ΕΡΓΑΣΊΑ 3 – ΔΟΜΈΣ ΔΕΔΟΜΈΝΩΝ

Συμμετέχοντες:

-Αναστάσιος Χρήστος Χατζής , ΑΜ:**3190216**

-Θεόδωρος Πάνου , ΑΜ: **3190154**

Υλοποίηση Μεθόδων:

-Μέθοδος load(String filename):

Η μέθοδος χρησιμοποιείται για να διαβάσει ένα text αρχείο που περιέχει Suspects, και να ενημερώσει το δέντρο. Ο χρήστης δίνει το όνομα του αρχείου, η μέθοδος μας εντοπίζει το αρχείο και σε κάθε γραμμή του αρχείου δημιουργεί ένα νέο αντικείμενο τύπου Suspect , το οποίο γεμίζει κατά το διάβασμα της γραμμής και στο τέλος της το κάνει insert στο δέντρο. Η πολυπλοκότητα της μεθόδου αυτής είναι O(N) ,για το διάβασμα κάθε γραμμής του αρχείου επί την πολυπλοκότητας της μεθόδου insert η οποία ισούται με O(N). Άρα συνολικά η πολυπλοκότητα της load είναι $O(N^2)$.

-Μέθοδος updateSavings(int AFM, double savings):

Στην μέθοδο αυτή ο χρήστης μπορεί να ενημερώσει τις καταθέσεις ενός Suspect δίνοντας το ΑΦΜ του. Η μέθοδος μας παίρνει το ΑΦΜ που έδωσε ο χρήστης καλεί την συνάρτηση searchByAFM(int AFM) , η οποία μας επιστρέφει τον Suspect με το δοσμένο ΑΦΜ. Τότε με την μέθοδο setSavings(double savings) της κλάσης Suspect ενημερώνουμε τις καταθέσεις του Suspect. Η πολυπλοκότητα της μεθόδου αυτής είναι όση η πολυπλοκότητα της searchbyAFM(int AFM) που ισούται με O(logn) όπου N το πλήθος των TreeNodes του δέντρου.

-Μέθοδος searchbyAFM(int AFM):

Στη μέθοδο αυτή ο χρήστης δίνει ένα ΑΦΜ και αν υπάρχει Suspect με τέτοιο ΑΦΜ επιστρέφει όλα τα στοιχεία του Suspect αυτού, αλλιώς τυπώνει μήνυμα ότι δεν υπάρχει Suspect με τέτοιο ΑΦΜ. Καθώς το κλειδί με το οποίο εισάγωνονται οι Suspects στο δέντρο είναι το ΑΦΜ τους , η μέθοδος μας κάνει πρακτικά μια δυαδική

αναζήτηση στο δέντρο ξεκινώντας από την ρίζα. Δηλαδή πρώτα κοιτάει την ρίζα του υποδέντρου, αν το κλειδί της ρίζας ισούται με το ΑΦΜ που δώσαμε επιστρέφει τον Suspect της ρίζας, αλλιώς κοιτάει αν είναι μικρότερο ή μεγαλύτερο από το κλειδί της ρίζας. Αν είναι μικρότερο επαναλαμβάνει την ίδια διαδικασία στο αριστερό υποδέντρο, αν όχι τότε στο δεξί υποδέντρο. Η διαδικασία αυτή επαναλαμβάνεται μέχρι το τέλος του δέντρου, αν δεν έχει βρεθεί το ΑΦΜ. Επειδή επιτελείται δυαδική αναζήτηση προφανώς πολυπλοκότητα της μεθόδου αυτής είναι O(logN), όπου N το πλήθος των TreeNodes του δέντρου.

-Μέθοδος List searchByLastName(String last_name):

Η μέθοδος αυτή επιστρέφει μια λίστα με όλους τους χρήστες που έχουν επίθετο το last_name. Η μέθοδος μας αρχικά χρησιμοποιεί την ουρά από την 1^{η} εργασία την οποία ονομάζουμε list, η οποία δέχεται αντικείμενα τύπου Suspect. Με αναδρομική διαδικασία διασχίζεται όλο το δέντρο και όποιου κόμβος το Suspect έχει ίδιο επίθετο ίδιο με το last_name , τότε το προσθέτουμε στην list , με την μέθοδο AddLast(). Αν το μέγεθος της list γίνει μεγαλύτερο του 5 τότε σταματάμε την διαδικασία.

Πολυπλοκότητα της μεθόδου αυτής είναι O(N) επειδή πρέπει να διασχίσουμε όλο το δέντρο, όπου N το πλήθος των TreeNodes του δέντρου.

-Μέθοδος double getMeanSavings():

Η μέθοδος αυτή επιστρέφει τον μέσο όρο Savings όλων των Suspects. Η μέθοδος μας ξεκινάει από την ρίζα του δέντρου και προσθέτει τα savings του Suspects της ρίζας , έπειτα καλείται η ίδια μέθοδος για το δεξί υποδέντρο της ρίζας και τέλος για το αριστερό υποδέντρο της ρίζας, έως ότου να φτάσουμε στο τέλος του δέντρου. Τελικά διαιρούμε την μεταβλητή που αποθηκεύσαμε το άθροισμα των savings με το πλήθος των κόμβων του δέντρου, για να επιστρέψουμε τον μέσο όρο.
Πολυπλοκότητα της μεθόδου αυτής είναι O(N) επειδή πρέπει να διασχίσουμε όλο το δέντρο, όπου N το πλήθος των TreeNodes του δέντρου.

void insert(Suspect item):

