Πανεπιστήμιο Θεσσαλίας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

# Διπλωματική εργασία

**ΤΙΤΛΟΣ : Μελέτη Δομής Λεξικού με Ευαισθησία στην Κατανομή Προσπέλασης**

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : ΜΠΟΖΑΝΗΣ ΠΑΝΑΓΙΩΤΗΣ

Ζερβός Ιωάννης

Βόλος 2011

# Πίνακας περιεχομένων

# ΚΕΦΑΛΑΙΟ 1

## 1. Εισαγωγή

Ο χρόνος που απαιτείται για μια ακολουθία ενεργειών σε μια δομή δεδομένων συνήθως μετριέται με την χειρότερη δυνατή τέτοια ακολουθία. Αυτός όμως ο τρόπος υπερβαίνει τον χρόνο που απαιτείται στην πραγματικότητα. **Δομές δεδομένων με ευαισθησία στη κατανομή προσπέλασης** προσπαθούν να εκμεταλλευτούν επαναλαμβανόμενα μοντέλα με σκοπό να μειώσουν την χρονική πολυπλοκότητα, εφόσον τα μοντέλα προσπέλασης είναι μη-τυχαία. Δυστυχώς, πολλές από τις *δομές δεδομένων με ευαισθησία στη κατανομή προσπέλασης* απαιτούν αρκετό χώρο (space overhead) με την μορφή δεικτών. Εδώ παρουσιάζουμε μία δομή λεξικού που χρησιμοποιεί τυχαιότητα (randomization) και υπάρχουσες χωρο-αποδοτικές (space-efficient) δομές δεδομένων για να επιτύχουμε χαμηλή κατανάλωση σε χώρο και να διατηρήσουμε την ευαισθησία στη κατανομή προσπέλασης στα προσδοκώμενα επίπεδα.

Για το πρόβλημα λεξικού (dictionary problem), θα θέλαμε να υποστηρίξουμε τις λειτουργίες της **εισαγωγής**, **διαγραφής** και **αναζήτησης** σε ένα ολικά διατεταγμένο σύνολο. Υπάρχουν αρκετές τέτοιες δομές δεδομένων: AVL-δέντρα, Ερυθρό-μαυρα δέντρα και Εξαρθρωμένα (splay) δέντρα. Τα εξαρθρωμένα (splay) δέντρα έχουν ιδιαίτερο ενδιαφέρον επειδή έχουν ευαισθησία στη κατανομή προσπέλασης, αφού ο χρόνος που απαιτείται για ορισμένες λειτουργίες μπορεί να μετρηθεί με όρους της κατανομής αυτών των λειτουργιών.

Δυστυχώς, τέτοιες δομές λεξικού συχνά απαιτούν σημαντικό χώρο. Όμως αυτό είναι γενικό πρόβλημα με τις δομές δεδομένων. Η κατανάλωση χώρου (space-overhead) συχνά έχει την μορφή δεικτών: για παράδειγμα ένα δυαδικό δέντρο αναζήτησης μπορεί να έχει 3 δείκτες για κάθε κόμβο του δέντρου: ένα προς τον πατέρα και ένα προς κάθε παιδί. Σε μια τέτοια περίπτωση και αν υποθέσουμε ότι δείκτες και κλειδιά (keys) έχουν το ίδιο μέγεθος, τότε εύκολα βλέπουμε ότι τα 3/4 του αποθηκευτικού χώρου που χρησιμοποιεί το δυαδικό δέντρο αναζήτησης αποτελούνται από δείκτες. Αυτό είναι σπάταλο, εφόσον ενδιαφερόμαστε για τα ίδια τα δεδομένα και δεν θα θέλαμε να αφιερώνουμε τόσο ποσοστό χώρου σε δείκτες. Για να διορθώσουμε αυτή την κατάσταση, έχει αφιερωθεί αρκετή έρευνα στην περιοχή των implicit δομών δεδομένων (implicit data structures). Μία implicit δομή δεδομένων χρησιμοποιεί μόνο τον χώρο που απαιτείται για να συγκρατήσει τα ίδια τα δεδομένα (σε συνδυασμό με έναν σταθερό αριθμό από λέξεις (words) καθεμία μεγέθους O(log n) bits).

Σκοπός μας είναι να συνδυάσουμε χαρακτηριστικά της ευαισθησίας κατανομής με ιδέες από τα implicit λεξικά για να επιτύχουμε ένα λεξικό με ευαισθησία στη κατανομή προσπέλασης και με χαμηλό space-overhead (distribution-sensitive dictionary with low space overhead).

Εδώ παρουσιάζουμε ένα λεξικό με **χρόνους χειρότερης περίπτωσης εισαγωγής και διαγραφής O (log n)** και **αναμενόμενο χρόνο αναζήτησης O(log t(x))**, όπου x είναι το κλειδί το οποίο αναζητείται και **t(x)** είναι ο αριθμός των ξεχωριστών αιτήσεων που έχουν γίνει από όταν το x είχε αναζητηθεί για τελευταία φορά, ή **n** εάν το x δεν έχει αναζητηθεί μέχρι στιγμής. Το space-overhead που απαιτείται για αυτή τη δομή δεδομένων είναι **O(log log n)**, π.χ. O(log log n) επιπλέον λέξεις μνήμης (words of memory) (κάθε μία μεγέθους O(log n)bits) απαιτούνται εκτός από τα ίδια τα δεδομένα. Υπάρχουσες δομές δεδομένων που ταιριάζουν στο χρόνο των αιτήσεων είναι

τα splay trees (με την αφηρημένη έννοια) και το working set structure (στη χειρότερη περίπτωση) , αν και αυτά απαιτούν 3n και 5n δείκτες αντίστοιχα , υποθέτοντας 3 δείκτες για κάθε κόμβο. Επίσης δείχνουμε πώς να μετατρέψουμε αυτή τη δομή (και κατά προέκταση το working set structure) για να υποστηρίξουμε προηγούμενες αιτήσεις σε λογαριθμικό χρόνο στον αριθμό του συνόλου εργασίας (working set) του προκατόχου.

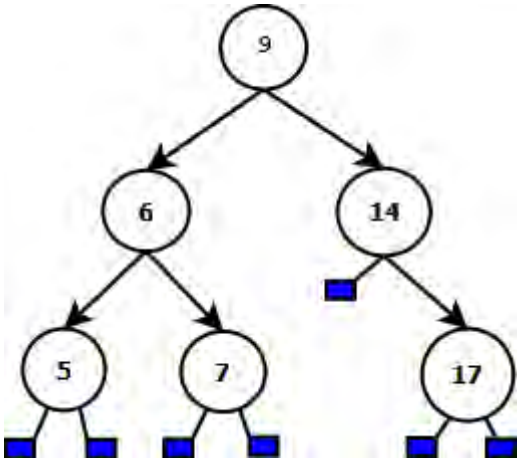
Στο **2ο** κεφάλαιο θα παρουσιάσουμε την δομή και τις λειτουργίες των δέντρων AVL, που θα μας χρειαστούν στην κατασκευή του working set structure. Θα δούμε πώς δομούνται, πώς συμπεριφέρονται στις πράξεις της εισαγωγής, διαγραφής και αναζήτησης, και θα κάνουμε μια ανάλυση της πολυπολοκότητάς τους. Στο **3ο** κεφάλαιο θα περιγράψουμε τη δομή του working set structure και θα αναλύσουμε τις πράξεις εισαγωγής, διαγραφής και αναζήτησης. Στη συνέχεια, στο κεφάλαιο **4** παρουσιάζουμε προτάσεις ώστε να απλοποιηθεί η υλοποίηση της δομής. Στο κεφάλαιο **5** θα διεξάγουμε σενάρια πειραμάτων ώστε να επαληθεύσουμε τους θεωρητικούς χρόνους. Τέλος, στο **6ο** κεφάλαιο καταλήγουμε με πιθανές κατευθύνσεις για μελλοντική έρευνα.

# ΚΕΦΑΛΑΙΟ 2

## 2. Περιγραφή του AVL-tree

### 2.1 AVL-tree

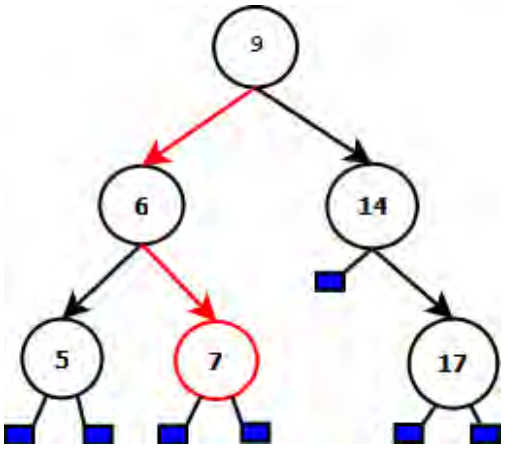
Τα AVL-trees είναι **ισορροπημένα δυαδικά δέντρα αναζήτησης** (balanced binary search trees), δηλαδή το ύψος των παιδιών κάθε κόμβου διαφέρουν το πολύ κατά ένα. Το κόστος εισαγωγής, διαγραφής και αναζήτησης είναι O(log n) στη μέση και στη χειρότερη περίπτωση, όπου n είναι ο αριθμός των κόμβων του δέντρου. Κατά την εισαγωγή και την διαγραφή μπορεί να χρειαστούν επαναζυγιστικές πράξεις, σε περίπτωση που παραβιαστεί η συνθήκη ισορροπίας μεταξύ αδερφών κόμβων.



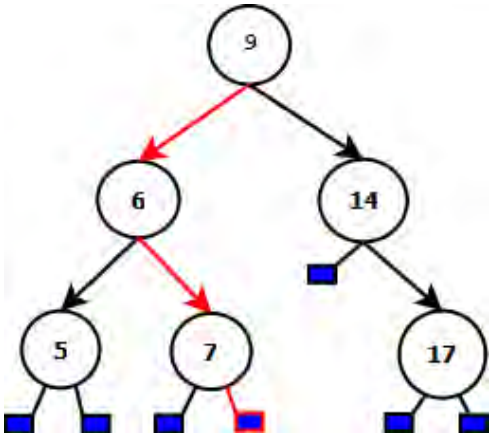

Εικόνα 2.1 Παράδειγμα AVL-tree

## 2.2 Αναζήτηση

Όπως σε κάθε δυαδικό δέντρο αναζήτησης, η αναζήτηση πραγματοποιείται ως εξής: Ξεκινάμε εξετάζοντας αν η ρίζα του δέντρου ισούται με το κλειδί το οποίο ψάχνουμε. Αν ναι, η αναζήτηση τελειώνει επιτυχημένα. Αν το κλειδί είναι μικρότερο από την τιμή της ρίζας, συνεχίζουμε να ψάχνουμε αναδρομικά στο αριστερό υποδέντρο. Σε αντίθετη περίπτωση η αναζήτηση συνεχίζεται στο δεξί υποδέντρο . Η διαδικασία επαναλαμβάνεται μέχρι να βρούμε τον κόμβο ή να καταλήξουμε σε φύλλο. Στην πρώτη περίπτωση, η αναζήτηση είναι επιτυχημένη, ενώ στη δεύτερη απότυχημένη και το κλειδί το οποίο ψάχνουμε δεν βρίσκεται στο δέντρο. Το κόστος αναζήτησης είναι O(log n) εξαιτίας της συνθήκης ισορροπίας που ισχύει στα δέντρα AVL.




Εικόνα 2.2 Παράδειγμα επιτυχημένης αναζήτησης του 7.



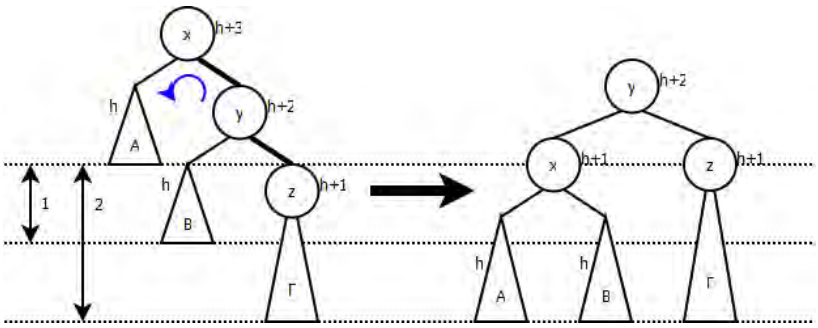

Εικόνα 2.3 Παράδειγμα αποτυχημένης αναζήτησης του 8.

## 2.3 Εισαγωγή

Αφού εισάγουμε ένα κόμβο στο δέντρο, είναι απαραίτητο να εξετάσουμε αν έχει διαταραχθεί η ισορροπία του δέντρου. Για κάθε κόμβο ελέγχουμε τον παράγοντα ισορροπίας (balance factor), δηλαδή την διαφορά του ύψους του αριστερού παιδιού με το ύψος του δεξιού. Αν για όλους τους κόμβους είναι -1,0 ή +1 τότε καμία περιστροφή δεν απαιτείται. Αν σε έστω και ένα κόμβο είναι ±2, τότε το υποδέντρο που έχει αυτό τον κόμβο ρίζα είναι μη ισορροπημένο. Για να το διορθώσουμε, θα πρέπει να προβούμε στις αντίστοιχες επαναζυγιστικές πράξεις. Οι περιπτώσεις μη ισορροπίας είναι οι ακόλουθες:

**(α)** Οι κόμβοι x,y,z σχηματίζουν δεξιό ή αριστερό «ευθές» μονοπάτι, το νέο στοιχείο έχει εισαχθεί στο υποδέντρο Γ και ο x παραβιάζει την συνθήκη ισοζύγισης λόγω της αυξήσεως του ύψους του από h+2 σε h+3. Τότε αρκεί μία αριστερή ή δεξιά **απλή περιστροφή (single rotation-rot)** στον x για να επανέλθει το ύψος του υποδέντρου στην προηγούμενη τιμή του h+2 και άρα να αποκατασταθεί η ισορροπία του δέντρου.

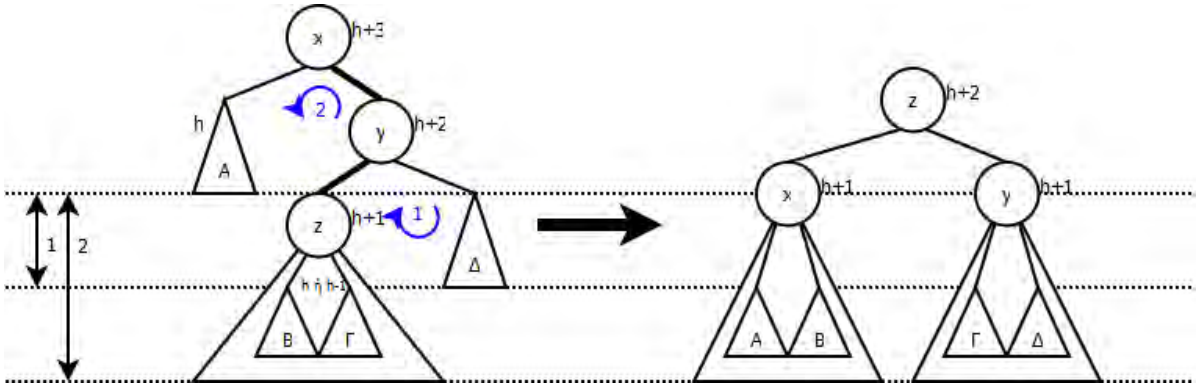
Με τον όρο **απλή περιστροφή** εννοούμε την ανταλλαγή ρόλων των δύο εμπλεκομένων κόμβων. Π.χ. στην περίπτωσή μας, ο x γίνεται γιος του y και αποκτά ως υποδέντρο το B και ο y μετατρέπεται σε πατέρα του x. Η «φορά» της περιστροφής είναι αντίθετη προς την κατεύθυνση όπου «γέρνει» το δέντρο, ώστε να εξισορροπηθεί η ασυμμετρία.




Εικόνα 2.4 Περίπτωση απλής περιστροφής

**(β)**Οι κόμβοι x,y,z σχηματίζουν δεξιά ή αριστερή «γωνία», το νέο στοιχείο έχει εισαχθεί στο υποδέντρο Β ή Γ και ο x παραβιάζει την συνθήκη λόγω της αύξησης, κατά μία μονάδα, του ύψους του σε h+3. Τότε αρκεί μία αριστερή ή δεξιά **διπλή περιστροφή (double rotation-drot)** για να επανέλθει το ύψος του z στην προηγούμενη τιμή και να αποκατασταθεί η ισορροπία στο δέντρο.

Η **διπλή περιστροφή** συνιστάται από δύο απλές περιστροφές αντίθετης φοράς. Για παράδειγμα, μία αριστερή διπλή περιστροφή αποτελείται από μία δεξιά απλή περιστροφή στον ενδιάμεσο κόμβο y (με επακόλουθη την υποβάθμισή του), ακολουθούμενη από μία αριστερή απλή περιστροφή στον υψηλότερο κόμβο x.

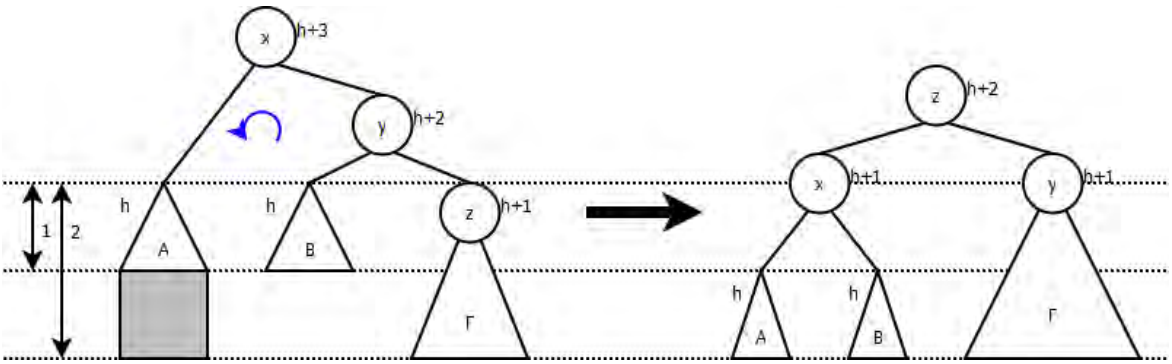



Εικόνα 2.5 Περίπτωση διπλής περιστροφής

## 2.4 Διαγραφή

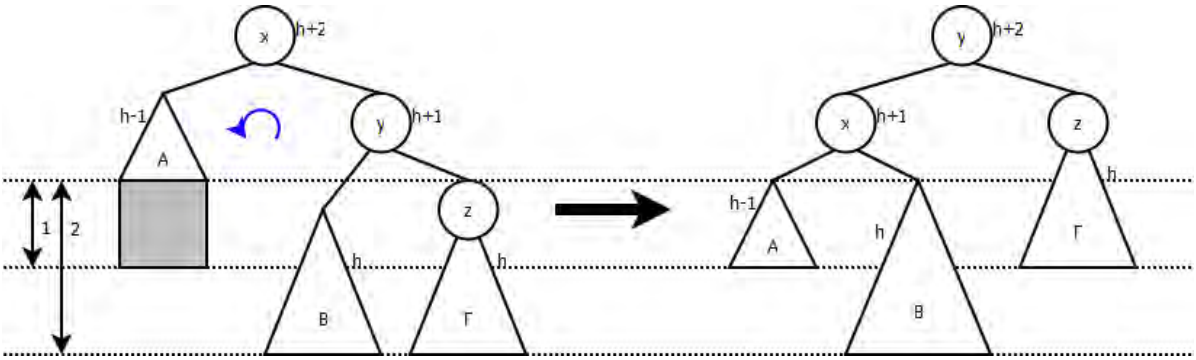
Αν ο προς διαγραφή κόμβος είναι φύλλο ή έχει μόνο ένα παιδί, απλά τον αφαιρούμε. Αλλιώς, τον αντικαθιστούμε είτε με το μεγαλύτερο στοιχείο του αριστερού υποδέντρου του, δηλαδή με το δεξιότερο παιδί του αριστερού υποδέντρου, είτε με το μικρότερο στοιχείου του δεξιού υποδέντρου, δηλαδή με το αριστερότερο στοιχείο του δεξιού υποδέντρου. Μετά την διαγραφή ακολουθούμε το μονοπάτι αντίστροφα προς την ρίζα, προσαρμόζοντας τον δείκτη ισορροπίας όπου χρειάζεται. Παρακάτω περιγράφουμε τις τρεις περιπτώσεις που μπορεί να εμφανιστούν μετά την διαγραφή ενός στοιχείου:

**(α)** Οι κόμβοι x,y,z σχηματίζουν δεξιό ή αριστερό «ευθές» μονοπάτι, το στοιχείο έχει διαγραφεί από το υποδέντρο A, μειώνοντας το ύψος του κατά μία μονάδα σε h ,και ο x παραβιάζει την συνθήκη. Τότε μία αριστερή ή δεξιά απλή περιστροφή στον x ισοζυγίζει το υποδέντρο, μειώνοντας όμως το ύψος του κορυφαίου κόμβου κατά μία μονάδα. Γεγονός που καθιστά υποχρεωτική εξέταση των προγόνων του y για τυχόν δημιουργία παραβιάσεως. Αυτού του είδους οι περιστροφές, οι οποίες θεραπεύουν τοπικά το πρόβλημα, αλλά δεν εγγυώνται την οριστική εξάλειψή του, καθώς ενδεχομένως το μεταθέτουν σε ψηλότερους κόμβους, καλούνται **μη τερματικές.**



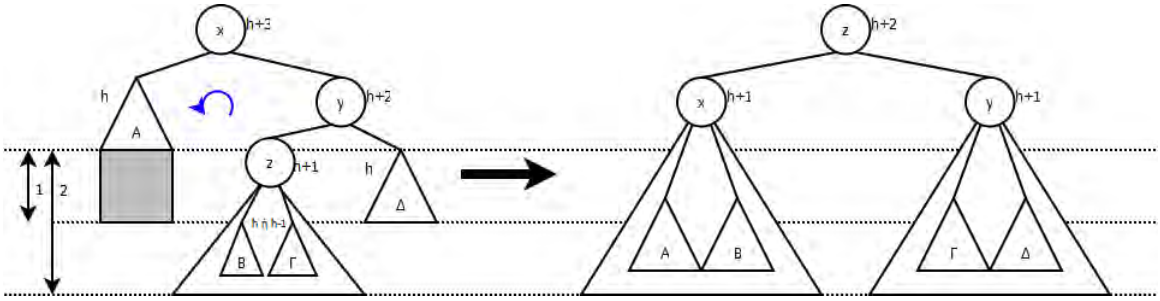

Εικόνα 2.6 Περίπτωση (α) αριστερή μη τερματική περιστροφή

**(β)** Οι κόμβοι x,y,z σχηματίζουν δεξιό ή αριστερό «ευθές» μονοπάτι, το στοιχείο έχει διαγραφεί από το υποδέντρο A, μειώνοντας το ύψος του κατά μία μονάδα, σε h-1 και ο x παραβιάζει την συνθήκη. Τότε μία αριαστερή ή δεξιά απλή περιστροφή στον x, ισοζυγίζει, κατά τρόπο **τερματικό**, το υποδέντρο διατηρώντας το προηγούμενο ύψος.




