ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ 3

* p3190179 - Προμπονάς Αντώνιος
* p3190031 - Γεωργιάδης Ελευθέριος

Στην εργασία 3 υλοποιήσαμε ένα δέντρο δυαδικής αναζήτησης βασισμένοι στην διεπαφή που μας δόθηκε.

Οι μέθοδοι είναι οι εξής:

* **Insert**: πραγματοποιεί την προσθήκη ενός νέου κόμβου στο δέντρο. Η μέθοδος υπολογίζει και τον τυχαίο παράγοντα, δηλαδή το δέντρο είναι τυχαιοποιημένο. Πρώτα αναζητείται το ΑΦΜ του υπόπτου που θέλουμε να εισάγουμε. Αν υπάρχει εμφανίζει αντίστοιχο μήνυμα και η εισαγωγή δεν γίνεται. Αν δεν υπάρχει, προσθέτουμε τον ύποπτο (χρησιμοποιούνται και οι βοηθητικές συναρτήσεις insertR, insertT, rotR, rotL). Πολυπλοκότητα Ο(h), όπου h το ύψος του δέντρου, στην χειρότερη περίπτωση (επειδή το δέντρο είναι τυχαιοποιημένο μπορεί να γίνει εισαγωγή του κόμβου και πιο ψηλά).
* **Load**: δημιουργεί αντικείμενο τύπου File και μέσω του scanner. Διατρέχουμε το txt αρχείο (έχουμε ένα δοκιμαστικό, το test.txt) και κάθε φορά που αλλάζει η γραμμή καλούμε την insert για να προσθέσει τον ύποπτο στο δέντρο. Έχει πολυπλοκότητα Ο(Μ), όπου Μ οι γραμμές του txt αρχείου.
* **updateSavings**: αναζητά τον ύποπτο με το ΑΦΜ που της περνάμε. Αν υπάρχει ανανεώνει τα savings του. Αν δεν υπάρχει εμφανίζεται κατάλληλο μήνυμα. Πολυπλοκότητα Ο(logn), και Ο(Ν) στην χειρότερη περίπτωση.
* **searchByAFM**: χρησιμοποιεί την βοηθητική αναδρομική μέθοδο searchByAFMR. Μέσω διαδοχικών αναδρομών συγκρίνει το τρέχον ΑΦΜ με αυτό που έχουμε περάσει. Για λόγους αισθητικής του προγράμματος κυρίως (και για να μην αλλάξουμε την διεπαφή) φτιάξαμε την searchByAFMNoPrint η οποία είναι ίδια με την searchByAFM αλλά δεν κάνει print κάποια δεδομένα εκεί που δεν χρειάζονται. Πολυπλοκότητα Ο(logn), και Ο(Ν) στην χειρότερη περίπτωση.
* **searchByLastName**: διασχίζουμε το δέντρο (level order search) και σε κάθε επανάληψη του βρόγχου ελέγχουμε αν το όνομα του αντικειμένου του τρέχοντος κόμβου είναι ίσο με την παράμετρο της συνάρτησης. Τότε το προσθέτουμε στην λίστα suspects τύπου SingleLinkedList. Ο τύπος αυτός είναι ίδιος με την λίστα της εργασίας 1 αλλά έχει διαγραφεί η δυνατότητα διπλής σύνδεσης (έγινε μονής σύνδεσης) και υποστηρίζει generics. Αν η λίστα έχει μέχρι 5 ύποπτους αυτοί εμφανίζονται πριν την έξοδο της μεθόδου. Διαφορετικά, μετά την κλήση της. Η πολυπλοκότητα είναι Ο(Ν) αφού διασχίζουμε όλο το δέντρο.
* **getMeanSavings:** Για να υλοποιήσω τη μέθοδο αυτή χρειάστηκε να δημιουργήσουμε 2 βοηθητικές συναρτήσεις. Την inorder και την inorder\_Recursive. Και οι δυο συναρτήσεις λειτουργούν εν μέρη σαν getters αφού επιστρέφουν μία τιμή double, η οποία αποτελεί το μ.ο. των συνολικά "κακών" αποταμιεύσεων των υπόπτων. Η τελική τιμή που επιστρέφεται και εμφανίζεται εν τέλη στην οθόνη στην ουσία υπολογίζεται από την inOrder\_Recursive. Η συγκεκριμένη συνάρτηση προσθέτει κάθε φορά στη μεταβλητή sum τα mean savings του κάθε υπόπτου και στη συνέχεια το αποτέλεσμα αυτό διαιρείται με την μεταβλητή count, η οποία μετράει το πόσα αντικείμενα προστίθενται στη sum. Η πολυπλοκότητα της συγκεκριμένης μεθόδου είναι Ο(Ν), λόγω των αναδρομών που καλούνται μέσα στην inOrder\_Recursive και έχουν ως αποτέλεσμα την διερεύνηση όλων των φύλλων ενός δέντρου.
* **printTopSuspects:** Για την συγκεκριμένη μέθοδο χρειαστήκαμε δύο βοηθητικούς πίνακες cheat && help. Ο πρώτος περιέχει τη διαφορά μεταξύ getMeanSavings && getTaxedSavings και ο 2ος τα περιεχόμενα του h που δίνει κάθε φορά η αναδρομική μέθοδος. Η μέθοδος αυτή είναι η void preoder, η οποία χρησιμεύει στο να ενημερώνει τους πίνακες μέσα από την προδιατεταγμένη διάσχιση των κόμβων. Ωστόσο, επειδή το αρχικό μέγεθος των κόμβων δηλώνεται με βάση το πλήθος των εισαγωγών που έχουν γίνει μέχρι και τη στιγμή που θα κληθεί για πρώτη φορά η μέθοδος αυτή από το μενού , αναγκαστικά θα πρέπει να αυξάνουμε το μέγεθός της κάθε φορά που η global παράμετρος που χρησιμοποιείται ειδικά ως δείκτης των πινάκων, πλησιάζει το υπάρχον μέγεθος του πίνακα.