Η insert το μόνο που πρακτικά κάνει είναι να καλεί την **insertR(TreeNode h,Suspect x)** η οποία με την χρήση της Math.random() μας βοηθάει στην δημιουργία του Τυχαιοποιημένου Δέντρου Δυαδικής Αναζήτησης. Ειδικότερα ,με αυτόν τον τρόπο βοηθούμαστε στο να

χτίσουμε ισοζυγισμένα ΔΔΑ με πολύ μικρή στατιστική πιθανότητα να καταλήξουμε στην χειρότερη περίπτωση ,ενός δηλαδή εκφυλισμένου ΔΔΑ.Η πολυπλοκότητα της είναι κατά την μέση και χειρότερη περίπτωση O(N). Με πιθανότητα 1/h.N+1- ο κόμβος ο οποίος εισάγεται γίνεται ρίζα του δέντρου με την μέθοδο **insertT(TreeNode h,Suspect x)**. Περισσότερα για την εν λόγω μέθοδο ακολουθούν παρακάτω.

TreeNode insertT(TreeNode h,Suspect x):

Η ύπαρξη της μεθόδου μας διευκολύνει στην εισαγωγή ενός κόμβου σαν ρίζα στο δέντρο ή υποδέντρο μας. Πρώτα εισάγει τον κόμβο στην ορθή θέση που του αναλογεί σύμφωνα με τους κανόνες των ΔΔΑ και στην συνέχεια με περιστροφές (δεξιές ή αριστερές) το τοποθετεί ως ρίζα έχοντας πλέον αλλάξει την δομή του δέντρου μας(παραμένοντας όμως σωστό).Με αυτόν τον τρόπο μειώνουμε την πιθανότητα ενός εκφυλισμένου ΔΔΑ.

Void Remove(int val):

Η μέθοδος αυτή αφαιρεί έναν κόμβο από το ΔΔΑ μας καλώντας το **removeR (TreeNode h,int v)** η οποία αναδρομικά καλείται και επιστρέφει την νέα ρίζα του ΔΔΑ.Η remove έχει πολυπλοκότητα O(logN) αφού επιλέγει κάθε φορά σε ποιο μέρος του δέντρου θα κοιτάξει (h.left ή h.right).Η χειρότερη περίπτωση είναι αυτή οπού το δέντρο καταλήγει σε εκφυλισμένο και εμείς ζητήσουμε το μικρότερο ή μεγαλύτερο κλειδί ,σε αυτή την περίπτωση έχουμε πολυπλοκότητα O(N).Περισσότερα για την **removeR(TreeNode h,int v)** ακολουθούν παρακάτω.

TreeNode removeR(TreeNode h, int v):

Η συγκεκριμένη μέθοδος με αναδρομικές κλήσεις αναζητά δυαδικά το στοιχείο με κλειδί v(εξού και η πολυπλοκότητα O(logN)).Όταν το βρεί το αφαιρεί και στην συνέχεια καλούμε την μέθοδο **joinLR(TreeNode a,TreeNode b)** του οποίου τα ορίσματα είναι το δεξί και αριστερό φύλλο της ρίζας έτσι ώστε να τα ενώσουμε και να μην έχουμε διαχωρίσει το ένα υποδέντρο με το άλλο.

Καλείται και αυτή η μέθοδος αναδρομικά ώσπου να βρούμε null στοιχείο δεξιά η αριστερά των φύλλων.

Στο τέλος της κλήσης αυτής της μεθόδου έχουμε καταφέρει να αφαιρέσουμε το στοιχείο με το κλειδί που ζητήσαμε και «γεμίσαμε» το κενό μεταξύ των δύο υποδέντρων επιστρέφοντας την νέα ανανεωμένη ρίζα.

Void printTopSuspects(int k):

Στην μέθοδο αυτή χρησιμοποιούμε την δομή από την προηγούμενη εργασία 2.Συγκεκριμένα δημιουργούμε μια priorityqueue η οποία γεμίζει με αντικείμενα τύπου suspects.Μέσα στην μέθοδο κάνουμε insert(K) ώστε να θέσουμε το όριο των υπόπτων στην λίστα και να κρατήσουμε μνήμη O(k).

Στην συνέχεια καλούμε την **printTopSuspectsR(TreeNode h)**(δέχεται σαν όρισμα την ρίζα του δέντρου) η οποία καλείται αναδρομικά και γεμίζει με την μέθοδο insert την λίστα μας. Όταν φτάσουμε στο όριο k τότε συγκρίνουμε τον Suspect που περιμένει να μπει στην λίστα με τον min της λίστας και συνεχίζουμε μέχρι να προσπελάσουμε όλο το δέντρο μας αναδρομικά. Στο τέλος εκτυπώνουμε την λίστα μέσω της ήδη υπάρχουσας μεθόδου στην List.H

πολυπλοκότητα της μεθόδου είναι O(N)(η προσπέλαση του δέντρου) *O(logN)(η insert της List μέσω swim κλπ)=O(NlogN).

void printbyAFM():

Η μέθοδος αυτή υλοποιείται πολύ εύκολα καθώς το κλειδί που χρησιμοποιούμε για την ταξινόμηση στο δέντρο είναι το ΑΦΜ των υπόπτων. Η μέθοδος μας κάνει μια ενδοδιατεταγμένη διάσχιση στο δέντρο ξεκινώντας από την ρίζα (δηλαδή πηγαίνει πρώτα στο αριστερό υποδέντρο μετά στον ίδιο τον κόμβο και τέλος στο δεξί υποδέντρο). Η πολυπλοκότητα της μεθόδου προφανώς είναι O(N) καθώς κάνει διάσχιση όλο το δέντρο.

Για την υλοποίηση της εργασίας χρησιμοποιήθηκαν: Eclipse,Notepad++,CMD.