Εικόνα 2.7 Περίπτωση (β) αριστερή τερματική περιστροφή

**(γ)** Οι κόμβοι x,y,z σχηματίζουν δεξιά ή αριστερή «γωνία», το στοιχείο έχει αποσβεστεί από το υποδέντρο A, μειώνοντας το ύψος του κατά μία μονάδα σε h και ο x παραβιάζει την συνθήκη. Τότε μία αριστερή ή δεξιά διπλή περιστροφή ισοζυγίζει το υποδέντρο, μειώνοντας όμως το ύψος του κατά μία μονάδα, με συνέπεια η επαναζυγιστική πράξη να είναι **μη τερματική.**




Εικόνα 2.8 Περίπτωση (γ) αριστερή διπλή μη τερματική περιστροφή

# ΚΕΦΑΛΑΙΟ 3

## 3. Περιγραφή του Αλγορίθμου

### 3.1 Γενική Περιγραφή

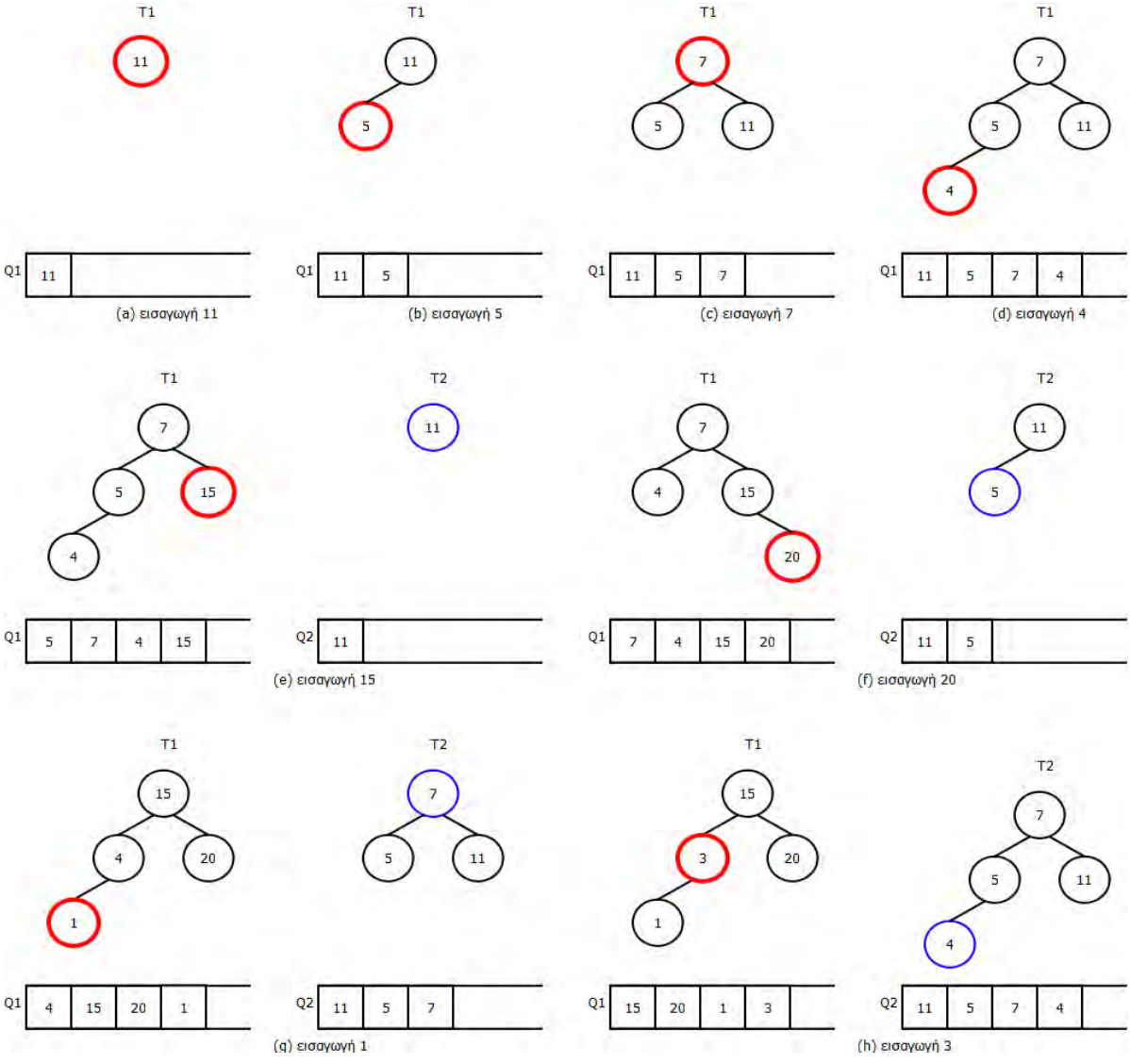
Το working set structure αποτελείται από k ισορροπημένα δυαδικά δέντρα αναζήτησης (balanced binary search trees) T1,…,Tk και από k ουρές (queues) Q1,…,Qk. Κάθε ουρά έχει ακριβώς τα ίδια στοιχεία με το αντίστοιχο δέντρο, και το μέγεθος των Ti και Qi είναι $2^{2^i}$ , εκτός από το Tk και Qk που απλά περιέχουν τα εναπομείναντα στοιχεία. Επομένως, αφού υπάρχουν **n** στοιχεία, **k=O(log log n)**.

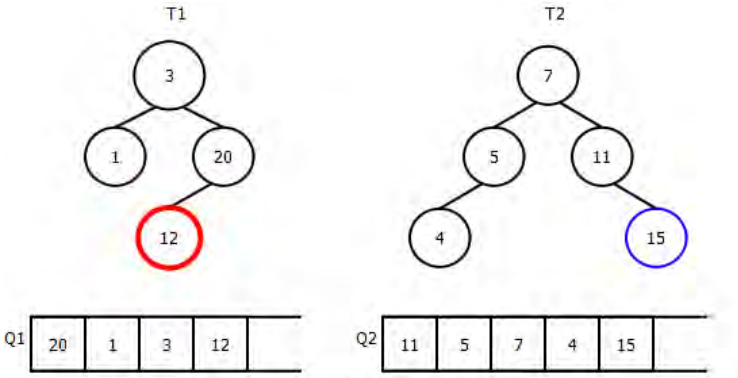
Η δομή μεταχειρίζεται με μια **λειτουργία εναλλαγής (shift operation)** με τον ακόλουθο τρόπο: Μία εναλλαγή από το **i** στο **j** πραγματοποιείται εξάγοντας ένα στοιχείο από την ουρά Qi και αφαιρώντας το αντίστοιχο στοιχείο από το Ti. Έπειτα το αφαιρούμενο στοιχείο εισάγετε στο επόμενο δέντρο και ουρά (όπου «επόμενο» ορίζεται το εγγύτερο δέντρο στο Tj), και η διαδικασία επαναλαμβάνεται μέχρι να φτάσουμε στο Tj και Qj. Με αυτό τον τρόπο, τα παλιότερα στοιχεία αφαιρούνται από τα δέντρα κάθε φορά. Το αποτέλεσμα της εναλλαγής είναι ότι το μέγεθος των Ti και Qi έχει μειωθεί κατά ένα και το μέγεθος των Tj και Qj έχει αυξηθεί κατά ένα.

## 3.2 Εισαγωγή

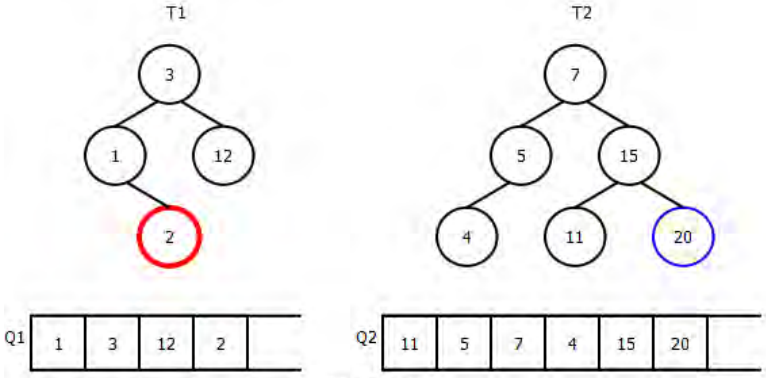
**Εισαγωγές (insertions)** πραγματοποιούνται εκτελώντας μια συνηθισμένη εισαγωγή λεξικού στο T1 και Q1, και στη συνέχεια κάνοντας εναλλαγές (shifts) από το πρώτο index μέχρι το τελευταίο. Μία τέτοια εναλλαγή κάνει χώρο για το νέο εισαγόμενο στοιχείο στο πρώτο δέντρο και ουρά προωθώντας το παλιότερο στοιχείο σε κάθε δέντρο και ουρά στο επόμενο. Στην συνέχεια απεικονίζουμε διαδοχικές ενθέσεις των εξής εικοσιδύο στοιχείων: (a)11,(b)5,(c)7,(d)4,(e)15,(f)20,(g)1,(h)3,(i)12,(j)2,(k)18,(l)8,(m)13 ,(n)14,(o)10,(p)17,(q)16,(r)9,(s)6,(t)19,(u)22,(v)21

Με κόκκινο χρώμα αναπαραστούμε τους κόμβους που εισήχθησαν και με μπλε τους κόμβους που εκτοπίστηκαν σε επόμενα δέντρα.