Αφού ολοκληρωθεί η διαδικασία ενημέρωσης των πινάκων, οι πίνακες ταξινομούνται με την τεχνική ταξινόμησης bubble sort, η οποία έχει πολυπλοκότητα Ο(Ν^2). Για όλες τις μεθόδους διάταξης έχουμε πολυπλοκότητα Ο(Ν).

Η παραπάνω διαδικασία πραγματοποιείται εφόσον η παράμετρος k είναι μικρότερη από το πλήθος των εισαγωγών κατά τη διάρκεια ολόκληρου του προγράμματος.

Σε διαφορετική περίπτωση εκτυπώνεται το κατάλληλο μήνυμα.

* **printByAFM**: Για την υλοποίηση αυτής της μεθόδου χρησιμοποιήσαμε την ενδοδιατεταγμένη διάσχιση καθώς η διάσχιση αυτή σύμφωνα με την θεωρία εκτυπώνει τα κλειδιά του δέντρου σε αύξουσα σειρά. Έχει πολυπλοκότητα Ο(Ν), αφού έχουμε εισάγει Ν στοιχεία στο δέντρο.
* **remove**: αφαιρεί ένα στοιχείο από το δέντρο.

Πρώτα βρίσκουμε τον κόμβο που θέλουμε να διαγράψουμε και τον πατέρα του. Έπειτα, ελέγχουμε αν ο κόμβος που πρέπει να διαγραφεί έχει παιδιά ή όχι, για να μεταφερθούν στον πατέρα. Η πολυπλοκότητα είναι Ο(logn) ή Ο(Ν) στην χειρότερη περίπτωση που χρειάζεται να διασχίσουμε ολόκληρο το δέντρο.

Γενικά, είχαμε κάποιες δυσκολίες με το πεδίο Ν της κλάσης TreeNode (είναι πάντα μηδέν) για αυτό χρησιμοποιήσαμε την μεταβλητή counter. Επίσης είχαμε προβλήματα με την recursive remove και την join, οπότε χρησιμοποιήσαμε την υλοποίηση του εργαστηρίου με κάποιες μικρο-αλλαγές δικές μας εννοείται. Επίσης, για λόγους αισθητικής του προγράμματος πάλι, χρησιμοποιούμε την μεταβλητή removed και r για να αποθηκεύσουμε τον κόμβο που αφαιρείται (λόγω αναδρομής άλλαζε η τιμή του συνέχεια), και τυπώνουμε τα στοιχεία του μέσα στην main.

Ο κώδικας μεταγλωττίζεται και τρέχει κανονικά και στο command line.