοίωτη μνήμη (q '>q - 1/2t). Έτσι χρειαζόμαστε μόνο αποδείξεις ότι ο ελεγκτής αποκωδικοποιήσει επιτυχώς αναλλοίωτες μνήμες με μεγάλη πιθανότητα για να δείξει ότι ο αντίπαλος μπορεί να ξεγελάσει τον ελεγκτή με μεγάλη πιθανότητα.

Θεωρούμε τώρα την περίπτωση όπου ο ελεγκτής επιτρέπεται ένα ορισμένο ποσό της αξιόπιστη μνήμη που ο αντίπαλος μπορεί επίσης να δει. Εάν το μέγεθος αυτής της δημόσιας άφθαρτης μνήμης ήταν n, τότε ο ελεγκτής θα μπορούσε να απλά να αποθηκεύσει την είσοδο στην μνήμη. Θεωρούμε την περίπτωση όπου το μέγεθος της μνήμης που είναι άφθαρτο είναι λιγότερο από n κατά ένα σταθερό τμήμα και φαίνεται ότι ο ελεγκτής χρειάζεται ακόμη Ω(logn) ιδιωτική μνήμη.

Ισχυρισμός 3: Ένας ελεγκτής ο οποίος λειτουργεί σωστά με πιθανότητα p≥1/2 +(1/2)$^{(l+1)}$ στην ακολουθία εισόδου αποθηκεύοντας n ψηφία δεδομένων χρησιμοποιώντας μία δημόσια άφθαρτη ταινία με μέγεθος dn (d<1) πρέπει να έχει μία ιδιωτική μνήμη μεγέθους m≥logn-log(l/(l-d)) όπου l ανήκει στο Z+.

Απόδειξη: Υποθέτουμε m<logn-log(l/(l-d))Ορίζουμε σφαίρες όπως στην παραπάνω απόδειξη αλλά τώρα επιτρέπουμε στην σφαίρα να είναι μία συνάρτηση της εισόδου x, της κύριας μνήμης M και της άφθαρτης μνήμης B. Χρησιμοποιώντας την ίδια διάκριση όπως στην προηγούμενη απόδειξη, ξέρουμε ότι υπάρχουν το πολύ $2^{(l*2\^m)}$ διακριτές σφαίρες ανά άφθαρτη κατασκευή μνήμης. Υπάρχουν 2^{dn} πιθανές άφθαρτες κατασκευές μνήμης και το πολύ $2^{l2m}*2^{dn}<2^{n}$ είσοδοι που έχουν μία διακριτή σφαίρα για κάθε μία από τις πιθανές άφθαρτες κατασκευές μνήμης. Έτσι υπάρχει x και B τέτοιο ώστε ,για κάθε M να υπάρχει y διαφορετικό του x και M' τέτοιο ώστε σφαίρα (y,M',B)=σφαίρα(x,M,B).Ένας αντίπαλος ίσως αντικαταστεί πάντα το M με το M' και η συνολική πιθανότητα να παγιδευτεί ο ελεγκτής θα είναι τουλάχιστον p-1/2^{l} αν η είσοδος του ελεγκτή είναι x.Για το p ισχύει πάντα ότι p=1/2+ (1/2)$^{l+1}$

Για το δεύτερο όριο που αφορά ειδικά τους μη συνδεδεμένους στο διαδίκτυο και μη επεκτατικούς ελεγκτές θα δώσουμε μία μέση λύση σε επιβάρυνση σε χώρο και χρόνο. Ο χρόνος συμβολίζεται με t,ο αριθμός των κελιών μνήμης στην RAM που εξετάζονται από τον ελεγκτή όταν ελέγχει την εγκυρότητα μιας λειτουργίας. Χώρος είναι μεγέθους m η αξιόπιστη μνήμη του ελεγκτή. Έστω ότι είναι n το μέγεθος της μνήμης RAM.Δείχνουμε ότι το n ανήκει στο O(mt).Για χάρη απλότητας υποθέτουμε ότι κάθε κελί μνήμης RAM περιέχει μόνο ένα ψηφίο. Υποθέτουμε ακόμη ότι ο ελεγκτής είναι σωστός με πιθανότητα p οποτεδήποτε πιστοποιεί τα περιεχόμενα μίας θέσης μνήμης. Εμάς μας ενδιαφέρει η περίπτωση που η πιθανότητα αυτή είναι μεγαλύτερη από ½. Έστω M είναι τα περιεχόμενα της μνήμης RAM και θέτουμε το R να είναι ένα αλφαριθμητικό r ψηφίων το οποίο υποβάλλεται σε επεξεργασία σαν πηγή τυχαιότητας για τον ελεγκτή. Δεδομένου ότι ο ελεγκτής είναι μη επεμβατικός, όλα τα περιεχόμενα της μνήμης είναι πιθανά και ο αριθμός των ζευγών (M, R) είναι 2^{n+r}. Η ιδέα της απόδειξης είναι να δείξουμε ότι μπορούμε να χρησιμοποιήσουμε έναν συνδεδεμένο στο διαδίκτυο και μη επεκτατικό ελεγκτή για να κωδικοποιήσει το ζεύγος (M, R) ως ένα μικρότερο αλφαριθμητικό από το οποίο το αρχικό ζευγάρι μπορεί να ανακατασκευαστεί. Αν μπορούμε να εκτελέσουμε αυτή την

κωδικοποίηση για πάρα πολλά ζευγάρια, παίρνουμε μια αντίφαση. Συγκεκριμένα δείχνουμε ότι τουλάχιστον ένα σταθερό τμήμα γ από τα ζευγάρια (M,R) μπορεί να προσδιοριστεί μοναδικά από αλφαριθμητικά μήκους m+(t+β)n/(t+1)+r με β<1.

n+ r +logγ≤m+(t+β)n/(t+1)+r

(1-β)n≤(m-logγ)(t+1)

n ϵ O(mt)

Η κωδικοποίηση του (R,R) είναι (C,M',Z',R) και προσδιορίζεται από τα ακόλουθα βήματα:

1) Προσομοιώνουμε την αποθήκευση του M στη μνήμη RAM χρησιμοποιώντας το R ως πηγή τυχαιότητας προκειμένου να επικρατήσει η μνήμη του ελεγκτή

2)Θέτουμε J=0,j=0

3)επανέλαβε το παρακάτω n/(t+1) φορές:

Επέλεξε τον μικρότερο θετικό αριθμό I διαφορετικό του J

Θέσε M_i=0 και έλεγξε (διαβάζοντας t θέσεις που εξετάστηκαν από τον ελεγκτή χρησιμοποιώντας τα C και R)

Αν ο ελεγκτής επιστρέψει BUGGY θέτουμε M_i=1.

(Σ' αυτό το σημείο έχουμε θέσει είτε M_i=0 είτε M_i=1)

Αν είναι σωστή η τιμή αυτή τότε Z_j=1 αλλιώς Z_j=0

j=j+1

Ενώσετε τα περιεχόμενα των t περιοχών που διαβάζονται από τον ελεγκτή στο M '.
Βάλτε i και οι θέσεις t διαβάζονται από τον ελεγκτή στο J

4)Αν τουλάχιστον ένα από τα n/(t+1) από τα Z_j είναι 1,τότε η έξοδος (C,M',Z',R) όπου Z' και a ορίζονται στη συνέχεια.

Η πιθανότητα ο αλγόριθμος να θέτει την σωστή τιμή στο M_i είναι>=p.Ο προσδοκώμενος αριθμός των άσσων στο Z είναι >=pn/(t+1).Από την ανισότητα του Markov, τουλάχιστον γ=(p-a)/(1-a) από τα αλφαριθμητικά R εξαιτίας του Z έχει ≥an/(t+1) άσσους. Δεδομένου ότι αυτό κρατάει περιεχόμενα από κάθε μνήμη, τουλάχιστον τα γ από τα ζευγάρια (M,R) έχουν σαν έξοδο ένα κωδικοποιημένο (C,M',Z',R). Όταν μας δίνονται τα C, M', Z, και R μπορούμε να ανακατασκευάσουμε τα M και R. Τώρα θα δείξουμε τον τρόπο να συμπιέσουμε το Z (n / (t + 1) ψηφίων) σε Z ' (β n / (t + 1) ψηφίων με β <1) χωρίς απώλεια οποιωνδήποτε
πληροφοριών. Γνωρίζουμε ότι το Z περιέχει ≥ αn /(t+1) άσσους. Χρησιμοποιώντας τα όρια Chernoff, η πιθανότητα ένα τυχαία επιλεγμένο αλφαριθμητικό n /(t+1) ψηφίων έχει ≥an /(t+1) άσσους, είναι ≤ $e^{-(\alpha-1/2)2n/(2(t+1))}$.Έτσι, ο αριθμός από τέτοια αλφαριθμητικά είναι ≤2^(βn/(t+1)) όπου β=1-loge(α-1/2) ^{2}/2. Μπορούμε να κωδικοποιήσουμε αυτά τα αλφαριθμητικά χρησιμοποιώντας βn/(t+1) ψηφία. Το μήκος του (C,M',Z',R) είναι m+(t+β)*n/(t+1)+r.Ο ορισμός του ελεγκτή επιτρέπει p=3/4,α=5/8,γ=1/3 και β=1-log(e)/128. Αυτό συνεπάγεται ότι το n ανήκει στο O(mt).

2.8 ΜΕΛΕΤΗ ΓΙΑ ΑΝΟΧΗ ΛΑΘΩΝ ΣΕ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ (AUMANN & BENDER)

Στην ενότητα που ακολουθεί θα εξετάσουμε την ανοχή των δομών δεδομένων (λίστες, στοίβες, δυαδικά δέντρα) σε λάθη[3]. Αρχικά θα δώσουμε μερικούς ορισμούς που θα μας διευκολύνουν στην ανάλυσή τους. Κατόπιν, θα ακολουθήσει λεπτομερή ανάλυση της κάθε δομής χωριστά και θα κλείσουμε με τα αποτελέσματα της ανάλυσης.

ΟΡΙΣΜΟΣ 1: Έστω ότι συμβολίζουμε ως S=(H,P) μία δομή δεδομένων και ότι R={R1,…,Rt} είναι το σύνολο των σχέσεων των κόμβων που έχουν την πληροφορία συνδυασμένοι στο το γράφημα του H,που είναι ένα στιγμιότυπο του S.Έστω S=(V,E) που ανήκει στο H είναι ένα στιγμιότυπο του S.Το S'=(V',E') είναι μία ανακατασκευή του S που δεν καταπατά το R αν ισχύουν οι ακόλουθες συνθήκες.

-Το γράφημα S' είναι ένα έγκυρο στιγμιότυπο του S με S' να ανήκει στο H

-Οι κόμβοι που περιέχουν την πληροφορία στο S' είναι ένα υποσύνολο των κόμβων που ανήκουν στο S

-Για κάθε R_i που ανήκει στο R, αν το R_i έχει κ σχέσεις, τότε για κάθε κ-πλειάδα u ανήκει $(V')^K$ από τους κόμβους πληροφορίας έχει σχέση R_i στο S αν και μόνο αν έχει σχέση R στο S.

Αποκαλούμε τους κόμβους πληροφορίας που εμφανίζονται στο S αλλά όχι στο S' χαμένους κόμβους. Ένας κόμβος μπορεί να χαθεί ακόμα και αν δεν είναι εσφαλμένος. Πιο συγκεκριμένα, ένας κόμβος μπορεί να είναι απρόσιτος στην περίπτωση που όλα τα μονοπάτια προς τον κόμβο αυτό έχουν χαθεί από λάθη. Εμείς ψάχνουμε δομές δεδομένων για τις οποίες όλα τα στιγμιότυπα είναι εφικτό να ανακατασκευαστούν με τον ελάχιστη απώλεια μη λανθασμένων κόμβων.

ΟΡΙΣΜΟΣ 2: Έστω S=(H,P) είναι ένας συμβολισμός μίας δομής δεδομένων και $R=\{R_1,\ldots,R_t\}$ είναι ένα σύνολο από σχέσεις στους κόμβους πληροφορίας συνδυασμένοι στο γράφημα H. Έστω ότι d είναι ένα στιγμιότυπο και g: N->N είναι μία συνάρτηση. Λέμε ότι το S έχει (d,g) ανοχή βλαβών, ενώ παράλληλα να ισχύει και το R αν υπάρχει ένας αλγόριθμος ανακατασκευής A που να ικανοποιεί τα ακόλουθα. Για κάθε στιγμιότυπο s του S αν υπάρχουν f≤d λάθη στο S,τότε ο αλγόριθμος A με είσοδο την λανθασμένη s, έχει σαν έξοδο μία ανακατασκευή S' του S ,ενώ παράλληλα ισχύει και το R,έτσι ώστε ο αριθμός των χαμένων κόμβων πληροφοριών να έχει όριο το g(f).Ο χρόνος εκτέλεσης του αλγόριθμου ανακατασκευής πρέπει να είναι πολυονιμικός στα f και d.Εμείς προσπαθούμε να πετύχουμε δομές δεδομένων για τις οποίες η συνάρτηση g(f) και ο χρόνος εκτέλεσης του A να είναι συναρτήσεις που θα αυξάνονται αργά και ανεξάρτητα από το μέγεθος της δομής δεδομένων S.

Στις βασιζόμενες σε δείκτες δομές δεδομένων οι κόμβοι προσεγγίζονται από δείκτες. Οι περισσότεροι από αυτούς τους δείκτες τοποθετούνται σε άλλους κόμβους του γραφήματος. Παρ' όλα αυτά οι δομές δεδομένων πρέπει να επιτρέπουν την πρόσβαση στην δομή από το εξωτερικό περιβάλλον της. Για παράδειγμα ο δείκτης στην κορυφή της λίστας είναι σε μία σταθερή θέση και η δομή προσεγγίζεται μέσω αυτού του δείκτη. Η ουρά έχει δύο τέτοιους δείκτες που είναι σε σταθερές τοποθεσίες και υπάρχει ένας σταθερός αριθμός από τέτοιους δείκτες. Αν χαθούν όλοι αυτοί οι δείκτες ολόκληρη η δομή δεδομένων θα γίνει απρόσιτη. Αποκαλούμε το σύνολο των δεικτών αυτών στην δομή ως χειριστές της δομής. Τυπικά ο χειριστής H(S) της δομής δεδομένων S είναι ένα σύνολο από κόμβους που συμβολίζονται με $\{h_1,\ldots,h_k\}$.Κάθε χειριστής hi αποθηκεύει έναν δείκτη, ο οποίος μπορεί να δείξει σε οποιονδήποτε δείκτη της δομής δεδομένων. Ακόμη, ο κόμβος χειριστής ίσως αποθηκεύσει βοηθητική πληροφορία προστατεύοντας τον δείκτη ή την δομή δεδομένων. Για κάθε δομή δεδομένων ο

αριθμός των κόμβων στον χειριστή είναι ένα πάνω όριο του αριθμού των λαθών που η δομή δεδομένων μπορεί να ανεχθεί. Αυτό εξηγεί γιατί ο ορισμός δύο έχει ένα πάνω όριο ,d, στον αριθμό των λαθών που η δομή μπορεί να αντέξει στην χειρότερη περίπτωση. Οι περισσότερες δομές δεδομένων δεν προσφέρονται για μία αποτελεσματική ανακατασκευή. Έτσι κάνουμε μία εισαγωγή σε εκδόσεις ανοχής βλαβών δομών δεδομένων. Η νέα έκδοση έχει παρόμοια συμπεριφορά με την γνήσια ενώ υποστηρίζει τον μεγαλύτερο βαθμό σε ανοχή βλαβών. Ο ακόλουθος ορισμός προσδιορίζει τι σημαίνει για την έκδοση αυτή της ανοχής βλαβών να μοιάζει με την γνήσια.

Ορισμός 3:Έστω S=(H,P) είναι ένας συμβολισμός μίας δομής δεδομένων με P={P_1 ,…, P_k } (P_i είναι οι διαδικασίες) και έστω S''=(H'',P'') είναι ένας άλλος συμβολισμός δομής δεδομένων με P''={P_1'',…,P_k''}.S'' είναι μία προσομοίωση το S αν ισχύουν οι παρακάτω συνθήκες:

-Για κάθε i, η διαδικασία Pi'' του S'' έχει την ίδια διεπαφή με την διαδικασία Pi του S,δηλαδή παίρνουν είσοδο με την ίδια μορφή και η έξοδος τους έχει πάλι την ίδια μορφή

-Για κάθε ακολουθία κλήσεων των διαδικασιών από το S,καλούν την αντίστοιχη διαδικασία και έχουν σαν αποτέλεσμα την ίδια έξοδο στον χρήστη.

Την ποιότητα της προσομοίωσης την χαρακτηρίζουν δύο χαρακτηριστικά γνωρίσματα ,ο χρόνος και ο χώρος.

Ορισμός 4:Έστω S'' μία προσομοίωση του S.Λέμε ότι είναι μία (α ,β) προσομοίωση αν ικανοποιούνται τα ακόλουθα κριτήρια.

-Χρόνος: Για κάθε ακολουθία κλήσεων των διαδικασιών του S και των αντίστοιχων διαδικασιών του S'' ,ο χρόνος εκτέλεσης των διαδικασιών του S'' είναι το πολύ α φορές το κόστος εκτέλεσης των διαδικασιών του S.

-Χώρος: Για κάθε στιγμιότυπο Σ της δομής δεδομένων S, το αντίστοιχο στιγμιότυπο Σ'' του S'' καταλαμβάνει το πολύ α * β περισσότερο χώρο.

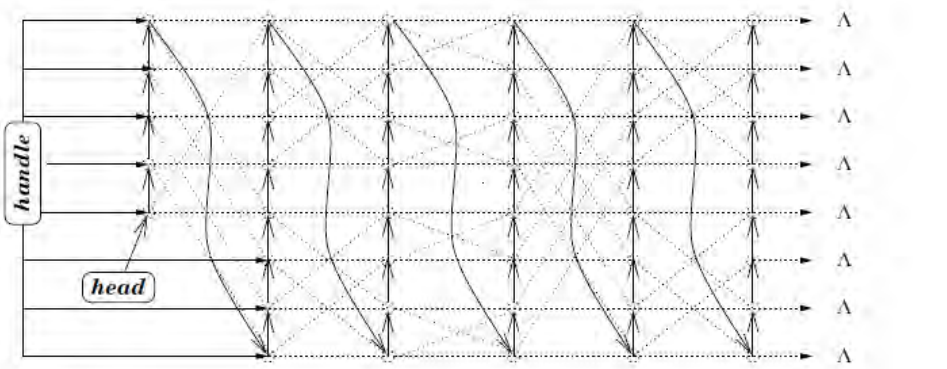
Χαρακτηρίζουμε S'' μία σταθερή προσομοίωση αν είναι O(1)*O(1) προσομοίωση.

2.8.1 ΑΝΟΧΗ ΒΛΑΒΩΝ ΣΕ ΣΤΟΙΒΑ

2.8.1.1 ΔΟΜΗ

Ας προχωρήσουμε στην μελέτη της ανοχής βλαβών στις στοίβες. Μια στοίβα, είναι μία δομή δεδομένων που υποστηρίζει δύο λειτουργίες. Την ένθεση στοιχείου (Push(x)) και την αφαίρεση στοιχείου από τη στοίβα (POP).Ένα στιγμιαίο γράφημα έχει τη μορφή ενός κατευθυνόμενου μονοπατιού, με το κορυφαίο σημείο που γίνονται οι χειρισμοί στον πρώτο κόμβο του μονοπατιού και τον τελευταίο κόμβο να δείχνει στο κενό. Κάθε κόμβος x έχει ένα πεδίο δεδομένων x.value και έναν δείκτη που βγαίνει έξω (x.next).Θεωρητικά συναντάμε δείκτες που πηγαίνουν προς τα κάτω και έτσι βάζουμε και βγάζουμε κόμβους από την κορυφή της στοίβας. Η τοπολογική ταξινόμηση της στοίβας έχει σχέση «πάνω κάτω» κατά μήκος των κόμβων. Η στοίβα έχει υψηλή μη-ανθεκτικότητα σε λάθη επειδή ένα λάθος μνήμης μπορεί να προκαλέσει την απώλεια O(n) κόμβων. Θα περιγράψουμε μία οικογένεια στοιβών, τις d-FT στοίβες, για d ανήκει 2^i (για άλλους d απλούς γύρους στο κοντινότερη δύναμη

του 2).Για κάθε τέτοιο d, η στοίβα d-FT ,είναι (d,O(f*logf)) ανοχή βλαβών με σεβασμό στην σχέση «πάνω-κάτω» και είναι μία σταθερά προσομοίωση της στοίβας. Η δομή του γραφήματος της στοίβας d-FT αποτελείται από μία ακολουθία επιπέδων. Κάθε επίπεδο,L_i, εκτός ίσως από το κορυφαίο επίπεδο, αποτελούνται από 2d κόμβους,$L_i=\{x_{l,0},\ldots x_{l,2d-1}\}$.Σημειώνουμε τα επίπεδα $L_{(n/2d)-1},\ldots, L_0$, έτσι ώστε αυτό το επίπεδο L0 να είναι η βάση της στοίβας και το $L_{(n/2d)-1}$ να είναι στην κορυφή. Κάθε logd+1 επίπεδα αποτελούν μία δομή πεταλούδας .Ένα παράδειγμα γραφήματος παρουσιάζουμε αμέσως μετά.



Εικόνα 3:Μία D-FT στοίβα με d=4 και n=45 [3].

Κάθε κόμβος $x_{i,j}$ στο γράφημα πεταλούδας έχει δύο ακμές που φεύγουν , μία διαγώνια και μία ευθεία. Η ευθεία ακμή κατευθύνεται από τον κόμβο $x_{i,j}$ στον κόμβο $x_{i-1,j}$ (αν i≠0 και στο μηδέν αλλιώς). Η διαγώνια ακμή ορίζεται ως εξής: Έστω $j^{(i)}$ είναι ο ακέραιος ο οποίος έχει την ίδια δυαδική αντιπροσώπευση με τον j, εκτός του imod2d ψηφίο το οποίο αλλάζει. Ακόμη η διαγώνια ακμή κατευθύνεται από το $x_{i,j}$ στο $x_{i-1,j^{i}}$. Η λεξικογραφική διάταξη των κόμβων αντιστοιχεί στη διάταξή τους σ τη στοίβα. Το άνω επίπεδο μπορεί να είναι ατελές. Ο χειριστής της d-FT στοίβας αποτελείται από 2d +1 κόμβους. Ένας κόμβος, κορυφή, αποθηκεύει ένα δείκτη προς την τρέχουσα κορυφή της στοίβας. Οι επιπρόσθετοι 2d κόμβοι διαχείρισης είναι δείκτες για τα κορυφαία 2δ κόμβους της στοίβας. Οι διαδικασίες της FT στοίβας, Push () και Pop ,προσθέτουν και αφαιρούν τους κόμβους στη λεξικογραφική διάταξη και να διατηρούν τη δομή δεικτών.

Ισχυρισμός 1: Για κάθε d, η στοίβα FT είναι μία σταθερή προσομοίωση της αρχικής στοίβας.

Απόδειξη: Οι στοίβες και οι στοίβες d-FT έχουν την ίδια διεπαφή χρήστη και συμπεριφέρονται στον χρήστη με την ίδια συμπεριφορά .Έτσι οι στοίβες FT είναι μία προσομοίωση των απλών στοιβών. Η αναλογία εκτέλεσης ανάμεσα σε μία στοίβα και σε μία FT στοίβα είναι ως εξής:

-Χρόνος: Καθεμία από τις λειτουργίες των FT στοιβών έχει ένα σταθερό αριθμό βημάτων. Έτσι η αναλογία είναι επίσης σταθερή.

-Χώρος: Οι κόμβοι μιας FT στοίβας είναι σε ένα προς ένα αντιστοιχία με εκείνους μιας απλής στοίβας. Κάθε κόμβος FT στοίβας απαιτεί μία σταθερή ποσότητα χώρου. Επίσης, μία FT στοίβα έχει 2d+1 κόμβους χειριστές Συνολικά καθώς αυξάνεται το n, η αναλογία ανάμεσα στις απαιτήσεις χώρου της στοίβας και της FT στοίβας είναι O(1).

2.8.1.2 ΣΦΑΛΜΑΤΑ ΚΑΙ Η ΑΝΤΙΜΕΤΩΠΙΣΗ ΤΟΥΣ

Όταν εντοπίζεται ένα σφάλμα, ξεκινά η ανακατασκευή. Η διαδικασία ανακατασκευής πραγματοποιείται σε δύο φάσεις, που περιγράφονται παρακάτω.

-Φάση αφαίρεσης: Σ' αυτήν την φάση, αφαιρούμε τους κόμβους από την FT στοίβα και τους τοποθετούμε σε μία βοηθητική αποθήκη, όπως μία άλλη στοίβα FT.Προσπαθούμε να εντοπίσουμε κάθε κόμβο μέσω από τους δύο δείκτες που εισέρχονται σ' αυτόν. Σε περίπτωση που και οι δύο δείκτες δεν είναι διαθέσιμοι ή φτάνουν σε λάθος κόμβο, αγνοούμε τον κόμβο. Η φάση αφαίρεσης τερματίζει όταν 2d διαδοχικοί προσεγγίσιμοι κόμβοι ανακαλυφθούν ή όταν προσεγγιστεί η βάση της FT στοίβας. Σ' αυτό το σημείο η FT στοίβα που έχει παραμείνει είναι λειτουργική και δεν παρουσιάζει άλλα σφάλματα. Επιπρόσθετα σφάλματα μπορεί να υπάρξουν στην FT στοίβα. Σ αυτήν την περίπτωση θα τρέξουμε τη διαδικασία ανακατασκευής οποτεδήποτε εμφανιστούν αυτά τα λάθη.

-Φάση επανεισαγωγής: Εδώ επανεισάγουμε τους κόμβους χρησιμοποιώντας την διαδικασία εισαγωγής στις FT στοίβες.

Επειδή επανεισάγουμε κόμβους με την αντίθετη ταξινόμηση από ότι μπήκαν στη λίστα, η ανακατασκευή διατηρεί την ταξινόμηση των κόμβων της γνήσιας FT στοίβας.

Ισχυρισμός 2:Αν υπάρχουν f σφάλματα, τότε το πολύ O(flogf) κόμβοι έχουν χαθεί.

Απόδειξη: Έστω F είναι το σύνολο των εσφαλμένων κόμβων. Χωρίς απώλεια γενικότητας, θεωρούμε ότι ο αριθμός των λαθών f=|F| είναι δύναμη του 2.Ονομάζουμε έναν κόμβο μη-προσβάσιμο αν έχει χαθεί αλλά δεν έχει καταστραφεί. Ξεκινούμε την ανάλυσή μας θεωρώντας έναν διακριτό μη προσβάσιμο κόμβο ,έστω x0, ο οποίος βρίσκεται στο επίπεδο Li.Υπάρχουν δύο περιπτώσεις. Πρώτον, υποθέτουμε ότι υπάρχει το επίπεδο Li+logf+1 (το επίπεδο logf+1 πάνω από το x0). Έστω A(xo) είναι το σύνολο των κόμβων που ανήκουν στο Li+logf+1 που είναι πρόγονος του x0.Ένα χαρακτηριστικό της πεταλούδας (δεδομένου ότι ο αριθμός των λαθών f<2d) είναι ότι αν διαπεράσουμε τις ακμές των logf+1 επιπέδων από τον κόμβο x0 στο επίπεδο Li στο επίπεδο Li+logf+1, δημιουργούμε ένα δυαδικό δέντρο με 2f φύλλα και έχουμε |A(x0)|=2f. Με σκοπό να είναι το x0 απρόσιτο θα πρέπει να ισχύει ένα από τα παρακάτω για όλους τους κόμβους y του A(x0):

(1)είτε να έχει χαθεί το y(λανθασμένο ή απροσπέλαστο)

(2)είτε το y να είναι προσβάσιμο αλλά να έχουν καταστραφεί όλα τα μονοπάτια από το y στο x0.Αυτό μπορεί να ισχύει επειδή όλα τα ευθεία μονοπάτια του

γραφήματος της πεταλούδας μπορεί να παρέχουν μονοπάτι στα οποία υπάρχουν κόμβοι που έχουν χαθεί από τον χειριστή του καθένα από τους 2f κόμβους του A(x0). Δεδομένου ότι έχουμε μόνο f σφάλματα, αυτά μπορούν να σταματήσουν μόνο f από αυτά τα μονοπάτια. Συμβολίζουμε με B(x0) το σύνολο των προσβάσιμων κόμβων του A(x0). Ξέρουμε από πάνω ότι |B(x0)|≥f. Έστω T(x0) είναι το δυαδικό δέντρο από τους κόμβους που βρίσκονται στο B(x0) και στο x0,συμπεριλαμβανομένου και των εσφαλμένων κόμβων. Για κόμβο z που ανήκει στοT(x0),έστω d(z,x0) είναι η απόσταση από το z στο x0.Θεωρούμε έναν εσφαλμένο κόμβο z στο T(x0).Μετράμε τον αριθμό των κόμβων y που ανήκουν στο B(x0) για τους οποίους το z μπορεί να καταστρέψει ένα μονοπάτι από το y στο x0.Η απόσταση από το z στο B(x0) είναι logf+1-d(z,x0).Ο αριθμός των κόμβων του B(x0) κάτω από το z είναι το πολύ

$2^{\log f+1-d(z,x0)} = f*2^{-d(z,x0)+1}$. Αυτός είναι ο αριθμός των y που μπορούν να σταματήσουν τα εσφαλμένα z.

Για τους κόμβους z και x0 έχουμε:

$w(z,x0)= 2^{-d(z,x0)+1}$ όταν το z ανήκει στο T(x)

σε κάθε άλλη περίπτωση w(z,x0)=0

Από την πιο πάνω ανάλυση, ένας εσφαλμένος κόμβος z μπορεί να καταστρέψει ένα μονοπάτι για το πολύ f*w(z,x0) μονοπάτια από τους κόμβους B(x0) στο x0.Δεδομένου ότι τουλάχιστον f μονοπάτια πρέπει να καταστραφούν από το B(x0) στο x0, για να είναι το x0 απρόσιτο, πρέπει να ισχύει $\sum_{z\epsilon F} w(z,x0)f\geq f$ και έτσι καταλήγουμε στο $\sum_{z\epsilon F} w(z,x0)\geq 1$.

Για την περίπτωση που το Llogf+1 δεν υπάρχει, που σημαίνει ότι ο κόμβος x0 είναι κοντά στην κορυφή, ένα παρόμοιο σκεπτικό μας οδηγεί στην εξίσωση 1.Μέχρι εδώ έχουμε υποθέσει ένα διακριτό απρόσιτο κόμβο x0.Τώρα θα θεωρήσουμε το σύνολο I όλων των απρόσιτων κόμβων. Θεωρούμε έναν λανθασμένο κόμβο z,ο οποίος έχει απόσταση ένα από τους δύο απόγονούς του, απόσταση δύο από τέσσερις κόμβους απόγονούς του κτλ. Αθροίζοντας όλα τα δέντα που περιέχουν το z έχουμε:

$$\sum_{x\epsilon I} w(z,x) \leq \sum_{i=1}^{\log f} 2^{i}*2^{-i+1}=2\log f$$

Από την εξίσωση 1 έχουμε:

$$|I|=\sum_{x\epsilon I}1\leq\sum_{x\epsilon I}(\sum_{z\epsilon I} w(z,x))$$

Από το άθροισμα και την εξίσωση 2 προκύπτει:

$$|I|\leq\sum_{z\epsilon F}(\sum_{x\epsilon I} w(z,x))\leq 2f\log f$$

Επομένως ο συνολικός αριθμός των χαμένων κόμβων είναι |I|+|F|≤2flogf+f

Ας οριοθετήσουμε τώρα την πολυπλοκότητα της διαδικασίας ανακατασκευής.

Ισχυρισμός 3 :Για κάθε f και d, η διαδικασία ανακατασκευής της στοίβας d-FT ολοκληρώνεται σε O(df*logf) βήματα.

Απόδειξη: Η αφαίρεση και η επανατοποθέτηση στοιχείων απαιτούν έναν σταθερό αριθμό βημάτων. Η φάση αφαίρεσης ολοκληρώνεται μόλις εντοπιστούν 2d

διαδοχικοί προσβάσιμοι κόμβοι. Με f σφάλματα, το πολύ flogf διαδοχικά επίπεδα έχουν απρόσιτους κόμβους .Δεδομένου ότι κάθε επίπεδο αποτελείται από 2d κόμβους ,η συνολική διαδικασία θέλει O(dflogf).

ΘΕΩΡΗΜΑ 1ο: Για κάθε d, η d-FT στοίβα έχει O(d,f*logf) ανεκτικότητα σε λάθη και είναι μία σταθερή προσομοίωση της στοίβας.