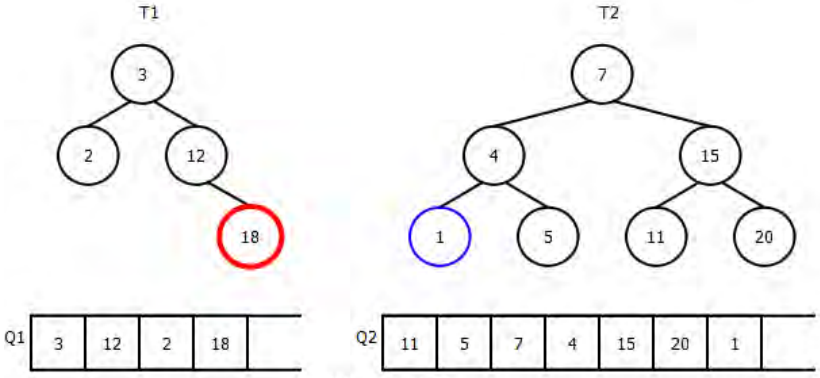




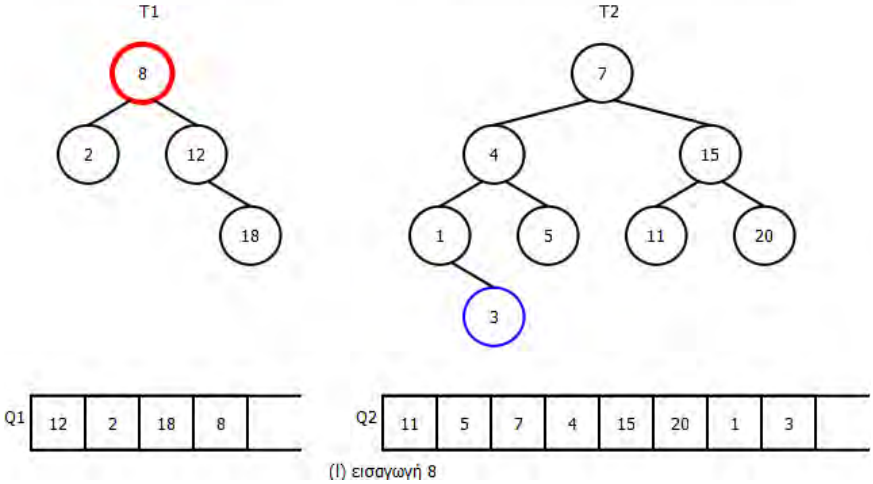

(i) εισαγωγή 12



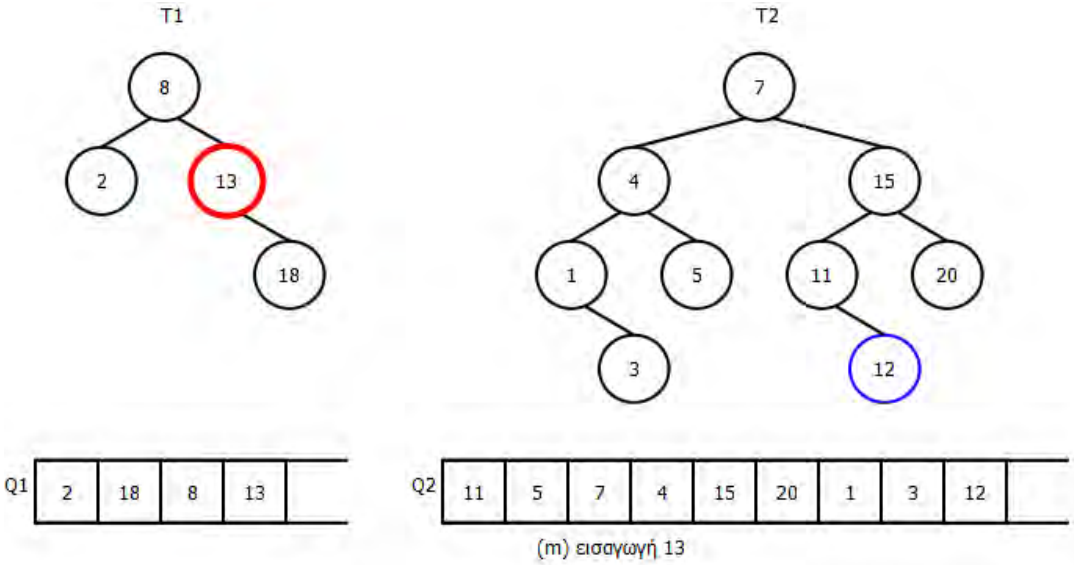

(j) εισαγωγή 2



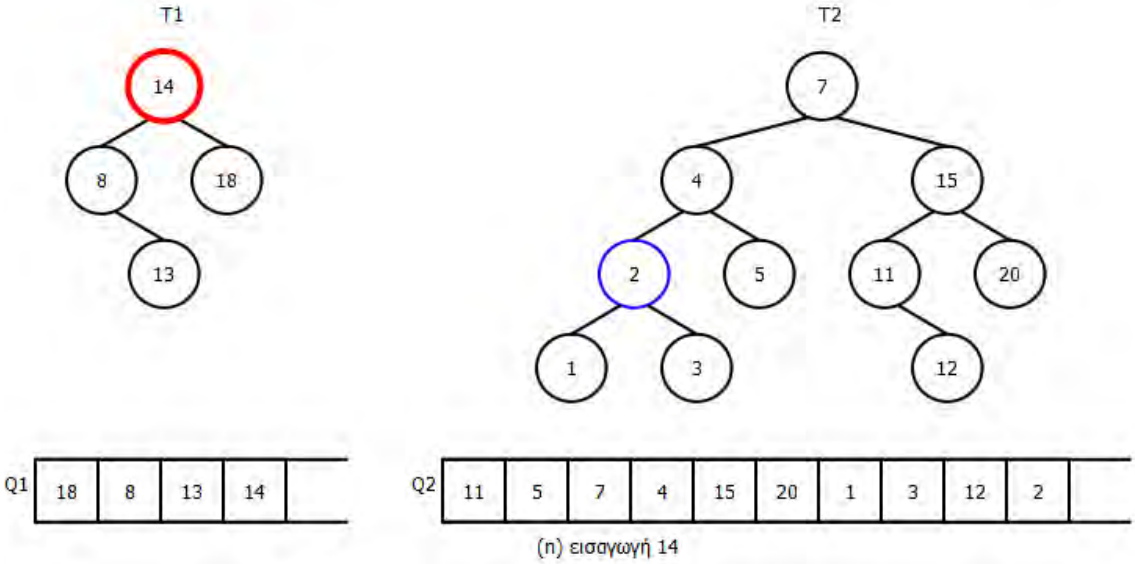

(k) εισαγωγή 18



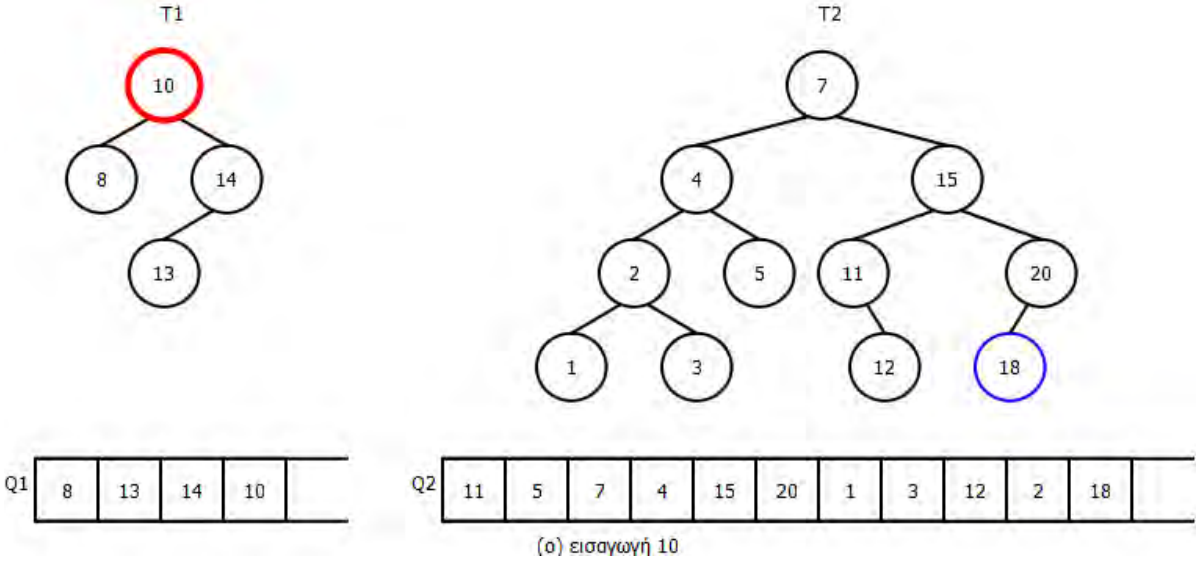

(l) εισαγωγή 8



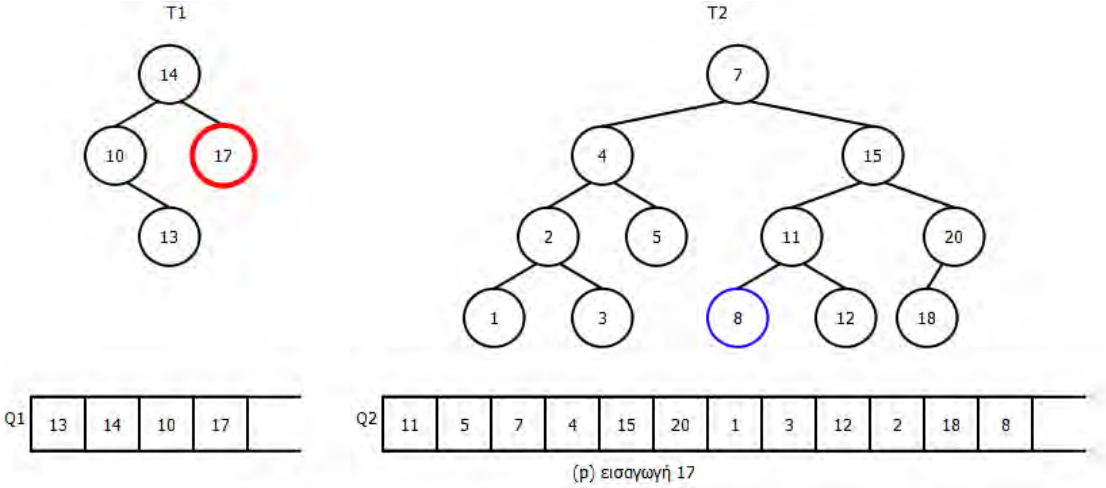

(m) εισαγωγή 13



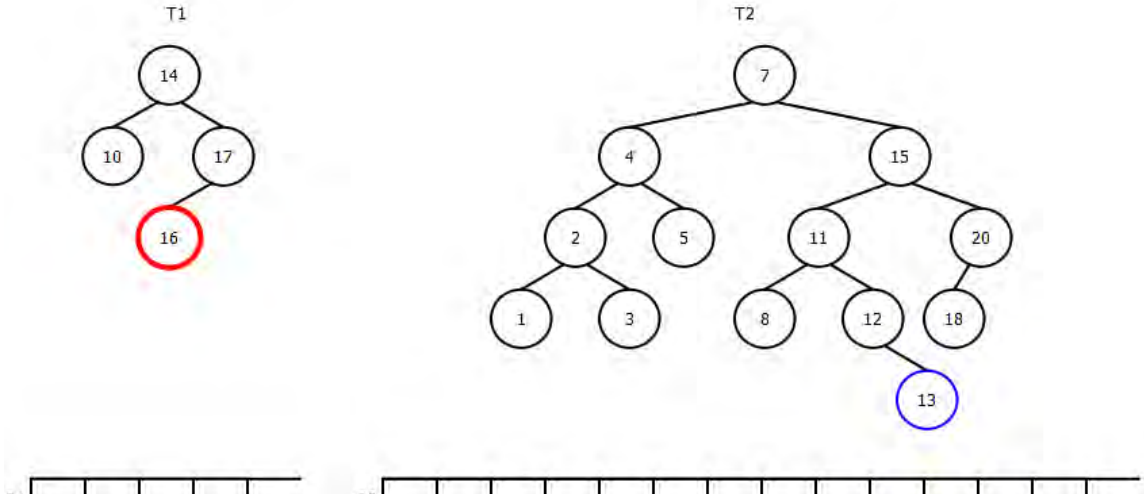

(n) εισαγωγή 14



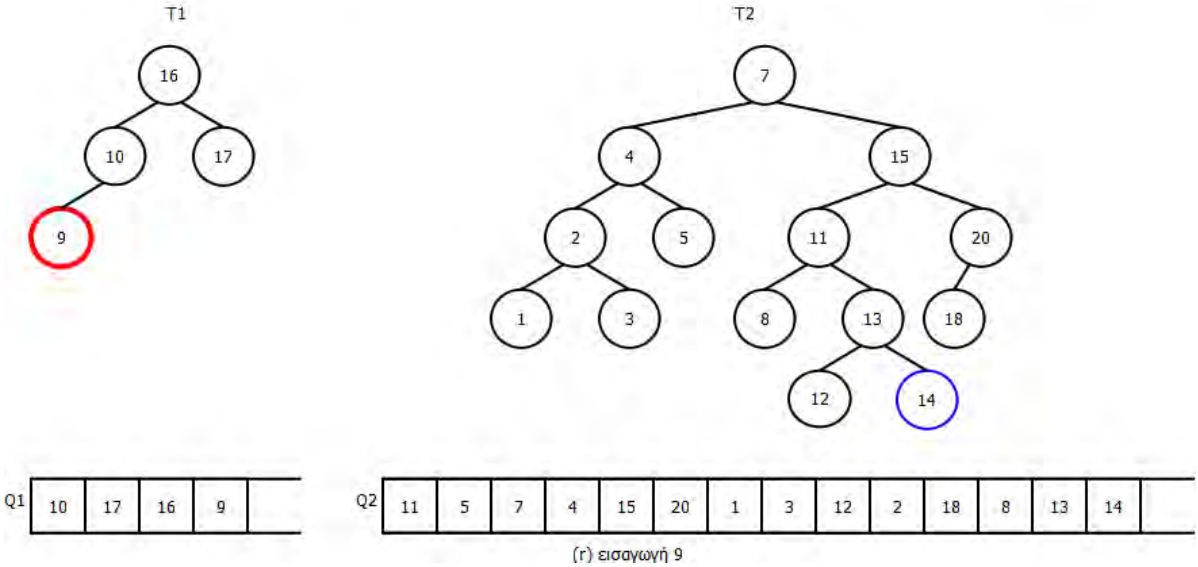

(o) εισαγωγή 10



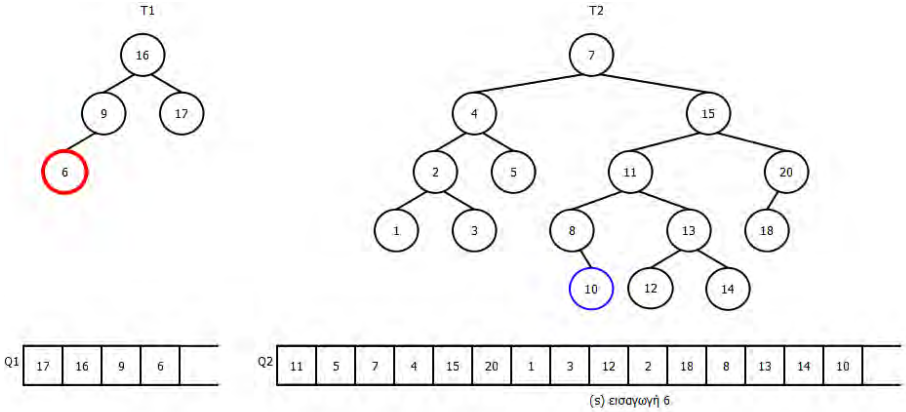

(p) εισαγωγή 17



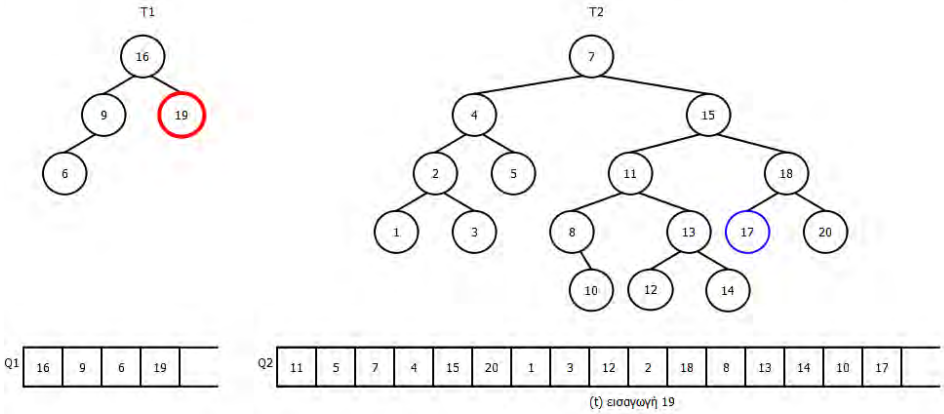

(q) εισαγωγή 16



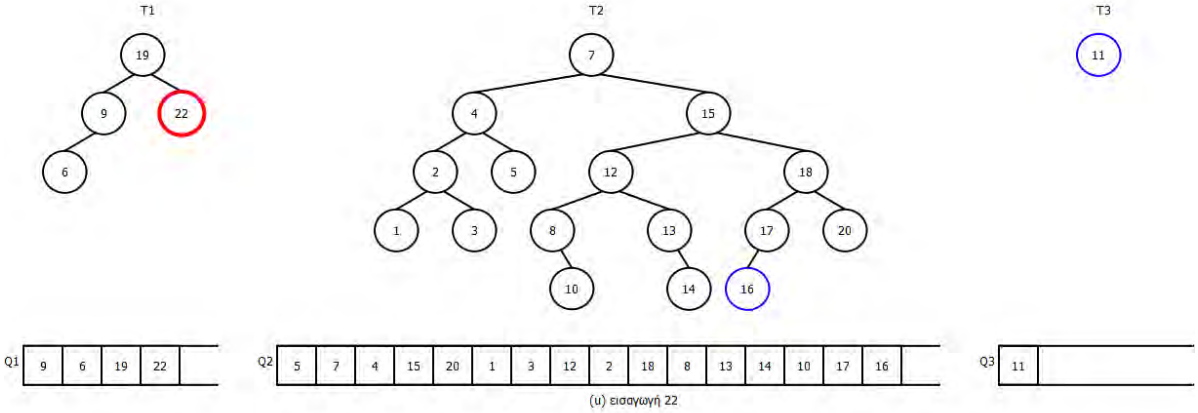

(r) εισαγωγή 9



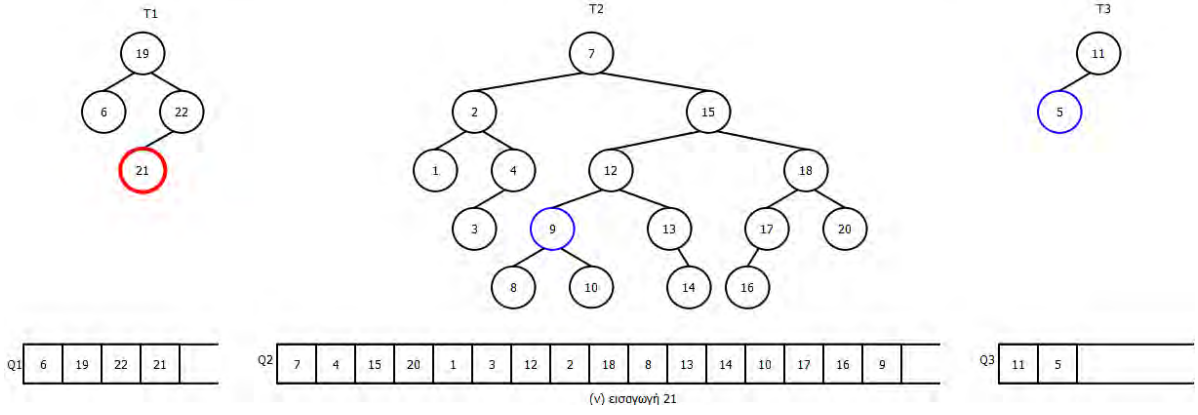

(s) εισαγωγή 6




(t) εισαγωγή 19




(u) εισαγωγή 22

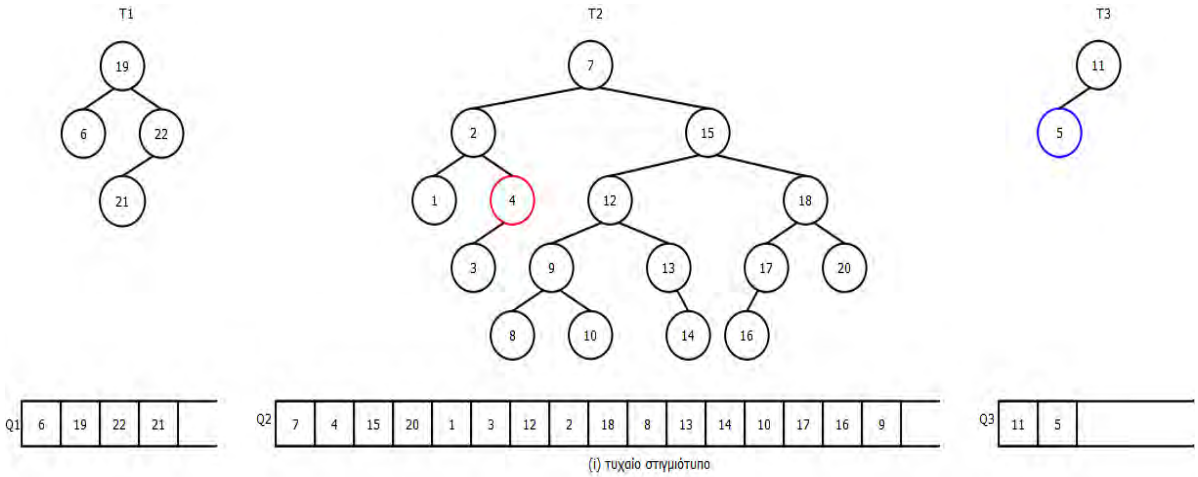



(v) εισαγωγή 21

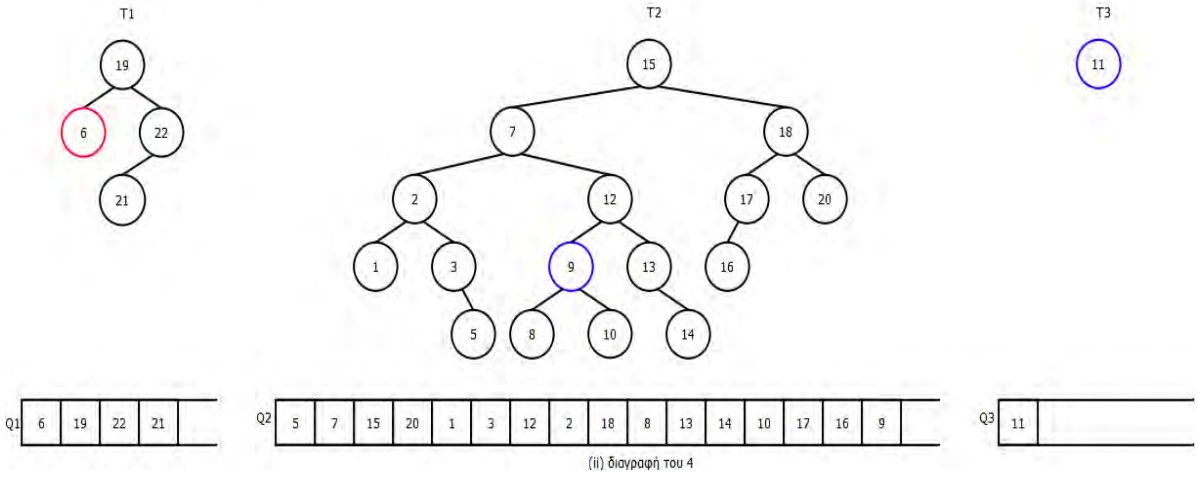
## 3.3 Διαγραφή

**Διαγραφές (deletions)** πραγματοποιούνται αναζητώντας το στοιχείο και διαγράφοντας το από το δέντρο (και την ουρά) στο οποίο βρέθηκε. Μία τέτοια εναλλαγή γεμίζει το κενό που δημιουργήθηκε από το αφαιρούμενο στοιχείο ανεβάζοντας στοιχεία από τις κατώτερες δομές. Στην συνέχεια απεικονίζουμε διαδοχικές διαγραφές στο δέντρο του προηγούμενου παραδείγματος των εξής έξι στοιχείων: (i)4,(ii)6,(iii)20,(iv)1,(v)9,(vi)15

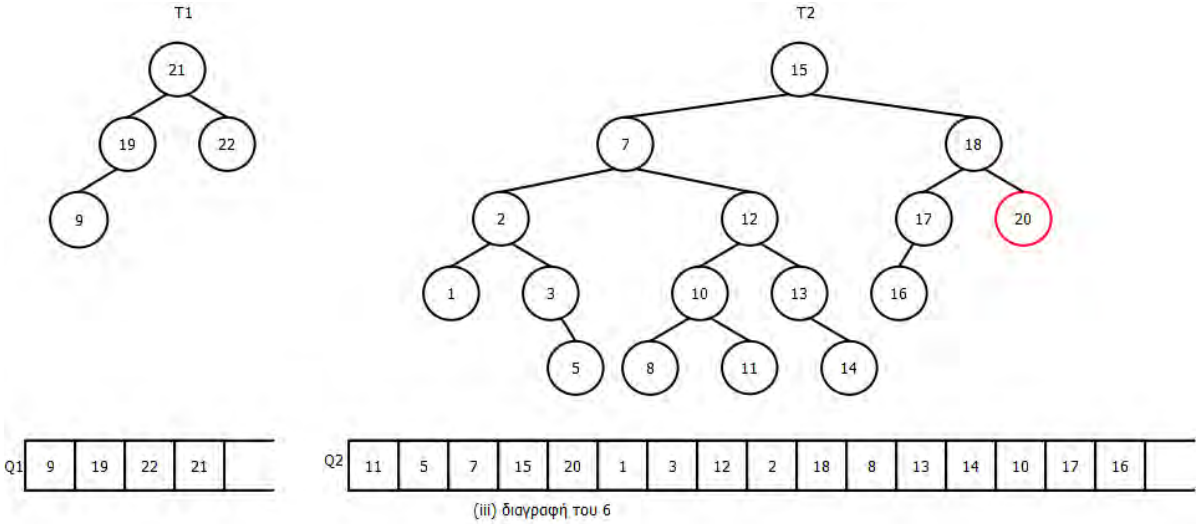
Με κόκκινο χρώμα αναπαριστούμε τον κόμβο που θα διαγραφεί σε επόμενο βήμα, και με μπλε τους κόμβους που θα προαχθούν σε προηγούμενα δέντρα.



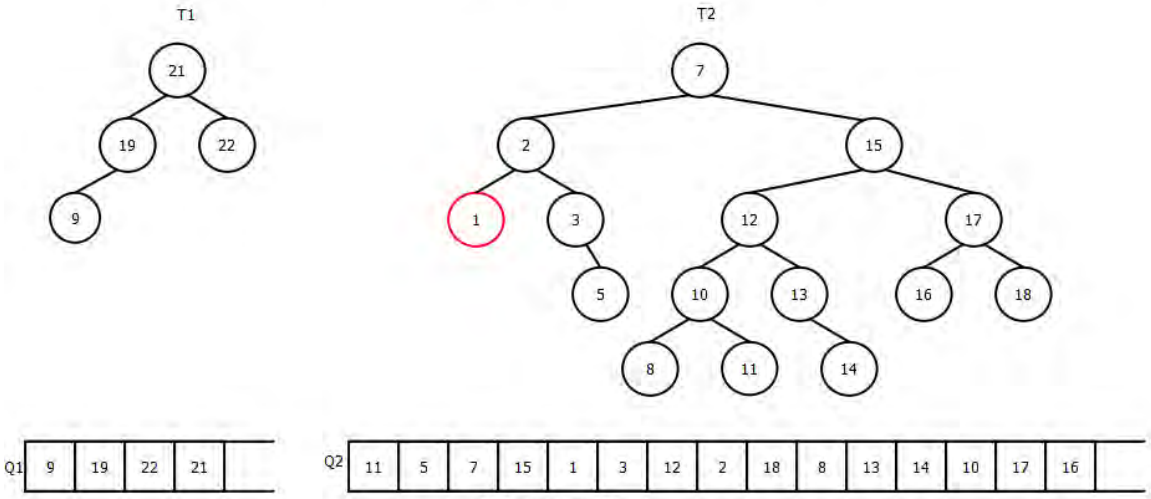
(i) τυχαίο στιγμιότυπο



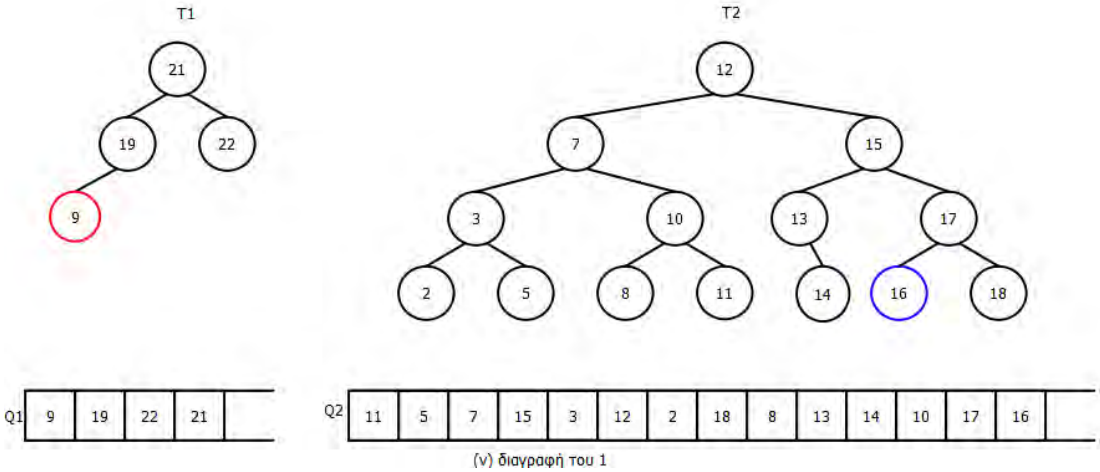
(ii) διαγραφή του 4



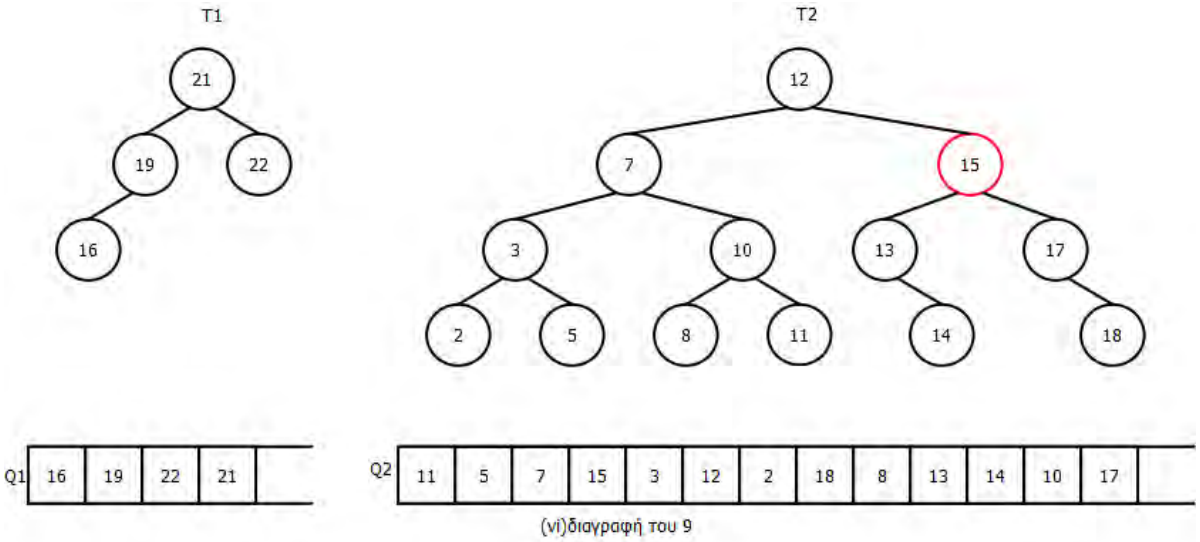

(iii) διαγραφή του 6



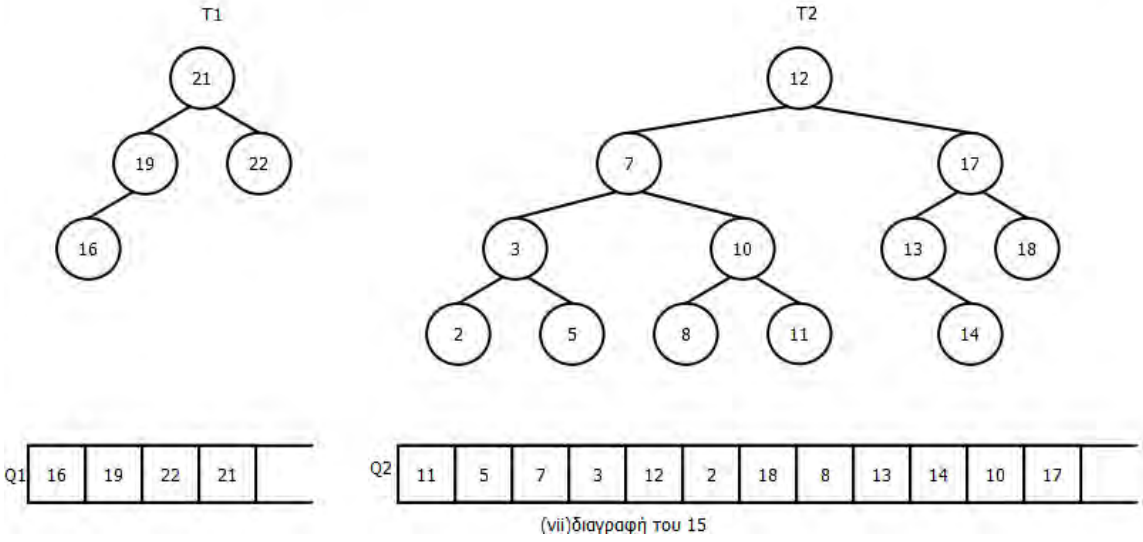

(iv) διαγραφή του 20




(v) διαγραφή του 1



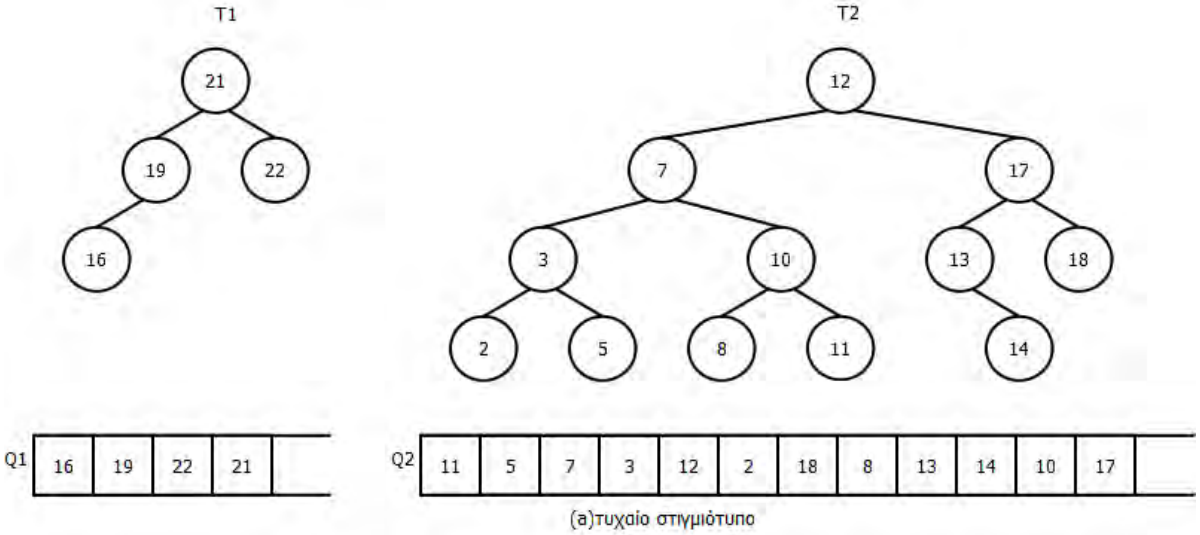

(vi)διαγραφή του 9



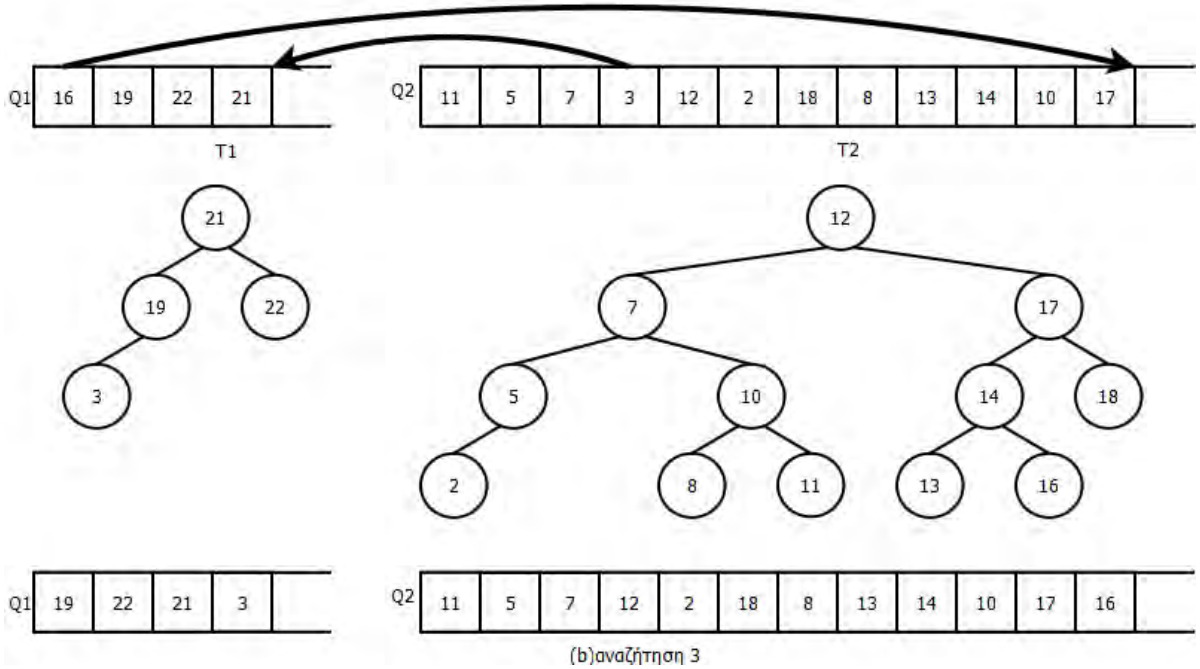

(vii)διαγραφή του 15

## 3.4 Αναζήτηση

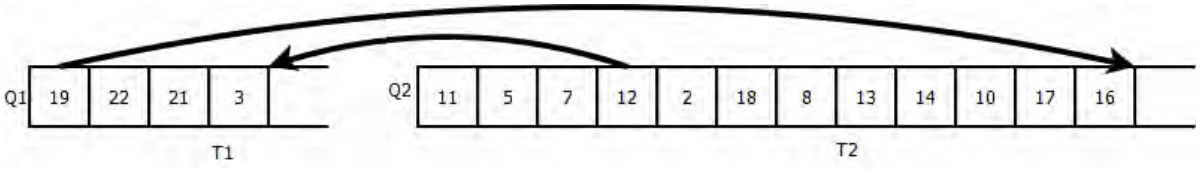
Τέλος, μία **αναζήτηση (search)** πραγματοποιείται αναζητώντας διαδοχικά τα δέντρα T1,T2,…,Tk μέχρι να βρεθεί το στοιχείο. Στη συνέχεια αυτό το στοιχείο **αφαιρείται** από το δέντρο και τη ουρά στο οποίο βρέθηκε και **εισάγεται** στο T1 και Q1, όπως περιγράφηκε παραπάνω. Εκτελώντας αυτή την εναλλαγή, διασφαλίζουμε ότι τα στοιχεία που έχουν αναζητηθεί πρόσφατα βρίσκονται στις αρχικές δομές και άρα θα βρεθούν πιο γρήγορα σε επόμενες αναζητήσεις.



