ΘΑΝΑΣ ΚΟΥΡΟ – 3170078

ΝΙΚΟΛΕΤΑ ΠΑΠΑΓΕΩΡΓΙΟΥ – 3170234

ΜΕΡΟΣ Δ

- void insert (Suspect item)

  Αρχικά ελέγχουμε με τη μέθοδο searchByAFM εάν υπάρχει ήδη κάποιος ύποπτος με το ΑΦΜ που έχει δοθεί για εισαγωγή. Εάν υπάρχει, τυπώνουμε κατάλληλο μήνυμα, διαφορετικά καλούμε την private μέθοδο insertR. Η εισαγωγή του item θα γίνει είτε στη ρίζα του υποδέντρου (ή δέντρου) με πιθανότητα 1/(h.N+1) είτε θα εισαχθεί σαν φύλλο. Εάν πρέπει να εισαχθεί στη ρίζα (του δέντρου ή κάποιου υποδέντρου) καλούμε την private μέθοδο insertAsRoot η οποία με κατάλληλες περιστροφές (μέθοδοι rotR και rotL) βάζει το item στη ρίζα του υποδέντρου που θα κληθεί. Διαφορετικά, όσο δε καλείται η insertAsRoot, ανάλογα με την τιμή του ΑΦΜ θα γίνεται αναδρομή από την insertR στο δεξί ή αριστερό υποδέντρο. Δεν ξεχνάμε την εντολή h.N++ (στην insertR και την insertAsRoot) έτσι ώστε το N κάθε κόμβου να αυξηθεί κατά 1. (Εφόσον προχωρήσει η αναδρομή σε κάποιο νέο υποδέντρο, σαφώς το item θα ανήκει σε αυτό το υποδέντρο , γι' αυτό και αυξάνουμε το N).

- void load (String filename)

  Χρησιμοποιούμε από τη βιβλιοθήκη της java την BufferedReader για το διάβασμα του αρχείου που έχει δοθεί. Διαβάζουμε γραμμή προς γραμμή το αρχείο και σε κάθε γραμμή δημιουργούμε ένα νέο αντικείμενο τύπου Suspect και το εισάγουμε στο δέντρο μας, μέσω της μεθόδου insert. Εάν υπάρχει άλλος ύποπτος με το ΑΦΜ αυτό, τότε δεν εισάγεται και τυπώνει κατάλληλο μήνυμα η insert.

- void updateSavings (int AFM, double savings)

  Πρώτα απ' όλα, ελέγχουμε εάν υπάρχει κάποιος ύποπτος με το ΑΦΜ που δόθηκε, πάλι με χρήση της searchByAFM. Εάν δεν υπάρχει τυπώνουμε κατάλληλο μήνυμα και επιστρέφει από τη μέθοδο. Διαφορετικά , έχουμε την εντολή searchByAFM(AFM).setSavings(savings); . Η μέθοδος επιστρέφει Suspect, οπότε έχουμε το αντικείμενο με το ΑΦΜ που δόθηκε και με τη μέθοδο setSavings της Suspect ενημερώνουμε τις καταθέσεις.

- void searchByAFM (int AFM)

  Υλοποιούμε την μέθοδο με δομή επανάληψης. Έχουμε τη μεταβλητή current τύπου Treenode, η οποία παίρνει την τιμή της ρίζας. Στην επανάληψη ελέγχουμε 4 καταστάσεις. Εάν current == null τότε έχουμε αποτύχει στην αναζήτηση και επιστρέφει null, καθώς δεν υπάρχει ύποπτος με το συγκεκριμένο ΑΦΜ. Εάν το ΑΦΜ που ψάχνουμε είναι ίσο με το ΑΦΜ του ύποπτου που βρίσκεται στον τρέχοντα κόμβο, τότε επιστρέφουμε το item τύπου Suspect του τρέχοντος κόμβου. Αν δεν ισχύει τίποτα από τα παραπάνω συγκρίνουμε το δοσμένο ΑΦΜ με το ΑΦΜ του κόμβου και εάν είναι μικρότερο συνεχίζουμε την αναζήτηση στο αριστερό παιδί του current (current = current.left), διαφορετικά πάμε στο δεξί παιδί.

- void searchByLastName (String LastName)

  Χρησιμοποιούμε τη κλάση List (πρώην StringDoubleEndedQueueImpl) της 1ης εργασίας με κάποιες αλλαγές (βασική αλλαγή είναι ότι θα περιέχει αντικείμενα τύπου Suspect) . Δημιουργούμε μία νέα λίστα τύπου List με όνομα queue. Έπειτα, καλούμε την private μέθοδο searchBLN η οποία αναδρομικά και με ενδοδιατεταγμένη διάσχιση ελέγχει κάθε κόμβο και συγκρίνει το επώνυμο του κάθε κόμβου με το επώνυμο που δόθηκε. ΄Κάθε φορά που βρίσκει ένα item με ίδιο επώνυμο, με την εντολή queue.addFirst(h.item), εισάγει το item στη λίστα. Επιστρέφοντας στην searchByLastName έχουμε τη λίστα. Εάν το μέγεθός της είναι 0, τότε η μέθοδος επιστρέφει null. Εάν το μέγεθος είναι <= 5 τυπώνει τους ύποπτους με την printQueue() της List και τέλος επιστρέφει τη λίστα (για size >=1).