2.8.2 ΑΝΟΧΗ ΒΛΑΒΩΝ ΣΕ ΣΥΝΔΕΔΕΜΕΝΕΣ ΛΙΣΤΕΣ

2.8.2.1 ΔΟΜΗ

Ας μελετήσουμε τώρα την ανθεκτικότητα σε λάθη μιας συνδεδεμένης λίστας. Αυτή η μορφή δομής δεδομένων υποστηρίζει τις εξείς λειτουργίες:

-Insert(u,p)(εισαγωγή): εισάγει έναν κόμβο με τιμή u πριν τον κόμβο p

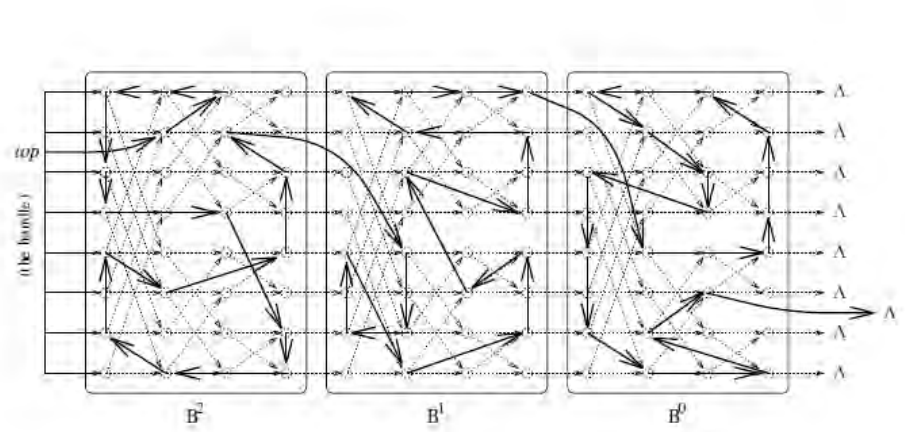
-Delete(p)(αφαίρεση): αφαιρεί τον κόμβο p

-Value(p)(τιμή): επιστρέφει την τιμή που είναι αποθηκευμένη στον κόμβο p

-Next(p)(επόμενος): επιστρέφει τον κόμβο που ακολουθεί τον p

-Head(κορυφή): επιστρέφει τον πρώτο κόμβο της λίστας

Η σχέση που έχουν οι κόμβοι μεταξύ τους είναι «προηγούμενος - επόμενος». Η συνδεδεμένη λίστα έχει υψηλή με ανθεκτικότητα σε λάθη. Θα μελετήσουμε μία νέα μορφή λίστας που ονομάζεται d-FT λίστα για d δύναμη του δύο. Για κάθε d η d-FT λίστα έχει (d,O(f*logf*logd)) ανεκτικότητα σε λάθη για να τηρείται πάντα η σχέση «προηγούμενος -επόμενος» ανάμεσα στους κόμβους. Το στιγμιότυπο μιας συνδεδεμένης λίστας έχει κατευθυνόμενα μονοπάτια. Η στοίβα και η λίστα έχουν την ίδια δομή γραφήματος και γι αυτό χρησιμοποιούμε και πάλι ένα γράφημα με επίπεδα για να προσομοιώσουμε την λίστα. Όμως, η βασική διαφορά ανάμεσα στη στοίβα και στην συνδεδεμένη λίστα είναι ότι στην δεύτερη οι κόμβοι μπορούν να προστεθούν και να αφαιρεθούν σε οποιοδήποτε σημείο του γράφου ,σε αντίθεση με τη στοίβα που οι εισαγωγές και διαγραφές γίνονται στην κορυφή της στοίβας. Η κύρια δυσκολία ,λοιπόν, στην κατασκευή μίας λίστας ανθεκτικής σε λάθη είναι να διατηρήσουμε την δομή της παρόλο που γίνονται δυναμικές αλλαγές. Στο σχήμα που ακολουθεί απεικονίζεται ένα γράφημα μιας FT-λίστας:



Εικόνα 4: Μία d-FT λίστα με d=4 και M=32 [3].

Το γράφημα μιας d-FT λίστας αποτελείται από μία ακολουθία από κομμάτια B^0,…, B^{ihead} .Κάθε κομμάτι Bi αποτελείται από 2d (log(2d)+1) κορυφές ενωμένες σε μία δομή πεταλούδας. Επίσης, κάθε κομμάτι έχει έναν κόμβο κεφαλή που συμβολίζεται με header(B^i).Οι κορυφές στο τελευταίο επίπεδο του κομματιού B^i κατευθύνονται σε εκείνες στο πρώτο επίπεδο του B^{I-1}.Ο χειριστής μιας d-FT λίστας αποτελείται από 2d+1 κόμβους. Ένας κόμβος χειριστής, κεφαλή, δείχνει στη κορυφή της λίστας. Οι υπόλοιποι κόμβοι χειριστές,h[k],k=0,..,2d-1 δείχνουν στο πρώτο επίπεδο που πρώτου κομματιού. Αυτό αποτελεί τον σκελετό του γραφήματος. Η πρωτότυπη συνδεδεμένη λίστα αποτυχαίνει στο σκελετό ως εξής. Κάθε κόμβος x της συνδεδεμένης λίστας αντιστοιχίζεται σε μια κορυφή s του σκελετού (χρησιμοποιούμε τον όρο «κόμβος» για τους κόμβους της συνδεδεμένης λίστας και «κορυφή» για εκείνους του σκελετού). Η κορυφή s αποθηκεύει το σύνολο των πληροφοριών του x, συμπεριλαμβανομένων των πεδίων δεδομένων του επόμενο δείκτη. Το πολύ ένας κόμβο λίστας αντιστοιχίζεται σε κάθε κορυφή του σκελετού. Το γράφημα διατηρεί τη σειρά των κόμβων κατά μήκος των κομματιών. Εάν τα x και y είναι κόμβοι του συνδέονται στην αρχική λίστα και το χ είναι πριν y, τότε τα x και y είναι στο ίδιο μπλοκ, ή το χ είναι σε ένα μπλοκ πριν από αυτό του y. Μέσα σε κάθε μπλοκ, οι κόμβοι τοποθετούνται αυθαίρετα. Οι κενές κορυφές του κάθε κομματιού B^i συνδέεται σε μία λίστα με όνομα $free_i$.Ένας δείκτης από την κορυφή της $frre_i$ αποθηκεύεται στον header(i).Ακόμα, ο header(i) διατηρεί μία μεταβλητή $Load_i$,η οποία μετράει τον συνολικό αριθμό κόμβων που υπάρχουν στην κορυφές του B^I. Έστω M=2dlog(4d) ο αριθμός των κορυφών σε ένα κομμάτι. Έχουμε την παρακάτω συνθήκη:

Κάθε στιγμή τουλάχιστον M/4 κόμβοι υπάρχουν σε κάθε κομμάτι Bi(i>1) και το πολύ M σε κάθε κομμάτι με i≥0.Δηλαδή μόνο το τελευταίο κομμάτι μπορεί περιέχει λιγότερα από M/4 κόμβους.

2.8.2.2 ΛΕΙΤΟΥΡΓΙΕΣ

Οι λειτουργίες Head,Next() και Value() είναι εύκολο να υλοποιηθούν καθώς υπάρχει ένα ολοκληρωμένο αντίγραφο στην d-FT λίστα. Σε μία λίστα d-FT δεν είναι αρκετός ένας μόνο δείκτης σε κάθε θέση του γραφήματος καθώς μπορεί να καταστραφεί. Έτσι εκτός από τον βασικό δείκτη p έχουμε ένα σύνολο από 2d δείκτες που δείχνουν στους κόμβους του πρώτου επιπέδου του τρέχοντος κομματιού. Οι δείκτες αρχικοποιούνται στον χειριστή και ενημερώνονται όταν αλλάζει η τρέχουσα τοποθεσία από ένα κομμάτι στο επόμενο. Το κόστος των λειτουργιών αυτών είναι O(1).Η λειτουργία της εισαγωγής ακολουθεί τα παρακάτω βήματα:

1.Δημιουργία ενός νέου κόμβου. Τοποθετούμε σ αυτόν ην τιμή u και τον τοποθετούμε στο αντίγραφο της συνδεδεμένης λίστας.

2.Υποθέτουμε ότι το x είναι στο κομμάτι B^i και το τοποθετούμε σε μία ελεύθερη κορυφή s στο B^i.

3.Ενημερώνουμε την λίστα free.

4.Αν $Load_i$=M, σπάμε το B^i σε δύο κομμάτια.

Για να σπάσουμε ένα κομμάτι πρώτα φτιάχνουμε δύο κομμάτια σκελετού και στη συνέχεια τοποθετούμε τα πρώτα μισά στο πρώτο και τα υπόλοιπα στο δεύτερο. Ο διαχωρισμός αυτός απαιτεί O(M) λειτουργίες.

Η διαγραφή είναι η αντίθετη διαδικασία από την εισαγωγή.

1.Αφαιρούμε το x από το αντίγραφο της λίστας

2.Υποθέτουμε ότι το x είναι τοποθετημένο στην κορυφή s του Bi.Διαγράφουμε το x από το s και προσθέτουμε το s στην freei.Αυξάνουμε την $Load_i$ κατά ένα.

3.Αν $Load_i$<M/4 και υπάρχει το B^{i-1} ,κάνουμε ένα από τα δύο. Αν $Load_i$<=3M/4 τότε ενώνουμε τα B^i και B^{i-1} ή χωρίζουμε τους κόμβους ανάμεσα στα B^i και B^{i-1}.

Οι ενώσεις/διαιρέσεις ολοκληρώνονται σε O(M) λειτουργίες.

2.8.2.3 ΣΦΑΛΜΑΤΑ ΚΑΙ Η ΑΝΤΙΜΕΤΩΠΙΣΗ ΤΟΥΣ

Σε περίπτωση που εντοπιστεί ένα λάθος, αρχίζει η διαδικασία ανακατασκευής, η οποία υλοποιείται σε τρεις φάσεις.

1.Συλλέγουμε όσους από τους κόμβους της λίστας μπορούμε.

2.Υποκαθιστούμε την σωστή διάταξη ανάμεσα στους κόμβους που συλλέξαμε.

3.Ανακατασκευάζουμε την δομή δεδομένων

Αναλυτικότερα, στην φάση 1,προκειμένου να βρεθούν οι υπόλοιποι κόμβοι που χρησιμοποιούμε στην υποκείμενη δομή της πεταλούδας του σκελετού. Υπενθυμίζουμε, ότι ανά πάσα στιγμή ο χρήστης κρατά 2d δείκτες στο πρώτο επίπεδο του τρέχοντος κομματιού. Έτσι, προκειμένου να βρει τους υπόλοιπους κόμβους αρχίζουμε από το πρώτο στρώμα του τρέχοντος κομματιού, αυξάνουμε ένα επίπεδο κάθε φορά και μαζεύω τους κόμβους της λίστας που βρίσκω στη διαδρομή. Συνεχίζουμε μέχρι να συναντήσουμε ένα κομμάτι χωρίς σφάλματα. Γνωρίζουμε ότι από f εσφαλμένους κόμβους ,το πολύ O(f*logf) κόμβοι μπορούν να γίνουν απρόσιτοι. Στη φάση δύο, το βασικό μας πρόβλημα είναι να διατηρήσουμε την σωστή ταξινόμηση των κόμβων οι οποίοι διασώθηκαν. Αυτό το πρόβλημα λύνεται με την χρήση σημαιών. Πιο συγκεκριμένα για κάθε κόμβο x

αποθηκεύουμε έναν ακέραιο ,έστω tag(x) έτσι ώστε δύο κόμβοι οι οποίοι βρίσκονται στο ίδιο κομμάτι και ο x βρίσκεται μπροστά από τον y, ισχύει tag(x)<tag(y).Έτσι ανακατασκευάζουμε εύκολα την λίστα. Η κατασκευή της λίστας με σημαίες διατηρεί τις σημαίες αυτές σε μία δυναμική λίστα με O(logM) μέγεθος και O(logM) κόστος ανά εισαγωγή ή διαγραφή για μία λίστα με μέγεθος M. Για να μειώσουμε το κόστος σε χρόνο αυτών των λειτουργιών χρησιμοποιούμε έναν αλγόριθμο «έμμεσης κατεύθυνσης» που μπορεί να επιτύχει O(1) κόστος ανά εισαγωγή/διαγραφή.

Ισχυρισμός 4:Με f λανθασμένους κόμβους το πολύ O(flogf) κόμβοι είναι απρόσιτοι.

ΘΕΩΡΗΜΑ 2ο: Για κάθε d η d-FT λίστα είναι μία πιστή προσομοίωση μιας συνδεδεμένης λίστας και η ανθεκτικότητά της σε λάθη είναι (d,O(logflogd)) με σεβασμό στην σχέση «προηγούμενος-επόμενος».

ΑΠΟΔΕΙΞΗ: Τουλάχιστον το ένα τέταρτο των εγγραφών δεν είναι άδειο. Μία d-FT λίστα απαιτεί ένα παράγοντα O(1) περισσότερο χώρο από μία συνδεδεμένη λίστα. Οι συναρτήσεις Value(),Next() και Head χρειάζονται γιαν α εκτελεστούν έναν σταθερό αριθμό βημάτων. Οι εισαγωγές και διαγραφές απαιτούν O(1) λειτουργίες. Μετά από Ω(dlogd) εισαγωγές ή διαγραφές σε ένα τμήμα, το τμήμα αυτό θα πρέπει να διαχωριστεί ή να ενωθεί, αντίστοιχα. Αυτές οι λειτουργίες αποτελούνται από O(d*logd) λειτουργίες στον σκελετό ακολουθούμενες από O(dlogd) εισαγωγές. Έτσι οι πράξεις ανά εισαγωγή και διαγραφή είναι O(1).

Από τον ισχυρισμό 4,με f<d σφάλματα, το πολύ O(f*logf) κορυφές του σκελετού χάνονται. Κάθε κορυφή αποθηκεύει το πολύ έναν κόμβο της λίστας. Επομένως, το πολύ O(f*logf) κόμβοι της λίστας μπορεί να χαθούν. Εξαιτίας των ανακατευθύνσεων O(f*logf) χαμένων κόμβων ίσως έχει ως αποτέλεσμα τον χαμό της πληροφορίας ταξινόμησης για O(f*logf*logd) κόμβους, κάνοντας τους μη χρηστικούς στην ανακατασκευή. Το πολύ f+1 συνεχόμενα κομμάτια έχουν απρόσιτους κόμβους και έτσι η ανακατασκευή ολοκληρώνεται ο σε $O(fd\log^2 d)$ λειτουργίες.

2.8.3 ΑΝΟΧΗ ΒΛΑΒΩΝ ΣΕ ΔΥΑΔΙΚΑ ΔΕΝΤΡΑ

2.8.3.1 ΔΟΜΗ

Για την μελέτη της ανθεκτικότητας σε λάθη των δέντρων, θα θεωρήσουμε ένα δυαδικό δέντρο που υποστηρίζει τις παρακάτω διαδικασίες:

-Insert(u,x):Εισάγει ένα κόμβο με την τιμή u σαν παιδί του κόμβου x.Ο κόμβος εισάγεται σαν φύλλο ή ανάμεσα σε παιδί και γιό.

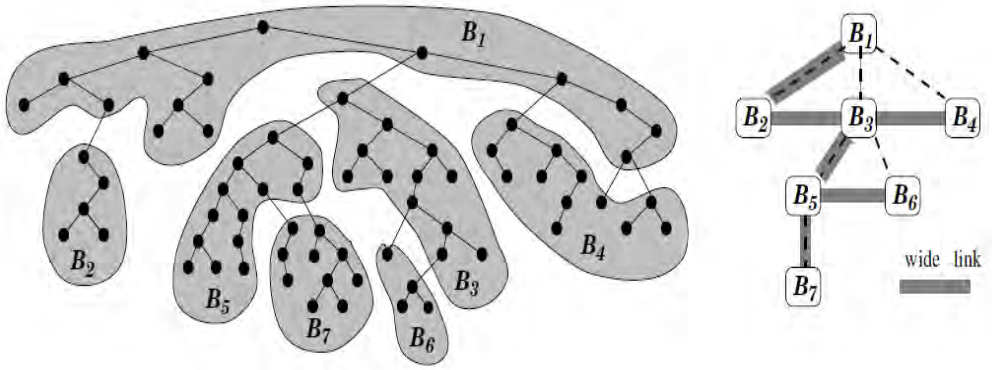
-Delete(x):διαγράφει τον κόμβο x από το δέντρο. Μόνο κόμβοι που έχουν το πολύ ένα παιδί μπορούν να διαγραφούν.

-Find(u):Αναζήτηση του κόμβου u ξεκινώντας από την ρίζα.

Κατασκευάζουμε μία οικογένεια δομών δεδομένων d-FT δέντρων, με το d να είναι δύναμη του δύο, που είναι ένα ακριβές αντίγραφο ενός δυαδικού δέντρου αναζήτησης. Για κάθε d, η d-FT λίστα έχει (d,O(f*logf*logd)) ανθεκτικότητα σε λάθη με παράλληλη ισχύ των σχέσεων «πάνω -κάτω» και «αριστερά-δεξιάά».

Επιβεβαιώνουμε ότι το βάθος του φύλλου στην ανακατασκευασμένη δομή δεν είναι περισσότερο από την παλιά δομή. Μικρότερες υλοποιήσεις κάνουν την ανοχή βλαβών στο δέντρο αναζητήσεις όλο και μικρότερες.

Θεωρούμε ένα δέντρο τμημάτων, έστω Τ. Όπως στις λίστες FT, χρησιμοποιούμε ένα γράφημα τμημάτων σαν σκελετό. Κάθε τμήμα του d-FT δέντρου αποτελείται από M=2d(log(2d)+1) κορυφές διασυνδεδεμένες σε μία δομή πεταλούδας. Κάθε τμήμα έχει μία συγκεκριμένη κορυφή οδηγό, που αποθηκεύει την ελεύθερη λίστα και φορτώνεται από το τμήμα. Οι κόμβοι του δέντρου Τ αντιστοιχίζονται στις κορυφές των τμημάτων έτσι ώστε κάθε κορυφή να έχει το πολύ έναν κόμβο. Με τον όρο κορυφή θα αναφερόμαστε στις κορυφές του τμήματος και με τον όρο node αναφερόμαστε στους κόμβους του γνήσιου δέντρου Τ. Και λέμε ότι ένα τμήμα «περιέχει» τους nodes που αντιστοιχούν σ αυτό. Τα τμήματα είναι λογικά συνδεδεμένα σε μία δομή δέντρου, την οποία αποκαλούμε δέντρο τμημάτων και συμβολίζεται με BT.Ένα παράδειγμα δέντρου που περιγράφουμε φαίνεται αμέσως μετά.



Εικόνα 5: Ένα d-FT δέντρο με d=4 και M=32.Στο αριστερό μέρος φαίνονται οι κόμβοι στα τμήματα που ανήκουν. Στο δεξί μέρος φαίνονται οι λογικές συνδέσεις των τμημάτων με διακεκομμένες γραμμές και οι άμεσες συνδέσεις με μεγάλες γκρι γραμμές. Τα τμήματα B_1 B_3 B_5 έχουν παιδιά και είναι ενιαία οπότε περιέχουν περισσότερους από M/6 κόμβους.Τα τμήματα B_4 B_6 B_7 είναι κομματιασμένα και δεν έχουν παιδιά. Το τμήμα B_6 περιέχει λιγότερα από M/6 κόμβους αλλά έχει ένα άμεσο αδελφό,το B_5, το οποίο είναι ενιαίο τμήμα και περιέχει περισσότερους από M/6 κόμβους [3].

Η αντιστοίχιση των κόμβων του Τ στα αντίστοιχα τμήματα διατηρούν τις ακόλουθες προτάσεις:

-Θεωρούμε ότι child(B) είναι ένα παιδί του B στο BT.Θεωρούμε ότι T1,…,Tκ είναι το δάσος των υποδέντρων του Τα που περιέχονται στο child(B).

-Αν R-Sib(B) είναι το τμήμα στα δεξιά του B στο δέντρο BT,τότε όταν φαίνονται μέσα από το Τα, οι κόμβοι περιέχονται στο R-Sib(B) είναι στα δεξιά από αυτά που περιέχονται στο B. Για ένα τμήμα B, αν όλοι οι κόμβοι περιέχονται στο B είναι συνδεδεμένοι σε ένα κομμάτι, λέμε ότι το B είναι ένα ενιαίο τμήμα.

Διαφορετικά είναι ένα κομματιασμένο τμήμα. Τα τμήματα συνδέονται μεταξύ τους με της έντονες γραμμές. Για την κατασκευή μιας τέτοιας έντονης γραμμής μεταξύ των τμημάτων Β και Β' διατηρούμε τους δείκτες μεταξύ των αντίστοιχων κορυφών στα Β και Β'. Οι έντονες γραμμές διατηρούνται μεταξύ των τμημάτων με τα εξείς χαρακτηριστικά:

-Μεταξύ του Β και του αριστερού του παιδιού

-Μεταξύ του Β και του δικού του άμεσου αδελφιού του στα δεξιά

Επομένως, όλα τα παιδιά ενός δοσμένου τμήματος είναι συνδεδεμένα σε μία λίστα με έντονη γραμμή, που έχει στην αρχή τον κόμβο πατέρα. Ο χειριστής του FT δέντρου αποτελείται από 2d δείκτες στο πρώτο επίπεδο του κομματιού της ρίζας.

Τα FT δέντρα έχουν την ακόλουθη συνθήκη:

Συνθήκη 2:Η δομή των FT δέντρων έχει ανά πάσα στιγμή τα ακόλουθα χαρακτηριστικά:

-Το πολύ M κόμβοι υπάρχουν σε κάθε τμήμα

-Μόνο τα ενιαία τμήματα έχουν παιδιά

-Αν ένα τμήμα Β έχει παιδί τότε το Β περιέχει τουλάχιστον M/6 κόμβους

-Αν το Β περιέχει λιγότερους από M/6 κόμβους, τότε είτε το Β είναι το μοναδικό τμήμα στο δέντρο, είτε ο άμεσος αδελφός του ,στα αριστερά ή στα δεξιά, είναι ενιαίο τμήμα και περιέχει τουλάχιστον M/6 κόμβους.

Από την συνθήκη 2 παρατηρούμε τον ακόλουθο ισχυρισμό ,που μας δίνει τα όρια του μεγέθους του FT δέντρου και το ποσοστό των τμημάτων στο δέντρο.

Ισχυρισμός 5: Το μέγεθος ενός FT δέντρου είναι γραμμικό στο μέγεθος του T.

Ισχυρισμός 6: Κάθε τμήμα έχει το πολύ 2M τμήματα-παιδιά

Οι αποδείξεις των ισχυρισμών 5 και 6 υπάρχουν στην πηγή [3].

Οι λειτουργίες που υποστηρίζει ένα δέντρο FT είναι: εισαγωγή, διαγραφή και διαχωρισμός και η υλοποίησή τους περιγράφεται αναλυτικότερα στην πηγή [3].

Ισχυρισμός 7: Το κόστος διαχωρισμού και ένωσης τμημάτων είναι O(1).

Απόδειξη στην πηγή [3].

2.8.3.2 ΣΦΑΛΜΑΤΑ ΚΑΙ Η ΑΝΤΙΜΕΤΩΠΙΣΗ ΤΟΥΣ

Ας προχωρήσουμε στον εντοπισμό λαθών και την ανακατασκευή της δομής δεδομένων μας. Με τον εντοπισμό λάθους, ξεκινάει και η ανακατασκευή. Η ανακατασκευή ακολουθεί την δομή δέντρου FT.Για την ανακατασκευή ενός τμήματος, πρέπει αρχικά να ανακατασκευάσουμε επαναληπτικά όλα τα παιδιά του.

Οι φαρδιές γραμμές, που συνδέουν την λίστα που περιέχει τα παιδιά, εγγυάται ότι όλα τα παιδία είναι αξιόπιστα. Αυτό ισχύει γιατί οι έντονες γραμμές παρέχουν 2d ξεχωριστούς κόμβους με διαφορετικά μονοπάτια από το τμήμα πατέρα προς όλα τα παιδιά και το πολύ d από αυτά μπορεί να περιέχουν λανθασμένα δεδομένα.

Αφού ανακατασκευαστούν όλα τα τμήματα-παιδιά, και το τμήμα πατέρας έχε ανακατασκευαστεί. Η ανακατασκευή κάθε τμήματος αποτελείται από τρία βήματα:

1.Διασώζουμε τους εναπομείναντες προσβάσιμους κόμβους.

2.Προσδιορίζουμε τη σωστή τοπολογική ταξινόμηση ανάμεσα στους εναπομείναντες κόμβους του τμήματος και μεταξύ των κόμβων του κομματιού και εκείνους των παιδιών αν υπάρχουν.

3.Ανακατασκευάζω το τμήμα

Η φάση (1) εκτελείται χρησιμοποιώντας την δομή πεταλούδας σα σκελετό του τμήματος. Σε αναλογία με τον ισχυρισμό 2, σε περίπτωση που t κόμβοι είναι κατεστραμμένοι, τότε το πολύ O(tlogt) κόμβοι του τμήματος έχουν χαθεί .Στο επόμενο βήμα, που προσπαθούμε να διατηρήσουμε τη σωστή διάταξη των τμημάτων ,η δυσκολία είναι ότι δεδομένου πως κάποιοι κόμβοι έχουν χαθεί, ίσως χάσουμε πληροφορία σχετικά με την διάταξη των κόμβων .Η αντιμετώπιση αυτού του προβλήματος είναι και πάλι η χρήση σημαιών .Όταν χάνονται κάποιοι κόμβοι, είναι πιθανό οι κόμβοι που περισώζονται από τη δομή να μην έχουν τη μορφή ενός δυαδικού δέντρου. Αν για παράδειγμα χαθεί η ρίζα του δέντρου αυτόματα δημιουργούνται δύο ξεχωριστά δέντρα. Ακόμη, αν ένας κόμβος, έστω u, έχει δύο παιδιά και ο πατέρας του ,έστω w,που και αυτός έχει δύο παιδιά ,χαθούν τότε ακόμα και να ενώσουμε τα παιδιά του u σαν παιδιά του w θα διατηρηθούν οι συνθήκες «πάνω- κάτω» και «αριστερά-δεξιάά» αλλά όμως το δέντρο δε θα είναι πλέον ένα δυαδικό δέντρο. Στις παραπάνω περιπτώσεις εισάγουμε ψεύτικους κόμβους στην διαδικασία ανακατασκευής. Οι κόμβοι αυτοί δεν περιέχουν δεδομένα και χρησιμοποιούνται για να διατηρήσει η δομή μας τη μορφή του δυαδικού δέντρου. Οι σημαίες μας διευκολύνουν να καταλάβουμε αν χρειαζόμαστε να βάλουμε ψεύτικους κόμβους στη δομή μας. Με την πάροδο των χρόνων, τρεις μελέτες έμειναν για τον τρόπο εισαγωγής σημαιών στους κόμβους. Η πρώτη με O(logn) επιβάρυνση ανά εισαγωγή/διαγραφή, στην οποία κάθε κόμβος έχει δύο σημαίες[3].Η δεύτερη που έχει O(logd) πολυπλοκότητα και η τελευταία η οποία έχει επιβάρυνση O(1),αλλά το κόστος αυξάνεται κατά ένα παράγοντα O(logd) στον αριθμό των χαμένων κόμβων.

Για να οδηγηθούμε σ αυτήν την τελευταία λύση χρησιμοποιούμε πλάγια μέσα. Όπως και στην συνδεδεμένη λίστα αυτά τα πλάγια μέσα μπορεί να επιφέρουν μία κατάσταση όπου ένας κόμβος μπορεί να είναι διασωσμένος αλλά να έχει χαθεί η θέση του στο δέντρο δεδομένου ότι οι σημαίες που ταξινομούνται υψηλότερα είναι αποθηκευμένες σε έναν άλλο κόμβο ο οποίος έχει χαθεί. Κάθε κόμβος αποθηκεύει τέτοιου τύπου σημαίες για το πολύ O(logd) άλλους κόμβους. Γι αυτό ο αριθμός των χαμένων κόμβων αυξάνεται το πολύ κατά ένα παράγοντα O(logd).Οδηγούμαστε έτσι στους ακόλουθους ισχυρισμούς.

Ισχυρισμός 10: Χρησιμοποιώντας της σημαίες με πλάγια μέσα, η κάθε εισαγωγή και διαγραφή απαιτούν O(1) βήματα και με f σφάλματα, το πολύ O(f*logf*logd) κόμβοι χάνονται.

Ισχυρισμός 11:Η ανακατασκευή απαιτεί O(poly(fd)) βήματα.

Η επαναληπτική διαδικασία ανακατασκευής σταματά όταν φτάσει σε ένα τμήμα χωρίς απρόσιτους κόμβους. Υπάρχουν το πολύ O (flogf) απρόσιτοι κόμβοι. Ως εκ τούτου, το πολύ O(flogf) τμήματα υποβάλλονται σε ανακατασκευή. Το έργο της ανακατασκευής ενός τμήματος είναι πολυονιμικό στον αριθμό των κόμβων του τμήματος που είναι O (dlogd).

ΘΕΩΡΗΜΑ 3ο: Για κάθε d, το d-FT δέντρο είναι ένα ακριβές αντίγραφο του δυαδικού δέντρου λεξικού και η ανοχή του σε λάθη είναι (d,O(f*logf*logd)) ενώ παράλληλα πάντα ισχύουν οι σχέσεις «πάνω- κάτω» και «αριστερά- δεξιά» .

Κλείνοντας η ανοχή βλαβών με αποσυμπιεστές στις τρεις δομές δεδομένων που μελετήσαμε έχει ως εξής: Χρησιμοποιώντας αποσυμπιεστές ο αριθμός των χαμένων κόμβων μπορεί να περιοριστεί σε O(f) και οδηγούμαστε στα ακόλουθα θεωρήματα:

ΘΕΩΡΗΜΑ 4ο:Για κάθε d, μία d-EFT στοίβα είναι ένα ακριβές αντίγραφο της στοίβας και έχει O(d,O(f)) ανοχή σε λάθη με σεβασμό στην σχέση «πάνω -κάτω»

ΘΕΩΡΗΜΑ 5ο:Για κάθε d, η d-EFT λίστα είναι ένα ακριβές αντίγραφο της συνδεδεμένης λίστας και έχει (d,O(flogd)) ανοχή βλαβών με σεβασμό στην σχέση «προηγούμενος- επόμενος»

ΘΕΩΡΗΜΑ 6ο:Για κάθε d, το d-EFT δέντρο είναι ένα ακριβές αντίγραφο του δυαδικού δέντρου αναζήτησης. Έχει (d,O(flogd)) ανοχή σε λάθη με σεβασμό στις σχέσεις «πάνω - κάτω» και «αριστερά -δεξιά».

2.9 ΑΝΘΕΚΤΙΚΕΣ ΣΕ ΛΑΘΗ ΟΥΡΕΣ ΠΡΟΤΕΡΑΙΟΤΗΤΑΣ

Σ' αυτήν την ενότητα θα μελετήσουμε το σχεδιασμό και την ανάλυση πολυπλοκότητας μιας ουράς προτεραιότητας με ένα μοντέλο εσφαλμένης μνήμης RAM[19].Αυτό χρησιμοποίει O(n) χώρο για την ταξινόμηση n στοιχείων και εκτελεί προσθέσεις και αφαιρέσεις σε O(logn+δ) χρόνο. Η ουρά προτεραιότητας που αναφερόμαστε ταιριάζει τα όρια μιας βέλτιστης ουράς προτεραιότητας που είναι βασισμένη στις συγκρίσεις ενώ μπορεί να ανέρθει O(logn) σφάλματα. Είναι σημαντική βελτίωση από τη χρήση του ανθεκτικού δέντρου αναζήτησης (εικόνα 8) σαν ουρά προτεραιότητας ,δεδομένου ότι χρησιμοποίει $O(\log n+\delta^2)$ χρόνο ανά λειτουργιά και ακόμη ανέχεται $O(\sqrt{\log n})$ σφάλματα για να διατηρήσει το O(logn) σαν όριο ανά λειτουργιά. Η ουρά προτεραιότητας που μελετάμε είναι η πρώτη ανθεκτική δομή δεδομένων που επιτρέπει O(logn) καταστροφές στοιχείων, ενώ παράλληλα διατηρεί τα βέλτιστα όρια στο μοντέλο RAM.Ακόμη, δεν αποθηκεύει στοιχειά σε αξιόπιστες θέσεις μνήμης ανάμεσα στις λειτουργιές, αλλά μόνο δομικές πληροφορίες όπως δείκτες. Αποδεικνύουμε ότι κάθε βασισμένη σε συγκρίσεις ανθεκτική ουρά προτεραιότητας που συμπεριφέρεται με αυτόν τον τρόπο απαιτεί στην χειρότερη περίπτωση Ω(logn+δ) χρόνο για λειτουργιές όπως εισαγωγές και διαγραφές στοιχείων.

Η ανθεκτική ουρά προτεραιότητας είναι βασισμένη στην ουρά προτεραιότητας που αγνοεί την κρυφή μνήμη cache.Η βασική ιδέα είναι να μαζεύουμε τα στοιχειά σε μεγάλα ταξινομημένα κομμάτια με αυξανόμενο μέγεθος, έτσι ώστε να μην απαιτούνται πολύ συχνές ενημερώσεις των κομματιών αυτών διαδικασία η όποια είναι ακριβή σε χώρο και χρόνο. Τα μικρότερα κομμάτια θα περιέχουν τα μικρότερα στοιχειά, έτσι ώστε να μπορούν να ανακτηθούν γρηγορότερα από την λειτουργιά της αφαίρεσης .Χρησιμοποιούμε πολλές φορές τον αλγόριθμο συγχώνευσης για να μετακινούμε τα στοιχειά από το ένα κομμάτι στο άλλο. Η επιπλέον δουλεία που απαιτείται, εξαιτίας του μεγάλου μεγέθους των κομματιών για την αντιμετώπιση και των σφαλμάτων με τον αλγόριθμο

συγχώνευσης γίνεται αμελητέα συγκρινόμενη με την συνολική δουλειά που γίνεται.

Ας ορίσουμε την ανθεκτική ουρά προτεραιότητας και ας κάνουμε μια εισαγωγή σε μερικές έννοιες που θα μας χρειαστούν στη συνεχεία για την μελέτη του αλγορίθμου. Δίνονται δυο ακολουθίες Χ και Υ και συμβολίζουμε την αλληλουχία αυτών των δυο ακολουθιών με ΧΥ. Μια ακολουθία Χ είναι απολυτά διατεταγμένη αν τα μη κατεστραμμένα κλειδιά της εμφανίζονται σε μη ελαττωμένη διάταξη. Τελικά, μια αξιόπιστη τιμή είναι μια τιμή που είναι αποθηκευμένη σε αναξιόπιστη μνήμη η όποια μπορεί ανακτηθεί αξιόπιστα παρά τις πιθανές καταστροφές. Αυτό επιτυγχάνεται με την επιστροφή της δοσμένης τιμής 2δ+1 φόρες. Ανακτώντας μια αξιόπιστη τιμή σπαταλούμε Ο(δ) χρόνο χρησιμοποιώντας των αλγόριθμο πλειοψηφίας που αναλύσαμε στην παράγραφο 3.2,ο όποιος διαπερνά τις 2δ+1 τιμές διατηρώντας τον υποψήφιο με την πλειοψηφία των ψήφων και τον μετρητή σε αξιόπιστη μνήμη.