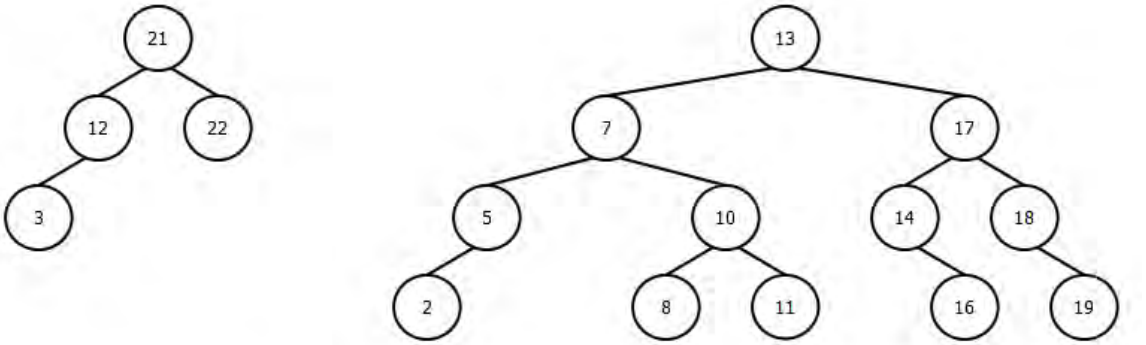

(a)τυχαίο στιγμιότυπο

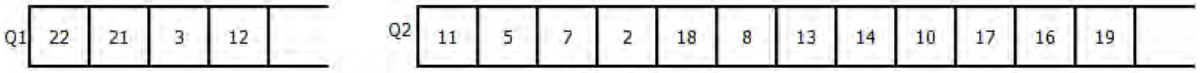



(b)αναζήτηση 3

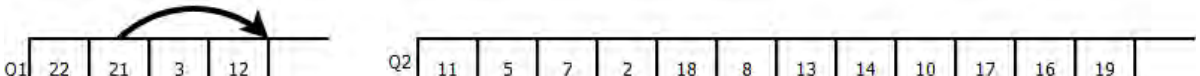



T1 T2

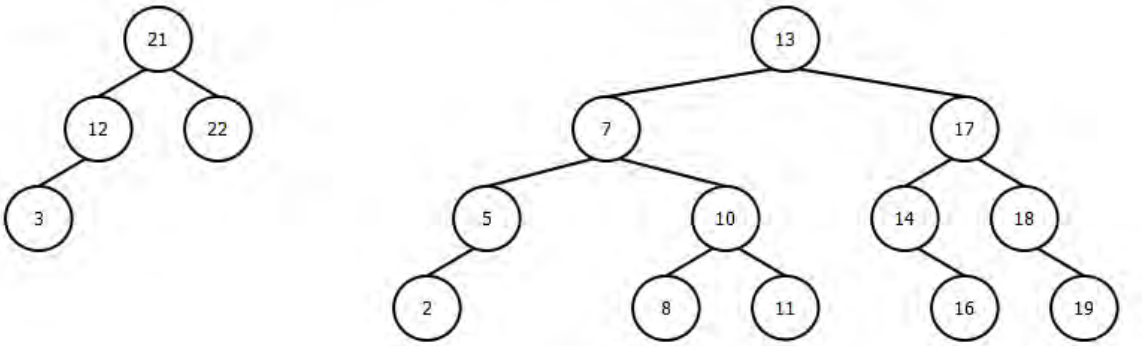


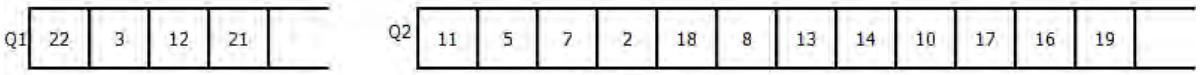




(c)αναζήτηση 12




T1 T2




(d)αναζήτηση 21

## 3.5 Ανάλυση

Το working set structure αποδείχτηκε από τον Iacono ότι έχει κόστος εισαγωγής και διαγραφής O(log n) και κόστος αναζήτησης O(log t(x)), όπου x είναι το κλειδί το οποίο έχει αναζητηθεί και t(x) είναι **1)** ο αριθμός των ξεχωριστών αιτήσεων που έχουν γίνει από όταν το x είχε αναζητηθεί τελευταία φορά ή **2) n** εάν το x δεν έχει αναζητηθεί ακόμα. Για να δούμε ότι το κόστος αναζήτησης είναι O(log t(x)), θυμηθείτε ότι εάν το x έχει βρεθεί στο Ti, θα πρέπει να έχει εξαχθεί από την ουρά $Q_{i-1}$ σε κάποιο σημείο. Εάν έχει γίνει αυτό, τότε $2^{i-1}$ προσβάσεις σε στοιχεία άλλα εκτός από το x έχουν γίνει από την τελευταία πρόσβαση στο x, και άρα $t(x) \geq 2^{i-1}$. Εφόσον ο χρόνος αναζήτησης για το x εξαρτάται από τον χρόνο αναζήτησης στο δέντρο στο οποίο βρέθηκε, το κόστος είναι: $O(\log 2^{2^i}) = O(\log t(x))$.

# ΚΕΦΑΛΑΙΟ 4

## 4. Παραλλαγή του Αλγορίθμου

### 4.1 Γενική Περιγραφή

Εδώ θα παρουσιάσουμε μια απλή χρήση της τυχαιότητας (randomization), ώστε να αφαιρέσουμε τις ουρές από την υλοποίηση της δομής. Αντί να βασιζόμαστε στις ουρές για να γνωρίζουμε ποιο είναι το αρχαιότερο στοιχείο στο δέντρο για την διαδικασία της εναλλαγής, εδώ απλά διαλέγουμε ένα **τυχαίο** στοιχείο και του συμπεριφερόμαστε σαν να ήταν το στοιχείο που αφαιρέσαμε από την κεφαλή της ουράς.

**Λήμμα [1]:** Το αναμενόμενο κόστος αναζήτησης στον νέο αλγόριθμο είναι

O(log t(x)).

**Απόδειξη :** Έστω το στοιχείο x και t=t(x) ο αριθμός των ξεχωριστών προσπελάσεων από την στιγμή που το x προσπελάστηκε τελευταία φορά. Υποθέτουμε ότι το x είναι στο δέντρο $T_i$ και πραγματοποιείται μια ακολουθία προσπελάσεων μεγέθους t. Εφόσον το $T_i$ έχει μέγεθος $2^{2^i}$, η πιθανότητα να μην έχει μετακινηθεί από το $T_i$, κατά την διάρκεια αυτών των προσβάσεων, είναι:

$$\Pr\{\text{να μην μετακινηθεί το x από το } T_i \text{ μετά από t προσπελάσεις}\} \geq (1 - \frac{1}{2^{2^i}})^t =$$

$$= (1 - \frac{1}{2^{2^i}})^{t\frac{2^{2^i}}{2^{2^i}}}$$

$$\geq (\frac{1}{4})^{\frac{t}{2^{2^i}}}$$

Τώρα εάν το x ανήκει στο $T_i$, θα πρέπει να έχει επιλεγεί για εκτόπιση από το δέντρο $T_{i-1}$ κάποια στιγμή. Πιθανότητα το x να είναι τουλάχιστον στο i-στο δέντρο είναι το πολύ:

$$\Pr\{\text{x ανήκει στο } T_j \text{ , για } j>i\} \leq 1 - \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{i-1}}}}$$

Ένα άνω όριο τη προσέγγισης E[S] του κόστους αναζήτησης είναι:

$$E[S] = \sum_{i=1}^{k} O(\log |T_i|) \times \Pr\{x \in T_i\}$$

$$\leq \sum_{i=1}^{k} O(\log |T_i|) \times \Pr\{x \in T_j \text{ , } \gamma\iota\alpha\ j \geq i\}$$

$$\leq \sum_{i=1}^{k} O(\log(2^{2^i})) \left(1- \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{i-1}}}}\right)$$

$$= \sum_{i=1}^{k} O(2^i) \left(1 - \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{i-1}}}}\right)$$

$$= \sum_{i=1}^{\lfloor \log\log t \rfloor} O(2^i) \left(1 - \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{i-1}}}}\right) + \sum_{i=1+\lfloor \log\log t \rfloor}^{k} O(2^i) \left(1- \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{i-1}}}}\right)$$

$$= O(\log t) + \sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^{i+\lfloor \log\log t \rfloor}) \left(1- \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{(i+\lfloor \log\log t \rfloor)-1}}}}\right)$$

$$= O(\log t) + O(\log t) \sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^i) \left(1- \left(\frac{1}{4}\right)^{\frac{t}{2^{2^{(i+\lfloor \log\log t \rfloor)-1}}}}\right)$$

$$\leq O(\log t) + O(\log t) \sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^i) \left(1- \left(\frac{1}{4}\right)^{\frac{t}{2^{2^i log t}}}\right)$$

$$= O(\log t) + O(\log t) \sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^i) \left(1- \left(\frac{1}{4}\right)^{\frac{1}{t^{2^i-1}}}\right)$$

Άρα, αρκεί να δείξουμε ότι το εναπομείναν άρθροισμα έχει κόστος O(1). Θα υποθέσουμε ότι $t \geq 2$, εφόσον αλλιώς το x θα μπορούσε να βρίσκεται το πολύ μέχρι το δεύτερο δέντρο, και θα μπορούσε να βρεθεί σε χρόνο O(1). Λαμβάνοντας υπόψιν μόνο το το άρθροισμα, έχουμε:

$\sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^i)\,(1-(\frac{1}{4})^{\frac{t}{t^{2^i-1}}}) \le \sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^i)\,(1-(\frac{1}{4})^{\frac{1}{2^{2^i-1}}})$

$\le \sum_{i=1}^{k-\lfloor \log\log t \rfloor} O(2^i)\,(1-(\frac{1}{16})^{\frac{1}{2^{2^i}}})$

$\le \sum_{i=1}^{\infty} O(2^i)\,(1-(\frac{1}{16})^{\frac{1}{2^{2^i}}})$

Μένει να δείξουμε ότι το άπειρο άρθροισμα είναι φραγμένο από μία φθίνουσα γεωμετρική σειρά (και άρα είναι σταθερό). Ο ρυθμός των συνεχόμενων όρων είναι:

$$\lim_{i\to\infty} \frac{2^{i+1}\,(1-(\frac{1}{16})^{\frac{1}{2^{2^{i+1}}}})}{2^i\,(1-(\frac{1}{16})^{\frac{1}{2^{2^i}}})} = \qquad 2\lim_{i\to\infty} \frac{(1-(\frac{1}{16})^{\frac{1}{2^{2^{i+1}}}})}{(1-(\frac{1}{16})^{\frac{1}{2^{2^i}}})}$$

Αντικαθιστώντας $u = \frac{1}{2^{2^i}}$, βρίσκουμε ότι $\frac{1}{2^{2^{i+1}}} = u^2$

Παρατηρούμε ότι όσο $i \to \infty$ , έχουμε $u \to 0$

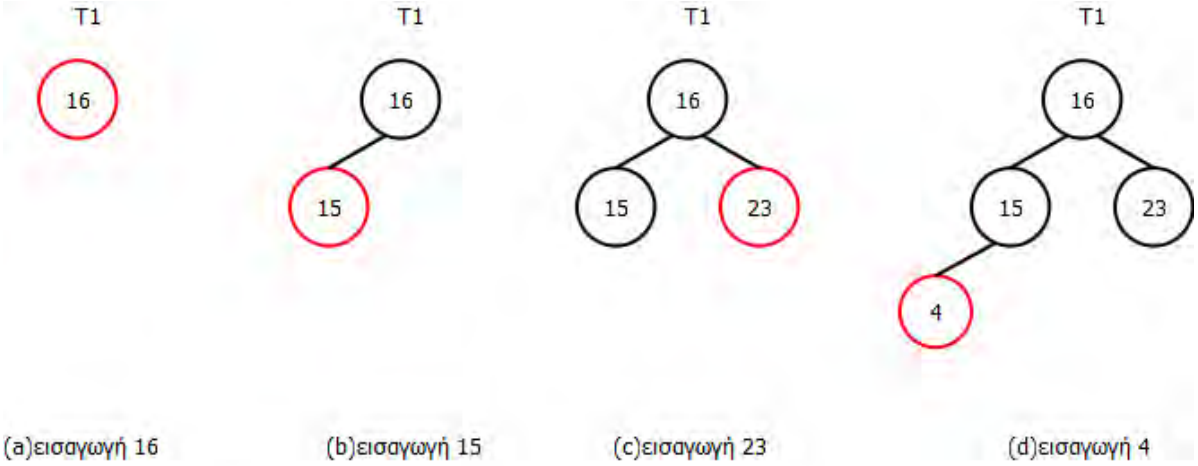
Άρα $2\lim_{i\to\infty} \frac{(1-(\frac{1}{16})^{\frac{1}{2^{2^{i+1}}}})}{(1-(\frac{1}{16})^{\frac{1}{2^{2^i}}})} = 2\lim_{u\to 0} \frac{(1-(\frac{1}{16})^{u^2})}{(1-(\frac{1}{16})^{u})} = 2\lim_{u\to 0} \frac{8(\frac{1}{16})^{u^2}\,u\ln 2}{4(\frac{1}{16})^{u}\ln 2} = 0$

Έτσι, ο ρυθμός των συνεχόμενων όρων είναι O(1) και άρα η σειρά είναι φραγμένη από μία φθίνουσα γεωμετρική σειρά.

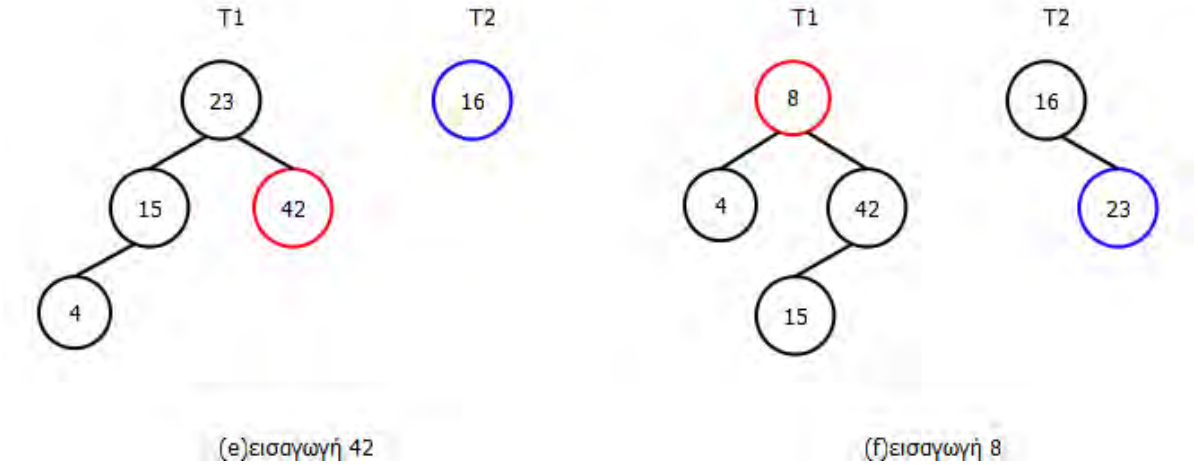
## 4.2 Εισαγωγή

Κατά την εισαγωγή, εφόσον έχει συμπληρωθεί ο αριθμός των στοιχείων που χωράει το κάθε δέντρο, σε επόμενο δέντρο εκτοπίζεται ένα **τυχαίο** στοιχείο. στο παράδειγμα που ακολουθεί παρουσιάζουμε την διαδοχική εισαγωγή των εξής στοιχείων: 16,15,23,4,42,8. Παρατηρήστε ότι κατά την εισαγωγή του 8, σε επόμενο δέντρο εκτοπίζεται το 23 αν και το 15 είναι «αρχαιότερο».

Με κόκκινο χρώμα αναπαραστούμε τους κόμβους που εισήχθησαν και με μπλε τους κόμβους που εκτοπίστηκαν σε επόμενα δέντρα.



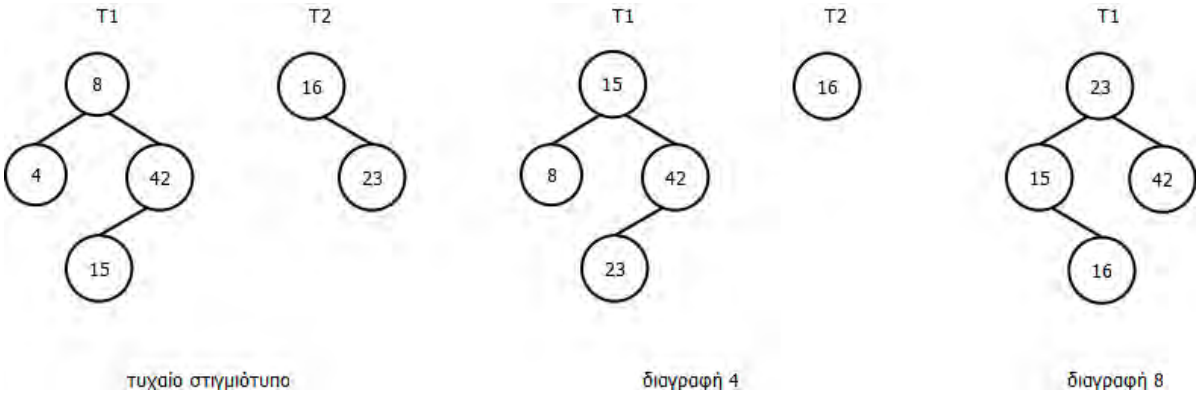

(a)εισαγωγή 16 (b)εισαγωγή 15 (c)εισαγωγή 23 (d)εισαγωγή 4




(e)εισαγωγή 42 (f)εισαγωγή 8

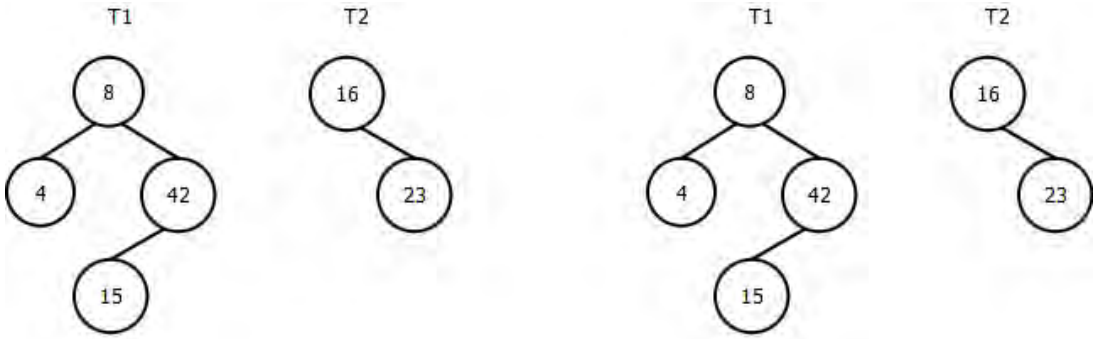
## 4.3 Διαγραφή

Όπως και στην εισαγωγή, έτσι και εδώ, όταν διαγράφουμε ένα στοιχείο, το κενό που δημιουργείται στο δέντρο από το οποίο διαγράφηκε, καλύπτεται από ένα **τυχαίο** στοιχείο του επόμενου δέντρου. Αυτό συμβαίνει επαναληπτικά σε όλα τα επόμενα δέντρα. Στη συνέχεια παραθέτουμε ένα παράδειγμα 2 διαγραφών στο δέντρο του προηγούμενου παραδείγματος. Παρατηρείστε ότι κατά την διαγραφή του 4, τυχαία το 23 προωθείται στο πρώτο δέντρο.



## 4.4 Αναζήτηση

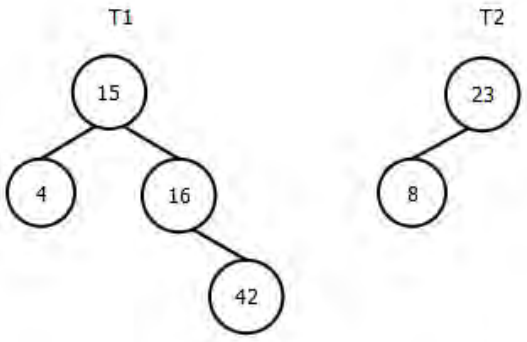
Τέλος, και στην αναζήτηση ισχύουν οι ίδιοι κανόνες. Το κλειδί το όποίο αναζητείται μεταφέρεται στο πρώτο δέντρο και στη συνέχεια ένα **τυχαίο** στοιχείο του πρώτου δέντρου εκτοπίζεται στο επόμενο. Το ίδιο συμβαίνει επαναληπτικά για όλα τα επόμενα δέντρα. Στο παράδειγμα που ακολουθεί, κατά την αναζήτηση του 4 δεν έχουμε καμία αλλαγή, εφόσον το 4 βρίσκεται ήδη στο πρώτο δέντρο. Αντίθετα, κατά την αναζήτηση του 16, το 16 μεταφέρεται στο πρώτο δέντρο και τυχαία επιλέγεται το 8 για να εκτοπιστεί στο επόμενο δέντρο.




(a)τυχαίο στιγμιότυπο




(b)αναζητηση 4




(c)αναζητηση 16

# ΚΕΦΑΛΑΙΟ 5

## 5. Πειραματική αξιολόγηση

### 5.1 Γενική Περιγραφή

Για να επαληθεύσουμε τους θεωρητικούς χρόνους των πράξεων εισαγωγής, διαγραφής και αναζήτησης θα διεξάγουμε σενάρια πειραμάτων. Για το σκοπό αυτό θα χρησιμοποιήσουμε μετρητές (counters) τους οποίους θα αυξάνουμε με την ολοκλήρωση κάθε στοιχειώδους πράξης, έτσι ώστε να υπολογίσουμε το συνολικό κόστος ανά πράξη. Τέτοιες στοιχειώδεις πράξεις είναι η **δημιουργία** και **διαγραφή** κόμβου, καθώς και η **εναλλαγή** δείκτη όταν διατρέχουμε το δέντρο.