- void remove (int AFM)

  Έχουμε την εντολή Suspect suspect = searchByAFM(AFM). Εάν η μεταβλητή suspect έχει τιμή null, δεν υπάρχει ύποπτος με το δοσμένο ΑΦΜ και τυπώνουμε κατάλληλο μήνυμα και επιστρέφουμε. Στην άλλη περίπτωση τυπώνουμε τα στοιχεία του ύποπτου και με την εντολή root = removeR(root, AFM) διαγράφουμε τον ύποπτο από το δέντρο. Η αναδρομική μέθοδος removeR είναι ιδιωτική και θα την περιγράψουμε τώρα. Κλασικά, συγκρίνουμε το ΑΦΜ με τον τρέχοντα κόμβο και εάν είναι μικρότερο κάνουμε αναδρομή στο αριστερό υποδέντρο, ενώ αν είναι μεγαλύτερο στο αριστερό. Ανάλογα με το υποδέντρο που θα προχωρήσουμε δε ξεχνάμε την εντολή h.N- - για να μειωθεί κατά 1 το N του κόμβου στο οποίο θα γίνει η αναδρομή (και εννοείται θα γίνεται για κάθε κόμβο έως ότου βρεθεί το στοιχείο που πρέπει να αφαιρεθεί από το δέντρο). Όταν βρούμε τον κόμβο, του οποίου θέλουμε να διαγράψουμε το item, τότε καλούμε την private μέθοδο joinLR.
  Η joinLR δέχεται ως όρισμα τα 2 υποδέντρα a ,b που είναι το αριστερό και το δεξί παιδί αντίστοιχα, του κόμβου που πρέπει να αφαιρεθεί. Η μέθοδος βάζει το υποδέντρο b κάτω από το a με πιθανότητα a.N/N ή το a κάτω από το b. Στην πρώτη περίπτωση κάνουμε a.N += b.N και στην δεύτερη b.N += a.N για να ενημερωθούν κατάλληλα τα πεδία N των κόμβων όλων των υποδέντρων μέχρι να τελειώσει η διαδικασία.

- void getMeanSavings()

  Εάν η ρίζα του δέντρου είναι ίση με null, σημαίνει πως δεν υπάρχει κάποιος ύποπτος και επιστρέφουμε 0. Διαφορετικά στην μεταβλητή meanSavings εκχωρούμε το αποτέλεσμα της ιδιωτικής μεθόδου MeanSavings. Η MeanSavings είναι αναδρομική και κάθε φορά προσθέτει τις καταθέσεις του τρέχοντος κόμβου και των παιδιών του (με αναδρομή), έως ότου προστεθούν οι καταθέσεις όλων των ύποπτων. Η getMeanSavings επιστρέφει meanSavings / root.N, δηλαδή το μέσο όρο καταθέσεων.

- void printTopSuspects (int k)

  Θα χρησιμοποιήσουμε την κλάση PQ, που είναι η ουρά προτεραιότητας της 2ης μας εργασίας. Πλέον τα στοιχεία της ουράς θα είναι τύπου Suspect.
  Δημιουργούμε στη μέθοδο την ουρά pq. Έπειτα καλούμε την βοηθητική, ιδιωτική μέθοδο setPQSuspects. Η μέθοδος πάλι με ενδοδιατεταγμένη διάσχιση ελέγχει τους κόμβους και αν το στοιχείο στον τρέχοντα κόμβο πρέπει να εισαχθεί στην pq,

εισάγεται. Συγκεκριμένα η λογική είναι ακριβώς είναι με τον τρόπο που υλοποιήσαμε το μέρος Γ της 2ης εργασίας. Ουσιαστικά, εισάγουμε στην ουρά αντικείμενα τύπου Suspect και στην ρίζα του δέντρου βρίσκεται κάθε φορά το αντικείμενο με την μεγαλύτερη προτεραιότητα. Μεγαλύτερη προτεραιότητα στην ουρά έχει ο ύποπτος που είναι λιγότερο ύποπτος από τους υπόλοιπους. Η σύγκριση γίνεται με χρήση της compareTo που έχουμε υλοποιήσει στην κλάση Suspect). Όταν το size της ουράς γίνει ίσο με κ και πρέπει να προσθέσουμε κάποιον ύποπτο, που είναι περισσότερο ύποπτος από τον στη ρίζα της pq, τότε διαγράφουμε το στοιχείο της ρίζας της pq (pw.getmax()) και εισάγουμε τον νέο. Έτσι η ουρά δε θα έχει ποτέ παραπάνω από k ενεργά στοιχεία. Μόλις τελειώσουμε τις αναδρομές επιστρέφουμε την prontTopSuspects και με μία for loop τυπώνουμε τα στοιχεία των υπόπτων της ουράς (που θα είναι σε αύξουσα σειρά, καθώς στη ρίζα θα βρίσκεται πάντα ο λιγότερο ύποπτος).

- void printByAFM()

  Η μέθοδος καλεί την αναδρομική, private inOrder, η οποία με ενδοδιατεταγμένη διάσχιση (δηλαδή ταξινομημένη σειρά) τυπώνει τα στοιχεία των ύποπτων.