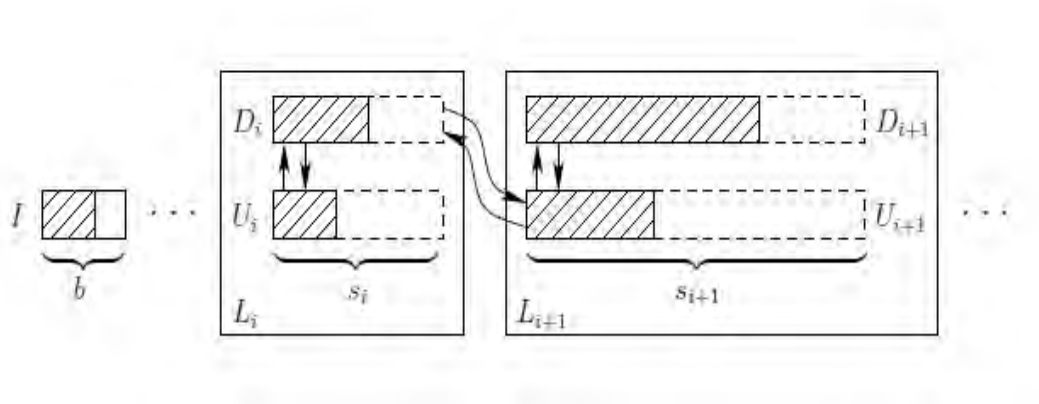
Ορισμός 1:

Μια ανθεκτική ουρά προτεραιότητας διατηρεί ένα σύνολο στοιχείων υπό τις λειτουργίες της εισαγωγής και της διαγραφής. Μια εισαγωγή προσθέτει ένα στοιχειό και μια διαγραφή αφαιρει και επιστρέφουν το ελάχιστο μη κατεστραμμένο στοιχειό ή ένα κατεστραμμένο.

Σημειώνουμε ότι ο ορισμός μιας ανθεκτικής ουράς προτεραιότητας συμφωνεί με τον ανθεκτικό αλγόριθμο ταξινόμησης. Δίνεται μια ακολουθία από n στοιχειά. Εισάγοντας αυτά τα στοιχειά σε μια ανθεκτική ουρά προτεραιότητας ακολουθουμένη από n λειτουργιές διαγραφής παράγεται μια απόλυτα ταξινομημένη ακολουθία.

Σ' αυτήν την παράγραφο θα κάνουμε μια εισαγωγή στην ανθεκτική ουρά προτεραιότητας που μοιάζει με την ουρά προτεραιότητας που δεν έχει σχέση με μνήμη cache.Τα στοιχειά αποθηκεύονται σε απολυτά ταξινομημένες λίστες και μετακινούνται χρησιμοποιώντας δυο θεμελιώδεις ενέργειες τα PUSH και PULL,βασισμένα σε έμπιστες συγχωνέψεις.

Όσον αφόρα ,τώρα, τη δομή της ουράς προτεραιότητας αποτελείται από έναν προσωρινό αποθηκευτικό χώρο(buffer) έστω I μαζί με έναν αριθμό από επίπεδα έστω $L_0,..,L_k$, με k=O(logn).Κάθε επίπεδο L_i περιέχει έναν πάνω αποθηκευτικό χώρο, έστω Ui, και έναν κάτω αποθηκευτικό χώρο, έστω D_i,που παρουσιάζονται σαν πινάκες. Οι πάνω αποθηκευτικοί χώροι περιέχουν μεγάλα στοιχειά που είναι στο δρόμο για τα πιο πάνω επίπεδα της ουρά προτεραιότητας και οι κάτω αποθηκευτικοί χώροι περιέχουν μικρά στοιχεία που είναι στο δρόμο προς τα χαμηλότερα επίπεδα. Οι προσωρινοί χώροι αποθήκευσης της ουράς προτεραιότητας είναι αποθηκευμένοι σαν μια διπλή λίστα $U_0,D_0,...,U_k,D_k$, όπως φαίνεται στη παρακάτω εικόνα.



Εικόνα 6: Απεικονίζονται οι προσωρινοί χώροι αποθήκευσης (buffers),D και U, δύο επιπέδων ,L_i και L_{i+1}.Ακομή φαίνεται η σύνδεση του D_i με τον L_{i+1}, του επόμενου επιπέδου και η σύνδεση των δύο αποθηκευτικών χώρων του κάθε επιπέδου [19].

Για κάθε πάνω και κάτω προσωρινό αποθηκευτικό χώρο αποθηκεύουμε αξιόπιστα τους δείκτες στους γειτονικούς προσωρινούς αποθηκευτικούς χώρους στη λίστα και το μέγεθος τους. Στην αξιόπιστη μνήμη αποθηκεύουμε δείκτες στο I,U_0 και D_0,μαζι με το |I|. Δεδομένου ότι η θέση του πρώτου στοιχείου στο U_0 και D_0 δεν είναι πάντα στο πρώτο κελί της μνήμης του αντίστοιχου προσωρινού χώρου αποθήκευσης, εμείς θα πρέπει πάντα να αποθηκεύουμε τον δείκτη του πρώτου στοιχείου σ' αυτούς τους χώρους προσωρινής αποθήκευσης σε αξιόπιστη θέση μνήμης. Ο προσωρινός χώρος αποθήκευσης εισαγωγής ,I, περιέχει περισσότερα από $b=\delta+\log n+1$ στοιχεία .Για το επίπεδο L_i ορίζουμε το κατώφλι s_i με $s_0=2(\delta^2+\log^2 n)$ και $s_i=2s_{i-1} = 2^{i+1}(\delta^2+\log^2 n)$όπου n είναι ο αριθμός των στοιχείων στην ουρά προτεραιότητας. Χρησιμοποιούμε αυτά τα κατώφλια για να αποφασίζουμε αν ένας πάνω-αποθηκευτικός χώρος περιέχει παρά πολλά στοιχεία ή αν ένας κάτω-αποθηκευτικός χώρος έχει πολύ λίγα. Για χάρη απλότητας, οι πάνω και κάτω προσωρινοί αποθηκευτικοί χώροι αναπτύσσονται και συρρικνώνονται όπως απαιτείται κατά τη διάρκεια της εκτέλεσης του αλγορίθμου έτσι ώστε να μην χρησιμοποιούμε επιπλέον χώρο.

Για για κατασκευάσουμε την ουρά προτεραιότητας διατηρούμε τα ακόλουθα αμετάβλητα χαρακτηριστικά για τους πάνω και κάτω προσωρινούς χώρους αποθήκευσης.

Αμετάβλητα χαρακτηριστικά ταξινόμησης:

1.Ολοι οι χώροι προσωρινής αποθήκευσης είναι απολυτά ταξινομημένοι

2.D_iD_{i+1} και D_iU_{i+1} είναι απόλυτα ταξινομημένοι για $0\le i<k$

Αμετάβλητα χαρακτηριστικά μεγέθους:

3.$s_i/2\le|D_i|\le s_i$ για $0\le i<k$

4.$|U_i|\le s_i/2$, για $0\le i<k$

Με το να διατηρούμε όλους τους πάνω και κάτω προσωρινούς χώρους αποθήκευσης απόλυτα ταξινομημένους, είναι πιθανό να μετακινούμε στοιχειά μεταξύ γειτονικών επίπεδων αποτελεσματικά, χρησιμοποιώντας την συνένωση. Από το χαρακτηριστικό 2, όλα τα μη κατεστραμμένα στοιχειά στο D_i είναι μικρότερα από όλα τα μη κατεστραμμένα στοιχειά στα D_{i+1} και U_{i+1}.Αυτο επιβεβαιώνει ότι τα μικρά στοιχειά ανήκουν στα χαμηλότερα επίπεδα της ουράς προτεραιότητας. Αξίζει να αναφέρουμε ότι δεν υπάρχει καμιά σχέση ανάμεσα στα στοιχειά του πάνω και κάτω προσωρινού αποθηκευτικού χώρου του ίδιου επίπεδου. Τα αμετάβλητα χαρακτηριστικά μεγέθους επιτρέπουν τα μεγέθη των προσωρινών αποθηκευτικών χωρών να κυμαίνονται σε ένα μεγάλο φάσμα. Έτσι, $\Omega(s_i)$ λειτουργίες εισαγωγής ή διαγραφής στοιχείου πραγματοποιούνται στον ίδιο χώρο στο L_i, δίνοντας τα επιθυμητά όρια στα κόστη.

Δεδομένου ότι οι τιμές s εξαρτώνται από το n,οποτεδήποτε το μέγεθος της ουράς προτεραιότητας αυξάνεται ή μειώνεται κατά Θ(n),εκτελούμε μια ολική ανοικοδόμηση. Αυτή η ανοικοδόμηση πραγματοποιείται συλλέγοντας όλα τα στοιχειά, ταξινομώντας τα με τον βέλτιστο αλγόριθμο ταξινόμησης και διανέμουμε ξανά την έξοδο στους κάτω αποθηκευτικούς χώρους από όλα τα επίπεδα ξεκινώντας από το L_0.Μετα το περάς αυτής της διαδικασίας οι πάνω αποθηκευτικοί χώροι είναι άδειοι και οι κάτω γεμάτοι, έκτος ίσως από τον τελευταίο κάτω αποθηκευτικό χώρο.

Θα μελετήσουμε τώρα τις δυο θεμελιώδεις λειτουργίες που χρησιμοποιούνται από την ουρά προτεραιότητας, Το PUSH χρησιμοποιείται όταν ένας πάνω προσωρινός αποθηκευτικός χώρος περιέχει πάρα πολλά στοιχειά και καταπατά το αμετάβλητο χαρακτηριστικό 4.Αυτο βάζει κάποια στοιχειά πάνω ,διορθώνοντας έτσι το τοπικό μέγεθος. Το PULL χρησιμοποιείται όταν ένας κάτω προσωρινός αποθηκευτικός χώρος περιέχει πολύ λίγα στοιχειά και παραβιάζει το αμετάβλητο χαρακτηριστικό 3.Αυτο γεμίζει τον κάτω αποθηκευτικό χώρο με το τράβηγμα στοιχείων από το πάνω επίπεδο, διορθώνοντας έτσι και αυτό το τοπικό μέγεθος. Και οι δυο λειτουργίες συνενώνουν επιτυχώς συνεχόμενους προσωρινούς χώρους αποθήκευσης στην ουρά προτεραιότητας και αναδιανέμουν την προκύπτουσα ακολουθία μεταξύ των αποθηκευτικών χωρών που συμμετέχουν. Μετά τη συγχώνευση, απελευθερώνουμε τους παλιούς προσωρινούς αποθηκευτικούς χώρους και διανέμουμε καινούριους πινάκες για τους καινούριους αποθηκευτικούς χώρους.

2.9.1 ΛΕΙΤΟΥΡΓΙΑ PUSH

Εξετάζοντας αναλυτικότερα το Push χρησιμοποιείται όταν ένας πάνω-αποθηκευτικός χώρος Ui καταπατά την συνθήκη 4,οταν για παράδειγμα περιέχει περισσότερα από si/2 στοιχεία. Σ' αυτήν την περίπτωση ενώνουμε τα U_i, D_i και U_{i+1} σε μια ακολουθία M χρησιμοποιώντας τον αλγόριθμο συνενωσης[28].Στη συνεχεία διαχωρίζουμε τα στοιχεία του M, τοποθετώντας τα πρώτα $|D_i|$-δ στοιχεία σε έναν καινούριο προσωρινό αποθηκευτικό χώρο,D_i', και τα εναπομείναντα $|U_{i+1}|+|U_i|+\delta$ στοιχεία σε έναν καινούριο U_{i+1}' προσωρινό αποθηκευτικό χώρο. Μετά την συνένωση δημιουργούμε έναν κενό αποθηκευτικό χώρο,U_i' και απελευθερώνουμε τους παλιούς. Αν ο U_{i+1}' περιέχει πάρα πολλά στοιχεία, καταπατώντας έτσι την συνθήκη 4, χρησιμοποιούμε την λειτουργιά Push στο U_{i+1}'.Σε περίπτωση που το L_i είναι το τελευταίο επίπεδο, γεμίζουμε το D_i' με τα πρώτα στοιχεία του M και δημιουργούμε ένα νέο επίπεδο ,L_{i+1},τοποθετωντας τα εναπομείναντα στοιχεία του M στο D_{i+1}' αντί για το U_{i+1}'.Δεδομένου ότι το $|D_i'|$ είναι

μικρότερο από το D_i,μπορεί να παραβιάζει την συνθήκη 3.Αυτη η κατάσταση αντιμετωπίζεται με τη χρήση της λειτουργιάς Pull ,που περιγράφεται πριν ξεκινήσουμε την μελέτη του Pull.Σε αντίθεση με άλλες ουρές προτεραιότητας, η λειτουργιά Push μειώνει το μέγεθος ενός κάτω προσωρινού αποθηκευτικού χώρου. Αυτό απαιτείται για να διατηρείται αληθής η συνθήκη 2,παρολο που υπάρχουν καταστροφές. Μετά την κλήση της ενέργειας Push,το Di' μπορεί να περιέχει στοιχεία από το σύνολο U_i U U_{i+1}.Δεδομενου ότι δεν υπάρχει καμιά υπόθεση για τη σχέση μεταξύ των στοιχείων στο U_i U U_{i+1} και στο D_{i+1} U U_{i+2},χρειαζεται να επιβεβαιώσουμε ότι κάθε στοιχειό στο Di' που προέρχεται από το U_i U U_{i+1} είναι σίγουρα μικρότερο από τα στοιχεία στο D_{i+1} U U $_{i+2}$. Υποθετουμε ότι το μέγεθος του Di διατηρείται και έτσι για παράδειγμα ισχύει $|D_i'|=|D_i|$.θεωρούμε μια καταστροφή η όποια αλλάζει ένα στοιχειό στο D_i σε μερικές μεγάλες τιμές πριν το Push.Αυτή η κατεστραμμένη τιμή θα μπορούσε να τοποθετηθεί στο U_{i+1}' και δεδομένου ότι $|D_i'|=|D_i|$,ένα στοιχειό από το U_i U U_{i+1} πρέπει να τοποθετηθεί στο D_i'.Αυτό το καινούριο στοιχειό στο D_i' ενδεχομένως να παραβιάζει την συνθήκη 2.

2.9.2 ΛΕΙΤΟΥΡΓΙΑ PULL

Όσον αφόρα τη λειτουργιά Pull καλείται στον κάτω προσωρινό αποθηκευτικό χώρο D_i όταν περιέχει λιγότερα από $s_i/2$ στοιχεία, καταπατώντας έτσι την συνθήκη 3. Σ' αυτήν την περίπτωση οι προσωρινοί αποθηκευτικοί χώροι $_{Di}$, U_{i+1} και D_{i+1} συγχωνεύονται σε μια ακολουθία M με τον ανθεκτικό αλγόριθμο συγχώνευσης. Τα πρώτα si στοιχεία από το M γράφονται σε ένα καινούριο προσωρινό χώρο αποθήκευσης τον Di' και τα υπόλοιπα| D_{i+1} |-(s_i-|D_i|)-δ στοιχεία γράφονται στον D_{i+1} '.Τα εναπομείναντα στοιχεία του M γράφονται στο U_{i+1} '.Μια ενέργεια Pull καλείται στον D_{i+1} ',αν είναι πολύ μικρός.

Παρομοίως και στην λειτουργιά Push.Τα επιπλέον δ στοιχεία που χάθηκαν από το Di_{+1} επιβεβαιώνουν ότι οι ταξινομημένες συνθήκες διατηρούνται αληθής πάρα τις πιθανές καταστροφές. Έτσι, μια καταστροφή ενός στοιχείου στο D_i U D_{i+1} σε μια πολύ μεγάλη τιμή ίσως προκαλέσει την τοποθέτηση ενός στοιχείου από το U_{i+1} στην θέση του κατεστραμμένου στοιχείου στο D_{j+1} ' και αυτό το στοιχείο θα είναι πιθανόν μεγαλύτερο από μερικά μη κατεστραμμένα στοιχεία στο $D_{i+2}UU_{i+2}$.Μετα την συγχώνευση, το U_{i+1} ' περιέχει δ περισσότερα στοιχεία από ότι το U_{i+1} είχε πριν την συγχώνευση και έτσι είναι πιθανόν να έχει πάρα πολλά στοιχεία, καταπατώντας την συνθήκη 4.Βρισκομαστε λοιπόν σε μια κατάσταση που αντιμετωπίζεται ως εξής: Θεωρούμε την μεγίστη σειρά από μεταγενέστερες λειτουργίες Pull που επικαλούνται στους κάτω αποθηκευτικούς χώρους D_i,D_{i+1},..,D_j με $0 \le i < j < k$. Μετά το πρώτο Pull που καλείται στον D_i και πριν την κλήση στον D_{i+1}, αποθηκεύουμε έναν δείκτη στο D_i σε αξιόπιστη θέση μνήμης. Μετά το τέλος όλων των κλήσεων της Pull εξετάζουμε όλους τους πάνω αποθηκευτικούς χώρους που επηρεάστηκαν, άπλα ακλουθώντας τους δείκτες ανάμεσα στους αποθηκευτικούς χώρος αρχίζοντας από τον Di και χρησιμοποιούμε την λειτουργιά Push οπουδήποτε χρειάζεται. Στην περίπτωση που η λειτουργιά Push επηρεάζει του κάτω αποθηκευτικούς χώρους, η ροή τους αντιμετωπίζεται παρομοίως

Σχετικά με τις εισαγωγές και διαγραφές στοιχείων τώρα. Ένα στοιχειό εισάγεται στην ουρά προτεραιότητας με απλή προσθήκη του στον προσωρινό αποθηκευτικό χώρο εισαγωγής έστω I. Αν το I είναι γεμάτο, τα στοιχεία που προστίθενται στο U_0 με τον πρώτη έμπιστη ταξινόμηση I και στη συνέχεια

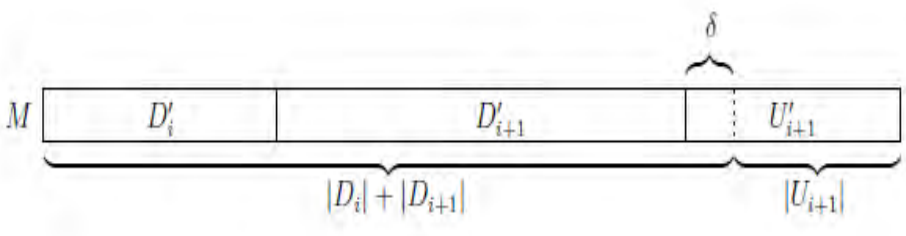
συγχωνεύοντας το I και το U_0.Αν το U_0 καταπατά την συνθήκη 4,χρησημοποιουμε την λειτουργιά Push.Αν το L_0 είναι το μοναδικό επίπεδο της ουράς προτεραιότητας και το D_0 παραβιάζει τον περιορισμό μεγέθους, συγχωνεύουμε τα στοιχεία στο I με αυτά του D_0.Για να διαγράψουμε το ελάχιστο στοιχείο της ουράς προτεραιότητας, αρχικά βρίσκουμε την ελάχιστη από τις πρώτες δ+1 τιμές στο D_0, την ελάχιστη από τις πρώτες δ+1 τιμές στο U_0 και το μικρότερο στοιχείο στον I. Έπειτα παίρνουμε το μικρότερο από αυτά τα τρία στοιχεία ,το διαγράφουμε από τον προσωρινό αποθηκευτικό χώρο από τον οποίο προέρχεται και το επιστρέφουμε. Μετά την διαγραφή του, κάνουμε μια δεξιά μετατόπιση όλα τα στοιχεία του αποθηκευτικού χώρου που βρισκόταν το στοιχείο που διαγράψαμε, από την αρχή μέχρι τη θέση του διαγραμμένου στοιχείου. Μ' αυτόν τον τρόπο είμαστε σίγουροι ότι τα στοιχεία σε κάθε αποθηκευτικό χώρο είναι αποθηκευμένα σε συνεχόμενες θέσεις. Σε περίπτωση που το D_0 υποχειλίσει, καλούμε την διαδικασία Pull στο D_0,εκτος αν το L_0 είναι το μοναδικό επίπεδο στην ουρά προτεραιότητας. Αν το U_0 ή D_0 περιέχουν Θ(logn+δ) άδεια κελία, δημιουργούμε έναν καινούριο προσωρινό αποθηκευτικό χώρο και αντιγράφουμε τα στοιχεία από τον παλιό στον καινούριο αποθηκευτικό χώρο.

2.9.3 ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΟΥ

Θα ξεκινήσουμε την ενότητα αυτή από την ανάλυση της ορθότητας του αλγορίθμου που μελετάμε. Για να αποδείξουμε την ορθότητα της ανθεκτικής ουράς προτεραιότητας, θα δείξουμε ότι οι λειτουργίες της αφαίρεσης επιστρέφουν τη μικρότερη μη κατεστραμμένη τιμή ή μια κατεστραμμένη. Αρχικά, θα αποδείξουμε ότι οι συνθήκες ταξινόμησης διατηρούνται με τις λειτουργίες Pull και Push.

ΛΗΜΜΑ 1ο :Οι λειτουργίες Pull και Push προστατεύουν τις συνθήκες ταξινόμησης

ΑΠΟΔΕΙΞΗ: Υπενθυμίζουμε ότι σε μια κλήση της λειτουργιάς Pull στον χώρο προσωρινής αποθήκευσης D_i,οι χώροι D_i, U_{i+1} και D_{i+1} συγχωνεύονται σε μια ακολουθία M. Τα στοιχεία στην M διαχωρίζονται σε τρεις νέους χώρους προσωρινής αποθήκευσης $D_{i'}$,U_{i+1}' και D_{i+1}',όπως φαίνεται στην εικόνα 7.



Εικόνα 7:Παρουσιάζεται μία ακολουθία M και τα τμήματα στα οποία διαχωρίζεται κατά την εκτέλεση των λειτουργιών Push και Pull [19].

Για να αποδείξουμε ότι οι συνθήκες ταξινόμησης ικανοποιούνται πρέπει να δείξουμε ότι τα στοιχεία του κάτω-αποθηκευτικού χώρου στο επίπεδο L_j για $0 \leq j < k$, είναι σίγουρα μικρότερα από τα στοιχεία των χωρών αποθήκευσης του επίπεδου L_{j+1},οπου κ είναι ο δείκτης του τελευταίου επιπέδου. Οι συνθήκες δεν επηρεάζουν τον αλγόριθμο όταν οι χώροι αποθήκευσης δεν επηρεάζονται. Η αξιόπιστη συγχώνευση εγγυάται ότι ο D_i' D_{i+1} 'και ο D_i ' U_{i+1} ' είναι αξιόπιστα ταξινομημένοι και ακόμη οι ατομικοί χώροι αποθήκευσης είναι επίσης αξιόπιστα ταξινομημένοι. Δεδομένου ότι η συνθήκη 2 ισχύει για τους αρχικούς προσωρινούς αποθηκευτικούς χώρους όλα τα άφθαρτα στοιχεία στο D_{i+1} και στο U_{i+1} είναι μεγαλύτερα από τα άφθαρτα στοιχεία του D_i,είναι εγγυημένο ότι ο D_{i-1} D_l' είναι αξιόπιστα ταξινομημένος. Κλείνοντας θα δείξουμε ότι ο D_{i+1} D_{i+2} και ο D_{i+1} U_{i+2} είναι αξιόπιστα ταξινομημένοι. Έστω m το μικρότερο μη κατεστραμμένο στοιχείο στο D_{i+2} U U_{i+2}.Χρειαζεται να δείξουμε ότι όλα τα άφθαρτα στοιχεία στο D_{i+1} ' είναι μικρότερα από το m.Αν κανένα άφθαρτο στοιχείο του U_{i+1} δεν είναι τοποθετημένο στο D_{i+1} ',η συνθήκη ισχύει από τις συνθήκες ταξινόμησης πριν την εκτέλεση της λειτουργίας. Διαφορετικά, υποθέτουμε ότι ένα άφθαρτο στοιχείο y που ανήκει στο U_{i+1} μετακινείται στο D_{i+1} '.Δεδομένου ότι $|U_{i+1}'| = |U_{i+1}| + \delta$ και το y μετακινείται στο D_{i+1}',τουλάχιστον δ+1 στοιχεία που προέρχονται από τον D_i U D_{i+1} περιέχονται στον U_{i+1} '.επειδή ισχύει από την υπόθεση ότι μπορεί να υπάρξουν το πολύ δ καταστροφές στοιχείων, υπάρχει τουλάχιστον ένα άφθαρτο στοιχείο, έστω χ, ανάμεσα τους. Από την αξιόπιστη συγχώνευση, όλα τα μη κατεστραμμένα στοιχεία στο D_{i+1} ' είναι μικρότερα του χ, γεγονός που σημαίνει ότι $y \leq \chi$. Γνωρίζουμε ότι το χ προέρχεται από τον D_i U D_{i+1} και είναι μικρότερο από το m. Έτσι προκύπτει ότι $y \leq m$.Με παρόμοιο τρόπο σκέψης αποδεικνύεται και η ορθότητα της λειτουργίας Push.Συμπεραίνουμε ότι και οι δυο συνθήκες ταξινόμησης προστατεύονται από τις λειτουργίες Push και Pull.

ΛΗΜΜΑ 2ο :Η διαδικασία διαγραφής επιστρέφει την ελάχιστη άφθαρτη τιμή στην ουρά προτεραιότητας ή μια κατεστημένη τιμή.

ΑΠΟΔΕΙΞΗ:

Υπενθυμίζουμε ότι η λειτουργιά της διαγραφής υπολογίζει το ελάχιστο από τα πρώτα δ+1 στοιχεία των προσωρινών χωρών αποθήκευσης U_0 και D_0.Συγκρινει αυτές τις τιμές με το ελάχιστο στοιχείο του l, που εντοπίζεται με ένα πέρασμα και επιστρέφει την μικρότερη από αυτές τις τρεις τιμές. Δεδομένου ότι U_0 και D_0 είναι απόλυτα ταξινομημένα, το ελάχιστο από τα δ+1 πρώτα στοιχεία τους είναι είτε το ελάχιστο άφθαρτο στοιχείο σε αυτούς τους αποθηκευτικούς χώρους μια κατεστραμμένη τιμή που είναι ακόμα μικρότερη.

Επιπλέον, συμφώνα με τις συνθήκες ταξινόμησης όλες οι τιμές στα επίπεδα $L_1,....,L_k$ είναι σίγουρα μεγαλύτερες από την ελάχιστη στο D0.Ετσι,το στοιχείο που αναφέρεται από την αφαίρεση είναι ή η ελάχιστη άφθαρτη τιμή ή μια εσφαλμένη τιμή.

Όσον αφόρα τώρα την πολυπλοκότητα του αλγορίθμου θα δείξουμε ότι η ανθεκτική ουρά προτεραιότητας χρειάζεται O(n) χώρο και οι εισαγωγές και διαγραφές απαιτούν $O(\log n + \delta)$ κόστος σε χρόνο. Αρχικά θα δείξουμε ότι οι λειτουργίες Push και Pull επαναφέρουν σε ισχύ τις συνθήκες μεγέθους.

ΛΗΜΜΑ 3ο : Αν μια συνθήκη μεγέθους καταπατάται από έναν αποθηκευτικό χώρο στο επίπεδο L_0,καλουμαι την Pull ή Push στον συγκεκριμένο αποθηκευτικό χώρο για να επαναφέρουν σε ισχύ τις συνθήκες μεγέθους το πολύ μια φορά. Καμιά άλλη συνθήκη δεν καταπατάται πριν ή μετά από αυτήν την λειτουργιά.

ΑΠΟΔΕΙΞΗ: Ας υποθέσουμε ότι το Push καλείται στον U_0 και επίσης καλείται επαναληπτικά σε μερικά επίπεδα L_l. Από την κατασκευή της Push,οι συνθήκες μεγέθους για όλους τους πάνω αποθηκευτικούς χώρους τώρα ισχύουν. Δεδομένου ότι μια λειτουργιά Push παίρνει δ στοιχεία από τους κάτω αποθηκευτικούς χώρους τα επίπεδα $L_0,..,L_l$ αντικρούονται ξανά και καλούμαι την διαδικασία Pull σ αυτά οπού χρειάζεται. Η τελευταία από αυτές τις λειτουργίες Pull ίσως εκτελείται στο παρελθοντικό επίπεδο L_l. Παρομοίως, μια λειτουργιά Pull ίσως προκαλέσει υπερχείλιση σε έναν πάνω-αποθηκευτικό χώρο. Παρ όλα αυτά όμως δεδομένου ότι γίνονται πολλές Push λειτουργίες απομένουν $|U_i|=0$ για $i\leq l$ και κάθε νέα λειτουργιά Push που καλείται στους πάνω αποθηκευτικούς χώρους μόνο στο επίπεδο L_{l+1} και ψηλότερα, και έτσι καλείται το η Push σε κάθε αποθηκευτικό χώρο το πολύ μια φορά. Με παρόμοιο τρόπο λειτούργει και η διαδικασία Pull.

ΛΗΜΜΑ 4ο :Η ανθεκτική ουρά προτεραιότητας χρησιμοποίει O(logn+δ) χώρο για να αποθηκεύσει n στοιχεία.

ΑΠΟΔΕΙΞΗ:Ο αποθηκευτικός χώρος των εισαγωγών χρησιμοποίει πάντα O(logn+δ) χώρο. Αποδεικνύουμε ότι τα εναπομείναντα επίπεδα χρησιμοποιούν O(n) χώρο. Για κάθε επίπεδο χρησιμοποιούμε O(δ) χώρο για να αποθηκεύει δομικές πληροφορίες αξιόπιστα. Σε όλα τα επίπεδα, έκτος από το τελευταίο ο κάτω αποθηκευτικός χώρος περιέχει $\Omega(\delta^2)$ στοιχεία όπως προκύπτει από την συνθήκη 3.Αυτο σημαίνει ότι σε καθένα από αυτά τα επίπεδα, τα στοιχεία αποθηκεύονται στον κάτω αποθηκευτικό χώρο κυριαρχώντας έτσι την πολυπλοκότητα του χώρου. Οι δομική πληροφορία του τελευταίου επιπέδου απαιτεί O(δ) επιπλέον χώρο. Η πολυπλοκότητα χώρου της ουράς προτεραιότητας μπορεί να περιοριστεί στο O(n) χωρίς να επηρεάσει την πολυπλοκότητα χρόνου, αποθηκεύοντας την δομική πληροφορία του L_0 σε ασφαλή θέση μνήμης και διπλασιάζοντας ή υποδιπλασιάζοντας τον αποθηκευτικό χώρο εισαγωγών κατά τη διάρκεια ζωής του αλγορίθμου καθώς αυτός θα χρησιμοποίει πάντα O(|I|) χώρο.

ΛΗΜΜΑ 5ο : Κάθε εισαγωγή ή διαγραφή έχει O(logn+δ) κόστος χρόνου.

ΑΠΟΔΕΙΞΗ: Ορίζουμε την συνάρτηση που εκμεταλλευόμαστε:

$$\Phi=\sum_{i=1}^{k} (c_1*(\log n-i)*|U_i|+c_2*i*|Di|)$$

Χρησιμοποιούμε το Φ για να αναλύσουμε το κόστος μιας λειτουργίας Push.Σε μια τέτοια λειτουργιά στον $_{Ui}$,οι αποθηκευτικοί χώρο U_i,D_i και U_{i+1} συγχωνεύονται. Τα στοιχεία στη συνέχεια ανακατανέμονται σε νέους αποθηκευτικούς χώρους U_i',D_i'

και U_{i+1}',έτσιώστε$|U_i'|=0$, $|D_l'|=|D_i|-\delta$ και $|U_{i+1}'|=|U_{i+1}|+|U_i|+\delta$. Αυτό μας δίνει την ακόλουθη αλλαγή στην συνάρτηση ΔΦ:

$$\Delta\Phi=-|U_i|*c_1*(\log n-i)-\delta*c_2*i+(|U_i|+\delta)*c_1(\log n-(i-1))=-c_1*|U_1|+\delta(-c_2*i+c_1*\log n-c_1*i-c_1)$$

Δεδομένου ότι η Push καλείται στον Ui,η συνθήκη 4 δεν ισχύει για τον U_i και επιπλέον $|U_i|\geq s_i/2=2^i(\log^2 n+\delta^2)$.

(1) $$\Delta\Phi\leq -c_1*|U_i|+c_1*\delta*\log n\leq -c_1*2^i*(\log^2 n+\delta^2)+c_1*\delta*\log n\leq -c_1*c'*|U_i|\ ,$$

για κάποια σταθερά c'>0.

Αφού η αξιόπιστη συγχώνευση δυο ακολουθιών με μέγεθος n απαιτεί $O(n+\delta^2)$ χρόνο, ο χρόνος που χρησιμοποιείται για μια Push στο U_i είναι άνω φραγμένη από το $C_m*(|U_i|+|D_i|+|U_{i+1}|+\delta^2)$, όπου cm εξαρτάται από την ανθεκτική σε λάθη συγχώνευση. Αυτή περιέχει το χρόνο που απαιτείται για να ανακτηθούν αξιόπιστα αποθηκευμένες μεταβλητές. Προσθέτοντας τον χρόνο και την αλλαγή στην συνάρτηση που χρησιμοποιούμε είμαστε ικανοί να πάρουμε μικρότερο από το μηδέν κόστος αλλάζοντας το c_1 βασισμένοι στην εξίσωση (1).Αυτο ισχύει επειδή ο $|U_i|$ είναι $\Omega(\delta^2)$ και έχει το πολύ ένα σταθερό τμήμα μικρότερο από αυτά που συμμετέχουν στην συγχώνευση.