### 5.2 Εισαγωγή στην πρώτη εκδοχή του αλγορίθμου

Σε αυτό το πείραμα θα εισάγουμε 30.000 τυχαία κλειδιά. Οι χρόνοι εισαγωγής , ανά 5.000 εισαγωγές , φαίνονται στον παρακάτω πίνακα:

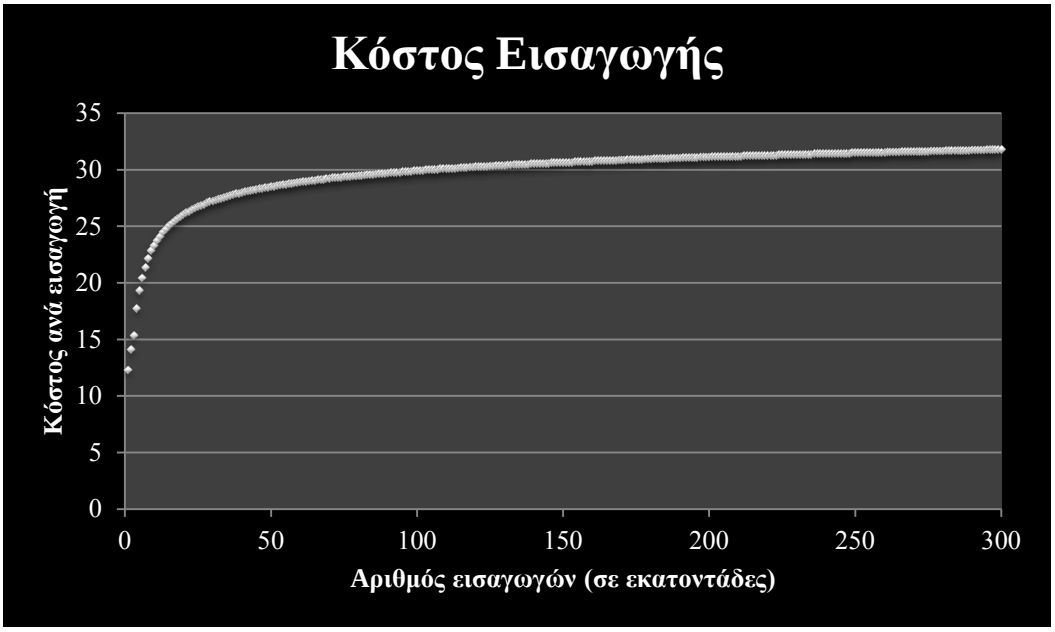
| | Χρόνος εισαγωγής |
|---|---|
| 5.000 | 0,16 |
| 10.000 | 0,32 |
| 15.000 | 0,48 |
| 20.000 | 0,66 |
| 25.000 | 0,83 |
| 30.000 | 1,01 |

Το συνολικό κόστος ανά πράξη φαίνεται στον πίνακα που ακολουθεί. Για ευκολία παραθέτουμε ανά 100 εισαγωγές.

| | | | | | |
|---|---|---|---|---|---|
| **100** | *12.340* | **5100** | *28.607* | **10100** | *29.988* |
| **200** | *14.195* | **5200** | *28.649* | **10200** | *30.005* |
| **300** | *15.417* | **5300** | *28.695* | **10300** | *30.026* |
| **400** | *17.745* | **5400** | *28.741* | **10400** | *30.044* |
| **500** | *19.326* | **5500** | *28.771* | **10500** | *30.063* |
| **600** | *20.482* | **5600** | *28.809* | **10600** | *30.079* |
| **700** | *21.424* | **5700** | *28.841* | **10700** | *30.093* |
| **800** | *22.200* | **5800** | *28.871* | **10800** | *30.111* |
| **900** | *22.873* | **5900** | *28.912* | **10900** | *30.126* |
| **1000** | *23.369* | **6000** | *28.954* | **11000** | *30.148* |
| **1100** | *23.786* | **6100** | *28.989* | **11100** | *30.162* |
| **1200** | *24.186* | **6200** | *29.024* | **11200** | *30.180* |
| **1300** | *24.535* | **6300** | *29.051* | **11300** | *30.192* |
| **1400** | *24.810* | **6400** | *29.083* | **11400** | *30.209* |
| **1500** | *25.083* | **6500** | *29.121* | **11500** | *30.227* |
| **1600** | *25.346* | **6600** | *29.153* | **11600** | *30.245* |
| **1700** | *25.547* | **6700** | *29.190* | **11700** | *30.263* |
| **1800** | *25.741* | **6800** | *29.221* | **11800** | *30.279* |
| **1900** | *25.932* | **6900** | *29.255* | **11900** | *30.295* |
| **2000** | *26.102* | **7000** | *29.284* | **12000** | *30.309* |
| **2100** | *26.259* | **7100** | *29.318* | **12100** | *30.320* |
| **2200** | *26.400* | **7200** | *29.344* | **12200** | *30.332* |
| **2300** | *26.540* | **7300** | *29.368* | **12300** | *30.348* |
| **2400** | *26.655* | **7400** | *29.393* | **12400** | *30.359* |
| **2500** | *26.774* | **7500** | *29.418* | **12500** | *30.371* |
| **2600** | *26.874* | **7600** | *29.445* | **12600** | *30.387* |
| **2700** | *27.007* | **7700** | *29.467* | **12700** | *30.402* |
| **2800** | *27.122* | **7800** | *29.491* | **12800** | *30.414* |
| **2900** | *27.213* | **7900** | *29.513* | **12900** | *30.429* |
| **3000** | *27.291* | **8000** | *29.537* | **13000** | *30.445* |
| **3100** | *27.373* | **8100** | *29.563* | **13100** | *30.463* |
| **3200** | *27.461* | **8200** | *29.584* | **13200** | *30.472* |
| **3300** | *27.528* | **8300** | *29.605* | **13300** | *30.488* |
| **3400** | *27.617* | **8400** | *29.626* | **13400** | *30.499* |
| **3500** | *27.709* | **8500** | *29.650* | **13500** | *30.513* |
| **3600** | *27.776* | **8600** | *29.672* | **13600** | *30.525* |
| **3700** | *27.838* | **8700** | *29.693* | **13700** | *30.534* |
| **3800** | *27.903* | **8800** | *29.716* | **13800** | *30.548* |
| **3900** | *27.980* | **8900** | *29.736* | **13900** | *30.561* |
| **4000** | *28.051* | **9000** | *29.759* | **14000** | *30.572* |
| **4100** | *28.102* | **9100** | *29.783* | **14100** | *30.586* |
| **4200** | *28.155* | **9200** | *29.804* | **14200** | *30.598* |
| **4300** | *28.207* | **9300** | *29.816* | **14300** | *30.612* |
| **4400** | *28.265* | **9400** | *29.840* | **14400** | *30.626* |
| **4500** | *28.321* | **9500** | *29.860* | **14500** | *30.638* |
| **4600** | *28.371* | **9600** | *29.884* | **14600** | *30.650* |
| **4700** | *28.416* | **9700** | *29.908* | **14700** | *30.661* |
| **4800** | *28.463* | **9800** | *29.933* | **14800** | *30.671* |
| **4900** | *28.514* | **9900** | *29.956* | **14900** | *30.680* |
| **5000** | *28.560* | **10000** | *29.972* | **15000** | *30.691* |

| | | | | | |
|---|---|---|---|---|---|
| **15100** | *30.703* | **20100** | *31.189* | **25100** | *31.546* |
| **15200** | *30.715* | **20200** | *31.195* | **25200** | *31.553* |
| **15300** | *30.725* | **20300** | *31.204* | **25300** | *31.560* |
| **15400** | *30.738* | **20400** | *31.210* | **25400** | *31.567* |
| **15500** | *30.747* | **20500** | *31.218* | **25500** | *31.573* |
| **15600** | *30.760* | **20600** | *31.228* | **25600** | *31.579* |
| **15700** | *30.775* | **20700** | *31.237* | **25700** | *31.586* |
| **15800** | *30.787* | **20800** | *31.243* | **25800** | *31.591* |
| **15900** | *30.797* | **20900** | *31.251* | **25900** | *31.598* |
| **16000** | *30.810* | **21000** | *31.258* | **26000** | *31.604* |
| **16100** | *30.821* | **21100** | *31.263* | **26100** | *31.612* |
| **16200** | *30.833* | **21200** | *31.270* | **26200** | *31.618* |
| **16300** | *30.843* | **21300** | *31.275* | **26300** | *31.625* |
| **16400** | *30.853* | **21400** | *31.283* | **26400** | *31.631* |
| **16500** | *30.863* | **21500** | *31.290* | **26500** | *31.635* |
| **16600** | *30.873* | **21600** | *31.300* | **26600** | *31.642* |
| **16700** | *30.883* | **21700** | *31.308* | **26700** | *31.647* |
| **16800** | *30.894* | **21800** | *31.314* | **26800** | *31.652* |
| **16900** | *30.903* | **21900** | *31.319* | **26900** | *31.656* |
| **17000** | *30.913* | **22000** | *31.328* | **27000** | *31.662* |
| **17100** | *30.922* | **22100** | *31.334* | **27100** | *31.666* |
| **17200** | *30.930* | **22200** | *31.342* | **27200** | *31.671* |
| **17300** | *30.942* | **22300** | *31.350* | **27300** | *31.678* |
| **17400** | *30.950* | **22400** | *31.357* | **27400** | *31.684* |
| **17500** | *30.960* | **22500** | *31.367* | **27500** | *31.691* |
| **17600** | *30.973* | **22600** | *31.374* | **27600** | *31.698* |
| **17700** | *30.983* | **22700** | *31.381* | **27700** | *31.702* |
| **17800** | *30.992* | **22800** | *31.391* | **27800** | *31.706* |
| **17900** | *31.001* | **22900** | *31.398* | **27900** | *31.712* |
| **18000** | *31.010* | **23000** | *31.405* | **28000** | *31.718* |
| **18100** | *31.016* | **23100** | *31.412* | **28100** | *31.725* |
| **18200** | *31.027* | **23200** | *31.420* | **28200** | *31.731* |
| **18300** | *31.038* | **23300** | *31.427* | **28300** | *31.738* |
| **18400** | *31.047* | **23400** | *31.433* | **28400** | *31.743* |
| **18500** | *31.056* | **23500** | *31.438* | **28500** | *31.750* |
| **18600** | *31.063* | **23600** | *31.444* | **28600** | *31.756* |
| **18700** | *31.071* | **23700** | *31.452* | **28700** | *31.761* |
| **18800** | *31.082* | **23800** | *31.457* | **28800** | *31.766* |
| **18900** | *31.091* | **23900** | *31.463* | **28900** | *31.773* |
| **19000** | *31.097* | **24000** | *31.471* | **29000** | *31.778* |
| **19100** | *31.106* | **24100** | *31.477* | **29100** | *31.784* |
| **19200** | *31.115* | **24200** | *31.486* | **29200** | *31.790* |
| **19300** | *31.127* | **24300** | *31.492* | **29300** | *31.796* |
| **19400** | *31.135* | **24400** | *31.499* | **29400** | *31.802* |
| **19500** | *31.144* | **24500** | *31.506* | **29500** | *31.807* |
| **19600** | *31.153* | **24600** | *31.511* | **29600** | *31.813* |
| **19700** | *31.159* | **24700** | *31.516* | **29700** | *31.820* |
| **19800** | *31.166* | **24800** | *31.522* | **29800** | *31.825* |
| **19900** | *31.173* | **24900** | *31.529* | **29900** | *31.830* |
| **20000** | *31.181* | **25000** | *31.538* | **30000** | *31.835* |

Η γραφική παράσταση του αριθμού των εισαγωγών συναρτήσει του κόστους ανά εισαγωγή είναι:




5.3 Διαγραφή στην πρώτη εκδοχή του αλγορίθμου

Για να υπολογίσουμε το κόστος της διαγραφής, στο στιγμιότυπο του προηγούμενου παραδείγματος, που περιέχει 30.000 στοιχεία, θα διαγράψουμε τα 29.800 από αυτά σε τυχαία σειρά και θα υπολογίσουμε το κόστος ανά διαγραφή. Όπως και πριν, αυξάνουμε τους μετρητές στις στοιχειώδεις πράξεις και διαιρούμε με τον αριθμό των εναπομείναντων στοιχείων. Οι χρόνοι διαγραφής , ανά 5.000 εισαγωγές , φαίνονται στον παρακάτω πίνακα:

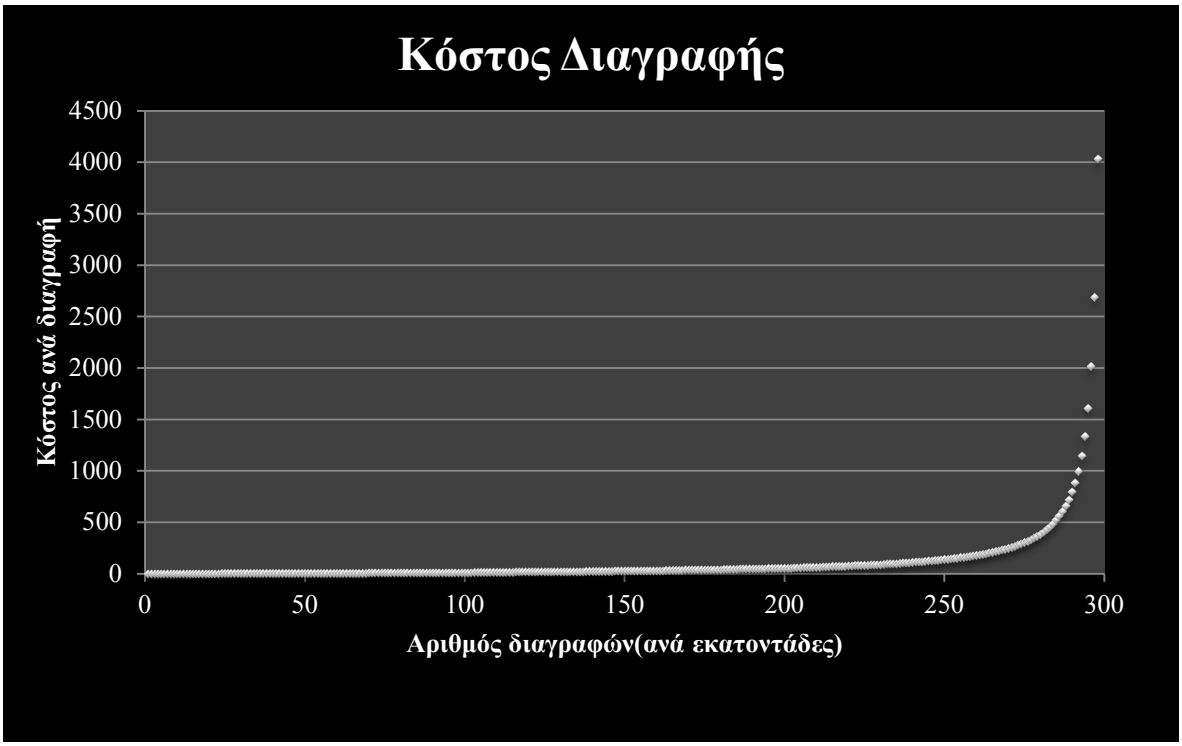
| | Χρόνος διαγραφής |
|---|---|
| 5.000 | 1,057 |
| 10.000 | 2,103 |
| 15.000 | 2,913 |
| 20.000 | 3,613 |
| 25.000 | 4,133 |

Τα ποσοστικά αποτελέσματα φαίνονται στους πίνακες που ακολουθούν.

| | | | | | |
|---|---|---|---|---|---|
| **100** | *0.10* | **5100** | *5.92* | **10100** | *14.59* |
| **200** | *0.20* | **5200** | *6.06* | **10200** | *14.81* |
| **300** | *0.29* | **5300** | *6.20* | **10300** | *15.03* |
| **400** | *0.39* | **5400** | *6.34* | **10400** | *15.25* |
| **500** | *0.49* | **5500** | *6.48* | **10500** | *15.48* |
| **600** | *0.59* | **5600** | *6.63* | **10600** | *15.70* |
| **700** | *0.69* | **5700** | *6.77* | **10700** | *15.93* |
| **800** | *0.79* | **5800** | *6.92* | **10800** | *16.16* |
| **900** | *0.90* | **5900** | *7.06* | **10900** | *16.39* |
| **1000** | *1.00* | **6000** | *7.21* | **11000** | *16.63* |
| **1100** | *1.10* | **6100** | *7.36* | **11100** | *16.87* |
| **1200** | *1.21* | **6200** | *7.52* | **11200** | *17.11* |
| **1300** | *1.31* | **6300** | *7.67* | **11300** | *17.35* |
| **1400** | *1.42* | **6400** | *7.82* | **11400** | *17.60* |
| **1500** | *1.52* | **6500** | *7.98* | **11500** | *17.84* |
| **1600** | *1.63* | **6600** | *8.13* | **11600** | *18.09* |
| **1700** | *1.74* | **6700** | *8.29* | **11700** | *18.35* |
| **1800** | *1.85* | **6800** | *8.45* | **11800** | *18.60* |
| **1900** | *1.96* | **6900** | *8.61* | **11900** | *18.86* |
| **2000** | *2.07* | **7000** | *8.77* | **12000** | *19.12* |
| **2100** | *2.18* | **7100** | *8.94* | **12100** | *19.39* |
| **2200** | *2.30* | **7200** | *9.10* | **12200** | *19.65* |
| **2300** | *2.41* | **7300** | *9.27* | **12300** | *19.92* |
| **2400** | *2.52* | **7400** | *9.44* | **12400** | *20.20* |
| **2500** | *2.64* | **7500** | *9.61* | **12500** | *20.48* |
| **2600** | *2.75* | **7600** | *9.78* | **12600** | *20.76* |
| **2700** | *2.87* | **7700** | *9.95* | **12700** | *21.04* |
| **2800** | *2.99* | **7800** | *10.12* | **12800** | *21.32* |
| **2900** | *3.10* | **7900** | *10.30* | **12900** | *21.61* |
| **3000** | *3.22* | **8000** | *10.47* | **13000** | *21.91* |
| **3100** | *3.34* | **8100** | *10.65* | **13100** | *22.20* |
| **3200** | *3.46* | **8200** | *10.83* | **13200** | *22.50* |
| **3300** | *3.58* | **8300** | *11.02* | **13300** | *22.81* |
| **3400** | *3.70* | **8400** | *11.20* | **13400** | *23.12* |
| **3500** | *3.83* | **8500** | *11.38* | **13500** | *23.43* |
| **3600** | *3.95* | **8600** | *11.57* | **13600** | *23.74* |
| **3700** | *4.08* | **8700** | *11.76* | **13700** | *24.06* |
| **3800** | *4.20* | **8800** | *11.95* | **13800** | *24.38* |
| **3900** | *4.33* | **8900** | *12.14* | **13900** | *24.71* |
| **4000** | *4.45* | **9000** | *12.33* | **14000** | *25.04* |
| **4100** | *4.58* | **9100** | *12.53* | **14100** | *25.37* |
| **4200** | *4.71* | **9200** | *12.72* | **14200** | *25.71* |
| **4300** | *4.84* | **9300** | *12.92* | **14300** | *26.05* |
| **4400** | *4.97* | **9400** | *13.13* | **14400** | *26.39* |
| **4500** | *5.10* | **9500** | *13.33* | **14500** | *26.75* |
| **4600** | *5.24* | **9600** | *13.54* | **14600** | *27.10* |
| **4700** | *5.37* | **9700** | *13.74* | **14700** | *27.46* |
| **4800** | *5.50* | **9800** | *13.95* | **14800** | *27.82* |
| **4900** | *5.64* | **9900** | *14.16* | **14900** | *28.20* |
| **5000** | *5.78* | **10000** | *14.38* | **15000** | *28.57* |

| | | | | | |
|---|---|---|---|---|---|
| **15100** | *28.95* | **20100** | *57.54* | **25100** | *143.31* |
| **15200** | *29.33* | **20200** | *58.41* | **25200** | *146.83* |
| **15300** | *29.72* | **20300** | *59.29* | **25300** | *150.51* |
| **15400** | *30.11* | **20400** | *60.19* | **25400** | *154.35* |
| **15500** | *30.51* | **20500** | *61.11* | **25500** | *158.35* |
| **15600** | *30.92* | **20600** | *62.04* | **25600** | *162.52* |
| **15700** | *31.33* | **20700** | *63.00* | **25700** | *166.90* |
| **15800** | *31.75* | **20800** | *63.98* | **25800** | *171.46* |
| **15900** | *32.17* | **20900** | *64.97* | **25900** | *176.26* |
| **16000** | *32.60* | **21000** | *66.00* | **26000** | *181.30* |
| **16100** | *33.03* | **21100** | *67.05* | **26100** | *186.61* |
| **16200** | *33.48* | **21200** | *68.12* | **26200** | *192.20* |
| **16300** | *33.93* | **21300** | *69.21* | **26300** | *198.09* |
| **16400** | *34.38* | **21400** | *70.33* | **26400** | *204.29* |
| **16500** | *34.84* | **21500** | *71.47* | **26500** | *210.83* |
| **16600** | *35.31* | **21600** | *72.65* | **26600** | *217.76* |
| **16700** | *35.78* | **21700** | *73.84* | **26700** | *225.13* |
| **16800** | *36.27* | **21800** | *75.07* | **26800** | *232.91* |
| **16900** | *36.76* | **21900** | *76.32* | **26900** | *241.21* |
| **17000** | *37.26* | **22000** | *77.61* | **27000** | *250.07* |
| **17100** | *37.76* | **22100** | *78.93* | **27100** | *259.53* |
| **17200** | *38.27* | **22200** | *80.28* | **27200** | *269.69* |
| **17300** | *38.79* | **22300** | *81.67* | **27300** | *280.56* |
| **17400** | *39.32* | **22400** | *83.09* | **27400** | *292.28* |
| **17500** | *39.86* | **22500** | *84.55* | **27500** | *304.94* |
| **17600** | *40.40* | **22600** | *86.05* | **27600** | *318.66* |
| **17700** | *40.95* | **22700** | *87.58* | **27700** | *333.56* |
| **17800** | *41.52* | **22800** | *89.17* | **27800** | *349.82* |
| **17900** | *42.09* | **22900** | *90.80* | **27900** | *367.60* |
| **18000** | *42.67* | **23000** | *92.47* | **28000** | *387.15* |
| **18100** | *43.26* | **23100** | *94.19* | **28100** | *408.71* |
| **18200** | *43.86* | **23200** | *95.97* | **28200** | *432.71* |
| **18300** | *44.47* | **23300** | *97.79* | **28300** | *459.49* |
| **18400** | *45.09* | **23400** | *99.68* | **28400** | *489.57* |
| **18500** | *45.72* | **23500** | *101.63* | **28500** | *523.66* |
| **18600** | *46.37* | **23600** | *103.63* | **28600** | *562.60* |
| **18700** | *47.03* | **23700** | *105.69* | **28700** | *607.47* |
| **18800** | *47.69* | **23800** | *107.81* | **28800** | *659.85* |
| **18900** | *48.37* | **23900** | *110.01* | **28900** | *721.74* |
| **19000** | *49.06* | **24000** | *112.28* | **29000** | *795.99* |
| **19100** | *49.76* | **24100** | *114.63* | **29100** | *886.62* |
| **19200** | *50.48* | **24200** | *117.05* | **29200** | *999.87* |
| **19300** | *51.20* | **24300** | *119.57* | **29300** | *1145.35* |
| **19400** | *51.94* | **24400** | *122.16* | **29400** | *1339.13* |
| **19500** | *52.69* | **24500** | *124.87* | **29500** | *1610.03* |
| **19600** | *53.46* | **24600** | *127.67* | **29600** | *2015.74* |
| **19700** | *54.24* | **24700** | *130.56* | **29700** | *2690.65* |
| **19800** | *55.05* | **24800** | *133.57* | **29800** | *4036.08* |
| **19900** | *55.86* | **24900** | *136.70* | | |
| **20000** | *56.70* | **25000** | *139.94* | | |

Η γραφική παράσταση του κόστους ανά διαγραφή είναι:




## 5.4 Εισαγωγή στην δεύτερη εκδοχή του αλγορίθμου

Εδώ εκτελούμε το ίδιο πείραμα, δηλαδή την εισαγωγή 30.000 στοιχείων, αλλά αυτή τη φορά στην βελτιωμένη εκδοχή του αλγορίθμου. Οι χρόνοι εισαγωγής ανά 5.000 στοιχεία φαίνονται στον παρακάτω πίνακα :

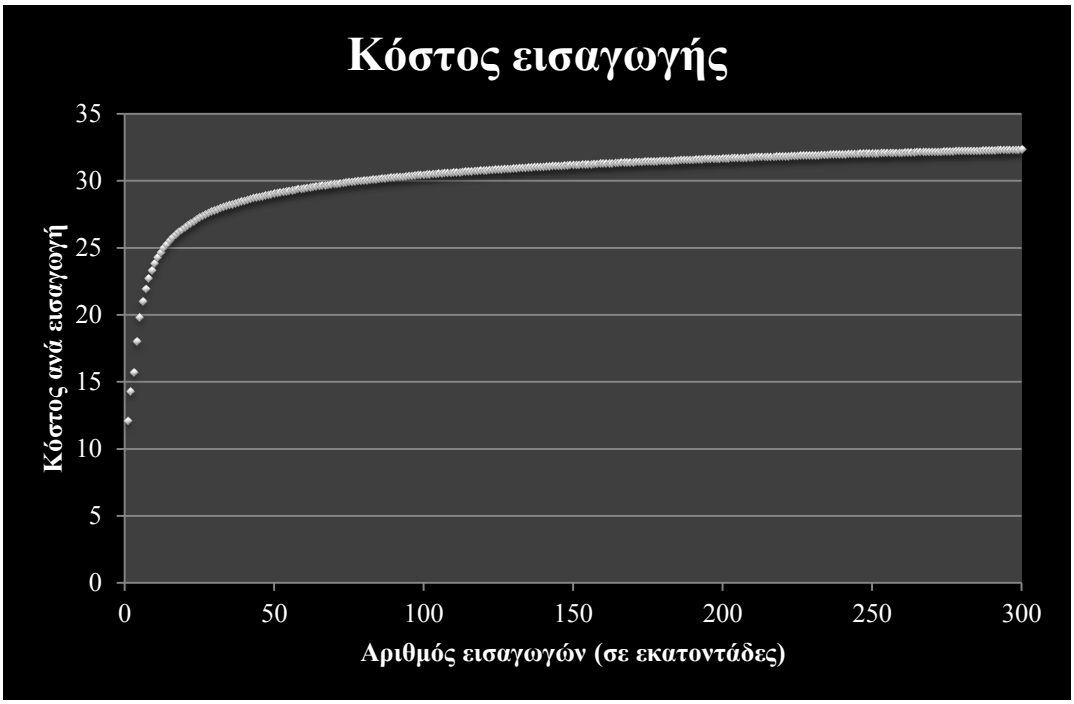
| | Χρόνος εισαγωγής |
|---|---|
| 5.000 | 0,17 |
| 10.000 | 0,31 |
| 15.000 | 0,45 |
| 20.000 | 0,59 |
| 25.000 | 0,73 |
| 30.000 | 0,87 |

Παρακάτω φαίνονται οι νέες μετρήσεις και η νέα γραφική παράσταση.

| | | | | | |
|---|---|---|---|---|---|
| **100** | 12,12 | **5100** | 29,115 | **10100** | 30,496 |
| **200** | 14,305 | **5200** | 29,158 | **10200** | 30,51 |
| **300** | 15,747 | **5300** | 29,195 | **10300** | 30,531 |
| **400** | 18,028 | **5400** | 29,237 | **10400** | 30,544 |
| **500** | 19,824 | **5500** | 29,278 | **10500** | 30,562 |
| **600** | 21,05 | **5600** | 29,319 | **10600** | 30,578 |
| **700** | 21,959 | **5700** | 29,359 | **10700** | 30,593 |
| **800** | 22,75 | **5800** | 29,394 | **10800** | 30,612 |
| **900** | 23,332 | **5900** | 29,431 | **10900** | 30,629 |
| **1000** | 23,866 | **6000** | 29,469 | **11000** | 30,646 |
| **1100** | 24,321 | **6100** | 29,508 | **11100** | 30,664 |
| **1200** | 24,684 | **6200** | 29,543 | **11200** | 30,679 |
| **1300** | 25,01 | **6300** | 29,581 | **11300** | 30,696 |
| **1400** | 25,284 | **6400** | 29,611 | **11400** | 30,713 |
| **1500** | 25,577 | **6500** | 29,641 | **11500** | 30,73 |
| **1600** | 25,817 | **6600** | 29,669 | **11600** | 30,743 |
| **1700** | 26,004 | **6700** | 29,696 | **11700** | 30,763 |
| **1800** | 26,212 | **6800** | 29,723 | **11800** | 30,776 |
| **1900** | 26,361 | **6900** | 29,756 | **11900** | 30,791 |
| **2000** | 26,535 | **7000** | 29,788 | **12000** | 30,802 |
| **2100** | 26,704 | **7100** | 29,816 | **12100** | 30,815 |
| **2200** | 26,855 | **7200** | 29,843 | **12200** | 30,832 |
| **2300** | 27,015 | **7300** | 29,869 | **12300** | 30,843 |
| **2400** | 27,158 | **7400** | 29,894 | **12400** | 30,859 |
| **2500** | 27,28 | **7500** | 29,924 | **12500** | 30,873 |
| **2600** | 27,398 | **7600** | 29,947 | **12600** | 30,889 |
| **2700** | 27,518 | **7700** | 29,972 | **12700** | 30,907 |
| **2800** | 27,639 | **7800** | 29,997 | **12800** | 30,919 |
| **2900** | 27,733 | **7900** | 30,023 | **12900** | 30,932 |
| **3000** | 27,816 | **8000** | 30,052 | **13000** | 30,947 |
| **3100** | 27,908 | **8100** | 30,077 | **13100** | 30,959 |
| **3200** | 27,984 | **8200** | 30,102 | **13200** | 30,974 |
| **3300** | 28,062 | **8300** | 30,123 | **13300** | 30,99 |
| **3400** | 28,136 | **8400** | 30,15 | **13400** | 31,004 |
| **3500** | 28,201 | **8500** | 30,175 | **13500** | 31,016 |
| **3600** | 28,285 | **8600** | 30,194 | **13600** | 31,027 |
| **3700** | 28,343 | **8700** | 30,22 | **13700** | 31,041 |
| **3800** | 28,414 | **8800** | 30,243 | **13800** | 31,056 |
| **3900** | 28,481 | **8900** | 30,267 | **13900** | 31,07 |
| **4000** | 28,549 | **9000** | 30,293 | **14000** | 31,083 |
| **4100** | 28,604 | **9100** | 30,315 | **14100** | 31,096 |
| **4200** | 28,666 | **9200** | 30,331 | **14200** | 31,109 |
| **4300** | 28,73 | **9300** | 30,346 | **14300** | 31,122 |
| **4400** | 28,786 | **9400** | 30,364 | **14400** | 31,134 |
| **4500** | 28,839 | **9500** | 30,387 | **14500** | 31,143 |
| **4600** | 28,887 | **9600** | 30,412 | **14600** | 31,155 |
| **4700** | 28,93 | **9700** | 30,43 | **14700** | 31,165 |
| **4800** | 28,982 | **9800** | 30,448 | **14800** | 31,175 |
| **4900** | 29,031 | **9900** | 30,466 | **14900** | 31,187 |
| **5000** | 29,072 | **10000** | 30,481 | **15000** | 31,2 |

| **15100** | 31,213 | **20100** | 31,697 | **25100** | 32,078 |
|---|---|---|---|---|---|
| **15200** | 31,227 | **20200** | 31,704 | **25200** | 32,085 |
| **15300** | 31,241 | **20300** | 31,712 | **25300** | 32,092 |
| **15400** | 31,25 | **20400** | 31,719 | **25400** | 32,096 |
| **15500** | 31,262 | **20500** | 31,728 | **25500** | 32,103 |
| **15600** | 31,273 | **20600** | 31,738 | **25600** | 32,108 |
| **15700** | 31,284 | **20700** | 31,747 | **25700** | 32,115 |
| **15800** | 31,295 | **20800** | 31,753 | **25800** | 32,121 |
| **15900** | 31,306 | **20900** | 31,763 | **25900** | 32,128 |
| **16000** | 31,314 | **21000** | 31,774 | **26000** | 32,134 |
| **16100** | 31,324 | **21100** | 31,782 | **26100** | 32,14 |
| **16200** | 31,337 | **21200** | 31,792 | **26200** | 32,148 |
| **16300** | 31,344 | **21300** | 31,799 | **26300** | 32,152 |
| **16400** | 31,355 | **21400** | 31,805 | **26400** | 32,158 |
| **16500** | 31,367 | **21500** | 31,813 | **26500** | 32,164 |
| **16600** | 31,376 | **21600** | 31,818 | **26600** | 32,17 |
| **16700** | 31,385 | **21700** | 31,826 | **26700** | 32,175 |
| **16800** | 31,393 | **21800** | 31,835 | **26800** | 32,18 |
| **16900** | 31,403 | **21900** | 31,843 | **26900** | 32,186 |
| **17000** | 31,414 | **22000** | 31,85 | **27000** | 32,189 |
| **17100** | 31,424 | **22100** | 31,858 | **27100** | 32,195 |
| **17200** | 31,434 | **22200** | 31,866 | **27200** | 32,199 |
| **17300** | 31,441 | **22300** | 31,876 | **27300** | 32,207 |
| **17400** | 31,448 | **22400** | 31,886 | **27400** | 32,213 |
| **17500** | 31,46 | **22500** | 31,893 | **27500** | 32,218 |
| **17600** | 31,47 | **22600** | 31,899 | **27600** | 32,223 |
| **17700** | 31,479 | **22700** | 31,908 | **27700** | 32,229 |
| **17800** | 31,488 | **22800** | 31,916 | **27800** | 32,234 |
| **17900** | 31,498 | **22900** | 31,923 | **27900** | 32,241 |
| **18000** | 31,507 | **23000** | 31,93 | **28000** | 32,247 |
| **18100** | 31,516 | **23100** | 31,938 | **28100** | 32,251 |
| **18200** | 31,528 | **23200** | 31,945 | **28200** | 32,255 |
| **18300** | 31,538 | **23300** | 31,953 | **28300** | 32,26 |
| **18400** | 31,55 | **23400** | 31,96 | **28400** | 32,266 |
| **18500** | 31,559 | **23500** | 31,969 | **28500** | 32,273 |
| **18600** | 31,568 | **23600** | 31,974 | **28600** | 32,278 |
| **18700** | 31,578 | **23700** | 31,979 | **28700** | 32,281 |
| **18800** | 31,587 | **23800** | 31,986 | **28800** | 32,285 |
| **18900** | 31,595 | **23900** | 31,996 | **28900** | 32,29 |
| **19000** | 31,604 | **24000** | 32,003 | **29000** | 32,295 |
| **19100** | 31,615 | **24100** | 32,009 | **29100** | 32,3 |
| **19200** | 31,624 | **24200** | 32,019 | **29200** | 32,306 |
| **19300** | 31,632 | **24300** | 32,025 | **29300** | 32,312 |
| **19400** | 31,641 | **24400** | 32,031 | **29400** | 32,32 |
| **19500** | 31,65 | **24500** | 32,039 | **29500** | 32,327 |
| **19600** | 31,659 | **24600** | 32,044 | **29600** | 32,332 |
| **19700** | 31,666 | **24700** | 32,051 | **29700** | 32,338 |
| **19800** | 31,672 | **24800** | 32,058 | **29800** | 32,345 |
| **19900** | 31,683 | **24900** | 32,066 | **29900** | 32,351 |
| **20000** | 31,688 | **25000** | 32,073 | **30000** | 32,356 |

Και η γραφική παράσταση είναι:




5.5 Διαγραφή στην δεύτερη εκδοχή του αλγορίθμου

Παρομοίως, από το αντίστοιχο σενάριο διαγραφής, δηλαδή τη διαγραφή 29.800 στοιχείων, στην δομή του προηγούμενου παραδείγματος, παίρνουμε τις εξής μετρήσεις και την γραφική παράσταση:

| | Χρόνος διαγραφής |
|---|---|
| 5.000 | 0,083 |
| 10.000 | 0,167 |
| 15.000 | 0,232 |
| 20.000 | 0,298 |
| 25.000 | 0,359 |

| | | | | | |
|---|---|---|---|---|---|
| **100** | *0.10* | **5100** | *5.92* | **10100** | *14.59* |
| **200** | *0.20* | **5200** | *6.06* | **10200** | *14.81* |
| **300** | *0.29* | **5300** | *6.20* | **10300** | *15.03* |
| **400** | *0.39* | **5400** | *6.34* | **10400** | *15.25* |
| **500** | *0.49* | **5500** | *6.48* | **10500** | *15.48* |
| **600** | *0.59* | **5600** | *6.63* | **10600** | *15.70* |
| **700** | *0.69* | **5700** | *6.77* | **10700** | *15.93* |
| **800** | *0.79* | **5800** | *6.92* | **10800** | *16.16* |
| **900** | *0.90* | **5900** | *7.06* | **10900** | *16.39* |
| **1000** | *1.00* | **6000** | *7.21* | **11000** | *16.63* |
| **1100** | *1.10* | **6100** | *7.36* | **11100** | *16.87* |
| **1200** | *1.21* | **6200** | *7.52* | **11200** | *17.11* |
| **1300** | *1.31* | **6300** | *7.67* | **11300** | *17.35* |
| **1400** | *1.42* | **6400** | *7.82* | **11400** | *17.60* |
| **1500** | *1.52* | **6500** | *7.98* | **11500** | *17.84* |
| **1600** | *1.63* | **6600** | *8.13* | **11600** | *18.09* |
| **1700** | *1.74* | **6700** | *8.29* | **11700** | *18.35* |
| **1800** | *1.85* | **6800** | *8.45* | **11800** | *18.60* |
| **1900** | *1.96* | **6900** | *8.61* | **11900** | *18.86* |
| **2000** | *2.07* | **7000** | *8.77* | **12000** | *19.12* |
| **2100** | *2.18* | **7100** | *8.94* | **12100** | *19.39* |
| **2200** | *2.30* | **7200** | *9.10* | **12200** | *19.65* |
| **2300** | *2.41* | **7300** | *9.27* | **12300** | *19.92* |
| **2400** | *2.52* | **7400** | *9.44* | **12400** | *20.20* |
| **2500** | *2.64* | **7500** | *9.61* | **12500** | *20.48* |
| **2600** | *2.75* | **7600** | *9.78* | **12600** | *20.76* |
| **2700** | *2.87* | **7700** | *9.95* | **12700** | *21.04* |
| **2800** | *2.99* | **7800** | *10.12* | **12800** | *21.32* |
| **2900** | *3.10* | **7900** | *10.30* | **12900** | *21.61* |
| **3000** | *3.22* | **8000** | *10.47* | **13000** | *21.91* |
| **3100** | *3.34* | **8100** | *10.65* | **13100** | *22.20* |
| **3200** | *3.46* | **8200** | *10.83* | **13200** | *22.50* |
| **3300** | *3.58* | **8300** | *11.02* | **13300** | *22.81* |
| **3400** | *3.70* | **8400** | *11.20* | **13400** | *23.12* |
| **3500** | *3.83* | **8500** | *11.38* | **13500** | *23.43* |
| **3600** | *3.95* | **8600** | *11.57* | **13600** | *23.74* |
| **3700** | *4.08* | **8700** | *11.76* | **13700** | *24.06* |
| **3800** | *4.20* | **8800** | *11.95* | **13800** | *24.38* |
| **3900** | *4.33* | **8900** | *12.14* | **13900** | *24.71* |
| **4000** | *4.45* | **9000** | *12.33* | **14000** | *25.04* |
| **4100** | *4.58* | **9100** | *12.53* | **14100** | *25.37* |
| **4200** | *4.71* | **9200** | *12.72* | **14200** | *25.71* |
| **4300** | *4.84* | **9300** | *12.92* | **14300** | *26.05* |
| **4400** | *4.97* | **9400** | *13.13* | **14400** | *26.39* |
| **4500** | *5.10* | **9500** | *13.33* | **14500** | *26.75* |
| **4600** | *5.24* | **9600** | *13.54* | **14600** | *27.10* |
| **4700** | *5.37* | **9700** | *13.74* | **14700** | *27.46* |
| **4800** | *5.50* | **9800** | *13.95* | **14800** | *27.82* |
| **4900** | *5.64* | **9900** | *14.16* | **14900** | *28.20* |
| **5000** | *5.78* | **10000** | *14.38* | **15000** | *28.57* |

| | | | | | |
|---|---|---|---|---|---|
| **15100** | *28.95* | **20100** | *57.54* | **25100** | *143.31* |
| **15200** | *29.33* | **20200** | *58.41* | **25200** | *146.83* |
| **15300** | *29.72* | **20300** | *59.29* | **25300** | *150.51* |
| **15400** | *30.11* | **20400** | *60.19* | **25400** | *154.35* |
| **15500** | *30.51* | **20500** | *61.11* | **25500** | *158.35* |
| **15600** | *30.92* | **20600** | *62.04* | **25600** | *162.52* |
| **15700** | *31.33* | **20700** | *63.00* | **25700** | *166.90* |
| **15800** | *31.75* | **20800** | *63.98* | **25800** | *171.46* |
| **15900** | *32.17* | **20900** | *64.97* | **25900** | *176.26* |
| **16000** | *32.60* | **21000** | *66.00* | **26000** | *181.30* |
| **16100** | *33.03* | **21100** | *67.05* | **26100** | *186.61* |
| **16200** | *33.48* | **21200** | *68.12* | **26200** | *192.20* |
| **16300** | *33.93* | **21300** | *69.21* | **26300** | *198.09* |
| **16400** | *34.38* | **21400** | *70.33* | **26400** | *204.29* |
| **16500** | *34.84* | **21500** | *71.47* | **26500** | *210.83* |
| **16600** | *35.31* | **21600** | *72.65* | **26600** | *217.76* |
| **16700** | *35.78* | **21700** | *73.84* | **26700** | *225.13* |
| **16800** | *36.27* | **21800** | *75.07* | **26800** | *232.91* |
| **16900** | *36.76* | **21900** | *76.32* | **26900** | *241.21* |
| **17000** | *37.26* | **22000** | *77.61* | **27000** | *250.07* |
| **17100** | *37.76* | **22100** | *78.93* | **27100** | *259.53* |
| **17200** | *38.27* | **22200** | *80.28* | **27200** | *269.69* |
| **17300** | *38.79* | **22300** | *81.67* | **27300** | *280.56* |
| **17400** | *39.32* | **22400** | *83.09* | **27400** | *292.28* |
| **17500** | *39.86* | **22500** | *84.55* | **27500** | *304.94* |
| **17600** | *40.40* | **22600** | *86.05* | **27600** | *318.66* |
| **17700** | *40.95* | **22700** | *87.58* | **27700** | *333.56* |
| **17800** | *41.52* | **22800** | *89.17* | **27800** | *349.82* |
| **17900** | *42.09* | **22900** | *90.80* | **27900** | *367.60* |
| **18000** | *42.67* | **23000** | *92.47* | **28000** | *387.15* |
| **18100** | *43.26* | **23100** | *94.19* | **28100** | *408.71* |
| **18200** | *43.86* | **23200** | *95.97* | **28200** | *432.71* |
| **18300** | *44.47* | **23300** | *97.79* | **28300** | *459.49* |
| **18400** | *45.09* | **23400** | *99.68* | **28400** | *489.57* |
| **18500** | *45.72* | **23500** | *101.63* | **28500** | *523.66* |
| **18600** | *46.37* | **23600** | *103.63* | **28600** | *562.60* |
| **18700** | *47.03* | **23700** | *105.69* | **28700** | *607.47* |
| **18800** | *47.69* | **23800** | *107.81* | **28800** | *659.85* |
| **18900** | *48.37* | **23900** | *110.01* | **28900** | *721.74* |
| **19000** | *49.06* | **24000** | *112.28* | **29000** | *795.99* |
| **19100** | *49.76* | **24100** | *114.63* | **29100** | *886.62* |
| **19200** | *50.48* | **24200** | *117.05* | **29200** | *999.87* |
| **19300** | *51.20* | **24300** | *119.57* | **29300** | *1145.35* |
| **19400** | *51.94* | **24400** | *122.16* | **29400** | *1339.13* |
| **19500** | *52.69* | **24500** | *124.87* | **29500** | *1610.03* |
| **19600** | *53.46* | **24600** | *127.67* | **29600** | *2015.74* |
| **19700** | *54.24* | **24700** | *130.56* | **29700** | *2690.65* |
| **19800** | *55.05* | **24800** | *133.57* | **29800** | *4036.08* |
| **19900** | *55.86* | **24900** | *136.70* | | |
| **20000** | *56.70* | **25000** | *139.94* | | |

Και η γραφική παράσταση είναι:

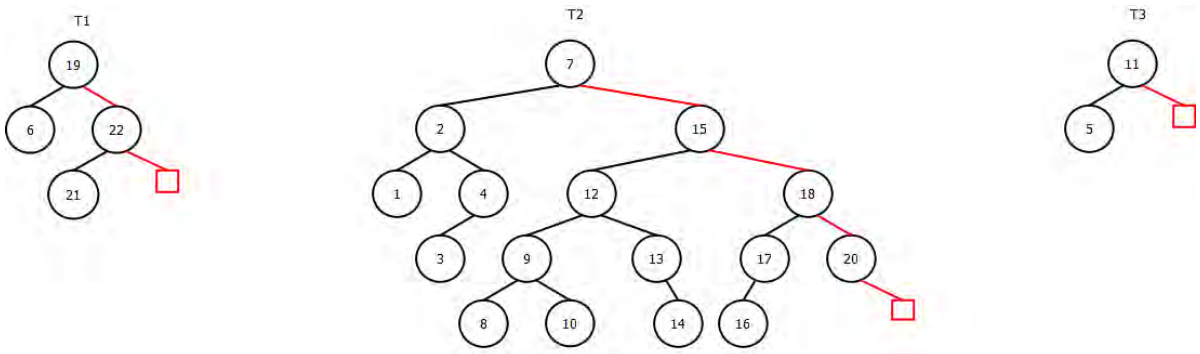
# ΚΕΦΑΛΑΙΟ 6

## 6. Μελλοντική έρευνα

### 6.1 Δείκτης προς επόμενο

Σε αυτό το κεφάλαιο θα περιγράψουμε μία αλλαγή που μπορεί να γίνει στη δομή δεδομένων του 4$^{ου}$ κεφαλαίου, η οποία βελτιώνει τις αναζητήσεις. Βέβαια αυτή η μετατροπή κοστίζει σε χώρο.

Μέχρι τώρα, υποθέταμε ότι κάθε αναζήτηση είναι πετυχημένη. Σε περίπτωση που το στοιχείο που αναζητούμε δεν βρίσκεται στη δομή, θα καταλήξουμε να ψάξουμε σε κάθε υποδομή με συνολικό κόστος O(log n), χωρίς να επιστρέψουμε τίποτα.