Παρομοίως αναλύουμε και την διαδικασία Pull.Σ' αυτήν την περίπτωση υπολογίζουμε το κόστος των εισαγωγών και διαγράφων στοιχείων. Αγνοούμε κάθε Push και Pull λειτουργίες, επειδή τα κόστη τους είναι αρνητικά. Το κόστος σε χρόνο, για την εισαγωγή ενός στοιχείου στον I, ταξινομώντας τον I, και συνενώνοντας τον με τον U_0 είναι $O(\log n+\delta)$ ανά λειτουργιά. Η ενδεχόμενη αλλαγή, όταν προσθέτουμε στοιχεία στον L_0 είναι $O(\log n)$ ανά στοιχείο. Ο χρόνος που απαιτείται για να βρούμε το μικρότερο στοιχείο κατά τη διάρκεια μιας αφαίρεσης είναι $O(\log n+\delta)$ και η ενδεχόμενη αλλαγή σε περίπτωση διαγραφής στοιχείου από το L_0 είναι αρνητική. Το συνολικό κόστος επαναδημιουργίας κυριαρχείται από το κόστος της ταξινόμησης, το οποίο είναι $O(n\log n+\delta^2)$.Υπαρχουν $\Theta(n)$ λειτουργίες ανάμεσα σε κάθε επαναδημιουργία οι οποίες οδηγούνται σε $O(\log n+\delta)$ χρόνο ανά λειτουργιά, γνωρίζοντας ότι $\delta\leq n$ και έτσι τελειώνει η απόδειξη.

ΘΕΩΡΗΜΑ 7ο: Η ανθεκτική ουρά προτεραιότητας απαιτεί $O(n)$ χώρο και απαιτεί $O(\log n+\delta)$ κόστος σε χρόνο ανά λειτουργιά.

2.9.4 ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΛΕΙΤΟΥΡΓΙΩΝ

Στο τελευταίο αυτό κομμάτι θα δείξουμε ότι κάθε ανθεκτική ουρά προτεραιότητας παίρνει $\Omega(\log n+\delta)$ χρόνο είτε για εισαγωγές είτε για διαγραφές στοιχείων με το μοντέλο συγκρίσεων. Αυτό συνεπάγεται την βέλτιστη συμπεριφορά από την ανθεκτική ουρά προτεραιότητας μας υπό αυτές τις υποθέσεις. Σημειώνουμε ότι η αξιόπιστη μνήμη μπορεί να περιέχει πληροφορίες για τη δομή όπως για παράδειγμα δείκτες και μεγέθη.

ΘΕΩΡΗΜΑ 8ο: Μία ανθεκτική ουρά προτεραιότητας περιέχει n στοιχεία, με n>δ, χρησιμοποίει Ω(logn+δ) συγκρίσεις για να εκτελεί εισαγωγές ακολουθούμενες από διαγραφές.

ΑΠΟΔΕΙΞΗ:

Θεωρούμε μια ουρά προτεραιότητας Q με n στοιχεία, με n>δ, που απαιτούν λιγότερες από δ συγκρίσεις για μια εισαγωγή που ακολουθητέοι από μια διαγραφή. Επίσης, η Q δεν αποθηκεύει στοιχεία σε αξιόπιστη μνήμη ανάμεσα στις λειτουργίες. Υποθέτουμε ότι καμιά καταστροφή δεν έχει προκληθεί ακόμη. Χωρίς να χάνουμε σε γενικότητα, υποθέτουμε ακόμη ότι όλα τα στοιχεία στην ουρά Q είναι διακριτά. Αποδεικνύουμε οι υπάρχει μια σειρά από καταστροφές στοιχείων, έστω C,με |C|≤δ, έτσι ώστε το αποτέλεσμα της εισαγωγής ενός στοιχείου e ακολουθείται από μια διαγραφή επιστρέφει το ίδιο στοιχείο ανεξάρτητα από την επιλογή του e.Με κ<δ να είναι ο αριθμός των συγκρίσεων που εκτελούνται από την Q κατά τη διάρκεια των δυο λειτουργιών. Αναγκάζουμε το αποτέλεσμα κάθε σύγκρισης να είναι το ίδιο ανεξάρτητα από το e με τις κατάλληλες καταστροφές. Με όλες τις συγκρίσεις να περιέχουν το e,διασφαλίζουμε ότι το e είναι το μικρότερο. Αυτό το κάνουμε καταστρέφοντας την τιμή με την οποία συγκρίθηκε το e,αν είναι απαραίτητο, προσθέτοντας μερικές θετικές σταθερές c≥e στις άλλες τιμές. Αν δυο στοιχεία διαφορετικά από το e συγκριθούν, είμαστε σίγουροι ότι το αποτέλεσμα είναι το ίδιο με την περίπτωση που δεν είχαν συμβεί καθόλου καταστροφές. Αν ένα από αυτά ήταν φθαρμένο, προσθέτοντας το c στο άλλο ,αποκαθιστάται η ταξινόμηση που είχαν τα στοιχεία από πριν. Αν πάλι και τα δύο ήταν αλλοιωμένα προθέτοντας το e,η ταξινόμηση του είναι αμετάβλητη και δεν απαιτούνται καταστροφές. Αναγκάζοντας κάθε σύγκριση να δίνει το επιθυμητό αποτέλεσμα απαιτείται το πολύ μια καταστροφή στοιχείου και επίσης |C|≤k<δ.

Θεωρούμε τώρα μια τιμή έστω e' που επιστρέφεται από την αφαίρεση στην Q.Αν e=e' τότε επιλεγούμε το e να είναι μεγαλύτερο από μερικά στοιχεία χ που ανήκουν στην ουρά Q και δεν επηρεάζονται από μια φθορά στοιχείου στην C.Μια τέτοια τιμή υπάρχει επειδή το μέγεθος της ουράς προτεραιότητας είναι μεγαλύτερο από το δ. Δεδομένου ότι e=e'>χ, η Q επιστρέφει ένα άφθαρτο στοιχείο το οποίο δεν ήταν το μικρότερο αναλλοίωτο στοιχείο στην Q.Αν e διαφορετικό από το e',επιλεγούμε το e να είναι το μικρότερο από κάθε άλλο στοιχείο στην Q.Με μια τέτοια επιλογή του e,δεν απαιτούνται καταστροφές και η τιμή που επιστρέφεται από την Q είναι άφθαρτη αλλά μεγαλύτερη από το e.Αυτό δείχνει ότι η Q δεν είναι ανθεκτική σε λάθη .

ΚΕΦΑΛΑΙΟ 3 ΤΕΧΝΙΚΕΣ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

3.1 ΑΠΟΔΟΣΗ ΑΛΓΟΡΙΘΜΟΥ

Μετά το σχεδιασμό ανθεκτικών αλγορίθμων για θεμελιώδεις λειτουργιές όπως ταξινόμηση και αναζήτηση, φαίνεται αρκετά φυσικό να αναρωτηθούμε αν μπορούμε να σχεδιάσουμε με επιτυχία ανθεκτικές δομές δεδομένων ,χωρίς να επιβαρυνθούμε επιπλέον σε χρόνο και χώρο. Σε πολλές εφαρμογές, όπως ο σχεδιασμός συστήματος αρχείων, είναι πολύ σημαντικό η υλοποίηση μιας δομής δεδομένων να είναι ανθεκτική σε λάθη μνήμης καθώς επίσης να παρέχει ένα μηχανισμό με την βοήθεια του οποίου να επανέρχεται γρήγορα από λανθασμένες καταστάσεις οι οποίες πιθανόν να οδηγήσουν σε λανθασμένη συμπεριφορά του αλγορίθμου. Ο σχεδιασμός των ανθεκτικών σε λάθη δομών δεδομένων παρουσιάζει έντονο ενδιαφέρον και η εξέλιξη και βελτίωση του αποτελεί πρόκληση . Πράγματι, κλασικές δομές δεδομένων, όπως για παράδειγμα δέντρα αναζήτησης και σειρές προτεραιότητας βασισμένες σε σορό εξαρτούνται άμεσα από την δόμηση και την τοποθέτηση της πληροφορίας. Η καταστροφή ενός κλειδιού σε ένα σταθερό δυαδικό δέντρο για παράδειγμα, ίσως βλάψει και καταργήσει την δυνατότητα αναζήτησης στο δέντρο και επομένως η αναζήτηση ίσως οδηγήσει σε λάθος υποδέντρα και κατ' επέκταση σε λανθασμένο αποτέλεσμα. Επιπλέον, πολλοί δείκτες βασισμένοι σε δομές δεδομένων είναι αυστηρώς μη ανθεκτικοί σε λάθη με αποτέλεσμα, η καταστροφή ενός μοναδικού δείκτη να κάνει ολόκληρη τη δομή δεδομένων απρόσιτη και ακόμα και οι μη κατεστραμμένες τιμές, πιθανόν να χαθούν.

Σε αυτό το σημείο, θα κάνουμε ένα πρώτο σημαντικό βήμα προς τον σχεδιασμό ανθεκτικών δομών δεδομένων με μη αξιόπιστες μνήμες. Επίσης, θεωρούμε την φυσική ανθεκτική έκδοση μια κλασικής δομής λεξικού στην οποία οι λειτουργιές, της εισαγωγής και της διαγραφής ορίζονται ως συνήθως, ενώ η λειτουργιά αναζήτησης θα πρέπει να λύσει το πρόβλημα της ανθεκτικής σε λάθη αναζήτησης που περιγράφεται πιο πάνω. Παρουσιάζουμε μια απλή τυχαία υλοποίηση μιας ανθεκτικής δομής λεξικού, που επιτυχαίνει $O(\log n+\delta)$ προσδοκώμενο χρόνο ανά λειτουργιά .Κατόπιν, περιγράφουμε μια ντετερμινιστική ανθεκτική δομή λεξικού με $O(\log n+ \delta^{1+\varepsilon})$ προσδοκώμενο χρόνο ανά λειτουργιά, όπου $\varepsilon>0$ είναι μια αυθαίρετη μικρή σταθερά. Πιο συγκεκριμένα, η ντετερμινιστική μας ανθεκτική δομή λεξικού υποστηρίζει κάθε ακολουθία από σ λειτουργιές σε $O(\sigma(\log n+\delta)+\delta^{(1+\varepsilon)})$ χρόνο. Για $\sigma \geq \delta^{\varepsilon}$ λειτουργίες, το αποτέλεσμα του ντετερμινιστικού αλγορίθμου είναι ίδιο με αυτό του τυχαίου. Τα αποτελέσματα αυτά συμπληρώνονται από ένα κάτω όριο σύμφωνα με το οποίο κάθε ανθεκτική δομή λεξικού βασισμένη στην σύγκριση κλειδιών, όπως αυτές που μελετάμε, πρέπει να έχουν $\Omega(\log n+\delta)$ προσδοκώμενο χρόνο ανά αναζήτηση. Όλες οι δίκες μας δομές χρησιμοποιούν στην καλύτερη περίπτωση $O(n)$ χώρο.

Η βασική ιδέα πάνω στην οποία στηρίζεται η προσέγγιση μας είναι το σύνολο των κλειδιών μέσα στα μη επικαλυπτόμενα διαστήματα και η διατήρηση των διαστημάτων σε ένα προσεκτικά προσαρμοζόμενο δέντρο αναζήτησης. Μετά από μερικές διαγραφές ή εισαγωγές τα διαστήματα μπορεί να χωριστούν, να

ενωθούν ή να τροποποιηθούν. Αυτό μπορεί να γίνει αρκεί να μην πάψουν να ισχύουν οι ακόλουθες ιδιότητες υψηλού επίπεδου.

(1)τα διαστήματα καλύπτουν τον χώρο των κλειδιών,

(2)καθε διάστημα περιέχει Θ(δ) κλειδιά, και

(3)το σύνολο των διαστημάτων και το δέντρο αναζήτησης τροποποιείται κάθε Ω(δ) εισαγωγές και/ή διαγραφές.

Η ιδιότητα (2) μας επιτρέπει να κάνουμε την αναζήτηση κλειδιού σε διαστήματα σε Ο(δ) χρόνο. Το πιο σημαντικό όμως είναι ότι μας επιτρέπει να αποθηκεύσουμε το δέντρο αναζήτησης αξιόπιστα, αποθηκεύοντας (2δ+1) αντίγραφα της κάθε σχετικής μεταβλητής, συμπεριλαμβανόμενων και των κλειδιών, ενώ παράλληλα ο χώρος στη μνήμη που απαιτείται διατηρείται γραμμικός. Εξαιτίας τώρα της ιδιότητας (3), μπορούμε να ενημερώνουμε το δέντρο αναζήτησης αξιόπιστα με χαμηλό κόστος σε χρόνο εκτέλεσης.

Ένα άλλο βασικό συστατικό της μεθόδου που ακολουθούμε είναι ο τρόπος με τον οποίο κάνουμε την αναζήτηση ενός διαστήματος που περιέχει το δοσμένο κλειδί, έστω κ. Από την ιδιότητα (1),προκυπτει ότι πρέπει πάντα να υπάρχει αυτό το διάστημα. Σχεδιάζουμε δυο διαφορετικούς αλγορίθμους διαστημάτων. Ο πρώτος από αυτούς με Ο(logn+δ) προσδοκώμενο χρόνο εκτέλεσης και ο δεύτερος με Ο(logn+$\delta^{1+\varepsilon}$)) χρόνο εκτέλεσης χειρότερης περίπτωσης. Ο αλγόριθμος αναζήτησης διαστήματος με χρόνο Ο(logn+δ), που περιέχει τυχαιότητα, βασίζεται σε ένα χρήσιμο Λήμμα, το οποίο αναφέρει ότι κάθε επιπόλαιος-ανθεκτικός ντετερμινιστικός αλγόριθμος, με κάποια επιπλέον χαρακτηριστικά, μπορεί να επιταχυνθεί κατά ένα παράγοντα Ο(δ), με την βοήθεια μιας έξυπνης χρήσης της τυχαίας δειγματοληψίας.

Ο ντετερμινιστικός μας Ο(logn+$\delta^{1+\varepsilon}$) αλγόριθμος αναζήτησης διαστημάτων βασίζεται σε μια νέα αντίληψη των ζημιογόνων αριθμών. Υπενθυμίζουμε ότι κάθε ανθεκτική μεταβλητή χ αποτελείται από (2δ+1) αντίγραφα, $\chi_1,\chi_2,...,\chi_{2\delta+1}$ μιας σταθερής μεταβλητής. Διαισθητικά ζημιογόνος αριθμός p ,που ανήκει σε {1,2,...,2δ+1} ,ενός αλγορίθμου/δομή δεδομένων είναι ο μικρότερος δείκτης από τα αντίγραφων των ανθεκτικών μεταβλητών ,που εμπιστευόμαστε ακόμα .Αρχικά,p=1,που σημαίνει ότι θεωρούμε όλα τα αντίγραφα της μεταβλητής αξιόπιστα. Οποτεδήποτε ,κατά τη διάρκεια ζωής του αλγορίθμου/δομή δεδομένων ,ανακαλύψουμε ότι το p-οστό αντίγραφο κάποιων ανθεκτικών μεταβλητών έχει καταστραφεί, θεωρούμε τα p αντίγραφα κάθε ανθεκτικής μεταβλητής αναξιόπιστα και αυξάνουμε το p κατά ένα. Επίσης ,θεωρούμε ότι όλα τα αναξιόπιστα αντίγραφα αγνοούνται. Η ιδέα μας είναι ότι αν μέσω του ελέγχου συνέπειας καταφέρουμε να ανακαλύψουμε λανθασμένες συμπεριφορές από τα p-οστά αντίγραφα σύντομα, θα είμαστε σε θέση να κάνουμε μερικές υπαναχωρήσεις σε ασφαλή σημεία ελέγχου και να επιβαρύνουμε με τον χρόνο υπολογισμών τα λανθασμένα αντίγραφα που αγνοούνται. Παρατηρούμε ότι ίσως αγνοούμε μερικά αντίγραφα μεταβλητών τα οποία ακόμη δεν έχουν λάβει μέρος σε υπολογισμούς. Το μοναδικό τους αρνητικό χαρακτηριστικό είναι ότι έχουν τον ίδιο δείκτη με μερικά λανθασμένα αντίγραφα. Από αυτήν την παρατήρηση, ίσως φαίνεται ότι οι ζημιογόνοι αριθμοί είναι μάλλον ένας δύσκολος τρόπος να

ενώσουμε τις πληροφορίες που έχουμε σχετικά με τα λάθη που ανακαλύφθηκαν. Παρ' όλα αυτά, θα δούμε στη συνέχεια ότι είναι ένας αρκετά καλός τρόπος για να πετύχουμε το σκοπό μας.

Όσο για το κάτω όριο, το όριο του Ω(logn) στις λειτουργιές αναζήτησης προκύπτει από τα σταθερά αποτελέσματα σε μη ανθεκτικές δομές λεξικού. Με σκοπό να αποδείξουμε το τμήμα Ω(δ) του κάτω ορίου, χρησιμοποιούμε μια μάλλον γενική κατασκευή. Το βασικό συστατικό που χρειαζόμαστε είναι ότι η έξοδος εξαρτάται από ένα επαρκώς μεγάλο τμήμα της ανασφαλούς μνήμης. Βασισμένοι σ' αυτό, θα δείξουμε το πως να δημιουργήσουμε ένα σενάριο το οποίο δημιουργεί μια λανθασμένη απάντηση σε κάθε ο(δ) χρόνο αναζήτησης.

3.2 ΠΡΟΚΑΤΑΡΚΤΙΚΑ ΑΛΓΟΡΙΘΜΟΥ

Υπενθυμίζουμε ότι όπου δ είναι ένα πάνω όριο του αριθμού των συνολικών λαθών τα οποία πιθανόν εμφανιστούν κατά τη διάρκεια ζωής της δομής δεδομένων. Επίσης, συμβολίζουμε με α τα λάθη τα οποία έχουν εμφανιστεί μέχρι στιγμής. Προφανώς, θα ισχύει ότι α ≤δ.

Μια ανθεκτική μεταβλητή $\chi=(\chi_1,\chi_2,...,\chi_{2\delta+1})$ αποτελείται από (2δ+1) αντίγραφα μιας κλασικής μεταβλητής. Βάζοντας μια τιμή στο χ σημαίνει ότι αυτή η τιμή αντιστοιχεί σε κάθε αντίγραφο του χ_i.Η τιμή του χ που αντιστοιχεί στην πλειοψηφία των αντίγραφων της μεταβλητής ,δηλαδή η τιμή που εμφανίζεται σε παραπάνω από τα μισά αντίγραφα. Έτσι, συμπεραίνουμε ότι κάθε τιμή είναι καλά ορισμένη μέχρι να καταστραφούν το πολύ δ αντίγραφα με την προϋπόθεση ότι τα υπόλοιπα δ+1 αντίγραφα να έχουν την ιδία τιμή. Η ενημέρωση του χ μπορεί να υλοποιηθεί σε Ο(δ) χρόνο και Ο(1) χώρο, άπλα ξαναγράφοντας κάθε αντίγραφο του χ με μια καινούρια τιμή. Το ίδιο ισχύει και για την ανάγνωση τιμών με τον αλγόριθμο των υπολογισμών πλειοψηφίας [7].Ο αλγόριθμος αυτός απαιτεί το πολύ 2n συγκρίσεις. Για την ευκολότερη κατανόηση του αλγορίθμου θα αναφερθούμε σε ένα παράδειγμα ψηφοφορίας, όπου η κάθε τιμή μιας ανθεκτικής μεταβλητής θα είναι μια ψήφος. Στον αλγόριθμο μας, δεν απαιτείται οι ψήφοι να μπορούν να ταξινομηθούν. Ακόμη, κατά την εκτέλεση του πραγματοποιούνται μονό συγκρίσεις ισότητας. Σε περίπτωση που είχαμε τη δυνατότητα να ταξινομήσουμε τις ψήφους θα χρησιμοποιούσαμε έναν αλγόριθμο που έχει n χρόνο εκτέλεσης. Αυτός, αρχικά, ελέγχει την διάμεσο ,που εντοπίζεται με τον αλγόριθμο Rivest-Tarjan ,αν έχει πάρει περισσότερες από τις μισές ψήφους. Αυτή η προσέγγιση οριοθετείται σε πάνω από (5,43n-163) συγκρίσεις για n>32. Ας επιστρέψουμε τώρα στη περίπτωση μας, που δεν μπορούμε τα ταξινομήσουμε τις ψήφους. Φανταστείτε ένα συνεδριακό κέντρο γεμάτο με αντιπροσώπους , δηλαδή ψηφοφόρους καθένας από τους οποίους κρατεί ένα πλακάτ που γραφεί το όνομα του υποψηφίου που υποστηρίζει. Έστω ότι γίνεται μια μάχη σε κάθε όροφο του συνεδριακού κέντρου και ο κάθε ψηφοφόρος χτυπά και χτυπιέται με το πλακάτ που έχει στα χεριά έναν ψηφοφόρο άλλου υποψήφιου. Το αποτέλεσμα είναι να πέφτουν και οι δυο κάτω .Στο τέλος θα πρέπει σε κάθε επίπεδο ένας υποψήφιος να έχει περισσότερους ψηφοφόρους από ότι όλοι οι άλλοι μαζί. Αυτός ο υποψήφιος όταν θα σταματήσει η σύρραξη θα έχει κερδίσει τη μάχη του κάθε ορόφου και έτσι θα έχει την πλειοψηφία των ψηφοφόρων. Σε περίπτωση όμως, που δεν υπάρχει κάποιος υποψήφιος που δεν έχει κερδίσει τη μάχη όλων των οροφών μπορεί να μην έχει απολυτή πλειοψηφία και το αποτέλεσμα δεν είναι τελείως ξεκάθαρο. Έτσι, στο τέλος της μάχης οι ψηφοφόροι που υποστηρίζουν το

πολύ έναν υποψήφιο τον διατηρούν στη θέση του αλλά είναι πιθανό να μην αντιπροσωπεύει την πλειοψηφία όλων των ψηφοφόρων. Γενικά αν κάποιος παραμείνει όρθιος μετά το πέρας της μάχης ο πρόεδρος της συνεδρίασης είναι υποχρεωμένος να μετρήσει τα πλακάτ των ψηφοφόρων, συμπεριλαμβανομένου αυτών που κρατηθήκαν από τους χαμένους, για να διαπιστώσει αν υπάρχει πλειοψηφία. Από την διαδικασία που περιγράψαμε καταλαβαίνουμε ότι ο αλγόριθμος μας έχει δυο τμήματα. Το πρώτο τμήμα, στο οποίο οι ψηφοφόροι αλληλοεξουδετερώνονται μέχρι να παραμείνουν όρθιοι μονό ψηφοφόροι του ίδιου υποψήφιου. Αυτό το πρώτο τμήμα του αλγορίθμου το ονομάζουμε “ζευγάρωμα”, καθώς ένας ψηφοφόρος εξουδετερώνει και παράλληλα εξουδετερώνεται από έναν με διαφορετικές αντιλήψεις. Πιθανώς, το ζευγάρωμα μπορεί να γίνει με n συγκρίσεις. Αν το ζευγάρωμα αφήσει μερικούς ψηφοφόρους όρθιους, αυτοί φυσιολογικά θα υποστηρίζουν τον ίδιο υποψήφιοι όποιος θα έχει και την πλειοψηφία αν υπάρχει. Το δεύτερο τμήμα του αλγορίθμου ονομάζεται “καταμέτρηση”. Αυτή υπολογίζει αν ο υποψήφιος έλαβε παραπάνω από τις μισές ψήφους. Αυτό το τμήμα απαιτεί το πολύ n συγκρίσεις. Ας επεξηγήσουμε τώρα τον πιο εύκολο τρόπο που μπορούμε να προσομοιώσουμε το τμήμα του ζευγαρώματος του αλγορίθμου .Ο αλγόριθμός μας, επισκέπτεται κάθε ψηφοφόρο διατηρώντας στη μνήμη τον τρέχον υποψήφιο και έναν μετρητή. Όπου υποψήφιο, έστω Υ, έχουμε το όνομα αυτού που υποστηρίζουν οι μέχρι τώρα ψηφοφόροι και ο μετρητής κρατεί το πόσοι ψηφοφόροι παραπάνω είναι με τον τρέχοντα υποψήφιο και αρχικοποιείται στο μηδέν. Κάθε φορά που ο πρόεδρος της συνεδρίασης επισκέπτεται έναν ψηφοφόρο ελέγχει αν η τιμή του μετρητή, έστω κ ,είναι μηδέν. Αν ισχύει θέτει στο Υ το όνομα του υποψήφιου που υποστηρίζει ο συγκεκριμένος ψηφοφόρος και αυξάνει το κ κατά ένα. Διαφορετικά ελέγχει την τιμή του Υ με το όνομα του υποψήφιου του τρέχοντα ψηφοφόρου, Αν είναι το ίδιο αυξάνει κατά ένα το μετρητή κ, αλλιώς το μειώνει κατά ένα. Κατόπιν προχωρεί στον επόμενο. Όταν ο πρόεδρος έχει περάσει από όλους τους ψηφοφόρους, ο υποψήφιος του οποίου το όνομα είναι η τιμή του Υ έχει την πλειοψηφία αν υπάρχει .Αρχικά υποθέτουμε ότι υπάρχουν Ν ψηφοφόροι. Στη συνεχεία ο πρόεδρος της συνεδρίασης επισκέπτεται τον i-οστό ψηφοφόρο ,με 1≤i≤N.Οι ψηφοφόροι μπορούν να διαιρεθούν σε δυο ομάδες .Η πρώτη ομάδα που αποτελείται από κ ψηφοφόρους που υποστηρίζουν τον ίδιο υποψήφιο και η δεύτερη που αποτελείται από τους ψηφοφόρους που μπορούν να ζευγαρώσουν με τέτοιο τρόπο ώστε οι ζευγαρωμένοι ψηφοφόροι να διαφωνούν μεταξύ τους. Από αυτό μπορούμε να συμπεράνουμε, αφού ασχοληθεί με όλους τους ψηφοφόρους ότι το όνομα που έχει το Υ είναι η πλειοψηφία αν υπάρχει. Υποθέτουμε ότι υπάρχει ένα όνομα υποψήφιου Χ διαφορετικό από αυτό του Υ ,με περισσότερες από Ν/2 ψήφους. Από τη στιγμή που οι ψηφοφόροι του δευτέρου τμήματος μπορούν να ζευγαρώσουν το Χ λαμβάνει το πολύ (Ν-κ)/2 ψήφους από αυτό το τμήμα. Έτσι το Χ θα έπρεπε να πάρει μια ψήφο από το πρώτο τμήμα, γεγονός που εναντιώνεται με την πραγματικότητα συμφώνα με την οποία όλες οι ψήφοι του πρώτου τμήματος είναι Υ. Καταλήγουμε λοιπόν σε άτοπο, πράγμα που σημαίνει ότι δεν υπάρχει τιμή Χ διαφορετική από την Υ που έχει περισσότερες από Ν/2 ψήφους. Κλείνοντας αξίζει να σημειωθεί ότι η ανάκληση των τιμών στον αλγόριθμο μας γίνεται γραμμικά .Σε περίπτωση που τα δεδομένα είναι παρά πολλά διαβάζονται από μια μαγνητική ταινία .Αυτή η ταινία θα πρέπει να ξαναδιαβαστεί στο δεύτερο μέρος του αλγορίθμου. Αθύμησε περίπτωση που μπορούμε να καταλάβουμε ότι υπάρχει πλειοψηφία το δεύτερο μέρος μπορεί να μην εκτελεστεί Έτσι ο αλγόριθμος μπορεί να υλοποιήσει την καταμέτρηση των ψηφοφόρων σε πραγματικό χρόνο., χωρίς να αποθηκεύσει τις ψήφους για μετέπειτα επεξεργασία.

Θα περιγράψουμε την διαδικασία ανάγνωσης, εφαρμόζοντας τον παραπάνω αλγόριθμο. Έστω $\chi_1,\chi_2,...,\chi_{2\delta+1}$ τα αντίγραφα του χ. Αποθηκεύουμε μια μεταβλητή y και έναν μετρητή z σε ασφαλείς θέσεις μνήμης .Αυτές οι τιμές θα καταστραφούν στο τέλος τις διαδικασίας. Αρχικά ,εξετάζοντας το πρώτο αντίγραφο, θέτουμε $y=\chi_1$ και z=1.Για i=2,3,...,2δ+1 συγκρίνουμε το χ_i με το y.Αν χ_i=y,αυξάνουμε το z.Διαφορετικά ελαττώνουμε το z. Σε περίπτωση που το z=0,θετουμε το $y=\chi_i$ και το z=1.Τελικα εξάγουμε το y σαν την τιμή με τα περισσότερα αντίγραφα.

Υποθέτουμε ένα μοντέλο βασισμένο στις συγκρίσεις των κλειδιών. Τα κλειδιά μπορούν μονό να συγκριθούν. Όλα τα πάνω και κάτω όρια που παρουσιάζονται βασίζονται στο μοντέλο αυτό. Για κάθε παρουσίαση υποθέτουμε ότι τα κλειδιά είναι πεπερασμένοι πραγματικοί αριθμοί. Επίσης, θέτουμε το +∞ την μεγαλύτερη τιμή από κάθε άλλη εφικτή τιμή που μπορούν να πάρουν τα κλειδιά. Αντίστοιχα το -∞ ως την μικρότερη δυνατή τιμή που μπορεί να τεθεί σε κάποιο κλειδί. Επειδή τα κλειδιά και οι μεταβλητές δεν είναι αποθηκευμένα σε ασφαλή μνήμη, όλες οι μεταβλητές που χρησιμοποιούνται από τη δομή δεδομένων μας είναι ανθεκτικές σε λάθη. Παρ όλα αυτά, μερικές φόρες θα διαβάζουμε μονό ένα υποσύνολο από τα αντίγραφα μιας ανθεκτικής μεταβλητής και θα υπολογίζουμε την τιμή που έχουν τα πιο πολλά από αυτά .Σε περίπτωση που δεν υπάρχει τιμή πλειοψηφίας θα επιστρέψουμε μια αυθαίρετη τιμή. Αυτό όπως είναι φυσικό συνεπάγεται ότι η επιστρεφόμενη τιμή με μεγάλη πιθανότητα θα είναι λανθασμένη.

ΚΕΦΑΛΑΙΟ 4^{Ο} ΤΟΠΟΘΕΤΩΝΤΑΣ ΚΛΕΙΔΙΑ ΣΤΑ ΔΙΑΣΤΗΜΑΤΑ

4.1 ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ

Σ' αυτό το σημείο θα περιγράψουμε την κοινή βασική δομή μιας ανθεκτικής δομής λεξικού σε μορφή δυαδικού δέντρου. Διατηρούμε ένα δυναμικά εξελισσόμενο σύνολο διαστημάτων. Για κάθε διάστημα έχουμε τα σύνορά του και τη λίστα των κλειδιών που περιέχει .Ας δούμε τον τρόπο που δημιουργείται ένα τέτοιο δέντρο. Αρχικά, υπάρχει ένα μοναδικό διάστημα, με τα όρια του να είναι το -∞ και το +∞ και η λίστα με τα κλειδιά του να είναι άδεια. Κατά τη διάρκεια των ενεργειών διατηρούμε αμετάβλητα τα εξείς χαρακτηριστικά των τμημάτων:

(i) Τα διαστήματα δεν είναι επικαλυπτόμενα και η ένωση τους έχει σύνορα το -∞ και το +∞,

(ii) Κάθε διάστημα περιέχει λιγότερα από 2δ κλειδιά.

(iii) Κάθε διάστημα περιέχει περισσότερα από δ/2 κλειδιά, εκτός ίσως από το πιο αριστερό η το πιο δεξιό διάστημα (ακραία διαστήματα).

Για την υλοποίηση κάθε αναζήτησης, εισαγωγής ή διαγραφής κάποιου κλειδιού, έστω κ, αρχικά απαιτείται να βρούμε το διάστημα I(κ) που περιέχει το κ (κάνοντας μια αναζήτηση διαστήματος). Τα αμετάβλητα χαρακτηριστικά (ii) και (iii) των διαστημάτων έχουν ένα πολύ σημαντικό ρολό στην ανάλυση πολυπλοκότητας του αλγορίθμου, όπως θα εξηγήσουμε αναλυτικότερα παρακάτω.

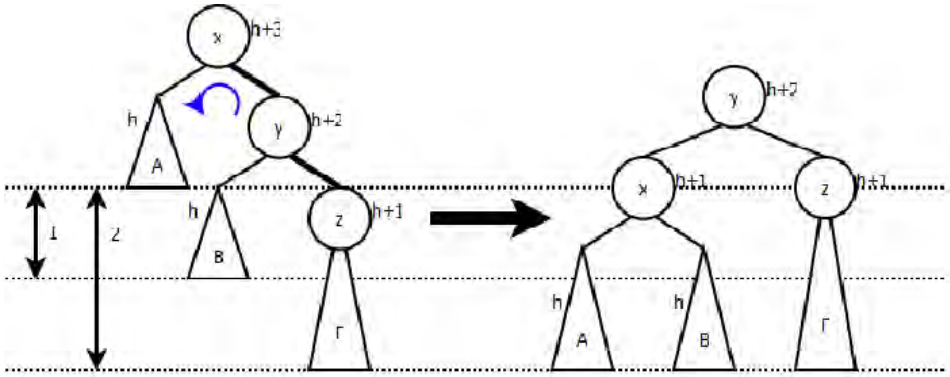
Τα διαστήματα αποθηκεύονται σε ένα σταθερό ισοζυγισμένο δέντρο αναζήτησης. Για την σαφήνεια της ανάλυσης θα στηρίξουμε την προσοχή μας στα δέντρα-AVL[1, 4, 15, 26, 30] .

AVL είναι ένα δυαδικό δέντρο αν και μόνον αν τα ύψη των δύο υποδέντρων κάθε εσωτερικού κόμβου u διαφέρουν το πολύ κατά ένα. Σε ένα δέντρο τέτοιου τύπου παρουσιάζεται δυσκολία κυρίως στις ενθέσεις και στις αποσβέσεις των στοιχείων του, καθώς δεν πρέπει να παραβιάζεται ο ορισμός του. Γι' αυτό πραγματοποιούνται κάποιες πράξεις ,τις οποίες ονομάζουμε επαναζυγιστικές που προσπαθούν να διατηρήσουν τη διάφορα του ύψους του αριστερού και δεξιού υποδέντρου ,δηλαδή των αδελφών κόμβων, το πολύ ίση με τη μονάδα.