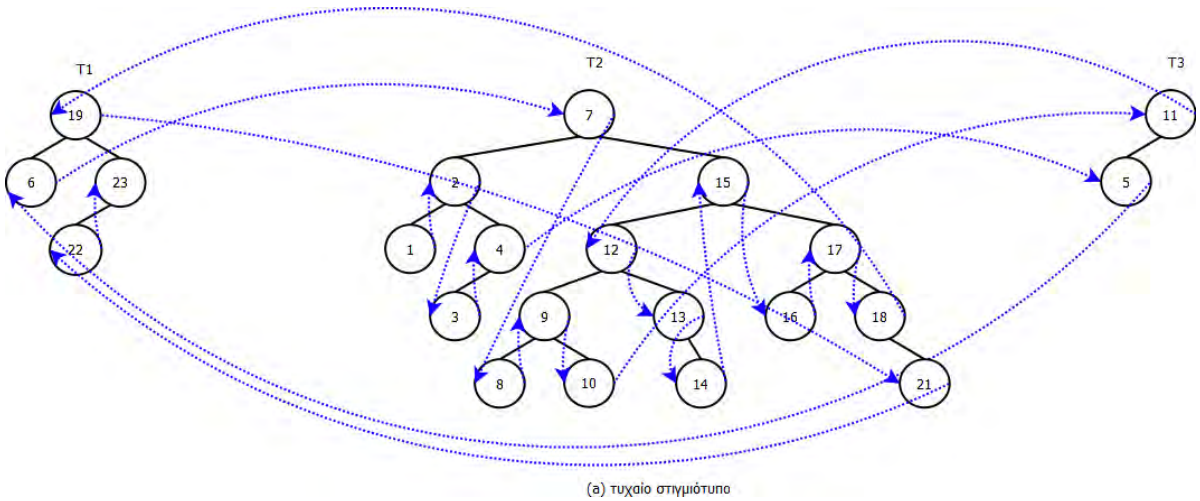



(a) αναζήτηση του 23

Συνήθως, ένα λεξικό επιστρέφει το μεγαλύτερο στοιχείο στο λεξικό, το οποίο είναι μικρότερο από το κλειδί το οποίο ψάχνουμε, ή το μικρότερο στοιχείο, το οποίο είναι μεγαλύτερο από το κλειδί. Αιτήσεις προς τον προηγούμενο είναι εύκολο να υλοποιηθούν στα δυαδικά δέντρα αναζήτησης, εφόσον αρκεί να εξετάσουμε πότε «πέφτουμε» εκτός του δέντρου. Αυτό όμως δεν λειτουργεί στην δομή δεδομένων μας, αφού έχουμε πολλές υποδομές και θα πρέπει να γνωρίζουμε πότε να σταματήσουμε.

Άρα ο στόχος μας είναι: Δεδομένου ενός κλειδιού x, θα πρέπει να επιστρέψουμε το x σε χρόνο O(log t(x)), εάν βρίσκεται στη δομή. Εάν το x δεν βρίσκεται στην δομή, θα πρέπει να επιστρέψουμε το pred(x) σε χρόνο O(log t(pred(x))), όπου pred(x) είναι ο αριθμητικά προηγούμενος του x.

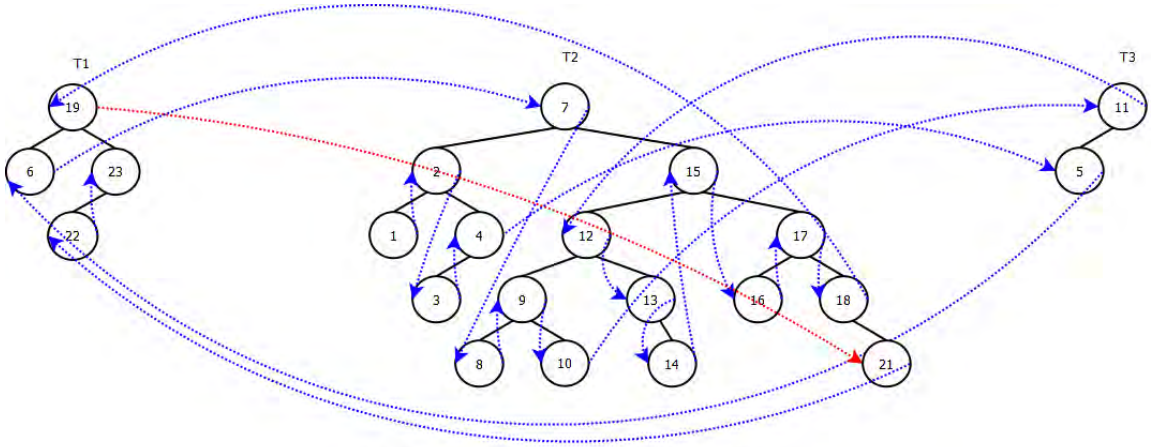
Για να το πετύχουμε αυτό, θα προσθέσουμε κάποιους δείκτες στη δομή μας. Πιο συγκεκριμένα, κάθε κόμβος στη δομή, θα έχει ένα δείκτη προς τον αριθμητικά επόμενό του.



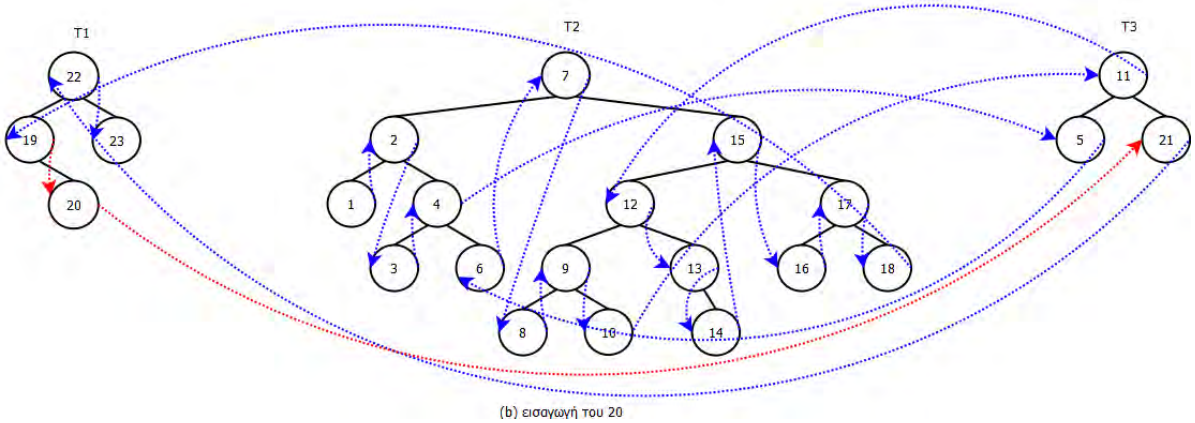
(a) τυχαίο στιγμιότυπο

### 6.1.1 Εισαγωγή

Κατά την **εισαγωγή**, κάθε υποδομή θα αναζητεί το προς εισαγωγή στοιχείο με συνολικό κόστος O(log n) και ο μικρότερος επόμενός του θα καταγράφεται. Ο μικρότερος τέτοιος επόμενος θα είναι ο επόμενος, σε ολόκληρη τη δομή, του στοιχείου που θα εισαχθεί. Για αυτό το λόγο, το κόστος εισαγωγής παραμένει O(log n).




(a) για την εισαγωγή του 20 , το 21 καταγράφεται ως επόμενο

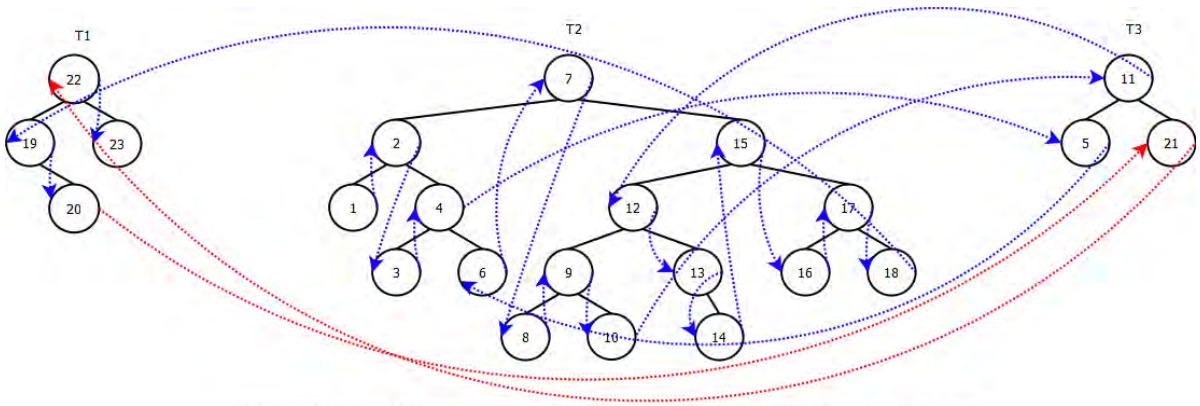



(b) εισαγωγή του 20

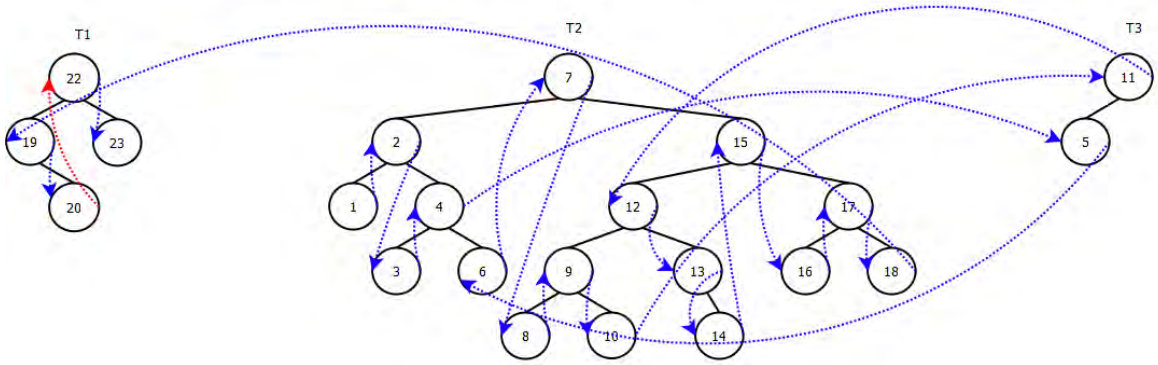
Παρατηρείστε ότι κατά την εισαγωγή του 20, ο επόμενος του 19 δείχνει πλέον το 20, αντί για το 21, και ο επόμενος του 20 δείχνει τον 21. Επίσης στο δεύτερο δέντρο εκτοπίζεται το 6, το οποίο με τη σειρά του εκτοπίζει το 21 στο τρίτο δέντρο.

### 6.1.2 Διαγραφή

Ομοίως, κατά την **διαγραφή**, μόνο ο προηγούμενος του προς διαγραφή στοιχείου θα πρέπει να ανανεώσει τον δείκτη του προς τον αριθμητικά επόμενο και για αυτό το συνολικό κόστος διαγραφής παραμένει O(log n).




(a) για την διαγραφή του 21 , σημειώνουμε τον επόμενό του , ώστε να ενημερώσουμε τον επόμενο του προηγούμενού του




(b) διαγραφή του 21

### 6.1.3 Αναζήτηση

Κατά την **αναζήτηση** πράττουμε όπως προηγουμένως. Έστω ότι ψάχνουμε στην υποδομή i. Εάν το x βρίσκεται στην υποδομή i, τότε η ανάλυση είναι όπως πριν, και επιστρέφουμε το x σε χρόνο O(log t(x)). Σε αντίθετη περίπτωση, αναζητούμε την υποδομή i για τον προηγούμενο του x, έστω pred(x). Εφόσον κάθε στοιχείο έχει δείκτη προς τον επόμενο του σε ολόκληρη τη δομή, μπορούμε να υπολογίσουμε το succ(pred(x)).

- Εάν succ(pred(x)) = x, τότε γνωρίζουμε ότι το x είναι στη δομή και άρα η αναζήτηση θα ολοκληρωθεί όπως προηγουμένως.
- Εάν succ(pred(x)) < x, τότε γνωρίζουμε ότι υπάρχει ένα στοιχείο μικρότερο από το x αλλά μεγαλύτερο από όσα έχουμε δει μέχρι αυτή τη στιγμή, και άρα συνεχίζουμε την αναζήτηση του x.
- Εάν succ(pred(x)) > x, τότε έχουμε φτάσει στο μεγαλύτερο στοιχείο που είναι ίσο ή μικρότερο του x, και άρα η αναζήτηση σταματάει.

Σε κάθε περίπτωση, αφού βρούμε το x ή το pred(x), μεταφέρουμε το στοιχείο που βρήκαμε στην πρώτη υποδομή όπως συνήθως.

Φυσικά με αυτή την βελτιστοποίηση, έχουμε πρόσθετο κόστος σε χώρο. Συγκεκριμένα, τώρα απαιτούμε n δείκτες και άρα έχουμε κόστος O(n). Ενώ έχουμε απαλλαγεί από τις ουρές, ακόμα έχουμε ένα δείκτη ανά στοιχείο στο λεξικό. Για να το διορθώσουμε αυτό, παρατηρείστε ότι μπορούμε να αγνοήσουμε τους δείκτες για τα λίγα τελευταία δέντρα και απλά να εκτελέσουμε μία αναζήτηση στοιχείο ανά στοιχείο, με ελάχιστα αυξημένη πολυπλοκότητα.

## 6.2 Χρήση ιδιοτήτων για βελτίωση του αλγορίθμου

Σε αυτή την εργασία χρησιμοποιήσαμε μία ιδιότητα της δομής avl-tree. Παρομοίως θα μπορούσαμε να χρησιμοποιήσουμε ιδιότητες άλλων δομών, όπως π.χ. η ιδιότητα του δυναμικού δαχτύλου **(dynamic finger property)** των ταξινομημένων πινάκων, υποθέτοντας ότι διατηρούμε έναν δείκτη στο αποτέλεσμα προηγούμενης αναζήτησης. Αυτή η ιδιότητα σημαίνει ότι ο χρόνος αναζήτησης είναι λογαριθμικός στη διαφορά βαθμού δύο διαδοχικών αναζητήσεων.

Μία άλλη ιδιότητα που θα μπορούσαμε να χρησιμοποιήσουμε είναι η ιδιότητα της ουράς **(queueish property)**, όπου ο χρόνος αναζήτησης είναι λογαριθμικός στον αριθμό των στοιχείων που δεν έχουν προσπελαστεί από την τελευταία φορά που προσπελάστηκε το αντικείμενο της αναζήτησης.

# ΚΕΦΑΛΑΙΟ 7

## 7. Βιβλιογραφία


1. Prosenjit Bose, John Howat, and Pat Morin: "A Distribution-Sensitive Dictionary with Low Space Overhead" (2009)

2. Παναγιώτης Δ. Μποζάνης: "Δομές Δεδομένων" (2006)

3. en.wikipedia.org/wiki/**AVL_tree**

4. Iacono J.: "Alternatives to splay trees with O(log n) worst-case access times". In SODA 2001: Proceeding of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 516-522 (2001)

# ΠΑΡΑΡΤΗΜΑ Α

## Α. Κώδικας 1ης εκδοχής

```
#include <stdio.h>
#include <stdlib.h>

//struct gia tous komvous tou dentrou
struct avl{
	int key;
	int height;
	struct avl *right;
	struct avl *left;
	struct avl *parent;
	struct queue *q;
};

//struct gia ta stoixeia tis listas , opou kathe kathe stoixeio deixnei sti riza
kathe upodomis kai stis antistoixi oura
struct header{
	int id;
	int tnodes;
	struct avl *root;
	struct queue *Qroot;
	struct header *next;
	struct header *previous;
};

//struct gia ti lista kathe dentrou
struct queue{
	int key;
	struct queue *nxt;
	struct queue *prv;
};

//global metavlites
struct header *Hroot;
struct avl *del;
float total_cost=0;
int acc=0,reb=0;

//stigmiotupa sunartisewn
avl *rotate_LL(struct avl *pointer, struct header *head);
avl *rotate_LR(struct avl *pointer, struct header *head);
avl *rotate_RR(struct avl *pointer, struct header *head);
avl *rotate_RL(struct avl *pointer, struct header *head);
void queue_init(struct header *pointer);
void printtree(struct avl *a, int m);

//sunartisi arxikopoisis tis listas pou periexei tis rizes twn upodomwn
void header_init(){
	Hroot = (struct header *)malloc(sizeof(struct header));
	Hroot->next = Hroot;
	Hroot->previous = Hroot;
	Hroot->id = 2;
}
```

```
//sunartisi gia eisagwgi sti lista pou krataei tis rizes twn upodomwn
void header_ins(){
	struct header *pointer;

	pointer = (struct header *)malloc(sizeof(struct header));

	pointer->tnodes = 0;
	pointer->root = NULL;
	pointer->next = Hroot->previous->next;
	pointer->previous = Hroot->previous;
	Hroot->previous->next = pointer;
	Hroot->previous = pointer;
	pointer->id = (pointer->previous->id * pointer->previous->id);
	queue_init(pointer);
}

//sunartisi gia diagrafi apo ti lista poy krataei tis rizes tvn upodomwn
void header_del(){
	struct header *point;

	point = Hroot->previous;
	point->previous->next = Hroot;
	Hroot->previous = point->previous;
	free(point);
}

//sunartisi gia arxikopoiisi tis ouras
void queue_init(struct header *pointer){
	pointer->Qroot = (struct queue *)malloc(sizeof(struct queue));
	pointer->Qroot->nxt = pointer->Qroot;
	pointer->Qroot->prv = pointer->Qroot;
}

//sunartisi gia diagrafi apo tin oura
//analoga me tin timi tou orismatos msg , ektelei diaforetika tin diagrafi
queue *queue_ins(int val, struct header *pointer, char msg){
	struct queue *point;

	point = (struct queue *)malloc(sizeof(struct queue));
	point->key = val;
	if( msg == 's'){
		point->nxt = pointer->Qroot->nxt;
		point->prv = pointer->Qroot;
		pointer->Qroot->nxt->prv = point;
		pointer->Qroot->nxt = point;
	}
	else{
		point->nxt = pointer->Qroot;
		point->prv = pointer->Qroot->prv;
		pointer->Qroot->prv->nxt = point;
		pointer->Qroot->prv = point;
	}
	return point;
}
```

```
//sunartisi gia diagrafi apo tin oura
//analoga me tin timi tou orismatos id , ektelei diagrafi apo tin kefali (r) ,
tin oura (t) h tin mesi (m) tis ouras
void queue_del(int val, struct header *pointer, char id){
	struct queue *point;

	if(id == 't'){
		point = pointer->Qroot->prv;
		point->prv->nxt = pointer->Qroot;
		pointer->Qroot->prv = point->prv;
	}
	else if(id == 'r'){
		point = pointer->Qroot->nxt;
		pointer->Qroot->nxt = point->nxt;
		point->nxt->prv = pointer->Qroot;
	}
	else if(id == 'm'){
		for(point=pointer->Qroot->nxt; point->key != val; point=point->nxt);
		point->nxt->prv = point->prv;
		point->prv->nxt = point->nxt;
	}
	free(point);
}

//sunartisi pou epistrefei to upsos tou komvou "point"
int height(struct avl *point){
	if(point == NULL) return -1;
	else return(point->height);
}

//sunartisi pou epistrefei to megisto apo ta duo orismata
int max(int Lheight, int Rheight){
	return Lheight > Rheight ? Lheight : Rheight;
}

//sunartisi tis epanazugistikis praksis LL
avl *rotate_LL(struct avl *pointer, struct header *head){
	struct avl *point;
	reb++;

	point = pointer->left;
	pointer->left = point->right;
	point->right = pointer;
	if(pointer->parent != pointer){
		point->parent = pointer->parent;
		if(point->parent->left == pointer) point->parent->left = point;
		else if(point->parent->right == pointer) point->parent->right =
point;
	}
	else{ point->parent = point; head->root = point; }
	pointer->parent = point;
	if(pointer->left != NULL) pointer->left->parent = pointer;

	pointer->height = max(height(pointer->left), height(pointer->right)) + 1;
	point->height =  max(height(point->left), pointer->height) + 1;

	return point;
}
```

```
//sunartisi epanazugistikis praksis RR
avl *rotate_RR(struct avl *pointer, struct header *head){
	struct avl *point;
	reb++;

	point = pointer->right;
	pointer->right = point->left;
	point->left = pointer;
	if(pointer->parent != pointer){
		point->parent = pointer->parent;
		if(point->parent->left == pointer) point->parent->left = point;
		else if(point->parent->right == pointer) point->parent->right =
                                                                point;
	}
	else{ point->parent = point; head->root = point; }
	pointer->parent = point;
	if(pointer->right != NULL) pointer->right->parent = pointer;

	pointer->height = max(height(pointer->left), height(pointer->right)) + 1;
	point->height =  max(height(point->right), pointer->height) + 1;

	return point;
}

//sunartisi epanazugistikis praksis LR
//tin metatrepei se RR
avl *rotate_LR(struct avl *pointer, struct header *head){
	pointer->left = rotate_RR(pointer->left, head);
	return rotate_LL(pointer, head);
}

//sunartisi epanazugistikis praksis RL
//tin metrepei se LL
avl *rotate_RL(struct avl *pointer, struct header *head){
	pointer->right = rotate_LL(pointer->right, head);
	return rotate_RR(pointer, head);
}

//sunartisi eisagwgis komvou sto avl dentro
avl *avl_ins(int val, struct avl *point, struct header *pointer, bool ins){

	if(point == NULL){
		point = (struct avl *)malloc(sizeof(struct avl));

		pointer->tnodes++;
		point->height = 0;
		point->left = NULL;
		point->right = NULL;
		point->key = val;
		if(pointer->root == NULL){ pointer->root = point; point->parent =
                                                                point; }
		if(ins) point->q = queue_ins(point->key, pointer, 's');
		else point->q = queue_ins(point->key, pointer, 't');

	}
	else if(val < point->key){
		point->left = avl_ins(val, point->left, pointer, ins);
		point->left->parent = point;
```

```
                if(height(point->left) - height(point->right) == 2){
                        if(val < point->left->key) point = rotate_LL(point, pointer);
                        else point = rotate_LR(point, pointer);
                }
        }
        else if(val > point->key){
                point->right = avl_ins(val,point->right,pointer, ins);
                point->right->parent = point;
                if(height(point->right) - height(point->left) == 2){
                        if(val > point->right->key) point = rotate_RR(point,pointer);
                        else point = rotate_RL(point,pointer);
                }
        }
        point->height = max(height(point->left),height(point->right)) + 1;
        return point;
}

//sunartisi diagrafis komvou apo to dentro avl
avl *avl_del(int val,struct avl *point, struct header *pointer){
        struct avl *move;
        int msg;

        if(point == NULL) return point;
        else if(val < point->key){
                if(point->left != NULL){ point->left = avl_del(val, point->left,
                                                                pointer); }
                else if(point->left == NULL){
                        move = point->parent;
                        if(move->key == val) msg = 1;
                        else msg = 0;

                        del->key = point->key;
                        del->height = point->height;
                        if(point->parent->left==point)point->parent->left=point->
                                                                        right;
                        else point->parent->right = point->right;
                        if(point->right != NULL) point->right->parent = point->
                                                                        parent;
                        free(point);
                        move->height = max(height(move->left),height(move->right))+1;
                        if(msg != 1)return move->left;
                        else return move->right;
                }
        }

        else if(val > point->key){ point->right = avl_del(val,point->right,Hroot->
                                                                        next); }

        else if(val == point->key){
                del = point;
                if(point->right != NULL && point->left != NULL){ point->right =
                                        avl_del(val,point->right,pointer); }
                else if(point->right == NULL && point->left != NULL){
                        del = point->left;
                        if(point->parent->left == point) point->parent->left =point->
                                                                        left;
                        else point->parent->right = point->left;
                        if(point->parent != point) point->left->parent = point->
                                                                        parent;
```

```
            else { del->parent = del; pointer->root = del; }
                    free(point);
                    return del;
                }
         else if(point->right != NULL && point->left == NULL){
            del = point->right;
                    if(point->parent->left == point) point->parent->left =point->
                                                                        right;
                    else point->parent->right = point->right;

            if(point->parent != point) point->right->parent = point->parent;
            else { del->parent = del; pointer->root = del; }
                    free(point);
                    return del;
                }
       else{ if(point->parent == point){ pointer->root = NULL; } free(point);
return NULL; }
       }
       if(height(point->right) - height(point->left) == 2){
              if(height(point->right->right) - height(point->right->left) == 1 ||
height(point->right->right) - height(point->right->left) == 0) point =
rotate_RR(point, pointer);
              else point = rotate_RL(point, pointer);
       }
       else if(height(point->left) - height(point->right) == 2){
              if(height(point->left->left) - height(point->left->right) == 1 ||
height(point->left->left) - height(point->left->right) == 0) point =
rotate_LL(point, pointer);
              else point = rotate_LR(point, pointer);
       }
       point->height = max(height(point->left),height(point->right)) + 1;
       return point;
}

//sunartisi eisagwgis komvou sti lista pou krataei tis rizes twn upodomwn
void insert(int val, struct header *point){
       int temp;

       if(point == Hroot){ header_ins(); point = point->next; }
       avl_ins(val, point->root, point, true);
       if(point->tnodes > point->id){
              point->tnodes--;
              temp = point->Qroot->prv->key;
              avl_del(temp, point->root, point);
              queue_del(point->Qroot->prv->key,point, 't');
              if(point->next == Hroot) header_ins();
              insert(temp, point->next);
       }
}

//diagrafi komvou apo ti lista pou krataei tis rizes twn upodomwn
void remove(int val, struct header *point){
       struct avl *pointer;
       struct header *p;
       int temp;

       for(point = Hroot->next; point != Hroot; point = point->next){
              pointer = point->root;
              while(pointer != NULL){
```