Ας αναφερθούμε τώρα στην λειτουργιά εισαγωγής στοιχείου σε ένα τέτοιο δέντρο. Η διαδικασία που ακολουθείται κάθε φόρα έχει ως εξής. Τοποθετούμε το στοιχειό στην σωστή θέση στο δέντρο. Στη συνεχεία ξεκινώντας από το στοιχείο που

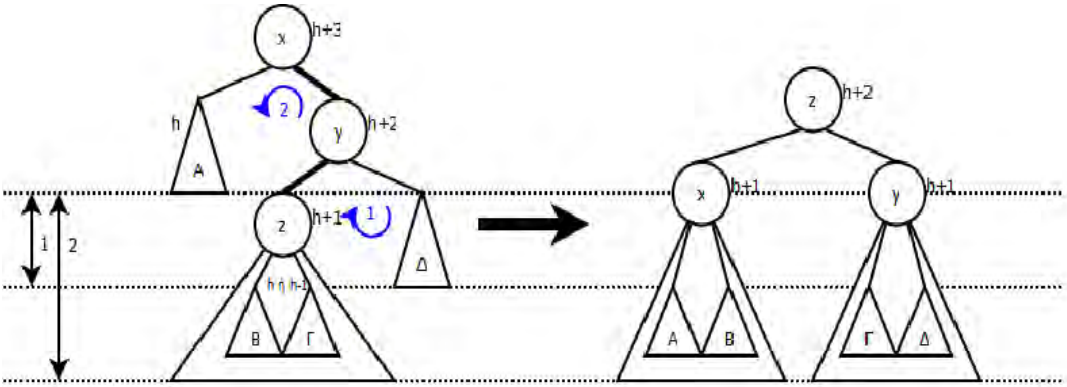
μόλις προσθέσαμε στο δέντρο, ανεβαίνουμε προς τη ριζά του δέντρου κάνοντας επιδιόρθωση των υψών μέχρι να εντοπίσουμε έναν κόμβο, έστω u, που δεν είναι ισοζυγισμένος. Εκεί σταματάμε και προσπαθούμε κάνοντας τις κατάλληλες επαναζυγιστικές πράξεις για να κάνουμε τη διαφορά των κόμβων-παιδιών του να είναι το πολύ ένα. Οι πράξεις που μπορούν να μας φέρουν το επιθυμητό αποτέλεσμα είναι δύο: η απλή και η διπλή περιστροφή.

Η απλή περιστροφή μπορεί να είναι είτε αριστερή είτε δεξιά και γίνεται όταν τρεις κόμβοι (έστω x,y,z) δημιουργούν ένα δεξιό ή αριστερό “ευθές” μονοπάτι και παραβιάζουν έτσι τη συνθήκη της ισοζύγισης. Η φορά της περιστροφής είναι αντίθετη προς την κατεύθυνση που γέρνει το δέντρο εξαιτίας του “ευθύγραμμου” αυτού τμήματος .Γενικά με τον όρο περιστροφή εννοούμε την ανταλλαγή ρόλων των δυο εμπλεκόμενων κόμβων, μετατρέποντας τον κόμβο-γιο σε κόμβο-πατέρα και αντίστοιχα. Όπως φαίνεται και στο σχήμα (α), με την εισαγωγή ενός στοιχείου στο υποδέντρα Γ, ο κόμβος x παραβιάζει τη συνθήκη ισοζύγισης λόγω της αύξησης του ύψους του από h+2 σε h+3.Σ' αυτήν την περίπτωση πρέπει να κάνουμε μια απλή περιστροφή για να επανέλθει το ύψος του υποδέντρου σε h+2 και να γίνει το δέντρο ισοζυγισμένο.



Σχήμα (α): Απλή περιστροφή. Στο αριστερό μέρος φαίνεται η μορφή του δέντρου πριν την περιστροφή και στο δεξί η τελική μορφή του δέντρου. Το βέλος μας δείχνει την ενέργεια που πραγματοποιείται.

Η δεύτερη ενεργεία που μπορούμε να κάνουμε για την ισοζύγιση ενός δέντρου είναι η διπλή περιστροφή. Σε περίπτωση που οι κόμβοι x,y,z σχηματίζουν γωνία, είτε δεξιά είτε αριστερή, το στοιχειό που εισάγεται στο υποδέντρο Β ή Γ, που φαίνονται στο σχήμα (β), παραβιάζει την συνθήκη ισοζύγισης καθώς αυξάνει το ύψος σε h+3.Σ' αυτήν την περίπτωση πρέπει να κάνουμε μια διπλή περιστροφή για να επανέλθει το ύψος του z στην προηγούμενη τιμή του και άρα να αποκατασταθεί η ισορροπία στο δέντρο. Η διπλή περιστροφή ανεξάρτητα τη φορά της αποτελείται από δυο απλές περιστροφές αντίθετης φοράς. Παράδειγμα μια διπλή αριστερή περιστροφή αποτελείται από μια δεξιά και μια αριστερή απλή περιστροφή. Η πρώτη γίνεται πάνω στον ενδιάμεσο κόμβο y,που προκαλεί την υποβάθμισή του και η δεύτερη πάνω στον υψηλότερο κόμβο x.

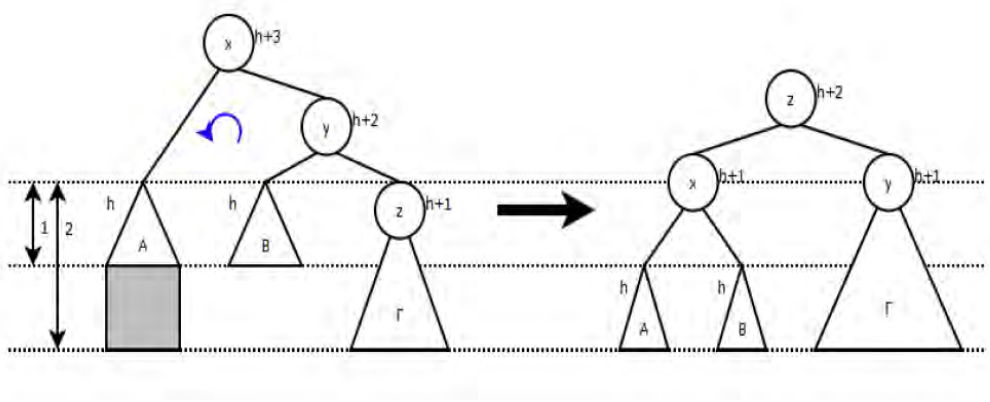


Σχήμα (β): Διπλή περιστροφή. Στο αριστερό μέρος φαίνεται η αρχική μορφή του δέντρου και στο δεξί η τελική μορφή του. Με τα έγχρωμα κυκλικά βελάκια απεικονίζονται οι ενέργειες που πραγματοποιούνται και η σειρά με την οποία πραγματοποιούνται.

Κάθε περιστροφή ανεξάρτητα από το αν είναι μόνη η διπλή και υποκαθιστά την ισοζύγιση στο δέντρο ονομάζεται τερματική. Λογικές δομικές πράξεις ονομάζονται αυτές που μεταβάλλουν τη μορφή του δέντρου. Αντίθετα, μη δομικές επαναζυγιστικές πράξεις ονομάζονται αυτές που απλώς αλλάζουν τη βοηθητική πληροφορία, όπως στην περίπτωση μας το ύψος.

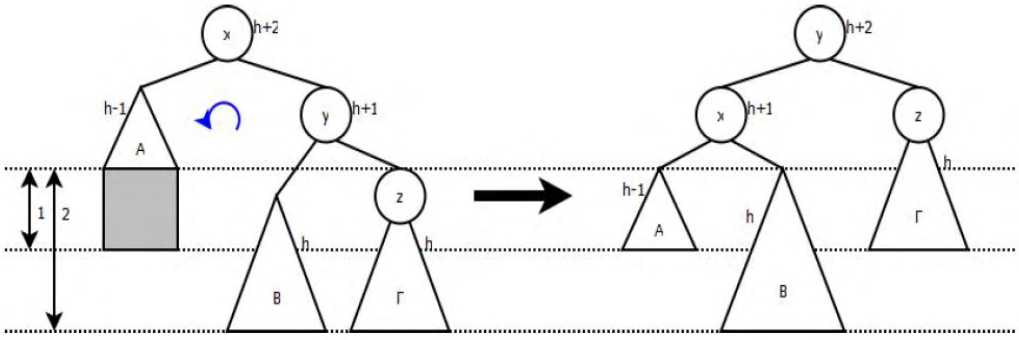
Σχετικά με τη διαδικασία διαγραφής στοιχείου, ακολουθούμε την παρακάτω διαδικασία .Αρχικά έχουμε ένα κλειδί, έστω κ. Διαγράφουμε τον κόμβο με το κλειδί αυτό και στη συνέχεια με αφετηρία των κόμβο αυτό της απόσβεσης, αναρριχούμαστε προς τη ρίζα με παράλληλη επιδιόρθωση των υψών .Σε κάθε κόμβο που εντοπίζουμε παράβαση εκτελούμε και την αντίστοιχη ενέργεια επιδιόρθωσης. Εδώ θα αναφερθούμε στις τρεις διαφορετικές περιπτώσεις σφάλματος που μπορούμε να συναντήσουμε.

Στην πρώτη περίπτωση, οι κόμβοι x,y,z σχηματίζουν δεξιό η αριστερό ευθές μονοπάτι και το στοιχείο έχει διαγραφεί από το υποδέντρο Α στο (γ), μειώνοντας έτσι το ύψος του κατά ένα σε h και επομένως το ύψος του x παραβιάζει τη συνθήκη. Η αντιμετώπιση θα είναι μια αριστερή η μια δεξιά απλή περιστροφή στο x,η οποία ισοζυγίζει το υποδέντρο μειώνοντας το ύψος του κορυφαίου κόμβου κατά ένα. Αυτό όμως έχει ως αποτέλεσμα να μην είμαστε σίγουροι αν οι πρόγονοι του y ικανοποιούν τη συνθήκη ή έχει δημιουργηθεί κάποια παραβίασή της. Έτσι, απαιτείται η εξέταση των προγόνων του y. Εδώ πρέπει να αναφέρουμε ότι τέτοιου τύπου ενέργειες που δεν εγγυώνται την οριστική εξάλειψη του προβλήματος ονομάζονται μη τερματικές.



Σχήμα (γ): Απεικόνιση πρώτης περίπτωσης. Αριστερή μη τερματική περιστροφή. Εμφάνιση σχήματος πριν και μετά την ενέργεια.

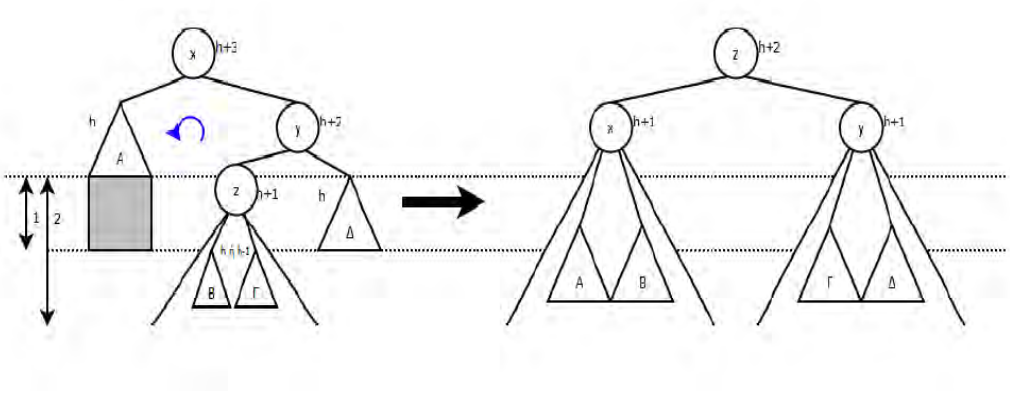
Στην δεύτερη τώρα περίπτωση, σχήμα (δ), με τους ίδιους κόμβους, διαγράφοντας στοιχειό από το υποδέντρο Α το ύψος γίνεται h-1 και ο κόμβος x παραβιάζει πάλι αρχική την συνθήκη .Η λύση είναι πάλι μια δεξιά η αριστερή απλή περιστροφή στον κόμβο x η όποια ισοζυγίζει το υποδέντρο διατηρώντας το ύψος του και μειώνοντας τη διαφορά ύψους των υποδέντρων σε ένα. Η ενεργεία αυτή είναι τερματική.



.

Σχήμα (δ): Απεικόνιση δεύτερης περίπτωσης. Το σχήμα πριν και μετά την εκτέλεση της απλής περιστροφής

Η τρίτη και τελευταία περίπτωση που μπορούμε να συναντήσουμε στην απόσβεση στοιχείου είναι όπως φαίνεται στο σχήμα (ε). Έστω ότι έχουμε πάλι τους ίδιους κόμβους x,y,z. Οι κομβόι σχηματίζουν γωνία είτε αριστερή είτε δεξιά. Το στοιχείο που σβήνεται ήταν στο υποδέντρο Α, μειώνοντας το ύψος κατά ένα σε h,και ο x παραβιάζει την συνθήκη. Αυτό αντιμετωπίζεται με μια δεξιά η αριστερή διπλή περιστροφή η όποια ισοζυγίζει το δέντρο μειώνοντας έτσι το ύψος του κατά ένα. Συμπεραίνουμε από αυτό ότι η πράξη δεν είναι τερματική.



Σχήμα (ε): Απεικόνιση τρίτης περίπτωσης. Εμφάνιση σχήματος πριν και μετά την μη τερματική διπλή περιστροφή.

Όσον αφόρα την πολυπλοκότητα της ένθεσης και της απόσβεσης στοιχείου σε AVL δέντρα πρέπει να αναφέρουμε αρχικά ότι κάθε υποδέντρο ενός AVL δέντρου είναι και αυτό AVL δέντρο, όπως προκύπτει άμεσα από τον ορισμό των AVL δέντρων. Το ύψος ενός AVL δέντρου n στοιχείων είναι O(logn).Για να το αποδείξουμε αυτό θα πρέπει να μελετήσουμε τις δυο ακραίες περιπτώσεις που είναι το κοντύτερο δυαδικό δέντρο n εσωτερικών κόμβων που είναι το πλήρες δυαδικό δέντρο, το οποίο έχει ύψος h=log(n+1).Στο άλλο άκρο έχουμε τα δέντρα Fibonacci, που έχουν και αυτά λογαριθμική συμπεριφορά. Ακόμη τα δέντρα αυτά έχουν για κάθε ένθεση O(1) και για κάθε απόσβεση O(logn) κόστος σε επαναζυγιστικές πράξεις. Η ίδια βασική προσέγγιση μπορεί εύκολα να προσεγγιστεί και από άλλους τύπους δέντρων αναζήτησης, όπως τα RED-BLACK και τα B-δέντρα.

Τα δέντρα των διαστημάτων είναι ταξινομημένα συμφώνα με το αριστερό τους άκρο. Για τον κάθε κόμβο του δέντρου αναζήτησης κρατάμε αξιόπιστα τις ακόλουθες μεταβλητές:

1.Το άκρο του κάθε διαστήματος I(v) και ο αριθμός |I(v)| των κλειδιών την τρέχουσα στιγμή περιέχονται στο I(v).

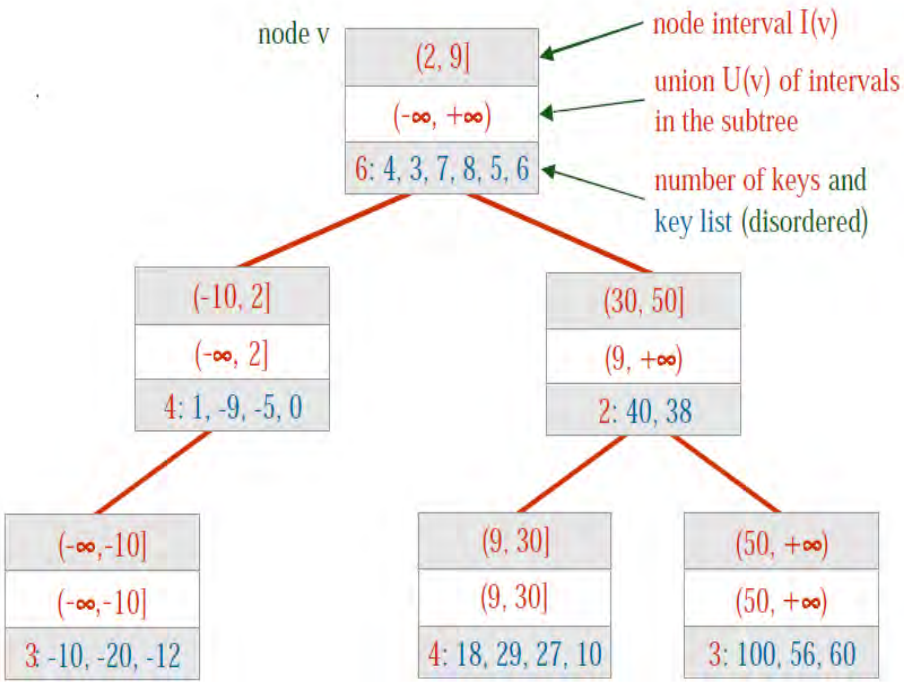
2.το διάστημα U(u), του οποίου τα όρια είναι το μικρότερο και το μεγαλύτερο άκρο των διαστημάτων που περιέχονται στο υποδέντρο αρχικοποιείται στο u.

3.Οι διευθύνσεις του αριστερού παιδιού, του δεξιού παιδιού και του γονέα του u, και οι πληροφορίες που απαιτούνται για να διατηρηθεί το δέντρο αναζήτησης ζυγισμένο εξετάζονται στην υλοποίηση.

Επιπλέον, κρατάμε αναξιόπιστα ένα αντίγραφο, σε ένα πινάκα 2δ θέσεων:

4.το σύνολο των κλειδιών την τρέχουσα στιγμή, όχι ταξινομημένο, που περιέχονται στο I(u).

Στην εικόνα 8,που ακολουθεί, φαίνεται ένα δέντρο του τύπου που περιγράψαμε.



Εικόνα 8: Ένα ανθεκτικό δέντρο με έξι κόμβους καθένας από τους οποίους έχει το διάστημα I(u),στο οποίο ανήκουν τα κλειδιά του, το διάστημα U(u) στο οποίο ανήκουν τα υποδέντρα των παιδιών του, και τον αριθμό των δικών του κλειδιών καθώς και το ποια συγκεκριμένα κλειδιά περιέχει [31].

Τα διαστήματα U(u) παίζουν πολύ σημαντικό ρολό στην προσέγγισή μας. Χρησιμοποιούνται για να καταλάβουμε αν η αναζήτηση εξελίσσεται προς τη σωστή κατεύθυνση στο δέντρο. Επίσης, το κλειδί που ψάχνουμε πρέπει να περιέχεται και σε κάθε U(u) που περνάμε κατά τη διάρκεια της αναζήτησης, έκτος και αν έχει γίνει κάποιο λάθος στην αναζήτηση. Ακόμη, κάθε παιδί u' του u πρέπει να είναι υποσύνολο του. Στην αρχή κάθε πράξης, το U(u) είναι ακριβώς η ένωση των άλλων διαστημάτων που περιέχονται στο υποδέντρο που έχει ως ρίζα το u. Αυτή η ιδιότητα μπορεί να πάψει να ισχύει για προσωρινά κατά τη διάρκεια μια εισαγωγής η μιας διαγραφής ενός κλειδιού. Σημειώνουμε ότι σε ένα δέντρο AVL μπορούμε να διατηρούμε τα διαστήματα U(u) χωρίς να αυξάνεται το κόστος των εισαγωγών και διαγράφων των κλειδιών.

Οι κόμβοι ενός δέντρου αναζήτησης αποθηκεύονται σε ένα πίνακα. Ο κυρίως λόγος που το κάνουμε αυτό είναι ότι έτσι είναι εύκολο να καταλάβουμε αν ο δείκτης, δείχνει σε έναν κόμβο του δέντρου. Διαφορετικά ο αλγόριθμος θα βρισκόταν έξω από το δέντρο αναζήτησης, ακολουθώντας έναν κατεστραμμένο δείκτη χωρίς να το καταλάβει. Αυτό μπορεί να είχε σαν αποτέλεσμα την απροσδόκητη συμπεριφορά του αλγορίθμου. Η διεύθυνση του πινάκα αυτού και του αριθμού των κόμβων αποθηκεύονται σε ασφαλή μνήμη, μαζί με τη διεύθυνση

του κόμβου-ριζά. Χρησιμοποιούμε μια σταθερή τεχνική διπλασιασμού [9] για να είμαστε σίγουροι ότι το μέγεθος του πινάκα είναι γραμμικό στον τρέχοντα αριθμό των κόμβων. Σε περίπτωση που ο πίνακας γεμίσει, δημιουργούμε ένα νέο πινάκα με μέγεθος δυο φόρες το αρχικό του μέγεθος. Στην αντίθετη περίπτωση, όταν δηλαδή ο πίνακας είναι άδειος κατά τα 3/4,δημουργουμε ένα πινάκα με μέγεθος το μισό του αρχικού. Και στις δυο περιπτώσεις, όλοι οι κόμβοι μετακινούνται στο νέο πινάκα και ο παλιός πίνακας καταστρέφεται. Η επιβάρυνση για κάθε εισαγωγή/διαγραφή κόμβου εξαιτίας του (υπο) διπλασιασμού είναι Ο(1) φορές το κόστος της εισαγωγής/διαγραφής κόμβου. Στη περίπτωση μας η επιβάρυνση θα είναι Ο(δ), καθώς ο πίνακας θα περιέχει δ στοιχειά. Στη συνέχεια θα δείξουμε ότι κάθε πράξη η όποια περιέχει εισαγωγή/διαγραφή κόμβου παίρνει Ω(δ) χρόνο στην πραγματικότητα, εξαιρώντας τον χρόνο που ξοδεύεται για τον (υπο) διπλασιασμό. Έτσι, από δω και πέρα μπορούμε να επιβαρύνουμε το κόστος του διπλασιασμού, στο κόστος των άλλων λειτουργιών. Αυτό έχει ως αποτέλεσμα να μη μας ενδιαφέρει το κόστος του διπλασιασμού από δω και πέρα στην ανάλυση μας. Στη συνέχεια θα περιγράψουμε τον τρόπο που γίνεται η αναζήτηση η εισαγωγή και η διαγραφή κλειδιού.

4.2 ΛΕΙΤΟΥΡΓΙΕΣ ΑΛΓΟΡΙΘΜΟΥ

4.2.1 ΑΝΑΖΗΤΗΣΗ ΚΛΕΙΔΙΟΥ

Κάθε αναζήτηση εκτελείται σε δυο στάδιο. Αρχικά, εκτελείται μια αναζήτηση στην οποία ψάχνουμε το διάστημα Ι(κ) που περιέχει το κ. Η αναζήτηση αυτή είναι μια αναζήτηση διαστήματος. Στη συνέχεια, εκτελούμε μια γραμμική αναζήτηση του κ μέσα στο διάστημα Ι(κ) που βρήκαμε στην πρώτη αναζήτηση. Η υλοποίηση της αναζήτησης διαστήματος είναι πολύ σημαντική και θα την αναλύσουμε στις ενότητες 3 και 4 που ακολουθούν.

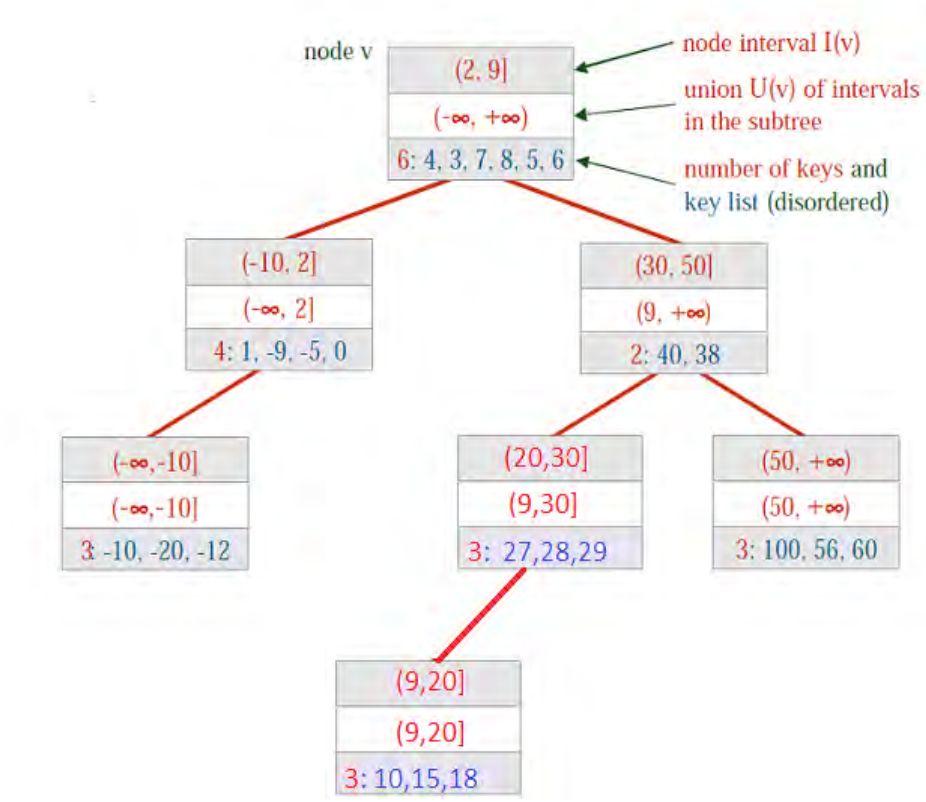
4.2.2 ΕΙΣΑΓΩΓΗ ΚΛΕΙΔΙΟΥ

Για την υλοποίηση της εισαγωγής ενός κλειδιού κ, αρχικά πρέπει να βρούμε το διάστημα Ι στο οποίο ανήκει το κ. Στη συνέχεια, σε περίπτωση που το κ δεν είναι ήδη στη λίστα των κλειδιών που ανήκουν στο Ι, το προσθέτουμε στη λίστα. Εάν το μέγεθος της λίστας του εν λόγο διαστήματος γίνει 2δ από αυτήν την εισαγωγή, εκτελούμε την ακόλουθη λειτουργιά για να μην καταπατάται το (ΙΙ) από τα χαρακτηριστικά των διαστημάτων που έχουμε αναφέρει προηγουμένως. Η πρώτη κίνηση μας είναι να διαγράψουμε το διάστημα Ι από το δέντρο αναζήτησης και τα το χωρίσουμε σε δυο μη επικαλυπτόμενα υποδιαστήματα αριστερό (Α) και δεξί (Δ), τέτοια ώστε ΑUΔ=Ι. Τα διαστήματα Α και Δ παίρνουν τα μισά μικρότερα κλειδιά και τα μισά μεγαλύτερα κλειδιά, αντίστοιχα. Με σκοπό να μοιράσουμε τα κλειδιά του Ι σε δυο ίσα κομμάτια, χρησιμοποιούμε τον αλγόριθμο Bubblesort-δύο δρόμων. Ο αλγόριθμος αυτός είναι γνωστός και ως ταξινόμηση αναμικτήρα και εφαρμόζει τον αλγόριθμο Bubblesort και προς τις δύο κατευθύνσεις. Έτσι ο πίνακας κατά τη διάρκεια της εκτέλεσης του αλγορίθμου χωρίζεται σε τρία τμήματα που είναι το αριστερό με τα μικρότερα στοιχεία του πίνακα ταξινομημένα ,το δεξί με τα μεγαλύτερα στοιχεία του πίνακα ταξινομημένα και τέλος το μεσαίο που περιέχει τα στοιχεία του πίνακα που δεν έχουν ταξινομηθεί ακόμη. Αυτό

παίρνει $O(\delta^2)$ χρόνο. Στο τέλος, εισάγουμε τα διαστήματα Α και Δ στο δέντρο αναζήτησης. Και οι διαγραφές και οι εισαγωγές διαστημάτων από/προς το δέντρο αναζήτησης εκτελούνται με καθιερωμένο τρόπο, κάνοντας δηλαδή περιστροφές για να είναι το δέντρο ζυγισμένο, χρησιμοποιώντας όμως μόνο ανθεκτικές μεταβλητές σε χρόνο O(δlogn). Αξίζει να σημειωθεί ότι τα αμετάβλητα χαρακτηριστικά (i) και (iii) τον διαστημάτων ισχύουν σε όλη τη διάρκεια της παραπάνω διαδικασίας.

4.2.2.1 ΠΑΡΑΔΕΙΓΜΑ ΕΙΣΑΓΩΓΗΣ ΚΛΕΙΔΙΟΥ

Για να γίνει πιο κατανοητή η διαδικασία ένθεσης κλειδιού στο δέντρο διαστημάτων που μελετάει ο αλγόριθμός μας θα παρουσιάσουμε ένα παράδειγμα εισαγωγής στοιχείων. Αρχικά πραγματοποιώντας μία ένθεση του στοιχείου 27 στο δέντρο της εικόνας 6,η ένθεση θα εντοπίσει αρχικά το διάστημα I(u) που ανήκει το 27 και θα διαπιστώσει ότι αυτό το κλειδί υπάρχει ήδη στον συγκεκριμένο κόμβο οπότε θα σταματήσει η λειτουργία εδώ. Προσπαθώντας στη συνέχεια να εισάγουμε το κλειδί 28 στο ίδιο στιγμιότυπο δέντρου θα πραγματοποιηθεί ξανά από την αρχή η αναζήτηση του κόμβου που περιέχει το διάστημα I(u) που ανήκει το 28.Στη συνέχεια θα παρατηρήσουμε ότι το συγκεκριμένο κλειδί δεν υπάρχει στο δέντρο και έτσι θα το εισάγουμε στον κόμβο. Αν ο αριθμός των κλειδιών στον κόμβο μετά την εισαγωγή του κλειδιού είναι μικρότερος από 2δ η λειτουργία εισαγωγής τερματίζει. Με την ίδια ακριβώς διαδικασία εισάγουμε και το κλειδί 15. Ας υποθέσουμε τώρα ότι ο κόμβος αυτός μετά την ένθεση του 15 έχει 2δ κλειδιά, με δ=3. Εφαρμόζουμε στα κλειδιά του διαστήματος των αλγόριθμο Bubblesort-δύο δρόμων. Μετά το πέρας του αλγορίθμου τα στοιχεία έχουν μπει σε αύξουσα σειρά. Αφού αφαιρέσουμε τον κόμβο από το δέντρο χωρίζουμε τα κλειδιά σε δύο ίσα κομμάτια καθένα από τα οποία θα αποτελέσει έναν καινούριο κόμβο στο δέντρο. Αναλυτικότερα, στην περίπτωσή μας τα τρία μεγαλύτερα κλειδιά (27, 28, 29) θα αποτελέσουν το περιεχόμενο του ενός από τους δύο νέους κόμβους και ο κόμβος αυτός θα μπει στη θέση του αρχικού κόμβου. Τα άλλα τρία μικρότερα κλειδία (10,15,18) θα αποτελέσουν το περιεχόμενο του αριστερού παιδιού του κόμβου με τα τρία μεγαλύτερα κλειδιά (σχήμα 1).



Σχήμα 1: Δέντρο διαστημάτων μετά τις εισαγωγές των κλειδιών 15 και 28 και μετά την διάσπαση του κόμβου που περιείχε 6 (2δ) κλειδιά.

4.2.3 ΔΙΑΓΡΑΦΗ ΚΛΕΙΔΙΟΥ

Η διαδικασία της διαγραφής είναι πιο δυσνόητη από τις άλλες δυο διαδικασίες που περιγράφτηκαν λίγο νωρίτερα ,καθώς υπάρχουν περισσότερες περιπτώσεις τροποποίησης των κόμβων του δέντρου. Γι αυτό και θα αναφερθούμε αναλυτικά στην διαδικασία της διαγραφής και στις ενέργειες που πρέπει να ακολουθηθούν σε κάθε περίπτωση.

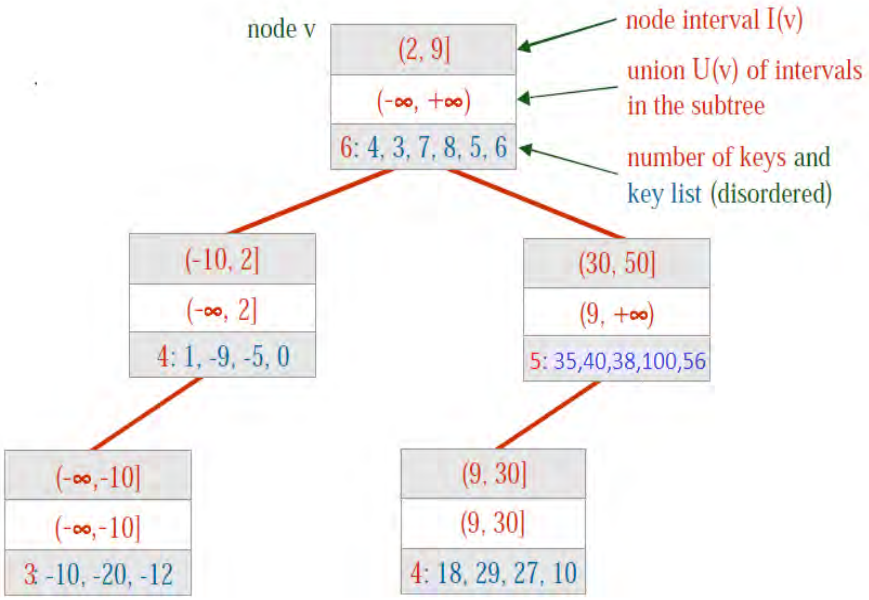
4.2.3.1 ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ ΔΙΑΓΡΑΦΗΣ

Για να διαγράψουμε ένα κλειδί, έστω κ, από το δέντρο αναζήτησης ,αρχικά, βρίσκουμε το διάστημα I που περιέχει το συγκεκριμένο κλειδί. Αν το κλειδί αυτό υπάρχει στον κόμβο, το διαγράφουμε. Στη συνέχεια, παρατηρούμε τον αριθμό των κλειδιών που περιέχει το διάστημα I. Σε περίπτωση που ισχύει |I|= δ/2 και το I δεν είναι ένα ακραίο διάστημα, κάνουμε την εξής διαδικασία για να διατηρείται η ιδιότητα (III) των διαστημάτων. Αρχικά, ψάχνουμε το διάστημα A στα αριστερά του I και διαγράφουμε και τα δύο από το δέντρο αναζήτησης. Έπειτα κάνουμε

δυο διαφορετικές ενέργειες ανάλογα με το μέγεθος του Α. Αν το |Α|≤δ, ενώνουμε τα Α και Ι και δημιουργούμε ένα μοναδικό διάστημα έστω Ι'=ΑUΙ και εισάγουμε το Ι' στο δέντρο αναζήτησης. Αλλιώς, σε περίπτωση δηλαδή που το |Α|>δ, δημιουργούμε δυο καινούργια μη- επικαλυπτόμενα διαστήματα Α' και Ι', τέτοια ώστε Α'UΙ'=ΑUΙ. Το διάστημα Α' θα περιέχει όλα τα κλειδιά του Α, έκτος από τα δ/4 μεγαλύτερα κλειδιά που θα ανήκουν στο Ι' μαζί με τα αρχικά κλειδιά του Ι. Έτσι καταφέρνουμε να μην καταπατάται καμιά από τις ιδιότητες του δέντρου αναζήτησης .Σ' αυτήν την περίπτωση δημιουργώντας το Α' και το Ι' ξοδεύουμε $O(\delta^2)$ χρόνο χρησιμοποιώντας τον αλγόριθμο Bubblesort δύο δρόμων. Έπειτα, εισάγουμε τα διαστήματα Α' και Ι' στο δέντρο αναζήτησης.

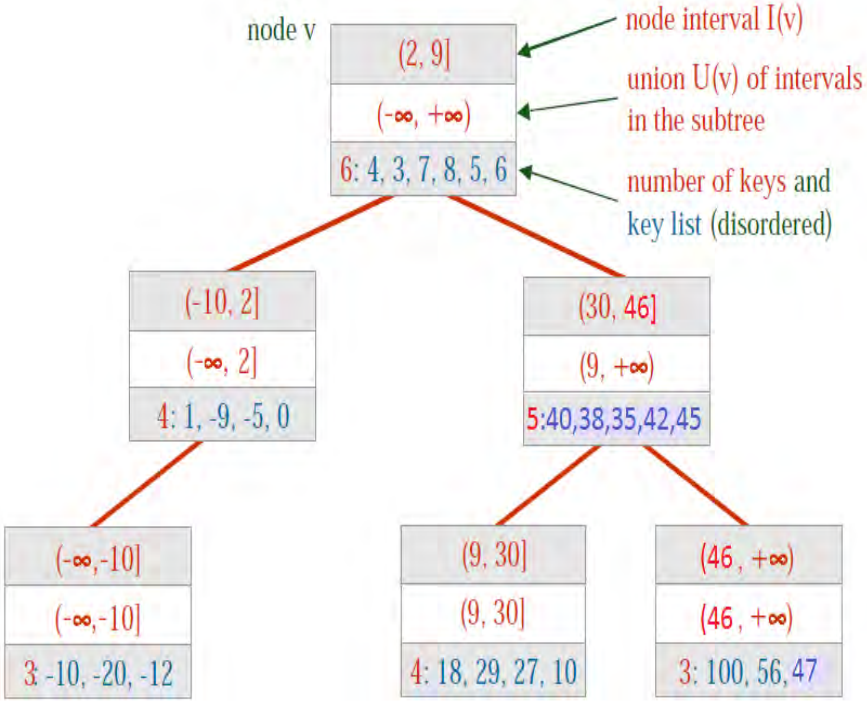
4.2.3.2 ΠΑΡΑΔΕΙΓΜΑ ΔΙΑΓΡΑΦΗΣ ΚΛΕΙΔΙΟΥ

Για την καλύτερη κατανόηση των δύο περιπτώσεων διαγραφής στοιχείου θα παρουσιάσουμε παραδείγματα. Αρχικά ας υποθέσουμε ότι δ=4 και ότι θέλουμε να διαγράψουμε από το δέντρο της εικόνας 6 το κλειδί 58 , θεωρώντας ότι ο κόμβος γονέας του περιέχει τρία κλειδιά εκτός δηλαδή από τα 40 , 38 και το 35.Επίσης αγνοούμε ότι το διάστημα που θα αναφερόμαστε αποτελεί ακραίο διάστημα. Το πρώτο βήμα του αλγορίθμου διαγραφής είναι να εντοπίσει το διάστημα στο οποίο θα ανήκει το κλειδί που θέλουμε να διαγράψουμε και κατ' επέκταση των κόμβο που θα το περιέχει. Παρατηρούμε, εντοπίζοντας τον συγκεκριμένο κόμβο ότι δεν περιέχει το κλειδί 58 και έτσι η διαδικασία διαγραφής του τερματίζει εδώ. Προσπαθώντας στη συνέχεια να διαγράψουμε το κλειδί 60 και αφού εντοπιστεί ο κόμβος που το περιέχει το διαγράφουμε. Παρατηρούμε όμως ότι ο συγκεκριμένος κόμβος περιέχει πλέον δ/2=4/2=2 στοιχεία. Επομένως, διαγράφουμε τον κόμβο αυτό και τον κόμβο Α, που βρίσκεται στα αριστερά του. Προσέχουμε ότι ο κόμβος Α περιέχει 3 στοιχεία. Καταλαβαίνουμε ότι βρισκόμαστε στην περίπτωση που ο κόμβος Α περιέχει |Α|≤4 στοιχεία. Έτσι ενώνουμε τα στοιχεία του Α και του Ι σε έναν κόμβο και δημιουργούμε έναν κόμβο με όλα τα κλειδιά. Ο νέος κόμβος θα περιέχει τα κλειδιά 40, 38, 35, 100, 56, όπως φαίνεται στο σχήμα 2.



Σχήμα 2: Η τελική μορφή του δέντρου διαστημάτων μετά τη διαγραφή του στοιχείου 60 και μετά την ένωση των κόμβων.

Ας υποθέσουμε τώρα ότι ο κόμβος Α του προηγούμενου παραδείγματος περιέχει τα κλειδιά 40, 38, 35, 42, 45, 47 και διαγράφουμε ξανά το κλειδί 60.Στο πρώτο βήμα ο αλγόριθμος διαγραφής εντοπίζει τον διάστημα που ανήκει το συγκεκριμένο κλειδί και ψάχνει τα περιεχόμενα του κόμβου στα οποία και βρίσκει το κλειδί και το διαγράφει. Ελέγχοντας τον κόμβο Α παρατηρούμε σ αυτήν την περίπτωση ότι ισχύει |A|>4. Επομένως, δημιουργούμε δύο νέους κόμβους ,έστω Α' και Ι' που τα περιεχόμενά τους είναι τα εξείς. Ο Α είχε αρχικά 6 κλειδιά. Από αυτά τα δ/4=4/4=1 μεγαλύτερα κλειδιά τα μεταφέρουμε στο Ι. Επομένως το νέο Α' θα έχει τα 5 κλειδιά (40, 38, 35, 42, 45).Το νέο Ι' θα έχει τα παλιά κλειδιά του Ι συν το ένα που μεταφέρουμε από το Α δηλαδή τα (47, 100, 56) όπως απεικονίζεται στο σχήμα 3.



Σχήμα 3: Το τελικό δέντρο διαστημάτων με |A|>δ μετά τη τελική μετατόπιση κλειδιών και συγκεκριμένα του κλειδιού 47.

Από τη στιγμή που χρησιμοποιούμε ανθεκτικές μεταβλητές, το κόστος για κάθε εισαγωγή/διαγραφή ενός διαστήματος είναι O(δlogn). Παρατηρούμε ότι οι ιδιότητες (i) και (ii) των διαστημάτων ισχύουν καθ όλη τη διάρκεια των ενεργειών που περιγράψαμε.

Από την ιδιότητα (iii) προκύπτει ότι ο συνολικός αριθμός των κόμβων στο δέντρο αναζήτησης είναι O(1+ n/δ). Αν ο κάθε κόμβος λαμβάνει Θ(δ) χώρο, εξαιτίας του διπλασιασμού, ο χώρος που λαμβάνεται από το δέντρο αναζήτησης είναι O(n+δ). Αυτό είναι ,επίσης, ένα άνω όριο της συνολικής πολυπλοκότητας του χώρου. Η πολυπλοκότητα του χώρου μπορεί να περιοριστεί στο O(n) αποθηκεύοντας τις μεταβλητές που ανήκουν στα οριακά διαστήματα σε O(1) μέγεθος ασφαλούς μνήμης, και διαχειρίζοντας το αντίστοιχο σύνολο κλειδιών μέσω του διπλασιασμού. Αυτή η αλλαγή μπορεί να πραγματοποιηθεί χωρίς να επηρεαστεί ο χρόνος εκτέλεσης τον πράξεων και μειώνει το χώρο που δεσμεύεται σε O(n).

Αυτό που απομένει τώρα είναι να περιγράψουμε την υλοποίηση της αναζήτησης διαστημάτων. Όπως προκύπτει από το Λήμμα που ακολουθεί, υπάρχει μια κρίσιμη διαφορά στην εκτέλεση της παραπάνω δομής δεδομένων .

4.2.3.3 ΛΗΜΜΑ 6ο

ΛΗΜΜΑ 6ο:Αν S(n,δ) η χειρότερη περίπτωση από άποψη χρόνου που απαιτείται για την εκτέλεση μιας αναζήτησης διαστήματος, τότε η χειρότερη περίπτωση χαμένου χρόνου ανά λειτουργιά της παραπάνω δομής δεδομένων είναι O(S(n,δ)+logn+δ). Η πολυπλοκότητα του είναι O(n).

Απόδειξη:

Η πολυπλοκότητα χώρου είναι O(n), όπως αναφέραμε πιο πάνω. Από την αμετάβλητη ιδιότητα (ii), κάθε αναζήτηση παίρνει O(S(n,δ)+δ) χρόνο στην χειρότερη περίπτωση. Το ίδιο ισχύει και για τις πράξεις της εισαγωγής και της διαγραφής, σε περίπτωση που το δέντρο αναζήτησης δεν χρειάζεται να τροποποιηθεί. Αλλιώς για κάθε εισαγωγή/διαγραφή έχουμε ένα επιπλέον κόστος κατά $O(\delta \log n+\delta^2)$. Στη συνέχεια θα δείξουμε ότι από τις ιδιότητες (ii) και (iii) το δέντρο αναζήτησης τροποποιείται κάθε Ω(δ) εισαγωγές και/η διαγραφές. Από αυτό συνεπάγεται ότι το κόστος των εισαγωγών και διαγράφων είναι O(S(n,δ)+logn+δ).

Αρχικά, το δέντρο αναζήτησης είναι κενό και υπάρχει ένα μοναδικό άδειο ακραίο διάστημα. Αυτό το διάστημα χωρίζεται υστέρα από τουλάχιστον 2δ εισαγωγές. Θεωρούμε τώρα ένα διάστημα I το οποίο μόλις έχει δημιουργηθεί. Είναι εφικτό να δείξουμε ότι μετά την δημιουργία του I, απαιτούνται Ω(δ) εισαγωγές ή διαγραφές για να φτάσει το I σε ένα από τα δυο κατώφλια ,δ/2 για ένωση και 2δ για διαχωρισμό.

Προσοχη!!! Σε περίπτωση που το διάστημα είναι ακραίο διάστημα, κοιτάζουμε μόνο το δεύτερο κατώφλι.

Το διάστημα I μπορεί να δημιουργηθεί με τους τέσσερις ακόλουθους πιθανούς τρόπους:

(1) Με διαχωρισμό του διαστήματος I' σε δυο ίσα διαστήματα. |I'|=2δ. Επομένως |I|=δ.

(2)Με προσθήκη δ/4 κλειδιών σε ένα διάστημα I'.|I'|=δ/2.Ετσι,|I|=3δ/4

(3)Με τη διαγραφή δ/4 κλειδιών από ένα διάστημα I',|I'|>δ. Σ αυτή τη περίπτωση 3δ/4<|I|<7δ/4

(4) Με την ένωση δυο διαστημάτων I' και I'',|I'|=δ/2 και

|I''|≤δ.Σ αυτή τη περίπτωση |I|≤3δ/2

θεωρούμε τώρα ότι το διάστημα |I| φτάνει στο κατώφλι δ/2 και δεν είναι ακραίο διάστημα. Στις περιπτώσεις (1),(2) και (3), που αναφέραμε λίγο πριν αυτό μπορεί

να συμβεί υστέρα από τουλάχιστον δ/4 διαγραφές από το διάστημα αυτό. Στην περίπτωση (4), το διάστημα I'' δε μπορεί να είναι ακραίο διάστημα, γιατί και το διάστημα I θα ήταν επίσης ακραίο διάστημα .Αν |I''|>δ/2 και |I|>δ, ο αριθμός των διαγράφων πρέπει να είναι τουλάχιστον δ/2.

Στην συνέχεια θα παρουσιάσουμε μια απλή τυχαία διαδικασία αναζήτησης διαστήματος με O(logn+δ) προσδοκώμενο χρόνο εκτέλεσης. Έπειτα θα περιγράψουμε μια ντετερμινιστική διαδικασία για την ίδια περίπτωση με $O(\log n+\delta+\alpha\delta^{\varepsilon})=O(\log n+\delta^{1+\varepsilon})$ χειρότερη περίπτωση χρόνου εκτέλεσης, όπου ε>0 μια αυθαίρετη μικρή σταθερά. Από το Λήμμα 1, συνεπάγεται άμεσα ανθεκτικά δέντρα αναζήτησης με ανάλογο κόστος εκτέλεσης.

ΚΕΦΑΛΑΙΟ 5ο ΤΥΧΑΙΑ ΑΝΑΖΗΤΗΣΗ ΔΙΑΣΤΗΜΑΤΩΝ

5.1 ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΖΗΤΗΣΗΣ

Τώρα θα παρουσιάσουμε έναν απλό τυχαίο αλγόριθμο για την εκτέλεση μιας αναζήτησης διαστήματος, όπως παραδείγματος χάριν να βρούμε ένα διάστημα I(κ) που περιέχει το δοσμένο κλειδί κ, που ψάχνουμε. Υπενθυμίζουμε ότι το διάστημα I(κ) υπάρχει και μάλιστα είναι μοναδικό όπως προκύπτει από την ιδιότητα (I) του προηγούμενου κεφαλαίου.Ο συγκεκριμένος αλγόριθμος έχει O(logn+δ) προσδοκώμενο χρόνο εκτέλεσης. Για την εκτέλεση του αλγορίθμου αυτού, έχουμε μια γραμμική σε χώρο ανθεκτική δομή λεξικού με O(logn+δ) προσδοκώμενο χαμένο χρόνο για κάθε πράξη.

Η βασική βλέψη πίσω από αυτήν την προσέγγιση είναι η ακόλουθη βελτιστοποίηση. Θεωρούμε κάθε ντετερμινιστικό αλγόριθμο A' που λύνει μερικά προβλήματα και έχει O(t') χρόνο εκτέλεσης. Υποθέτουμε ότι ο A' διαβάζει τη κάθε μεταβλητή μια φόρα το πολύ. Αντικαθιστούμε κάθε μεταβλητή του A' με μια ανθεκτική μεταβλητή και επιτρέπουμε στον A να είναι ένας επιπόλαιος-ανθεκτικός αλγόριθμος αποτελέσματος.

Παρατηρούμε ότι ο χρόνος εκτέλεσης του A είναι O(δt').Ακόμη, έχουμε την C,που είναι μια προσαρμοζόμενη διαδικασία που ελέγχει αν ισχύει το αποτέλεσμα και έχει σαν χρόνο εκτέλεσης O(t'').Στη συνέχεια είναι δυνατόν να βρούμε έναν τυχαίο προσαρμοζόμενο αλγόριθμο R, ο οποίος επιλύει το ίδιο πρόβλημα με τον A σε προσδοκώμενο χρόνο O(t'+t'').

Ο αλγόριθμος R αποτελείται από μερικούς γύρους. Στον κάθε γύρο, ο αλγόριθμος R τρέχει τον αλγόριθμο A, αλλά θεωρεί μόνο ένα αντίγραφο κάθε ανθεκτικής μεταβλητής επιλεγμένο ανομοιόμορφα και τυχαία. Στο τέλος του κάθε γύρου ο αλγόριθμος R ελέγχει την ορθότητα του αποτελέσματος που παράγεται από τον C.Αν η απάντηση δεν είναι σωστή, ο R ξεκινά ένα νέο γύρο, διαφορετικά, βγάζει σαν αποτέλεσμα την υπολογισμένη λύση.

5.2 ΛΗΜΜΑ 7ο

ΛΗΜΜΑ 7ο :

Ο αλγόριθμος R,που περιγράφεται πιο πάνω, λύνει το ίδιο πρόβλημα με τον αλγόριθμο A σε O(t'+t'') χρόνο, όπου ο A έχει O(t') χρόνο εκτέλεσης και η εγκυρότητα του αποτελέσματος μπορεί να διαπιστωθεί σε O(t'') χρόνο.

ΑΠΟΔΕΙΞΗ:

Η εγκυρότητα του αλγορίθμου R εξαρτάται άμεσα από την εγκυρότητα των αλγορίθμων A και C.Στην πραγματικότητα, ο R ποτέ δεν παγώνει με μια λανθασμένη απάντηση. Επίσης, σταματά αν κανένα λανθασμένο αντίγραφο επιλεχθεί σε έναν γύρο, πράγμα που συμβαίνει με μεγάλη πιθανότητα. Από τη στιγμή που ο κάθε γύρος παίρνει O(t'+t'') χρόνο , απομένει να δείξουμε ότι ο προσδοκώμενος αριθμός γύρων είναι σταθερός. Ας μελετήσουμε τον εξής γύρο. Θέτουμε στο a_i τον ενεργό αριθμό των λαθών που έχουν συμβεί μέχρι τώρα στην i-στη ανθεκτική μεταβλητή κατά τη διάρκεια του γύρου, με i=1,2,3,...,p =O(t'). Η πιθανότητα όλα τα αντίγραφα που έχουν επιλεχθεί κατά τη διάρκεια του συγκεκριμένου γύρου να είναι λανθασμένα είναι τουλάχιστον

$$\left(1-\frac{\alpha_1}{2\delta+1}\right)\left(1-\frac{\alpha_2}{2\delta+1}\right)\cdots\left(1-\frac{\alpha_p}{2\delta+1}\right)\geq\left(1-\frac{\sum_{i=1}^{p}\alpha_i}{2\delta+1}\right).$$

Υπό την ίδια υπόθεση (ότι δηλαδή όλα τα αντίγραφα που επιλέχθηκαν να είναι λανθασμένα), η συμπεριφορά του A και του R σε κάθε βήμα του γύρου είναι ακριβώς το ίδιο. Επιπρόσθετα, οι δυο αλγόριθμοι εξετάζουν ακριβώς την ίδια ακολουθία μεταβλητών. Υπενθυμίζουμε ότι από την υπόθεση ο αλγόριθμος A ποτέ δεν εξετάζει δυο φορές την ίδια μεταβλητή. Αυτό έχει σα συνέπεια, οι μεταβλητές που εξετάζονται από τον R να είναι όλες ξεχωριστές και $\sum_{i=1}^{p} a_i \leq \alpha \leq \delta$. Οπότε έχουμε :

$$\left(1-\frac{\sum_{i=1}^{p}\alpha_i}{2\delta+1}\right)\geq\left(1-\frac{\delta}{2\delta+1}\right)\geq\frac{1}{2}.$$

αυτό συνεπάγεται ότι ο αναμενόμενος αριθμός των γύρων είναι το πολύ 2.Τωρα θα δείξουμε τον τρόπο που χρησιμοποιούμε το Λήμμα 2 με σκοπό να λύσουμε το πρόβλημα αναζήτησης διαστημάτων.

5.3 ΛΗΜΜΑ 8ο

ΛΗΜΜΑ 8ο:

Υπάρχει αλγόριθμος αναζήτησης διαστήματος με αναμενόμενο χρόνο εκτέλεσης O(logn+δ).

ΑΠΟΔΕΙΞΗ:

Αρχικά σημειώνουμε ότι, όταν μας δοθεί κάποια διεύθυνση μνήμης μπορούμε εύκολα να διαπιστώσουμε σε σταθερό χρόνο αν αυτή η διεύθυνση ανήκει σε κάποιο κόμβο του δέντρου αναζήτησης. Επιπρόσθετα, όταν μας δοθεί κάποιος κόμβος u του δέντρου, μπορούμε να διαπιστώσουμε αξιόπιστα σε O(δ) χρόνο αν ένα κλειδί , έστω κ, ανήκει σ αυτό το κόμβο δηλαδή στο διάστημα I(u). Έτσι, υπάρχει μια O(t'')=O(δ) αξιόπιστη διαδικασία C να ελέγχουμε την εγκυρότητα της εξόδου ενός αλγορίθμου αναζήτησης διαστήματος.

Υπάρχει ένας επιπόλαιος-προσαρμοζόμενος αλγόριθμος αναζήτησης διαστήματος Α, ο οποίος εκτελεί μια καθιερωμένη δυαδική αναζήτηση στο δέντρο αναζήτησης η οποία περιγράφεται πιο πάνω, που χρησιμοποιώντας αξιόπιστες μεταβλητές σε κάθε κόμβο, οδηγεί την αναζήτηση. Εδώ πρέπει να σημειώσουμε ότι ο αλγόριθμος Α διαβάζει κάθε ανθεκτική μεταβλητή το πολύ μια φόρα και ο χρόνος που σπαταλιέται είναι O(δt')=O(δlogn).

Εφαρμόζοντας την ερμηνεία του Λήμματος 2 στους αλγορίθμους Α και C,εκτελούμε τον επιθυμητό αλγόριθμο αναζήτησης R, σε προσδοκώμενο χρόνο O(t'+t")=O(logn+δ).

5.4 ΘΕΩΡΗΜΑ 9ο

ΘΕΩΡΗΜΑ 9ο : Υπάρχει μια ανθεκτική δομή λεξικού που παίρνει O(n) χώρο και O(logn+δ) χρόνο ανά λειτουργιά.

Η απόδειξη του θεωρήματος 9 είναι άμεση συνέπεια των Λημμάτων της πηγής [31].

ΚΕΦΑΛΑΙΟ 6ο ΝΤΕΤΕΡΜΙΝΙΣΤΙΚΗ ΑΝΑΖΗΤΗΣΗ ΔΙΑΣΤΗΜΑΤΟΣ

6.1.ΠΕΡΙΓΡΑΦΗ ΚΕΦΑΛΑΙΟΥ

Στο κεφάλαιο αυτό θα μελετήσουμε αναλυτικότερα έναν ντετερμινιστικό αλγόριθμο αναζήτησης διαστήματος, του οποίου ο χρόνος εκτέλεσης είναι $O(\log n+\delta+\alpha\delta^{\varepsilon})= O(\log n+\delta^{1+\varepsilon})$, όπου ε>0 μια σταθερή αυθαίρετη παράμετρος. Το αποτέλεσμα αυτό σε συνδυασμό με τα αποτελέσματα των ενεργειών από το δέντρο διαστημάτων, δημιουργείται μια ανθεκτική δομή λεξικού με $O(\log n+\delta^{1+\varepsilon})$ χειρότερη περίπτωση χρόνου ανά λειτουργιά.

Όπως και στην περίπτωση του τυχαίου αλγορίθμου, αρχίζουμε ,εισάγοντας μια απλή αλλά χρήσιμη βελτίωση. Έστω ο A' ένας μη-ανθεκτικός ντετερμινιστικός αλγόριθμος με χρόνο εκτέλεσης O(t'), ο A είναι ο αντίστοιχος επιπόλαιος-ανθεκτικός αλγόριθμος με χρόνο εκτέλεσης O(δt') και ο C είναι ένας ελαστικός αλγόριθμος με χρόνο εκτέλεσης O(t''),ο οποίος ελέγχει την εγκυρότητα της εξόδου. Διαφορετικά, από την τυχαία περίπτωση, δε θα υπήρχε κανένας επιπλέον περιορισμός στον A'. Επίσης, ο A' μπορεί να διαβάσει την ίδια μεταβλητή περισσότερες από μια φορές.

Θα περιγράψουμε τώρα έναν ανθεκτικό ντετερμινιστικό αλγόριθμο D,ο οποίος εκτελεί την ίδια λειτουργιά με τον A, αλλά σε O((1+α)(t'+t'')) χρόνο χειρότερης περίπτωσης. Όπου α είναι ο αριθμός τον λαθών που έχουν γίνει μέχρι τη συγκεκριμένη στιγμή. Ο αλγόριθμος D, αποτελείται από μερικούς γύρους. Στο γύρο p,με p=1,2,...,2δ+1, ο αλγόριθμος D τρέχει τον A αλλά λαμβάνει υπόψη μόνο το p-οστό αντίγραφο της κάθε ανθεκτικής μεταβλητής του περιλαμβάνεται στους υπολογισμούς. Στο τέλος του γύρου ο D ελέγχει την εγκυρότητα της εξόδου με την χρήση του C. Αν ο έλεγχος αυτό αποτύχει ,ο D ξεκινά την εκτέλεση ενός νέου γύρου, έχοντας αυξήσει το p. Σε αντίθετη περίπτωση ο D βγάζει την απάντηση που υπολογίστηκε στον πρώτο γύρο.

6.2 ΛΗΜΜΑ 9ο

ΛΗΜΜΑ 9ο :

Ο αλγόριθμος D,που περιγράφεται από πάνω, λύνει το ίδιο πρόβλημα με τον A σε χρόνο χειρότερης περίπτωσης O((1+α)(t'+t'')),όπου A ο αλγόριθμος που έχει χρόνο εκτέλεσης O(t') και η εγκυρότητα της εξόδου μπορεί να ελεγχτεί σε χρόνο O(t'').

ΑΠΟΔΕΙΞΗ:

Η ορθότητα του D εξαρτάται άμεσα από την ορθότητα των A και C.Σε περίπτωση που σε έναν δοσμένο γύρο p,όλα τα αντίγραφα που μελετούνται έχουν λανθασμένη τιμή, ο D πρέπει να συμπεριφέρεται σαν τον A .Επίσης, ο D

πρέπει να τερματίσει με τη σωστή έξοδο. Σε περίπτωση που δεν έχει το σωστό αποτέλεσμα θα πρέπει να επιβαρύνουμε με το O(t'+t'') κόστος του κάθε γύρου την κάθε p-οστή λανθασμένη επανάληψη. Για να αποτύχει ένας γύρος του αλγορίθμου θα πρέπει να υπάρχει τουλάχιστον ένα λανθασμένο αντίγραφο Σημειώστε ότι το κάθε λανθασμένο αντίγραφο μπορεί να επιβαρυνθεί το πολύ μια φόρα. Έτσι ξέρουμε ότι μπορεί να υπάρξουν το πολύ (α+1) γύροι, οι οποίοι δίνουν την αναφερόμενη πολυπλοκότητα χρόνου.

6.3 ΜΙΑ O(logn+δ^2) ΥΛΟΠΟΙΗΣΗ

Στην αρχή της ενότητας αυτής θα παρουσιάσουμε μια διαδικασία αναζήτησης διαστήματος με O(logn+δ+ αδ)=O(logn+δ^2) βασισμένη στην βελτιστοποίηση που περιγράφτηκε παραπάνω. Αυτό θα μας βοηθήσει να το έχουμε σαν βάση στην δικιά μας πιο εκλεπτυσμένη και εξελιγμένη διαδικασία. Έστω ότι το p είναι μια ακέραια μεταβλητή αποθηκευμένη σε ασφαλή θέση μνήμης. Αρχικοποιούμε το p στην τιμή ένα (p=1). Η αναζήτηση εξελίσσεται σε γύρους. Στην αρχή του κάθε γύρου, δίνουμε έναν κόμβο u, ως σημείο αναφοράς, έτσι ώστε το κ να ανήκει στο διάστημα U(u) ελαστικά, δηλαδή να είναι μια ανθεκτική τιμή η όποια θα υπάρχει στην πλειοψηφία των αντίγραφων του διαστήματος U(u). Αυτό απαιτεί το I(κ) να περιέχεται στο υποδέντρο το οποίο έχει ως ρίζα τον κόμβο u.Το σημείο αναφοράς u είναι αποθηκευμένο σε ασφαλή θέση μνήμης. Στην αρχή της διαδικασίας το u είναι η ρίζα του δέντρου αναζήτησης, για το οποίο ισχύει U(u)=(-∞,+∞). Στον κάθε γύρο εκτελούμε δ, ίδια κάθε φόρα βήματα αναζήτησης. Αρχίζοντας από τον κόμβο w,σταματούμε την αναζήτηση ή την κατευθύνουμε στο αριστερό ή στο δεξί παιδί w'ή w,αναλόγως της τιμής του I(w).Σε κάθε τέτοιο βήμα, εξετάζουμε το p-οστό αντίγραφο της εκάστοτε σχετικής ελαστικής μεταβλητής (άκρα διαστημάτων και δείκτες) μόνο.

Στο τέλος του κάθε γύρου παρατηρούμε αν το κ ανήκει στο διάστημα U(u') το οποίο είναι υποσύνολο του U(u),όπου u' είναι ο τελικός κόμβος. Αν ο έλεγχος αυτός αποτύχει αυξάνουμε το p και ξανά ξεκινάμε τον γύρο από το σημείο του u που προκλήθηκε το λάθος. Την ίδια ενέργεια κάνουμε και σε περίπτωση που εντοπιστεί οποιαδήποτε αστοχία. Τέτοιου είδους αστοχίες μπορεί να είναι για παράδειγμα ένας δείκτης που χρησιμοποιείται να μην δείχνει σε κάποιο κόμβο του δέντρου ή το διάστημα U(w') να μην είναι υποσύνολο του U(w)για ένα παιδί w' του w.Σε κάθε άλλη περίπτωση, παρατηρούμε ελαστικά αν το κ ανήκει I(u').Αν όντως ανήκει, επιστρέφουμε το I(u'). Αλλιώς ξεκινάμε έναν καινούριο γύρο από το u', το οποίο τώρα πλέον έχει γίνει το καινούριο μας σημείο αναφοράς.

Προτού αναλύσουμε την από πάνω διαδικασία χρειαζόμαστε μερικές ακόμα σημειώσεις σχετικά με αυτή. Ας χαρακτηρίσουμε έναν γύρο ως:

-ανεπιτυχή, σε περίπτωση που μια αστοχία έχει ανακαλυφθεί, ή κατά τη διάρκεια της αναζήτησης ή στο τελικό έλεγχο συνέπειας

-φαινομενικά επιτυχημένο, σε περίπτωση που ο τελικός έλεγχος συνέπειας είναι επιτυχημένος και το κ ∉I(u') και ο κόμβος u' είναι λιγότερο από δ/2 επίπεδα κάτω από τον u στο δέντρο αναζήτησης

-πραγματικά επιτυχημένο σε περίπτωση που ο τελικός έλεγχος συνέπειας είναι επιτυχημένος και το κ ϵ I(u') ή ο κόμβος τερματισμού u'είναι τουλάχιστον δ/2 επίπεδα κάτω από τον u στο δέντρο αναζήτησης

Σ αυτό το σημείο πρέπει να σημειώσουμε ότι ο αλγόριθμος αυτός δε μπορεί να κάνει διάκριση ανάμεσα στον πραγματικά επιτυχημένο γύρο και στον φαινομενικά επιτυχημένο. Όμως, η διάκριση αυτή αποδεικνύεται χρήσιμη στην ανάλυση.

6.4 ΛΗΜΜΑ 10ο

ΛΗΜΜΑ 10ο: Η διαδικασία αναζήτησης διαστήματος που αναλύθηκε παραπάνω έχει ντετερμινιστικό χρόνο εκτέλεσης ίσο με O(logn+δ+αδ)

Απόδειξη:

Η ορθότητα της διαδικασίας είναι τετριμμένη. Τώρα αναλύουμε τον χρόνο εκτέλεσης. Ο κάθε γύρος θέλει για να εκτελεστεί O(δ) χρόνο. Εμείς επιβαρυνόμαστε κατά O(δ) χρόνο για τον κάθε μη επιτυχημένο γύρο για καθένα από τα p λανθασμένα αντίγραφα (πρέπει να υπάρχει τουλάχιστον ένα τέτοιο λανθασμένο αντίγραφο). Όπου p είναι η τιμή του ζημιογόνου αριθμού την στιγμή που αρχίζει ο γύρος που μελετάμε. Όπως και στην απόδειξη του Λήμματος 4,κάθε λανθασμένη τιμή εξετάζεται το πολύ μια φόρα. Έτσι το συνολικό κόστος των αποτυχημένων γύρων είναι O(αδ).

Θεωρώντας τώρα, έναν φαινομενικά επιτυχημένο γύρο και θέτοντας $u=(u_1,u_2,...,u_\delta)=u'$ να είναι η ακολουθία των ίσως όχι διαφορετικών κόμβων που επισκεπτόμαστε κατά τη διάρκεια του γύρου. Από τη στιγμή που δεν έχει εντοπιστεί κάποια ασυνέπεια , θα πρέπει να ισχύει $U(u_1)) \supset U(u_2) \supset \ldots \supset U(u_\delta)$ με σεβασμό στο p-οστά αντίγραφα.

Από τον ορισμό του φαινομενικά επιτυχημένου γύρου, προκύπτει ότι τουλάχιστον δ/2 από αυτά τα αντίγραφα πρέπει να έχουν καταστραφεί. Τώρα, παρατηρούμε ότι ο κόμβος u_i και το αντίστοιχο κατεστραμμένο αντίγραφο του $U(u_i)$ μπορεί να θεωρηθεί ως σωστό σε αρκετούς φαινομενικά επιτυχημένους γύρους. Στην πραγματικότητα, το κόστος των κατεστραμμένων δεικτών, ίσως εξαναγκάσει τον αλγόριθμο να οδηγηθεί κυκλικά στο ίδιο σύνολο κόμβων. Παρ όλα αυτά, αφού κανένα λάθος δεν έχει εντοπιστεί κατά τη διάρκεια του γύρου, η κατεστραμμένη τιμή του $U(u_i)$ πρέπει να τροποποιείται σε κάθε γύρο. Επομένως, όπως είναι φυσικό αυτό μειώνει κατά ένα τον προϋπολογισμό των λαθών. Αυτό έχει ως συνέπεια να μπορούν να υπάρχουν το πολύ δύο συνεχόμενοι επιτυχημένοι γύροι. Έτσι επιβαρύνουμε το προσδοκώμενο κόστος O(δ) στον επόμενο πραγματικά επιτυχημένο γύρο. Σημειώστε ότι ένας πραγματικά επιτυχημένος γύρος θα πρέπει να υπάρχει για να είναι σωστός ο αλγόριθμος.

Το πραγματικό κόστος ,των πραγματικά επιτυχημένων γύρων είναι O(logn+δ). Στην πραγματικότητα, μπορεί να υπάρχουν το πολύ 2logn/δ τέτοιοι γύροι, από τους οποίους ο καθένας θα απαιτεί O(δ) χρόνο. Μέσα σ' αυτό το

χρόνο συμπεριλαμβάνεται και το επιπλέον κόστος που προστέθηκε από τους φαινομενικά επιτυχημένους γύρους. Συμπερασματικά, παρατηρούμε ότι ο συνολικός χρόνος εκτέλεσης της διαδικασίας είναι O(logn+δ+ αδ).

6.5 ΜΙΑ ΙΕΡΑΡΧΕΙΑ ΖΗΜΙΟΓΩΝΩΝ ΑΡΙΘΜΩΝ

Μπορούμε να βελτιώσουμε τον χρόνο εκτέλεσης με τη χρήση ενός συνόλου ζημιογόνων αριθμών. Για λόγους ευκολίας, δεν λαμβάνουμε υπόψη το πάνω και το κάτω όριο της δομής. Για παράδειγμα, θεωρούμε το ριζά δ ακέραιο αριθμό. Χωρίζουμε ομοιόμορφα τα (2δ+1) αντίγραφα της κάθε ανθεκτικής μεταβλητής σε ριζά δ υπο-διαστήματα. Παρατηρούμε ότι κάθε διάστημα περιέχει περίπου 2 (√δ)= Θ(√δ) κλειδιά. Αναλογικά χωρίζουμε τον κάθε γύρο ,ο οποίος αποτελείται από δ βήματα αναζήτησης, σε ριζά δ μικρότερου γύρους με Θ(√δ) βήματα αναζήτησης ο καθένας. Στη συνεχεία ορίζουμε δυο ζημιογόνους αριθμούς p0 και p1 ,οι οποίοι αποθηκεύονται σε ασφαλείς θέσεις μνήμης. Θεωρούμε ότι τα p0 και p1 δείχνουν στο πρώτο υποδιάστημα και στο πρώτο αντίγραφο αντίστοιχα. Αυτά τα δυο είναι αυτά που μπορούμε να εμπιστευτούμε καθώς ανήκουν σε αξιόπιστες θέσεις μνήμης. Στο τέλος του κάθε υπό-κύκλου, πραγματοποιούμε έναν έλεγχο βασισμένο στο p1-οστο υποδιάστημα μόνο. Πραγματοποιούμε συνολικά Θ(√δ) ελέγχους. Σημειώνουμε σ' αυτό το σημείο ότι οι Θ(√δ) έλεγχοι είναι φθηνότεροι αλλά λιγότερο αξιόπιστοι. Με την καταστροφή Ο(√δ) τιμών ο αντίπαλος, μπορεί να οδηγήσει σε λανθασμένα αποτελέσματα. Σε κάθε περίπτωση, αν οι Θ(√δ) έλεγχοι αποτύχουν, γυρίζουμε πίσω στον κόμβο εκκίνησης u1 του υπό-γύρου, βασισμένοι στο γεγονός ότι αυτός είναι αποθηκευμένος σε ασφαλή θέση μνήμης. Σε αυτή τη περίπτωση αυξάνουμε και το p0. Όταν p0≥ √δ/2, αυξάνουμε το p1 και αρχικοποιούμε το p0 στο ένα. Γυρίζουμε πίσω στο u1 και αυξάνουμε το p0 οποτεδήποτε παρατηρείται μια ασυνέπεια κατά τη διάρκεια κάθε βήματος αναζήτησης του υπο-γύρου που μελετάμε. Ο αλγόριθμος εξελίσσεται όπως προηγουμένως. Στο τέλος του κάθε γύρου εκτελούμε έναν αξιόπιστο έλεγχο συνέπειας σε όλα τα 2δ+1 αντίγραφα (Θ(δ)-έλεγχοι). Αν ο έλεγχος αποτύχει, επιστρέφουμε στο αντίστοιχο σημείο αναφοράς u2,το οποίο είναι αποθηκευμένο σε ασφαλή μνήμη. Εδώ, πρέπει να τονίσουμε ότι πριν από τον καθένα από τους Θ(δ) ελέγχους ,θα πρέπει να υπάρχουν Θ(√δ) επιτυχημένοι έλεγχοι στο ίδιο σύνολο αξιόπιστων μεταβλητών. Σε κάθε άλλη περίπτωση ο αλγόριθμος επιστρέφει προτού εκτελέσει τους Θ(δ) ελέγχους.

Ισχυριζόμαστε ότι ο παραπάνω αλγόριθμος εκτελείται σε O(logn+δ+α*√δ). Στην πραγματικότητα, θεωρούμε ένα υποδιάστημα I. Το συνολικό κόστος της επιστροφής του αλγορίθμου στα υπο-διαστήματα περιέχει τουλάχιστον Ο(√δ) λανθασμένα αντίγραφα. Ο συνολικός αριθμός των υποδιάστημα που εξετάζονται στο I είναι Ο(√δ)*Ο(√δ)=Ο(δ). Από τη στιγμή που το κάθε υποδιάστημα που απορρίπτεται περιέχει τουλάχιστον Ω(√δ) λανθασμένα αντίγραφα, ο συνολικός αριθμός των υποδιάστημα που απορρίπτονται είναι Ο(α/√δ+1).Ετσι το συνολικό κόστος του γυρίσματος του αλγορίθμου στους υπο-γύρους είναι Ο(α*√δ+δ). Ας υπολογίσουμε τώρα το υπολειπόμενο κόστος μετά τον υπολογισμό του κόστους των υπαναχωρήσεων στους επιμέρους κύκλους. Κάθε γύρος κοστίζει μόνο Ο(δ). Θεωρώντας κάθε μη επιτυχημένο γύρο, όπου με μη επιτυχημένο εννοούμε ότι οι αντίστοιχοι Θ(δ) έλεγχοι είναι αποτυχημένοι. Κάθε άλλος τύπος αποτυχίας έχει ήδη υπολογιστεί στο κόστος των υπαναχωρήσεων των επιμέρους κύκλων. Από

τη στιγμή που εκτελείται ο αναφερόμενος έλεγχος, πρέπει να υπάρχουν Θ(√δ) επιτυχημένοι έλεγχοι οι όποιοι επέστρεψαν λανθασμένο αποτέλεσμα. Το αντίστοιχο υποδιάστημα το οποίο περιέχει τουλάχιστον Ω(√δ) λάθη, απορρίπτεται. Από αυτό μπορούμε να συμπεράνουμε ότι υπάρχουν το πολύ O(α/√δ) μη επιτυχημένοι κύκλοι ,με συνολικό κόστος O(α/√δ)*O(δ)= O(α√δ). Μπορούμε να οριοθετήσουμε το κόστος των κύκλων που απομένουν όπως στο ΛΗΜΜΑ 10.Ειναι πολύ σημαντικό το γεγονός ότι μπορούμε, το κόστος του κάθε φαινομενικά επιτυχημένου κύκλο να επιβαρύνει τον επόμενο πραγματικά επιτυχημένο κύκλο. Υπενθυμίζουμε εδώ ότι όπως εξηγήσαμε πιο νωρίς δε μπορεί να υπάρξουν περισσότεροι από δύο συνεχόμενοι φαινομενικά επιτυχημένοι κύκλοι. Έτσι, κάθε επιτυχημένος κύκλος επιβαρύνεται με O(δ). Μπορεί να υπάρξουν το πολύ 2logn/δ πραγματικά επιτυχημένοι κύκλοι και έτσι το συνολικό τους κόστος είναι O(logn+δ). Συμπεραίνουμε συνεπώς ότι συνυπολογίζοντας τα επιμέρους κόστη, καταλήγουμε στο συνολικό κόστος της διαδικασίας που είναι O(logn+δ+ α*√δ).

Μπορούμε να εξασφαλίσουμε έναν καλύτερο χρόνο εκτέλεσης από την περαιτέρω επέκταση της ιεραρχίας των ζημιογόνων αριθμών. Έστω ε<1/2 να είναι μια οποιαδήποτε θετική σταθερά και κ=1/ε=O(1) .Διαχωρίζουμε ομοιόμορφα τα (2δ+1) αντίγραφα της κάθε ανθεκτικής μεταβλητής σε $\delta^{1/\kappa}$ υποδιαστήματα με μέγεθος $\Theta(\delta^{\kappa-1)/\kappa})$.Στη συνεχεία επαναλαμβάνουμε τη διαδικασία στα υποδιαστήματα έως ότου φτάσουμε σε διάστημα με μέγεθος $\Theta(\delta^{1+\kappa})$.Με αυτό τον τρόπο, μπορούμε να δημιουργήσουμε μια ιεραρχία από (κ-1) επίπεδα διαστημάτων οπού τα διαστήματα του επίπεδου I ,με i=1,2,..,k-1 έχουν μήκος $\Theta(\delta^{i/\kappa})$. Επίσης, διαχωρίζουμε ανάλογα με πριν τον κάθε γύρο σε $\delta^{1/\kappa}$ επιμέρους κύκλους, ο καθένας από του οποίους έχει $\Theta(\delta^{\kappa-1/\kappa})$ βήματα αναζήτησης. Επαναλαμβάνουμε τη διαδικασία στους επιμέρους κύκλους μέχρι να φτάσουμε σε επιμέρους κύκλους με $\Theta(\delta^{1/\kappa})$ βήματα αναζήτησης. Ακόμη, σε αυτήν την περίπτωση αποκτούμε μια ιεραρχία από κ-1 επίπεδα γύρων, όπου υπάρχουν $\Theta(\delta^{(\kappa-i)/\kappa}$) κύκλοι του επίπεδου I ,με i=1,2,..κ-1,το καθένα από τα όποια αποτελείται από $\Theta(\delta^{i/\kappa})$ βήματα αναζήτησης. Παρατηρούμε επίσης ότι κάθε γύρος από τα επίπεδα για i≥ 2 αποτελούνται από $\Theta(\delta^{(i-j)/k}$) κύκλους από το επίπεδο j,1≤j<i.

Αντιστοιχούμε έναν ζημιογόνο αριθμό p_i σε κάδε επίπεδο διαστημάτων με I να ανήκει {1,2,..,κ-1} και ορίζουμε έναν επιπλέον αριθμό p0.Η λειτουργιά των ζημιογόνων αριθμών είναι όπως νωρίτερα. Επιπλέον, το πρώτο διάστημα $I_{\kappa-1}$ του επίπεδου κ-1,το οποίο μπορούμε ακόμη να εμπιστευτούμε είναι το $p_{\kappa-1}$-οστό.Το πρώτο διάστημα $I_{\kappa-2}$ του επίπεδου κ-2 το οποίο εμπιστευόμαστε ακόμα είναι το $p_{\kappa-2}$-οστό υποδιάστημα του $I_{\kappa-1}$.Παρομοιως μπορούμε να ορίσουμε το πρώτο διάστημα I_i του επίπεδου I που ακόμα εμπιστευόμαστε , i=1,2,...,κ-3.Τελικα το πρώτο αντίγραφο του I_1 που εμπιστευόμαστε είναι το p0-οστο. Στο τέλος του κάθε γύρου του επίπεδου i εκτελούμε τον έλεγχο συνέπειας στα υποδιαστήματα I_i. Εκτελούμε συνεπώς $\Theta(\delta^{i/k})$ ελέγχους. Επίσης, διατηρούμε έναν κόμβο u_i,ως σημείο αναφοράς για το κάθε ι. Υπαναχωρούμε από τον τρέχοντα κόμβο στην αναζήτηση στο u_i οποτεδήποτε ένας $\Theta(\delta^{i/k})$ έλεγχος αποτύχει. Σ αυτήν την περίπτωση αυξάνουμε και το p_{i-1}.Καθως το $p_{i-1}\geq(\delta^{1/\kappa})/2$, αγνοούμε το διάστημα I_i.Δηλαδή αυξάνουμε το p_i και αρχικοποιούμε το p_j στην τιμή ένα για κάθε j<i.Ακόμη υπαναχωρούμε στο u_1 και αυξάνουμε το p_0 οποτεδήποτε βρούμε μια ασυνέπεια κατά τη διάρκεια κάποιου βήματος αναζήτησης. Το υπόλοιπο του αλγορίθμου είναι όπως περιγράψαμε και παραπάνω. Επίσης, κάθε δ βήματα αναζήτησης εκτελούμε έναν αξιόπιστο έλεγχο συνέπειας και υπαναχωρούμε στο αντίστοιχο u_κ σημείο αναφοράς σε περίπτωση που ο έλεγχος αποτύχει. Όλα τα u_i σημεία ελέγχου και όλοι οι ζημιογόνοι αριθμοί p_i κρατούνται σε ασφαλείς θέσεις

μνήμης. Τονίζουμε ότι είναι πιθανό να είναι συνεχεία αρκετά. Ακόμη, τονίζουμε ότι κάθε $\Theta(\delta^{i/k})$ έλεγχος με $2 \leq i \leq k$ ακολουθητέοι από $\Theta(\delta^{i-1/k})$ επιτυχημένους ελέγχους στο ίδιο σύνολο από αξιόπιστες μεταβλητές.

6.6 ΛΗΜΜΑ 11ο

ΛΗΜΜΑ 11ο :Για κάθε ε>0 η παραπάνω διαδικασία αναζήτησης διαστήματος έχει ντετερμινιστικό χρόνο εκτέλεσης $O(\log n+\delta+ \alpha\delta^{\varepsilon})$.

ΑΠΟΔΕΙΞΗ:

Θεωρούμε το κόστος των υπαναχωρήσεων στους κύκλους του επίπεδου ένα. Κάθε υπαναχώρηση κοστίζει $O(\delta^{1/k})$ και υπάρχουν το πολύ $\delta^{1/k}/2$ υπαναχωρήσεις ανά διάστημα. Το κόστος των υπαναχωρήσεων σε ένα δοσμένο διάστημα I1 του επίπεδου 1 είναι $O(\delta^{2/k})$.Καθε διάστημα I_1 που απορρίπτεται του επίπεδου 1 περιέχει τουλάχιστον $\Omega(\delta^{1/k})$ λανθασμένα αντίγραφα. Έτσι μπορούμε να θεωρήσουμε το πολύ $O(\alpha/\delta^{1/k} +1)$ διαστήματα του επίπεδου 1,με συνολικό κόστος υπαναχωρήσεων $O(\alpha\delta^{1/k} + \delta^{2/k})$.

Θεωρούμε τώρα το κόστος των υπαναχωρήσεων στους επιμέρους κύκλους του επίπεδου i, με i=2,3,...,k-1, αγνοώντας το κόστος των υπαναχωρήσεων στα χαμηλότερα επίπεδα .Κάθε υπαναχώρηση κοστίζει $O(i\delta^{i/k})$. Στην πραγματικότητα οφείλουμε να λάβουμε υπόψη $\Theta(\delta^{i/k})$ βήματα αναζήτησης, έναν $\Theta(\delta^{i/k})$ έλεγχο και όλους τους επιτυχημένους ελέγχους που αντιστοιχούν στα επίπεδα j<i τα όποια δεν λαμβάνονται υπόψη στο κόστος υπαναχωρήσεων των χαμηλότερων επίπεδων. Για κάθε j,υπάρχουν $\Theta(\delta^{(i-j)/k})$ περισσότεροι έλεγχοι από $\Theta(\delta^{i/k})$ αυτού του τύπου. Κλείνοντας, το συνολικό κόστος είναι $O(\delta^{i/k})+\sum_{j=1}^{i-1} O(\delta^{(i-j)/k})*O(\delta^{j/k})=O(i\delta^{i/k})$.

Υποθέτουμε ότι κάθε διάστημα I_i του επίπεδου i. Μπορούμε να υπαναχωρήσουμε στο I_i το πολύ $\delta^{1/k}/2$ φόρες, με συνολικό κόστος $O(i\delta^{i/k})*O(\delta^{1/k})=O(i\delta^{i+1/k})$. Υποθέτουμε ότι το I_i έχει καταστραφεί. Αν ένας από τους $\Theta(\delta^{i/k})$ ελέγχους στο I_i ήταν λανθασμένος θα πρέπει να υπάρχουν $\Omega(\delta^{i/k})$ λανθασμένα αντίγραφα στον I_i.Διαφορετικά, θα πρέπει να ήταν λάθος ένας έλεγχος συνέπειας στον κάθε ένα από τα πρώτα $\delta^{1/k}/2$ υποδιαστήματα του I_i ,του επίπεδου (i-1).Επισης, σ' αυτήν την περίπτωση Ii πρέπει να περιέχει $\Omega(\delta^{i-1/k}*\Omega(\delta^{1/k})=\Omega(\delta^{i/k})$ λανθασμένες τιμές. Μπορούμε λοιπόν να συμπεράνουμε ότι ο συνολικός αριθμός των υποδιάστημα του επίπεδου i που αγνοείται από τον αλγόριθμο είναι $O(\alpha/\delta^{i/k+1})$.Το συνολικό, λοιπόν, κόστος των υπαναχωρήσεων στους επιμέρους κύκλους του επίπεδου ι είναι $O(\alpha/\delta^{i/k}+1)*O(i\delta^{(i+1)/k}) = O(\alpha i\delta^{1/k} + i\delta^{i+1)/k})$.

Συνολικά, το ολοκληρωτικό κόστος των υπαναχωρήσεων στους υποκύκλους είναι $\sum_{i=1}^{k-1} O(i\alpha\delta^{1/k}+i\delta^{(i+1)/k}) =O(k^2(\alpha\delta^{1/k}+\delta)) =O (\alpha\delta^{\varepsilon}+ \delta)$. Ας υπολογίσουμε τώρα το υπόλοιπο κόστος, αφού εξαιρέσουμε το κόστος των υπαναχωρήσεων στους επιμέρους κύκλους. Με τον ίδιο τύπο ορισμάτων, με πάνω, ο κάθε γύρος κοστίζει $O(\delta)+ \sum_{j=1}^{k-1} O(\delta^{(k-j)/k})*O(\delta^{j/k})=O(k\delta)$. Θεωρούμε κάθε δοσμένο μη αξιόπιστο κύκλο, οπού λέγοντας μη αξιόπιστο εννοούμε ότι οι αναμενόμενοι $\Theta(\delta)$ έλεγχοι θα αποτύχουν. Μέχρι να εκτελέσουμε τον επόμενο έλεγχο, ο όποιος είναι αξιόπιστος ,θα πρέπει να υπάρχει ένας επιτυχημένος $\Theta(\delta^{(k-1)/k})$ έλεγχος, ο όποιος θα απαντήσει λανθασμένα. Τα αντίστοιχα διαστήματα

του επίπεδου κ-1 θα πρέπει να περιέχουν τουλάχιστον $\Omega(\delta^{\kappa-1/\kappa})$ λανθασμένα αντίγραφα τα όποια δεν λαμβάνονται υπόψη. Από αυτό μπορούμε να συμπεράνουμε ότι υπάρχουν το πολύ $O(\alpha/\delta^{(\kappa-1)/\kappa})$ άχρηστοι κύκλοι με συνολικό κόστος $O(\alpha/\delta^{(\kappa-1)/\kappa})*O(\kappa\delta)= O(\kappa\alpha\delta^{1/\kappa})= O(\alpha\delta^{\varepsilon})$. Μπορούμε να περιορίσουμε το κόστος των απομεινάντων κύκλων όπως στο Λήμμα 5. Αυξάνουμε το κόστος των φαινομενικά επιτυχημένων γύρων, όχι παραπάνω από δύο διαδοχικούς, στον επόμενο πραγματικά επιτυχημένο γύρο. Μπορεί να υπάρχουν το πολύ 2logn/δ πραγματικά επιτυχημένοι γύροι και το συνολικό κόστος τους είναι O(logn+δ). Ακολουθεί ο ισχυριζόμενος χρόνος εκτέλεσης.

6.7 ΘΕΩΡΗΜΑ 10ο

ΘΕΩΡΗΜΑ 10ο :

Για κάθε σταθερά ε>0,υπάρχει μια ανθεκτική δομή λεξικού η όποια απαιτεί O(n) χώρο και $O(\log n+\delta+ \alpha\delta^{\varepsilon})= O(\log n+\delta^{1+\varepsilon})$ χρόνο για κάθε λειτουργιά.

ΑΠΟΔΕΙΞΗ:

Η απόδειξη προκύπτει άμεσα από τα Λήμματα 1 και 6.

Μπορούμε να πάρουμε καλύτερο αποτέλεσμα μελετώντας μια ακολουθία από λειτουργιές.

ΠΟΡΙΣΜΑ: Για κάθε σταθερά ε>0,υπάρχει μια ανθεκτική δομή λεξικού που λαμβάνει O(n) χώρο και $O(\sigma(\log n+\delta)+\delta^{1+\varepsilon})$ χρόνο σε κάθε ακολουθία από σ λειτουργιές.

ΑΠΟΔΕΙΞΗ:

Θεωρούμε την απόδειξη του Λήμματος 6.Η τιμή $O(\alpha\delta^{\varepsilon})$ στην πολυπλοκότητα του χρόνου μπορεί να προσαρμοστεί σε μια ακολουθία από αναζήτησης διαστημάτων. Είναι εφικτό να διατηρήσουμε τους ίδιους ζημιογόνους αριθμούς από τη μια αναζήτηση στην επομένη. Έτσι μια ακολουθία από σ αναζητήσεις διαστημάτων κοστίζουν $\sigma S(n,\delta)+O(\alpha\delta^{\varepsilon})= O(\sigma(\log n+\delta)+\alpha\delta^{\varepsilon})$. Από την απόδειξη του Λήμματος 1, κάθε ακολουθία από σ λειτουργιές κοστίζει $O(\sigma(\log n+\delta))$, περιλαμβάνοντας το κόστος των αναζητήσεων διαστημάτων. Αυτό έχει ως αποτέλεσμα ,για $\sigma\geq\delta^{\varepsilon}$ λειτουργιές, το κόστος σε χρόνο να είναι O(logn+δ) ανά λειτουργιά. Υπό αυτήν την υπόθεση το ντετερμινιστικό αποτέλεσμα αυτής της ενότητας ισούται με το τυχαίο αποτέλεσμα του αλγορίθμου που αναλύθηκε σε προηγούμενη παράγραφο.

ΚΕΦΑΛΑΙΟ 7ο ΕΝΑ ΚΑΤΩ ΟΡΙΟ

Σ' αυτήν την ενότητα θα δείξουμε ότι, σ' αυτό το μοντέλο που στηρίζεται στις συγκρίσεις, κάθε ανθεκτική δομή λεξικού απαιτεί Ω(logn+δ) χρόνο για κάθε αναζήτηση, με δ= O(n). Τονίζουμε ότι για n=o(δ), υπάρχει μια ασήμαντη δομή λεξικού με χρόνο εκτέλεσης O(n)=o(δ). Είναι εφικτό να διατηρήσουμε τα κλειδιά σε έναν πινάκα, του οποίου η διεύθυνση και το μήκος είναι αποθηκευμένα σε ασφαλείς θέσεις μνήμης.

7.1 ΘΕΩΡΗΜΑ 11ο

ΘΕΩΡΗΜΑ 11ο:

Για δ=O(n),κάθε ανθεκτική δομή λεξικού βασισμένη σε συγκρίσεις απαιτεί Ω(logn+δ) χρόνο ανά αναζήτηση.

ΑΠΟΔΕΙΞΗ:

Κάθε δομή λεξικού βασισμένοι σε συγκρίσεις, ακόμα και σε σύστημα χωρίς λάθη μνήμης, λαμβάνει Ω(logn) χρόνο ανά αναζήτηση. Αυτό το κάτω όριο επεκτείνεται άμεσα και στην περίπτωση των ανθεκτικών δομών λεξικών καθώς κάθε ανθεκτική διαδικασία μπορεί να προσαρμοστεί σε ένα ασφαλές σύστημα μνήμης χωρίς απώλειες από γενικότητα. Μας επιτρέπεται να υποθέσουμε ότι logn=o(δ). Με αυτήν την υπόθεση είναι εφικτό να δείξουμε ότι ο χρόνος που απαιτείται για μια ανθεκτική λειτουργιά αναζήτησης είναι Ω(δ).

Ας ονομάσουμε ΑΛ κάθε ανθεκτική δομή λεξικού. Θεωρούμε αρχικά την περίπτωση που η ΑΛ είναι ντετερμινιστική και υποθέτουμε ακόμη από αντίθεση ότι η ΑΛ εκτελεί κάθε λειτουργιά αναζήτησης σε ο(δ) χρόνο στην χειρότερη περίπτωση. Τώρα ας περιορίσουμε την προσοχή μας στο ακόλουθο σενάριο. Αρχικά εισάγουμε n διακριτά κλειδιά και υποθέτουμε ότι δεν προκαλούνται λάθη κατά τη διάρκεια της εισαγωγής τους. Στη συνεχεία εκτελούμε μια λειτουργιά αναζήτησης, με κλειδί αναζήτησης κ. Ακολούθως, δείχνουμε ότι υπάρχει επιλογή του κ η όποια κάνει την ΑΛ να επιστρέφει λανθασμένο αποτέλεσμα το οποίο δηλαδή καταλήγει σε άτοπο.

Έστω S,|S|=O(1), να είναι τα κλειδιά που βρίσκονται σε ασφαλή μνήμη και U,|U|=Θ(n), τα κλειδιά που είναι αποθηκευμένα σε μη ασφαλείς θέσεις μνήμης. Υποθέτουμε ότι το κλειδί αναζήτησης κ ανήκει στο U. Θεωρούμε την πρώτη θέση μνήμης α_1 σε μη ασφαλής θέση μνήμης να διαβάζεται από την ΑΛ.Η θέση μνήμης α_1 μπορεί να ερμηνευτεί σαν φυσική διεύθυνση μνήμης. Τα κλειδιά του συνόλου S χωρίζουν τα κλειδιά του συνόλου U σε |S|+1 το πολύ ισοδύναμες κλάσεις με συγκρίσεις με τα κλειδιά του S. Από αυτό συνεπάγεται ότι η ΑΛ μπορεί να γενικευτεί σε το πολύ |S|+1=O(1) διαφορετικές τιμές του α_1,μια για κάθε κλάση. Ας υποθέσουμε από τώρα ότι το κλειδί κ ανήκει στην μεγαλύτερη από αυτές τις κλάσεις, έστω U'.Σημειώνουμε ότι |U'|=Θ(n).Προτού η ΑΛ διαβάσει την α_1,ο "αντίπαλος" θέτει το περιεχόμενο της α_1 την ίδια σταθερή, αυθαίρετη τιμή διαφορετική από όλα τα κλειδιά, έστω ότι παίρνει την τιμή μηδέν. Η ίδια ενέργεια επαναλαμβάνεται από τον "αντίπαλο" για όλες της διεύθυνσης μνήμης

[($\alpha_2,\alpha_3,...,\alpha_{\delta'}$),δ'=ο(δ)] οι οποίες είναι αποθηκευμένες σε μη ασφαλή μνήμη και διαβάζονται από την ΑΛ. Δηλαδή ο “αντίπαλος” πιθανόν να θέτει όλες τις α_i τιμές σε μηδέν στην αρχή της λειτουργιάς αναζήτησης. Στο τέλος της διαδικασίας, η ΑΛ είτε θα εξάγει σαν αποτέλεσμα ΝΑΙ και θα επιστρέψει την τιμή μιας διεύθυνσης μνήμης που περιέχει ένα κλειδί με τιμή κ ή θα επιστρέψει ΟΧΙ. Παρατηρούμε ότι η έξοδος του αλγορίθμου είναι ακριβώς η ίδια για κάθε κ που ανήκει στο U'.Επιπλέον, αν το κ ανήκει στο U' και δεν είναι ένα από τα κλειδιά που αρχικά αποθηκευτήκαν στις θέσεις αi, με i=1,2,...,δ', τότε το κ είναι ψευδές. Στην πραγματικότητα μόνο τα αi καταστρέφονται. Ως εκ τούτου η ΑΛ οδηγείται σε έξοδο ΝΑΙ και επιστρέφει μερικές θέσεις μνήμης α', από την τιμή του κ. Αυτό οδηγεί σε άτοπο, από τη στιγμή που μπορούμε να επιλέξουμε δυο διακριτά κλειδιά κ' και κ΄΄ στο U',που ικανοποιούν τις παραπάνω ιδιότητες. Άδω χρησιμοποιούμε την υπόθεση δ=Ο(n) και έτσι |U'|-δ'=Θ(n)≥2 για αρκετά μεγάλο n.

Η περίπτωση με τυχεότητα είναι ανάλογη. Τώρα υποθέτουμε σαν αντιθεση οτι μερικές ΑΛ εκτελούν κάθε αναζήτηση σε Ο(δ) χρόνο. Στη συνεχεία μπορούμε να χρησιμοποιήσουμε τις ίδιες υποθέσεις με την προηγούμενη διαδικασία ,με μόνη βασική διάφορα να είναι ότι η ακολουθία $\alpha_1,\alpha_2,...,\alpha_{\delta'}$ είναι τυχαία. Παρ όλα αυτά ο “αντίπαλος”, προσομοιώνοντας την διαδικασία αναζήτησης του ΑΛ, μπορεί να γενικεύσει μια τέτοια ακολουθία, η όποια εμφανίζει με μεγάλη πιθανότητα και έτσι δ'=ο(δ). Με το να θέτει τις αντίστοιχες θέσεις μνήμης ίσες με το μηδέν, στην αρχή της αναζήτησης ο “αντίπαλος” κάνει την ΑΛ να αποτύχει με μεγάλη πιθανότητα για μια κατάλληλη επιλογή του κ. Αυτό εναντιώνεται με την ορθότητα της ΑΛ.

Κλείνοντας αξίζει να παρατηρήσουμε ότι η ιδέα από την όποια πρόεκυψε η απόδειξη του θεωρήματος 11,μπορει να προσαρμοστεί για να αποκτήσουμε Ω(δ) κάτω όριο σε ένα ευρύ φάσμα από ανθεκτικούς αλγορίθμους και δομές δεδομένων. Διαισθητικά , το βασικό χαρακτηριστικό που χρειαζόμαστε είναι ότι η έξοδος επηρεάζεται από την κατάσταση της μη ασφαλούς μνήμης. Το γεγονός ότι η ασφαλής μνήμη είχε μέγεθος Ο(1),μοιαζει να είναι μάλλον μια γενική ιδιότητα. Αυτό αποκλείει τις διαδικασίες που απαιτούν ο(δ) χρόνο, οι οποίες πιθανόν να μην είναι ικανές να διαβάσουν κάποια μη κατεστραμμένη θέση μνήμης η όποια βρίσκεται σε μη ασφαλή μνήμη.

ΚΕΦΑΛΑΙΟ 8 ΑΝΑΖΗΤΗΣΕΙΣ ΣΕ ΒΕΛΤΙΣΤΕΣ ΔΟΜΕΣ

Μετά την μελέτη της τυχαίας και ντετερμινιστικής αναζήτησης διαστήματος και τις εισαγωγές και διαγραφές που παρουσιάσαμε ακολούθησαν πολλές μελέτες πάνω σ' αυτά και έτσι οδηγηθήκαμε σε μια ανθεκτική τυχαία στατική δομή λεξικού η όποια υποστηρίζει αναζητήσεις σε βέλτιστο χρόνο με τον αντίπαλο να μην προσαρμόζεται στον αλγόριθμο μας ,να μην είναι δηλαδή δυναμικός και έτσι η καταστροφή των τιμών του πινάκα να προκαλούνται από φυσικές αιτίες όπως κοσμική ακτινοβολία και σωματίδια Α. Ακόμη, θα παρουσιάσουμε και μια ντετερμινιστική ανθεκτική δομή λεξικού. Αυτή σε αντίθεση με την αντίστοιχη περίπτωση που έχει τυχαιότητα, δεν κάνει καμιά υπόθεση σχετικά με τον τρόπο που εκτελούνται οι καταστροφές δεδομένων .

8.1 ΒΕΛΤΙΣΤΗ ΤΥΧΑΙΑ ΑΝΘΕΚΤΙΚΗ ΔΟΜΗ ΛΕΞΙΚΟΥ

Σ' αυτήν την παράγραφο θα μελετήσουμε έναν απλό τυχαίο ανθεκτικό αλγόριθμο αναζήτησης ο οποίος ψάχνει για ένα δοσμένο στοιχειό σε ταξινομημένο πινάκα χρησιμοποιώντας στη χειρότερη περίπτωση O(logδ) τυχαία ψηφιά και προσδοκώμενο χρόνο O(logn+δ), υποθέτοντας ότι οι καταστροφές των στοιχείων γίνονται από έναν μη δυναμικό αντίπαλο [8]. Ο χρόνος εκτέλεσης ταιριάζει με τον αλγόριθμο του Finocchi,που χρησιμοποιεί O(logn*logδ) τυχαία ψηφία. Η βασική ιδέα του αλγορίθμου είναι να διαιρέσει τον ταξινομημένο πινάκα εισόδου σε 2δ κομμάτια καθένα από τα όποια θα αποτελεί μια ταξινομημένη ακολουθία έστω $S_0,S_1,..S_{2δ-1}$ που θα έχουν το πολύ n/2δ μέγεθος. Το j-οστό στοιχειό της Si, που συμβολίζεται $S_i[j]$, είναι το στοιχειό στη θέση $pos_i(j)=2δj+i$ στον πίνακα εισόδου. Διαισθητικά, αυτό χωρίζει τον πίνακα εισόδου σε το πολύ n/2δ συνεχόμενα τμήματα με μέγεθος 2δ στοιχεία, οπού $S_i[j]$ είναι το i-οστό στοιχείο του j-οστού τμήματος. Σημειώνουμε πως ,δεδομένου ότι ορίζονται 2δ ξεχωριστές ακολουθίες από τον πίνακα εισόδου και ότι προκαλούνται δ λάθη, θα υπάρχουν τουλάχιστον δ ακολουθίες που δεν θα έχουν κανένα λάθος στοιχείο.

Ο αλγόριθμος παράγει έναν τυχαίο αριθμό κ που ανήκει στο {0,1,2,...,2δ-1} και εκτελεί μια επαναληπτική δυαδική αναζήτηση στο $S_κ$. Αποθηκεύουμε σε ασφαλή μνήμη κ, το κλειδί αναζήτησης e και τον αριστερό και τον δεξί δείκτη,l και r, που χρησιμοποιούνται από την δυαδική αναζήτηση. Η δυαδική αναζήτηση τερματίζει όταν τα l και r είναι συνεχόμενα με το Sκ και επομένως 2δ στοιχεία χωριστά στον πίνακα εισόδου, δεδομένου ότι $pos_k(r)-pos_k(l)=2δ$ όταν ισχύει ότι r=l+1.Αν η δυαδική αναζήτηση δεν παρασυρθεί από τα εσφαλμένε στοιχεία, η θέση του e είναι ανάμεσα στα $pos_k(l)$ και $pos_k(r)$ στον πίνακα εισόδου. Για να ελέγξουμε αν η αναζήτηση έχει παρασυρθεί εκτελούμε την ακόλουθη διαδικασία επιβεβαίωσης. Θεωρούμε τους γείτονες N_l και N_r,που περιέχουν τα 2δ+1 στοιχεία στον πίνακα εισόδου που βρίσκονται στα αριστερά της θέσης $pos_k(l)$ και στα δεξιά της θέσης $pos_k(r)$.Υπολογίζουμε το νούμερο $s_l = |\{ z \epsilon N_l \ | z \leq e \}|$ των στοιχείων στο N_l που είναι μικρότερα από το e σε O(δ) χρόνο διαπέρασης του N_l. Παρόμοια , υπολογίζουμε το νούμερο s_r,των στοιχείων στο Nr που είναι μεγαλύτερα από το e.Αν $s_l \geq δ+1$ και $sr \geq δ+1$ και το κλειδί αναζήτησης δεν βρίσκεται

στα N_l και N_r,αποφασίζουμε αν βρίσκεται στον πίνακα ή όχι με την εξέταση όλων των 2δ-1 στοιχείων μεταξύ του $pos_k(l)$ και του $pos_k(r)$.Αν s_l και s_r είναι μικρότερα από δ+1 ένα κατεστραμμένο στοιχείο έχει οδηγήσει σε λάθος την αναζήτηση. Σε αυτήν την περίπτωση ένα καινούριο κ επιλέγεται τυχαία και η δυαδική αναζήτηση ξαναξεκινάει.

ΘΕΩΡΗΜΑ 12ο: Η τυχαία δομή λεξικού υποστηρίζει αναζητήσεις σε O(logn+δ) προσδοκώμενο χρόνο και χρησιμοποίει O(logδ) τυχαία ψηφιά.

ΑΠΟΔΕΙΞΗ :

Αρχικά θα αποδείξουμε την ορθότητα του αλγορίθμου. Υποθέτουμε ότι $s_l \geq δ+1$ και e δεν ανήκει στο N_l.Δεδομένου ότι μόνο δ καταστροφές είναι δυνατό να γίνουν, υπάρχει ένα μη κατεστραμμένο στοιχείο στο N_l αυστηρώς μικρότερο από το e. Επειδή ο πινάκας εισόδου είναι ταξινομημένος κανένα μη κατεστραμμένο στοιχείο αριστερά του $pos_k(l)$ στον πίνακα εισόδου είναι όσο με το e.Με παρόμοιο επιχείρημα, αν $s_r \geq δ+1$ και το e δεν ανήκει στο N_r ,τότε κανένα μη κατεστραμμένο στοιχείο στα δεξιά του $pos_k(r)$ στον πίνακα εισόδου είναι όσο με το e.Αν κατά τη διάρκεια της δυαδικής αναζήτησης δεν συναντήσουμε κανένα κατεστραμμένο στοιχείο, όλα τα μη κατεστραμμένα στοιχεία του N_l είναι μικρότερα του e,και έτσι $s_l \geq δ+1$.Παρομοιως, έχουμε $s_r \geq δ+1$ και ο αλγόριθμος τερματίζει μετά τον έλεγχο των στοιχείων μεταξύ του l και r.

Θα αναλύσουμε τώρα τον χρόνο εκτέλεσης του αλγορίθμου. Κάθε επανάληψη δημιουργεί έναν τυχαίο αριθμό έστω κ που ανήκει στο {0,1,..,2δ-1},χρησημοποιωντας O(logδ) τυχαία ψηφιά. Οι ταξινομημένες ακολουθίες επηρεασμένες από διαφορετικά κ είναι κομματιασμένες, έτσι ώστε το πολύ δ από αυτές να περιέχουν κάποιο κατεστραμμένο στοιχείο. Δεδομένου ότι υπάρχουν 2δ ταξινομημένες ακολουθίες η πιθανότητα να επιλεχτεί μια τιμή κ που οδηγεί σε μια ακολουθία χωρίς κατεστραμμένα στοιχεία είναι τουλάχιστον ½ και ο προσδοκώμενος αριθμός των επαναλήψεων είναι το πολύ δυο. Κάθε επανάληψη χρησιμοποίει O(logn) χρόνο για την δυαδική αναζήτηση και O(δ) χρόνο για την επιβεβαίωση του αποτελέσματος. Συμπεραίνουμε ότι μια αναζήτηση χρησιμοποίει O(logδ) τυχαία ψηφιά και O(logn+δ) προσδοκώμενο χρόνο.

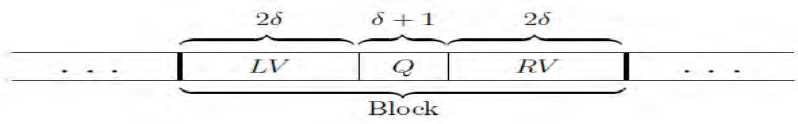
Σημειώνουμε σ' αυτό το σημείο ότι για κάθε επανάληψη ένας προσαρμοζόμενος αντίπαλος μπορεί να μάθει σχετικά με την υπακολουθία S_k στην οποία εκτελούμε την δυαδική αναζήτηση εντοπίζοντας τα στοιχεία που είχαμε πρόσβαση. Έπειτα θα είναι δυνατόν μια και μοναδική καταστροφή στοιχείου να αναγκάσει τη διαδικασία της αναζήτησης να τερματίσει μακριά από το σωστό αποτέλεσμα και έτσι η διαδικασία επιβεβαίωσης να αποτύχει. Σ' αυτήν την κατάσταση ο αλγόριθμος εκτελεί O(δ) επαναλήψεις και επομένως απαιτεί O(δ(logn+δ)) χρόνος ανεξάρτητα από την τυχαία επιλογή των υπακολουθιών στις οποίες εκτελείται η δυαδική αναζήτηση.

Εξασφαλίζουμε ένα όριο για τα τυχαία ψηφιά στην χειρότερη περίπτωση που είναι O(logδ), χρησιμοποιώντας μια σταθερή τεχνική που σταματά την τυχαιότητα των ψηφίων. Στην i-οστή επανάληψη εκτελούμε την δυαδική αναζήτηση στην ακολουθία Sh(i) με $h(i)=(r_0+ir_1+i^2r_2+i^3r_3)$υπολοιπο κ, όπου κ είναι ένας πρώτος αριθμός με $2δ \leq κ \leq 4δ$ και το r_i να είναι επιλεγμένο ομοιόμορφα στην τύχη μέσα στο {0,1,..,κ-1}.Απο την κατασκευή του h(i) αποτελεί μια 4-wise ανεξάρτητη συνάρτηση κατακερματισμού η οποία αρκεί για να ληφθέν ένας αναμενόμενος σταθερός αριθμός επαναλήψεων για τον αλγόριθμο μας.

8.2 ΒΕΛΤΙΣΤΗ ΝΤΕΤΕΡΜΙΝΙΣΤΙΚΗ ΑΝΘΕΚΤΙΚΗ ΔΟΜΗ ΛΕΞΙΚΟΥ

Στη παράγραφο που ακολουθεί θα κλείσουμε το χάσμα ανάμεσα στα κάτω και το πάνω όρια για ντετερμινιστικούς ανθεκτικούς αλγορίθμους αναζήτησης. Παρουσιάζουμε δηλαδή έναν ανθεκτικό αλγόριθμο με τον οποίο η αναζητήσεις για ένα στοιχείο σε ένα ταξινομημένο πίνακα σε O(logn+δ) χρόνο στην χειρότερη περίπτωση, η οποία παράλληλα είναι και βέλτιστη. Αυτό είναι μια βελτιστοποίηση από την προηγούμενη δημοσιευμένη καλύτερη ντετερμινιστική δομή λεξικού, η οποία υποστηρίζει αναζητήσεις σε $O(\log n+\delta^{1+\varepsilon})$ χρόνο. Ξαναχρησιμοποιούμε την ιδέα που παρουσιάστηκε στο σχεδιασμό του τυχαίου αλγορίθμου και όριζε κομματιασμένες ταξινομημένες ακολουθίες για να χρησιμοποιηθούν από έναν αλγόριθμο δυαδικής αναζήτησης. Παρόμοια με τον τυχαίο αλγόριθμο σχεδιάζουμε μια διαδικασία επιβεβαίωσης για να ελέγχουμε το αποτέλεσμα της δυαδικής αναζήτησης. Σχεδιάζουμε την προσαρμοσμένη δυαδική αναζήτηση και την διαδικασία επιβεβαίωσης έτσι ώστε να είναι εγγυημένη η εκτέλεση μόνο ενός επίπεδου στην δυαδική αναζήτηση για κάθε κατεστραμμένο στοιχείο οδηγώντας την αναζήτηση σε λάθος σημείο. Μετράμε το πλήθος των κατεστραμμένων στοιχείων που έχουν εντοπιστεί και προσαρμόζουμε τον αλγόριθμο μας αναλόγως για να σιγουρέψουμε ότι κανένα στοιχείο δεν χρησιμοποιείται περισσότερες από μια φόρες, εξαιρώντας βέβαια την τελευταία διαπέραση που εκτελείται μόνο μια φόρα στα δύο γειτονικά τμήματα. Ο συνολικός χρόνος που χρησιμοποιείται για την επιβεβαίωση του αποτελέσματος είναι O(δ).

Διαιρούμε τον πινάκα εισόδου σε τμήματα. Κάθε τμήμα αποτελείται από 5δ+1 διαδοχικά στοιχειά της εισόδου και είναι δομημένα σε τρία κομμάτια. Το αριστερό, που είναι το κομμάτι επιβεβαίωσης, που συμβολίζεται με LV,που αποτελείται από τα πρώτα 2δ στοιχειά. Τα επόμενα δ+1 στοιχειά αποτελούν το εξεταζόμενο κομμάτι ή το κομμάτι των ερωτήσεων, που συμβολίζεται με Q. Και το τρίτο και τελευταίο κομμάτι που είναι το δεξί κομμάτι επιβεβαίωσης και συμβολίζεται με RV. Το κομμάτι αυτό αποτελείται από τα τελευταία 2δ στοιχειά του τμήματος όπως φαίνεται στην εικόνα 9 πιο κάτω:



Εικόνα 9: Διακρίνονται τα τρία τμήματα στα οποία χωρίζει ο αλγόριθμος τον πίνακα καθώς και το μέγεθός τους[8].

Το αριστερό και το δεξί κομμάτι επιβεβαίωσης χρησιμοποιούνται μόνο από τη διαδικασία επιβεβαίωσης. Τα στοιχειά του εξεταζόμενου κομματιού χρησιμοποιούνται για να ορίσουν της ταξινομημένες ακολουθίες $S_0,S_1,..S_\delta$, όπως η τυχαία δομή λεξικού που δημιουργήθηκε προηγουμένως. Το j-οστό στοιχείο της

ακολουθίας Si, που συμβολίζεται με Si[j], είναι το i-οστό στοιχείο του εξεταζόμενου κομματιού του j-οστού τμήματος και βρίσκεται στην τοποθεσία $pos_i(j)=(5\delta+1)j+2\delta+i$ στον πίνακα εισόδου.

Αποθηκεύουμε μια τιμή κ που ανήκει στο {0,1,..,δ} σε ασφαλή μνήμη που προσδιορίζει την ακολουθία S_κ στην όποια εκτελούμε την δυαδική αναζήτηση. Επίσης, το κ προσδιορίζει τον αριθμό των κατεστραμμένων στοιχείων που έχουν εντοπιστεί .Όποτε εντοπίζουμε ένα κατεστραμμένο στοιχείο αλλάζουμε την ακολουθία που εκτελούμε την δυαδική αναζήτηση ,αυξάνοντας το κ κατά ένα. Δεδομένου ότι υπάρχουν δ+1 ξεχωριστές ακολουθίες, θα υπάρχει τουλάχιστον μια ακολουθία χωρίς κάποιο κατεστραμμένο στοιχείο.

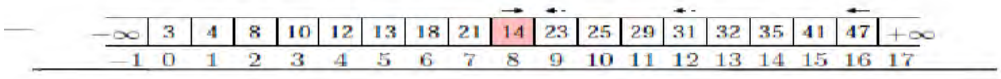
Η δυαδική αναζήτηση εκτελείται στα στοιχειά της S_κ. Παρόμοια με τον τυχαίο αλγόριθμο αποθηκεύουμε σε ασφαλή μνήμη το κλειδί αναζήτησης, έστω e, και την αριστερή και δεξιά ακολουθία, l και r, που χρησιμοποιούνται από την δυαδική αναζήτηση. Αρχικά, l=-1 είναι η θέση ενός υπονοουμένου στοιχείο στο -∞,αριστερά από το αριστερότερο στοιχείο. Παρόμοια, r είναι η θέση από ένα υπονοούμενο στοιχείο στο +∞ στα δεξιά του τελευταίου στοιχείου. Δεδομένου ότι κάθε στοιχείο στην S_κ ανήκει σε ένα ξεχωριστό τμήμα, τα l και r ανήκουν επίσης σε δυο τμήματα, B_l και B_r αντίστοιχα. Κάθε βήμα στην δυαδική αναζήτηση συγκρίνει το κλειδί αναζήτησης με το στοιχείο στην θέση i=(l+r)/2 στην ακολουθία S_k.Υποθέτουμε χωρίς να χάνουμε σε γενικότητα ότι το στοιχείο αυτό είναι μικρότερο από το e.Το l παίρνει την τιμή του i και μειώνουμε το r κατά ένα. Στη συνεχεία συγκρίνουμε το e με το $S_k[r]$. Αν αυτό το στοιχείο είναι μεγαλύτερο από το e,η αναζήτηση συνεχίζεται. Αλλιώς, αν δεν υπάρχει κάποιο κατεστραμμένο στοιχείο, η θέση του στοιχείου αναζήτησης είναι στο τμήμα B_r ή B_{r+1} του πίνακα εισόδου. Όταν δυο γειτονικά στοιχειά είναι ορισμένα όπως στην προηγούμενη περίπτωση, ή όταν l και r γίνονται διαφορετικά, καλούμε την διαδικασία επιβεβαίωσης στο αντίστοιχο τμήμα. Η περιγραφή του ψευδοκώδικα της δυαδικής αναζήτησης δίνεται στην εικόνα 10 που ακολουθεί .

Algorithm 1: Pseudo-code for the binary search procedure.

```
l ← −1
r ←last-block+1
while r − l > 1 do
    i ← ⌈(l+r)/2⌉
    if rep_k(block(i)) < e then
        l ← i
        r ← r − 1
        if rep_k(block(r)) < e then
            if verify(r,r+1) is successful then
                return success
            else
                Backtrack
    else if rep_k(block(i)) > e then
        Similar to previous case.
    else
        return success
if verify(l,r) is successful then
    return success
else
    Backtrack
```

Εικόνα 10: Ψευδοκώδικας της διαδικασίας δυαδικής αναζήτησης [8].

και ένα παράδειγμα του πίνακα του αλγορίθμου φαίνεται αμέσως μετά.



Εικόνα 11: Στιγμιότυπο του πίνακα κατά την εκτέλεση του αλγορίθμου της εικόνας 10 [8].

Η διαδικασία επιβεβαίωσης τερματίζει σε περίπτωση που τα δύο γειτονικά τμήματα, B_i και B_{i+1} είναι ορθά ορισμένα. Αν η επιβεβαίωση επιτύχει, η δυαδική αναζήτηση έχει ολοκληρωθεί και όλα τα στοιχειά στα δυο γειτονικά τμήματα έχουν διαπεραστεί. Η αναζήτηση επιστρέφει “αληθές” αν το e εντοπίστηκε κατά τη διάρκεια της διαπέρασης του τμήματος και “ψευδές” σε κάθε άλλη περίπτωση. Αν όμως αποτύχει η επιβεβαίωση η αναζήτηση ίσως έχει οδηγηθεί σε λάθος σημείο από καταστροφές στοιχείων και υπαναχωρούμε δύο βήματα. Για να διευκολύνουμε την υπαναχώρηση, αποθηκεύουμε έναν πίνακα ψηφίων με μέγεθος δύο λέξεων, d και f σε ασφαλή μνήμη. Το i-οστό ψηφίο του d υποδεικνύει την κατεύθυνση της αναζήτησης και το i-οστό ψηφίο του f μας δείχνει αν υπήρχε κάποια στρογγυλοποίηση στον υπολογισμό του μεσαίου στοιχείου στο i-οστό βήμα στην δυαδική αναζήτηση. Μπορούμε εύκολα να εντοπίσουμε τις τιμές των l και r στο προηγούμενο βήμα της δυαδικής αναζήτησης ανακτώντας τα σωστά ψηφιά από τα d και f. Σε περίπτωση που η επιβεβαίωση αποτύχει, σημαίνει ότι έχει εντοπίσει τουλάχιστον ένα κατεστραμμένο στοιχείο και αυξάνουμε το κ και η αναζήτηση συνεχίζεται σε μια διαφορετική ακολουθία $S_κ$.

Ας μελετήσουμε τώρα αναλυτικότερα τη φάση της επιβεβαίωσης. Η επιβεβαίωση εκτελείται σε δυο γειτονικά τμήματα, B_i και B_{i+1}. Αυτή είτε θα υπολογίσει ότι το e βρίσκεται στα B_i ή B_{i+1},ειτε θα εντοπίσει μια καταστροφή στοιχείου. Η επιβεβαίωση είναι ένας επαναληπτικός αλγόριθμος που διατηρεί μια τιμή από την οποία φαίνεται αν το κλειδί αναζήτησης βρίσκεται στα B_i ή B_{i+1}. Αρχικά υπολογίζουμε την αριστερή τιμή, c_l,η οποία πολιτικοποιεί το βαθμό εμπιστοσύνης για τον οποίο το e ανήκει στο B_i ή βρίσκεται πιο δεξιά από αυτό. Διαισθητικά, ένα στοιχείο στο τμήμα LV_i μικρότερο από το e είναι στη σωστή θέση αν το e βρίσκεται στο B_i ή ακόμα πιο δεξιά. Παρ’ όλα αυτά ένα στοιχείο στο LV_i που είναι μεγαλύτερο από το e σημαίνει ότι υπάρχει ασυνέπεια. Με τον ίδιο τρόπο υπολογίζουμε για το δεξί τμήμα την τιμή c_r για να εκφράσουμε την εμπιστοσύνη του e να βρίσκεται στο B_{i+1} ή ακόμα πιο αριστερά από αυτό.

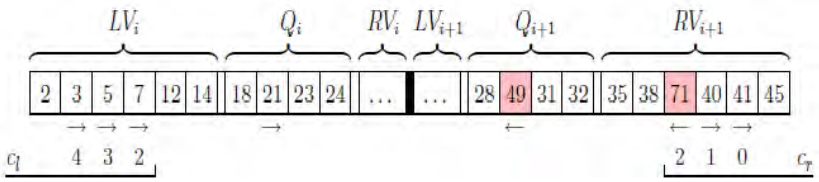
Υπολογίζουμε το c_l με τον έλεγχο ενός υποδιαστήματος του αριστερού τμήματος επιβεβαίωσης,LV_i, του B_i. Ομοίως, η δεξιά πιθανότητα υπολογίζεται με την διαπεράσει του δεξιού τμήματος επιβεβαίωσης ,RV_{i+1},του B_{i+1}. Αρχικά, θέτουμε c_l=1 και c_r=1. Διαπερνάμε το LV_i από τα δεξιά προς τα αριστερά ξεκινώντας από

το στοιχείο με δείκτη u_l=2δ-2κ στο LV_i. Με την επιλογή του u_l είμαστε σίγουροι ότι κανένα στοιχείο στο LV_i έχει μελετηθεί περισσότερες από μια φόρες. Με τον ίδιο τρόπο διαπερνάμε το RV_{i+1} από τα αριστερά στα δεξιά ξεκινώντας με το στοιχείο στην θέση u_r=2k. Σε μια επανάληψη συγκρίνουμε το $LV_i[u_l]$ και το $RV_{i+1}[u_r]$ με το e. Αν $LV_i[u_l] \leq e$,τότε το c_l αυξάνεται κατά ένα, αλλιώς μειώνεται κατά ένα και αυξάνεται το κ κατά ένα. Η διαδικασία επιβεβαίωσης σταματά όταν το ελάχιστο των (c_r,c_l) ισούται με το δ-κ-1 ή με μηδέν. Η επιβεβαίωση επιτυγχάνει στην πρώτη περίπτωση και αποτυγχάνει στην δεύτερη. Ακολουθεί ο ψευδοκώδικας της διαδικασίας επιβεβαίωσης και ένα παράδειγμα στην εικόνα αμέσως μετά.

Algorithm 2: Pseudo-code for the verification procedure.

```
input  : k: Number of errors identified so far
         δ: maximum number of errors
         l:index of the left block
         r:index of the right block
LV ← index of first element in LV_l
RV ← index of first element in RV_r
i_l ← LV + 2δ − 2k
i_r ← RV + 2k
c_r, c_l ← 1
while 0 < min(c_l, c_r) < δ − k + 1 do
    if A[i_l] < e then
        c_l ← c_l + 1
    else
        c_l ← c_l − 1
        k ← k + 1
    if A[i_r] > e then
        c_r ← c_r + 1
    else
        c_r ← c_r − 1
        k ← k + 1
    i_l ← i_l − 1
    i_r ← i_r + 1
if min(c_l, c_r) = 0 then
    return failure
else
    Scan left and right block and return result
```

Εικόνα 12: Ψευδοκώδικας της διαδικασίας επιβεβαίωσης [8].



Εικόνα 13: Στιγμιότυπο τρεξίματος του αλγορίθμου της εικόνας 12.Ξεχωρίζουμε τα τμήματα LV,Q και RV για i και i+1 με εξεταζόμενα τα στοιχεία 49 και 71 [8].

Θεώρημα:

Ο ανθεκτικός σε λάθη αλγόριθμος ψάχνει για ένα στοιχείο σε έναν ταξινομημένο πίνακα σε O(logn+δ) χρόνο.

ΑΠΟΔΕΙΞΗ:

Αρχικά θα αποδείξουμε ότι όταν το c_l ή c_r μειώνεται κατά τη διάρκεια της επιβεβαίωσης, σημαίνει ότι ένα κατεστραμμένο στοιχείο έχει εντοπιστεί. Αυξάνουμε το c_l όταν ένα στοιχείο μικρότερο από το e εντοπίζεται στο τμήμα LV_i ενώ στην αντίθετη περίπτωση το μειώνουμε. Το c_l μπορεί να θεωρηθεί ως το μέγεθος της στοίβας S. Όταν συναντάμε ένα στοιχείο μικρότερο του e, το αντιμετωπίζουμε σαν να ήταν βαλμένο στη στοίβα και σε αντίθετη περίπτωση σαν να βγαίνει από τη στοίβα. Αρχικά ,το στοιχείο g από το εξεταζόμενο κομμάτι του B_i χρησιμοποιείται από την δυαδική αναζήτηση βαλμένο στην S. Δεδομένου ότι το g χρησιμοποιήθηκε για να ορίσει το αριστερό άκρο στην δυαδική αναζήτηση, ισχύει g<e αυτή τη στιγμή. Κάθε φόρα που ένα στοιχείο, για το οποίο ισχύει $LV_i[u]$<e, αφαιρείται από τη στοίβα, θα ισούται με το τρέχον στοιχείο $Lvi[u_l]$.Δεδομένου ότι $LV_i[u]<e<LV_i[u_l]$ και $u_l<u$ τουλάχιστον ένα από τα $LV_i[u_l]$ και $LV_i[u]$ έχει καταστραφεί και ακόμη κάθε ταίριασμα αντιστοιχεί στον εντοπισμό τουλάχιστον μιας καταστροφής στοιχείου. Αυτό έχει ως αποτέλεσμα ότι αν 2τ-1 στοιχειά διαπεραστούν από την μια μεριά κατά τη διάρκεια της επιβεβαίωσης, θα έχουν εντοπιστεί τουλάχιστον τ καταστροφές στοιχείων.

Τώρα θα δείξουμε ότι κανένα κελί που έχει καταστραφεί μια φόρα δεν καταμετράται δυο φόρες. Μια καταστροφή εντοπίζεται αν και μόνο αν δυο στοιχειά συμμετέχουν σε περισσότερα από ένα ταιριάσματα. Πρώτα αναλύουμε τις καταστροφές που προκαλούνται στο αριστερό και στο δεξί τμήμα επιβεβαίωσης. Δεδομένου ότι η επιβεβαίωση αρχίζει από το δείκτη 2(δ-κ) στο αριστερό τμήμα επιβεβαίωσης και το κ αυξάνεται όταν μια καταστροφή στοιχείου εντοπίζεται, κανένα στοιχείο δεν διαβάζεται δυο φόρες και έτσι ούτε ταυτίζεται

δύο φόρες. Μια παρόμοια διαδικασία ακολουθείται και για το δεξί τμήμα επιβεβαίωσης. Κάθε αποτυχημένη επιβεβαίωση αυξάνει το κ και έτσι κανένα στοιχείο από το εξεταζόμενο τμήμα διαβάζεται περισσότερες από μια φορές. Σε κάθε βήμα της δυαδικής αναζήτησης και τα δυο γειτονικά τμήματα ενημερώνονται. Οποτεδήποτε υπαναχωρούμε στην δυαδική αναζήτηση, οι δυο τελευταίες ενημερώσεις των l και r επαναφέρονται. Αν το ίδιο τμήμα χρησιμοποιείται σε μια μεταγενέστερη διαδικασία επιβεβαίωσης, ένα νέο στοιχείο από το εξεταζόμενο τμήμα διαβάζεται και αυτό το νέο στοιχείο είναι αυτό που αρχικοποιεί την στοίβα. Συμπεραίνουμε λοιπόν ότι ούτε τα στοιχεία στο εξεταζόμενο τμήμα, τα όποια έχουν αρχικά τοποθετηθεί στην στοίβα, δεν ταυτίζονται ποτέ δυο φορές.

Για να σιγουρέψουμε την ορθότητα αποδεικνύουμε ότι αν η διαδικασία επιβεβαίωσης είναι επιτυχής και το e δεν εντοπιστεί στην διαπεράσει των δύο γειτονικών, τότε κανένα μη κατεστραμμένο στοιχείο ίσο με το e δεν υπάρχει στην είσοδο. Αν μια επιβεβαίωση επιτύχει και το e δεν βρεθεί σε κανένα από τα δύο τμήμα του πίνακα τότε c_l>=δ-κ+1.Δεδομενου ότι μόνο δ-κ επιπλέον καταστροφές είναι δυνατόν να γίνουν ,υπάρχει τουλάχιστον ένα μη κατεστραμμένο στοιχείο στο LV_i μικρότερο από το e και έτσι δε μπορεί να υπάρχουν μη κατεστραμμένα στοιχειά ίσα με το e στα αριστερά του B_i στον πίνακα εισόδου. Με παρόμοια διαδικασία ,αν c_r≥δ-k+1,τοτε όλα τα μη κατεστραμμένα στοιχειά στα δεξιά του B_{i+1} στον πίνακα εισόδου είναι μεγαλύτερα από το e.

Ας αναλύσουμε τώρα τον χρόνο εκτέλεσης. Επιβαρύνουμε κάθε υπαναχώρηση της δυαδικής αναζήτησης με την διαδικασία επιβεβαίωσης που την ενεργοποιεί. Ο συνολικός χρόνος του αλγορίθμου είναι O(logn) προστιθέμενος με τον χρόνο που απαιτείται από τις επιβεβαιώσεις. Για οριοθετήσουμε το χρόνο που απαιτείται για όλες τα βήματα της επιβεβαίωσης χρησιμοποιούμε το γεγονός ότι αν O(f) χρόνος απαιτείται για ένα βήμα επιβεβαίωσης, τότε Ω(f) καταστροφές στοιχείων εντοπίζονται στο τέλος του αλγορίθμου. Το πολύ O(δ) χρόνος απαιτείται στην τελευταία επιβεβαίωση για την διαπέραση των δύο τμημάτων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] G. M. Adel'son-Vel'skiˇi e Y. M. Landis, "An algorithm for the organization of informa-tion", Dokl. Akad. Nauk. Sssr, 146:263–266, 1962.

[2] J. A. Aslam and A. Dhagat. Searching in the presence of linearly bounded errors. Proc. 23rd ACM Symp. on Theory of Computing (STOC'91), 486–493, 1991.

[3] Y. Aumann and M. A. Bender. Fault-tolerant data structures. Proc. 37th IEEE Symp. on Foundations of Computer Science (FOCS'96), 580–589, 1996.

[4] R. Bayer ed E. McCreight, "Organization and maintenance of large ordered index-es", Acta Informatica, 1:173–189, 1972.

[5] M. Blum, W. Evans, P. Gemmell, S. Kannan and M. Naor. Checking the correct-ness of memories. Algorithmica, 12:225–244, 1994.

[6] R. S. Borgstrom and S. Rao Kosaraju. Comparison based search in the presence of errors. Proc. 25th ACM Symp. on Theory of Computing (STOC'93), 130–136, 1993.

[7] R. Boyer and S. Moore. MJRTY - A fast majority vote algorithm. University of TexasTech. Report, 1982.

[8] G. S. Brodal, R. Fagerberg, I. Finocchi, F. Grandoni, G. F. Italiano, A. G. Jørgensen,G. Moruz and T. Mølhave. Optimal resilient dynamic dictionaries. Proc. 15th AnnualEuropean Symp. on Algorithms (ESA'07), LNCS 4698, 347–358, 2007.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algo-rithms.The MIT Press/McGraw-Hill Book Company, 2nd Edition, 2001.

[10] A. Dhagat, P. Gacs, and P. Winkler. On playing "twenty questions" with a liar. Proc.3rd ACM-SIAM Symp. on Discrete Algorithms (SODA'92), 16–22, 1992.

[11] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy infor-mation.SIAM Journal on Computing, 23, 1001–1018, 1994.

[12] I. Finocchi, F. Grandoni, and G. F. Italiano. Optimal sorting and searching in the presence of memory faults. Proc. 33rd Int. Colloquium on Automata, Lang. and Prog. (ICALP'06), 286–298, 2006. To appear in Theoretical Computer Science.

[13] I. Finocchi, F. Grandoni, and G. Italiano. Resilient search trees. In Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA'07), 547–555, 2007.

[14] I. Finocchi and G. F. Italiano. Sorting and searching in faulty memories. To appearin Algorithmica. Extended abstract in Proc. 36th ACM Symposium on Theory of Com-puting (STOC'04), 101–110, 2004.

[15] L. J. Guibas e R. Sedgewick, “A dichromatic framework for balanced trees”, in Pro-ceedings of the 19th IEEE Symposium on Foundations of Computer Science, 8–21, 1978.

[16] S. Hamdioui, Z. Al-Ars, J. V. de Goor, and M. Rodgers. Dynamic faults in ran-dom- access-memories: Concept, faults models and tests. Journal of Electronic Test-ing:Theory and Applications, 19:195–205, 2003.

[17] M. R. Henzinger. The past, present and future of web search engines. In Interna-tional Colloquium on Automata, Languages and Programming (ICALP), 2004. Invit-ed talk.

[18] M. R. Henzinger. Combinatorial algorithms for web search engines - three successstories. In ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007. Invited talk.

[19] A. G. Jørgensen, G. Moruz and T. Mølhave. Priority queues resilient to memory faults. Proc. 10th Workshop on Algorithms and Data Structures (WADS’07), 127–138, 2007.

[20] D. J. Kleitman, A. R. Meyer, R. L. Rivest, J. Spencer, and K. Winklmann. Cop-ing with errors in binary search procedures. Journal of Computer and System Scienc-es,
20:396–404, 1980.

[21] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. IEEE Transactions on Electron Devices, 26(2), 1979.

[22] S. Muthukrishnan. On optimal strategies for searching in the presence of errors. Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA’94), 680–689, 1994.

[23] A. Pelc. Searching with known error probability. Theoretical Computer Science, 63,185–202, 1989. [24] A. Pelc. Searching games with errors: Fifty years of coping with liars.Theoretical Computer Science, 270, 71–109, 2002.

[25] J. von Neumann, Probabilistic logics and the synthesis of reliable organisms from un-reliable components. In Automata Studies, C. Shannon and J. McCarty eds., Princeton Univ. Press, 43–98, 1956.

[26]http://support.inf.uth.gr/vasi/upload/Zervos_Ioannis.pdf ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΙΩΑΝΝΗ ΖΕΡΒΟΥ

[27]www.wikipedia.org

[28]www.google.com

[29] http://www.mit.edu/~6.454/Slides/JohnSun_vonNeumann.pdf

[30] ΒΙΒΛΙΟ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ Π. Δ. ΜΠΟΖΑΝΗ

[31] I. Finocchi, F. Grandoni, and G. F. Italiano “Resilient Dictionaries”
http://www.idsia.ch/~grandoni/Pubblicazioni/FGI09talg.pdf