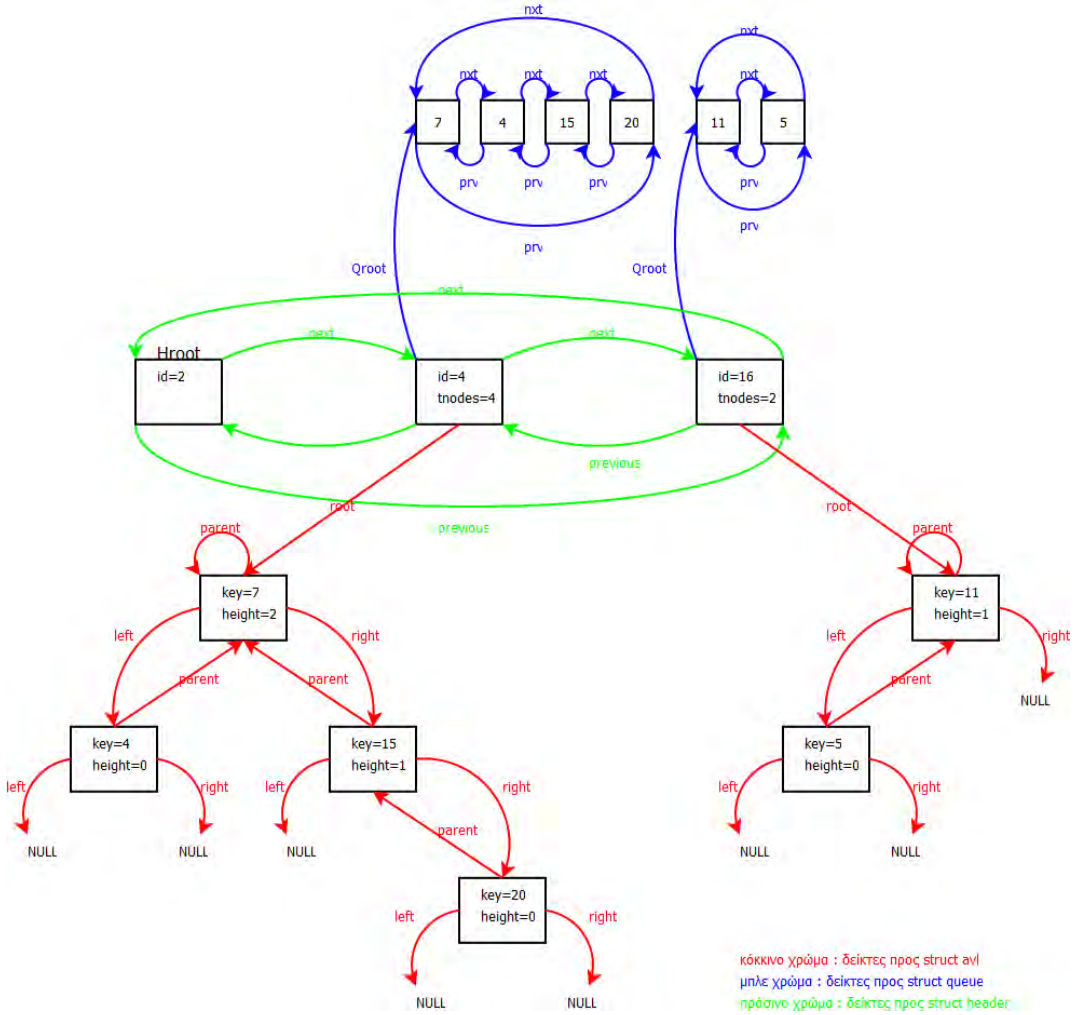
```
                        acc++;
                        if(val < pointer->key) pointer = pointer->left;
                        else if(val > pointer->key) pointer = pointer->right;
                        else if(val == pointer->key) break;
                }
                if(pointer != NULL) break;
        }
        p = point;
        if(point != Hroot){
                if(point->next != Hroot){
                        while(point->next != Hroot){
                                if(point == p){
                                        queue_del(pointer->key, point, 'm');
                                        avl_del(pointer->key, point->root, point);
                                        point->tnodes--;
                                }
                                temp = point->next->Qroot->nxt->key;
                                avl_del(point->next->Qroot->nxt->key, point->next->
                                                                        root, point->next);
                                point->next->tnodes--;
                                queue_del(temp, point->next, 'r');
                                avl_ins(temp, point->root, point, false);
                                point = point->next;
                                pointer=point->root;
                        }
                }
                else{
                        queue_del(pointer->key, point, 'm');
                        avl_del(pointer->key, point->root, point);
                        point->tnodes--;
                }
                if(point->root == NULL){ header_del(); }
        }
}

//sunartisi anazitisis
//ena vrethei to stoixeio pou anaziteitai (val) metaferetai sto 1o upodentro
void find(int val, struct header *point, bool state){
        int temp;
        struct avl *pointer;
        struct header *p = point;

        for(point = Hroot->next; point != Hroot; point = point->next){
                pointer = point->root;
                while(pointer != NULL){
                        acc++;
                        if(val < pointer->key) pointer = pointer->left;
                        else if(val > pointer->key) pointer = pointer->right;
                        else if(val == pointer->key) break;
                }
                if(pointer != NULL) break;
        }
        if(point == Hroot && !state){insert(val, p); return;}
        if(point != Hroot && state){
                if(point->previous != Hroot){
                        while(point->previous != Hroot){
                                queue_del(val, point, 'm');
                                avl_del(val, point->root, point);
                           avl_ins(val, point->previous->root, point->previous,true);
```

```
                        temp = point->previous->Qroot->prv->key;
                        avl_del(temp, point->previous->root, point->previous);
                        queue_del(temp , point->previous, 't');
                        avl_ins(temp, point->root, point, true);
                        point = point->previous;
                    }
                }
                else{ queue_del(val, point, 'm'); queue_ins(val, point, 's'); }
        }
        else printf("KEY: %d NOT FOUND\n", val);
}

//sunartisi ektupwsis dentrou
//i riza ektupwnetai arista kai ta fulla tou epektinontai pros ta deksia
void printtree(struct avl* a,int m){
        int n=m;
        if(a==0) return;
        if(a->right) printtree(a->right,m+1);
        while(n--) printf("    ");
        printf("%d\n",a->key);
        if(a->left) printtree(a->left,m+1);
}


int main(int argc, char *argv[]){
        FILE *f;
        FILE *output;
        FILE *output2;
        int i, val, selection = 1;
        struct header *pointer;
        struct queue *point;
        header_init();

        printf("1. Insert node\n2. Delete node\n3. Search node\n0. Exit\n");

        while(selection != 0){
                printf("SELECTION: ");
                scanf_s("%d",&selection);
                switch(selection){
                case 1:
                        fopen_s(&f, "randomnums.txt","r");
                        fopen_s(&output, "output.txt" , "w");
                        for(i=1; i<30001; i++){
                                //printf("insert node: ");
                                fscanf_s(f,"%d", &val);
                                //scanf("%d",&val);
                                find(val, Hroot->next, false);

                                total_cost+=acc+reb;
                                if(i%100==0){
                                        fprintf_s(output,"%6.3f\n",total_cost/i);
                                }
                                reb=0,acc=0;
                                        /*for(pointer=Hroot->next; pointer!=Hroot;
                                        pointer = pointer->next){
                                        printtree(pointer->root, 0);
                                        printf("------------\n");
                                }*/
                        }
```

```
	/*for(pointer=Hroot->next; pointer!=Hroot; pointer =pointer->
                                                        next){
		printtree(pointer->root, 0);
		printf("------------\n");

		printf("\n");
		for(point=pointer->Qroot->nxt;point!=pointer->
                                        Qroot;point=point->nxt){
			printf("%d ",point->key);
		}
		printf("\n");
	}
	*/
	fclose(f);
	fclose(output);
break;

case 2:
	fopen_s(&f, "randomnums2.txt","r");
	fopen_s(&output2,"output2.txt","w");

	total_cost=0;

	for(i=1;i<30001;i++){
	//for(i=0;i<6;i++){
		//printf("delete node: ");
		fscanf_s(f,"%d", &val);
		//scanf("%d",&val);
		remove(val, Hroot->next);
		//printf("%d deleted\n",val);

		total_cost+=acc+reb;
		//printf("total cost : %f\n",total_cost/i);
		if(i%100==0){
			fprintf_s(output2,"%.2f\n",total_cost);
		}

		reb=0,acc=0;
		/*for(pointer=Hroot->next; pointer!=Hroot; pointer =
                                                pointer->next){
			printtree(pointer->root, 0);

			printf("\n");
			for(point=pointer->Qroot->nxt;point!=pointer->
                                        Qroot;point=point->nxt){
				printf("%d ",point->key);
			}
			printf("\n");
			printf("------------\n");
		}*/
	}
	/*for(pointer=Hroot->next; pointer!=Hroot; pointer =pointer->
                                                        next){
		printtree(pointer->root, 0);
		printf("------------\n");
	}*/
	fclose(f);
	fclose(output2);
break;
```

```
            case 3:
                  for(i=0; i<6; i++){
                        printf("search node: ");
                        scanf_s("%d",&val);
                        find(val, Hroot->next, true);
                        for(pointer=Hroot->next; pointer!=Hroot; pointer =
                                                            pointer->next){
                              printtree(pointer->root, 0);

                              printf("\n");
                              for(point=pointer->Qroot->nxt;point!=pointer->
                                                Qroot;point=point->nxt){
                                    printf("%d ",point->key);
                              }
                              printf("\n");
                              printf("------------\n");
                        }
                  }
            }
      }
      system("PAUSE");
      return 0;
}
```

Η σχηματική αναπαράσταση ενός στιγμιότυπου είναι:



# ΠΑΡΑΡΤΗΜΑ Β

## Β. Κώδικας 2ης εκδοχής

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//struct gia tous komvous tou dentrou
struct avl{
	int key;
	int height;
	struct avl *right;
	struct avl *left;
	struct avl *parent;
};

//gia tous komvous tis listas pou krataei tis rizes tis kathe upodomis
struct header{
	int id;
	int tnodes;
	struct avl *root;
	struct header *next;
	struct header *previous;
};

struct header *Hroot;
struct avl *del;
float total_cost=0;
int acc=0,reb=0;

avl *rotate_LL(struct avl *pointer, struct header *head);
avl *rotate_LR(struct avl *pointer, struct header *head);
avl *rotate_RR(struct avl *pointer, struct header *head);
avl *rotate_RL(struct avl *pointer, struct header *head);
void printtree(struct avl *a, int m);

//sunartisi gia na dialegei tuxaia enan komvo tou dentrou
//ksekiname apo tin riza tou dentrou
//an random=0 epilegoume ton komvo pou vriskomaste
//an random=1 pigainoume ston aristero komvo
//an radnom=2 pigainoume stom deksi komvo
avl *random(struct header *pointer){
	int random;
	struct avl *point;
	random = rand()%3;
	point = pointer->root;
	if(!random) return point;
	while(random){
		if(point->left == NULL && point->right == NULL) break;
		else if(point->left != NULL && point->right != NULL){
			if(random == 1) point = point->left;
			else if(random == 2) point = point->right;
		}
```

```
		else if(point->left != NULL && point->right == NULL){ if(random==1)
			{ point = point->left; }  else{ break; } }
		else if(point->left == NULL && point->right != NULL){ if(random==2)
			{ point = point->right; }  else{ break; } }

		random = rand()%3;
	}
	return point;
}

//sunartisi arxikopoiisis tis listas pou krataei tis rizes twn upodomwn
void header_init(){
	Hroot = (struct header *)malloc(sizeof(struct header));
	Hroot->next = Hroot;
	Hroot->previous = Hroot;
	Hroot->id = 2;
}

//sunartsi gia eisagwgi sti lista pou krataei tis rizes twn upodomwn
void header_ins(){
	struct header *pointer;

	pointer = (struct header *)malloc(sizeof(struct header));

	pointer->tnodes = 0;
	pointer->root = NULL;
	pointer->next = Hroot->previous->next;
	pointer->previous = Hroot->previous;
	Hroot->previous->next = pointer;
	Hroot->previous = pointer;
	pointer->id = (pointer->previous->id * pointer->previous->id);
}

//sunartisi gia diagrafi apo i lista pou kraaei tis rizes twn upodomwn
void header_del(){
	struct header *point;

	point = Hroot->previous;
	point->previous->next = Hroot;
	Hroot->previous = point->previous;
	free(point);
}

//epistrefei to upsos tou komvou "point"
int height(struct avl *point){
	if(point == NULL) return -1;
	else return(point->height);
}

int max(int Lheight, int Rheight){
	return Lheight > Rheight ? Lheight : Rheight;
}

//epanazugistiki praksi peristrofis
avl *rotate_LL(struct avl *pointer, struct header *head){
	struct avl *point;
	reb++;

	point = pointer->left;
```

```
	pointer->left = point->right;
	point->right = pointer;
	if(pointer->parent != pointer){
		point->parent = pointer->parent;
		if(point->parent->left == pointer) point->parent->left = point;
		else if(point->parent->right == pointer) point->parent->right =
								point;
	}
	else{ point->parent = point; head->root = point; }
	pointer->parent = point;
	if(pointer->left != NULL) pointer->left->parent = pointer;

	pointer->height = max(height(pointer->left), height(pointer->right)) + 1;
	point->height =  max(height(point->left), pointer->height) + 1;

	return point;
}

//epanazugistiki praksi peristrofis
avl *rotate_RR(struct avl *pointer, struct header *head){
	struct avl *point;
	reb++;

	point = pointer->right;
	pointer->right = point->left;
	point->left = pointer;
	if(pointer->parent != pointer){
		point->parent = pointer->parent;
		if(point->parent->left == pointer) point->parent->left = point;
		else if(point->parent->right == pointer) point->parent->right =
								point;
	}
	else{ point->parent = point; head->root = point; }
	pointer->parent = point;
	if(pointer->right != NULL) pointer->right->parent = pointer;

	pointer->height = max(height(pointer->left), height(pointer->right)) + 1;
	point->height =  max(height(point->right), pointer->height) + 1;

	return point;
}

//epanazugistiki praksi peristrofis
avl *rotate_LR(struct avl *pointer, struct header *head){
	pointer->left = rotate_RR(pointer->left, head);
	return rotate_LL(pointer, head);
}

//epanazugistiki praksi peristrofis
avl *rotate_RL(struct avl *pointer, struct header *head){
	pointer->right = rotate_LL(pointer->right, head);
	return rotate_RR(pointer, head);
}
```

```
//sunartisi eisagwgis sto dentro avl
avl *avl_ins(int val, struct avl *point, struct header *pointer, bool ins){
	if(point == NULL){
		point = (struct avl *)malloc(sizeof(struct avl));

		pointer->tnodes++;
		point->height = 0;
		point->left = NULL;
		point->right = NULL;
		point->key = val;
		if(pointer->root == NULL){ pointer->root = point; point->parent =
								point; }
	}
	else if(val < point->key){
		point->left = avl_ins(val, point->left, pointer, ins);
		point->left->parent = point;
		if(height(point->left) - height(point->right) == 2){
			if(val < point->left->key) point = rotate_LL(point, pointer);
			else point = rotate_LR(point, pointer);
		}
	}
	else if(val > point->key){
		point->right = avl_ins(val,point->right,pointer, ins);
		point->right->parent = point;
		if(height(point->right) - height(point->left) == 2){
			if(val > point->right->key) point = rotate_RR(point,pointer);
			else point = rotate_RL(point,pointer);
		}
	}
	point->height = max(height(point->left),height(point->right)) + 1;
	return point;
}
//sunartisi diagrafis apo to dentro avl
avl *avl_del(int val,struct avl *point, struct header *pointer){
	struct avl *move;
	int msg;

	if(point == NULL) return point;
	else if(val < point->key){
		if(point->left != NULL){ point->left = avl_del(val, point->left,
								pointer); }
		else if(point->left == NULL){
			move = point->parent;
			if(move->key == val) msg = 1;
			else msg = 0;

			del->key = point->key;
			del->height = point->height;
			if(point->parent->left == point) point->parent->left =point->
								right;
			else point->parent->right = point->right;
			if(point->right != NULL) point->right->parent = point->
								parent;
			free(point);
			move->height = max(height(move->left),height(move->right))+1;
			if(msg != 1)return move->left;
			else return move->right;
		}
	}
```

```
        else if(val > point->key){ point->right = avl_del(val,point->right,Hroot->
                                                                    next); }

        else if(val == point->key){
               del = point;
               if(point->right != NULL && point->left != NULL){ point->right =
                                       avl_del(val,point->right,pointer); }
               else if(point->right == NULL && point->left != NULL){
                      del = point->left;
                      if(point->parent->left == point) point->parent->left =point->
                                                                    left;
                      else point->parent->right = point->left;
                      if(point->parent != point) point->left->parent = point->
                                                                    parent;
               else { del->parent = del; pointer->root = del; }
                      free(point);
                      return del;
               }
         else if(point->right != NULL && point->left == NULL){
               del = point->right;
                      if(point->parent->left == point) point->parent->left =point->
                                                                    right;
                      else point->parent->right = point->right;

               if(point->parent != point) point->right->parent = point->parent;
               else { del->parent = del; pointer->root = del; }
                      free(point);
                      return del;
               }
        else{ if(point->parent == point){ pointer->root = NULL; } free(point);
return NULL; }
        }
        if(height(point->right) - height(point->left) == 2){
               if(height(point->right->right) - height(point->right->left) == 1 ||
height(point->right->right) - height(point->right->left) == 0) point =
rotate_RR(point, pointer);
               else point = rotate_RL(point, pointer);
        }
        else if(height(point->left) - height(point->right) == 2){
               if(height(point->left->left) - height(point->left->right) == 1 ||
height(point->left->left) - height(point->left->right) == 0) point =
rotate_LL(point, pointer);
               else point = rotate_LR(point, pointer);
        }
        point->height = max(height(point->left),height(point->right)) + 1;
        return point;
}

//sunartisi eisagwgis
void insert(int val, struct header *point){
        int temp;
        struct avl *pointer;

        if(point == Hroot){ header_ins(); point = point->next; }
        avl_ins(val, point->root, point, true);
        if(point->tnodes > point->id){
               point->tnodes--;
               pointer = random(point);
```

```
                temp = pointer->key;
                avl_del(temp, point->root, point);
                if(point->next == Hroot) header_ins();
                insert(temp, point->next);
        }
}

//sunartisi diagrafis
void remove(int val, struct header *point){
        struct avl *pointer;
        struct header *p;
        int temp;

        for(point = Hroot->next; point != Hroot; point = point->next){
                pointer = point->root;
                while(pointer != NULL){
                        acc++;
                        if(val < pointer->key) pointer = pointer->left;
                        else if(val > pointer->key) pointer = pointer->right;
                        else if(val == pointer->key) break;
                }
                if(pointer != NULL) break;
        }
        p = point;
        if(point != Hroot){
                if(point->next != Hroot){
                        while(point->next != Hroot){
                                if(point == p){
                                        avl_del(pointer->key, point->root, point);
                                        point->tnodes--;
                                }
                                pointer = random(point->next);
                                temp = pointer->key;
                                avl_del(temp, point->next->root, point->next);
                                point->next->tnodes--;
                                avl_ins(temp, point->root, point, false);
                                point = point->next;
                                pointer=point->root;
                        }
                }
                else{
                        avl_del(pointer->key, point->root, point);
                        point->tnodes--;
                }
                if(point->root == NULL){ header_del(); }
        }
}

//sunartisi anazitisis
void find(int val, struct header *point){
        int temp;
        struct avl *pointer;
        struct header *p = point;

        for(point = Hroot->next; point != Hroot; point = point->next){
                pointer = point->root;
                while(pointer != NULL){
                        acc++;
                        if(val < pointer->key) pointer = pointer->left;
```

```
                        else if(val > pointer->key) pointer = pointer->right;
                        else if(val == pointer->key) break;
                }
                if(pointer != NULL) break;
        }
        if(point == Hroot){insert(val, p);return;}

        if(point->previous != Hroot){
                while(point->previous != Hroot){
                        avl_del(val, point->root, point);
                        avl_ins(val, point->previous->root, point->previous, false);
                        pointer = random(point->previous);
                        temp = pointer->key;
                        avl_del(temp, point->previous->root, point->previous);
                        avl_ins(temp, point->root, point, false);
                        point = point->previous;
                }
        }
}

//sunartisi ektupwsis dentrou
void printtree(struct avl* a,int m){
        int n=m;
        if(a==0) return;
        if(a->right) printtree(a->right,m+1);
        while(n--) printf("    ");
        printf("%d\n",a->key);
        if(a->left) printtree(a->left,m+1);
}


int main(int argc, char *argv[]){
        FILE *f;
        FILE *output;
        FILE *output2;

        int i, val, selection = 1;
        struct header *pointer;
        struct queue *point;

        header_init();

        srand( time(NULL) );

        printf("1. Insert node\n2. Delete node\n3. Search node\n0. Exit\n");

        while(selection != 0){
                printf("SELECTION: ");
                scanf_s("%d",&selection);
                switch(selection){
                case 1:
                        fopen_s(&f, "randomnums.txt","r");
                        fopen_s(&output, "output.txt" , "w");

                        for(i=1; i<30001; i++){
                                //printf("insert node: ");

                                fscanf_s(f,"%d", &val);
```

```
                //scanf("%d",&val);
                find(val, Hroot->next);
                total_cost+=acc+reb;
                //printf("total cost : %f\n",total_cost/i);


                if(i%100==0){
                        fprintf_s(output,"%6.3f\n",total_cost/i);
                }
                reb=0,acc=0;

                /*for(pointer=Hroot->next; pointer!=Hroot; pointer =
                                                        pointer->next){
                        printtree(pointer->root, 0);
                        printf("------------\n");
                }*/
        }
        /*for(pointer=Hroot->next; pointer!=Hroot; pointer =pointer->
                                                        next){
                printtree(pointer->root, 0);
                printf("------------\n");
        }*/
        fclose(f);
        fclose(output);
break;

case 2:
        fopen_s(&f, "randomnums2.txt","r");
        fopen_s(&output2,"output2.txt","w");

        total_cost=0;

        for(i=0;i<29971;i++){
                //printf("delete node: ");
                fscanf_s(f,"%d", &val);
                //scanf("%d",&val);
                remove(val, Hroot->next);
                //printf("%d deleted\n",val);

                total_cost+=acc+reb;
                //printf("total cost : %f\n",total_cost/i);
                if(i%100==0){
                      fprintf_s(output2,"%.2f\n",total_cost/(30001-i));
                }

                reb=0,acc=0;

                /*for(pointer=Hroot->next; pointer!=Hroot; pointer =
                                                    pointer->next){
                        printtree(pointer->root, 0);
                        printf("------------\n");
                }*/
        }
        /*for(pointer=Hroot->next; pointer!=Hroot; pointer =pointer->
                                                    next){
                        printtree(pointer->root, 0);
                        printf("------------\n");
        }*/
        fclose(f);
```

```
                break;

                case 3:
                        for(i=0; i<6; i++){
                                printf("search node: ");
                                scanf_s("%d",&val);
                                find(val, Hroot->next);
                                for(pointer=Hroot->next; pointer!=Hroot; pointer =
                                                                        pointer->next){
                                        printtree(pointer->root, 0);
                                        printf("\n");
                                        printf("------------\n");
                                }
                        }
                }
        }

        system("PAUSE");
        return 0;
}
```

Η σχηματική αναπαράσταση ενός στιγμιότυπου είναι: