Αναφορά Παράδοσης

Δομές Δεδομένων 2020-2021

3η Εργασία

Φοιτητές:

Αποστόλου Ιωάννης – p3190013

Καλαντζής Ηλίας – p3190068

Έκδοση Java 15

- **insert:** Στο insert χρησιμοποιούμε, μέθοδο δέντρα δυαδικής αναζήτησης. Ελέγχουμε πρώτα εάν το δέντρο είναι **null** (κενό) και εαν είναι προσθέτουμε έτσι το αντικείμενο ή βλέπουμε εάν υπάρχει ύποπτος με το ίδιο ΑΦΜ οπότε εμφανίζουμε μήνυμα στον χρήστη. Εάν περάσουμε από αυτά, ελέγχουμε το δέντρο και ορίζουμε μια πιθανότητα (με τυχαίο αριθμό) να βάλουμε το αντικείμενο στο κάτω μέρος του δέντρου και μετά να κάνουμε περιστροφές για την καλύτερη δομή του δέντρου (βασικό concept). Παρόλα αυτά ο τυχαίος αριθμός, δεν σημαίνει ότι είναι τόσο “τυχαίο” το δέντρο. Είναι O(N) διότι, αφού έχουμε δυαδικό δέντρο στην αναζήτηση της θέσης δεν διατρέχουμε όλο το δέντρο.
- **load** (Δεν μας ενδιαφέρει η πολυπλοκότητα) **:** Σε αυτήν την μέθοδο πρέπει να διαβάσουμε από ένα αρχείο txt, τα στοιχεία των υπόπτων. Σαν παράμετρο έχουμε ένα string όπου είναι το όνομα του αρχείου που διαβάζουμε. Χρησιμοποιούμε την συνάρτηση **readFile** με σκοπό να ελένξουμε εαν το αρχείο που πληκτρολόγησε ο χρήστης υπάρχει. Έπειτα χρησιμοποιώντας την **BufferReader** διαβάζουμε το αρχείο. Πολύ απλά έχουμε μια while η η οποία τελειώνει μόλις βρεθεί κενή γραμμή, ενώ παίρνουμε την κάθε γραμμή, ξεχωρίζουμε τις λέξεις και έπειτα τις καταχωρούμε στην **dome** η οποία είναι τύπου **RandomizedBST**.
- **UpdateSavings:** Χρησιμοποιώ την βοηθητική συνάρτηση **searchByAFMRoot** η οποία χρησιμοποιείται από την **searchByAFM**. Οπότε χρησιμοποιώ την ίδια τεχνική με την **searchByAFM** και το αντικείμενο που επιστρέφει του αλλάζω την τιμή του **saving**, με την μέθοδο **setSavings** που υπάρχει στην **Suspect** και απλά αλλάζει μια τιμή. Η πολυπλοκότητα είναι κάτω από **O(N)** καθώς χρησιμοποιούμε την **searchByAFM** και μετά απλά αλλάζουμε μια μεταβλητή.
- **searchByAFM:** Εφαρμόζουμε την μέδοθο της δυαδικής αναζήτησης. Ξεκινάμε από το την ρίζα και ελέγχουμε εαν είναι αυτό το στοχείο εαν όχι ελέγχουμε εαν το ΑΦΜ που ψάχνουμε είναι μικρότερο ή μεγαλύτερο από το ΑΦΜ της ρίζας, εαν είναι μικρότερο παίρνουμε το αριστερό υποδέντρο ενώ εαν ειναι μεγαλύτερο το δεξιό υποδέντρο. Όταν βρει φυσικά το στοιχείο σταματάει η αναζήτηση. Από την στιγμή που δεν διαπερνάει όλο το δέντρο είναι λιγότερο από O(N)
- **searchByLastName:** Η μέθοδος επιστρέφει αποτέλεσμα τύπου **List** όπου είναι μια λίστα από προηγούμενη εργασία και μπορείτε να την βρείτε στα αρχεία ως **List.java**. Επειδή μπορούν οι ύποπτοι να έχουν το ίδιο επίθετο οφείλουμε να διατρέξουμε όλο τον πίνακα. Έχουμε δημιουργήσει μια βοηθητική μέθοδο την **returnLastNames** η οποία διατρέχει όλο το δέντρο των υπόπτων με σειρά pre order και για κάθε ύποπτου που βρίσκει με το ίδιο επίθετο τον προσθέτει στην λίστα. Διατρέχει όλο το δέντρο μια φορά οπότε είναι O(N).
- **remove:** Η remove για αρχή με την απλή μέθοδος της αναζήτησης όπως αναφέραμε και παραπάνω προσπαθεί να βρεί το αντικείμενο. Όταν το βρεί, θα φροντίσει να διαγράψει αυτό το στοιχείο αλλά επίσης να ενώσει το υπόλοιπο δέντρο με αυτό το κενό που υπάρχει.
- **getMeanSavings:** στην **RandomizedBST** υπάρχει μια **int** μεταβλητή η **size.** Αυτή μετράει το πλήθος των στοιχείων που υπάρχουν μέσα στο δέντρο. Διατρέχουμε όλο το δέντρο με την μέθοδο pre order, και προσθέτουμε από κάθε αντικείμενο τα **savings**, στο τέλος διαιρούμε το άθροισμα με το **size** και βγάζουμε τον μέσο όρο.
- **printTopSuspect**s: (Δεν μας ενδιαφέρει η πολυπλοκότητα) Γι’αυτή την μέθοδο χρησιμοποιούμε την **PQ** (ουρά προτεραιότητας από προηγούμενη εργασία). Ουσιαστικά ψάχνουμε τους **k** πιο ύποτους που υπάρχουν. Από την Προηγούμενη εργασία είχαμε την PQ

η οποία ήταν μια ουρά η οποία μπορούσε να έχει έως **k** στοιχεία μέσα της τα οποία έβγαζε το μικρότερο και έβαζε το μεγαλύτερο. Με αυτήν την λογική όταν η ουρά είναι κάτω από **k**, βάζουμε το στοιχείο μέσα ότι και να είναι. Όταν το μέγεθος της PQ φτάσει ίσο με **k** τότε ελέγχουμε εαν το μικρότερο της PQ είναι μεγαλύτερο από το αντικείμενο που θέλουμε να προσθέσουμε. Εαν είναι, αφαιρούμε το μικρότερο και προσθέτουμε το καινούργιο. Για να διατρέξουμε όλο τον πίνακα τον κάνουμε με τον μέθοδος της αναδρομής και του pre order.

- **printByAFM:** Το δέντρο μας λόγο της **insert** κάθε φορά που μπαίνει ένα στοιχείο ταξινομείται. Οπότε ο πιο εύκολος και γρήγορος τρόπος είναι να τρέξουμε το δέντρο με pre order σειρά και να εμφανίσουμε τους υπόπτους. Χρησιμοποιούμε επίσης την βοηθητική συνάρτηση **printPreOrder**. Από την στιγμή που διατρέχουμε όλα τα στοιχεία του δέντρου μόνο μια φορά είναι O(N).

Οδηγίες

Εάν υπάρχει κάποιο πρόβλημα κατά το τρέξιμο του προγράμματος μπορείτε να δείτε τα παρακάτω:

- Δείτε εάν έχετε εγκατεστημένη την Java 15.0.1
- Ο κώδικας μας είναι σε πακέτο, ώστε να επικοινωνούν εύκολα τα αρχεία μας, άρα πρέπει να γίνουν compile κατά αυτόν τον τρόπο
- Το πρόγραμμα είναι σε πακέτα οπότε δοκιμάστε να κάνετε compile με “javac -d . src/*.java”.
- Έπειτα θα δημιουργηθούνε τα αρχεία .class μέσα στον ίδιο φάκελο ή σε διαφορετικό ανάλογα με το path που θα έχετε. Και τρέχετε την Main.
- Για το αρχείο μπορείτε να βάλετε τα στοιχεία στο αρχείο file.txt ή μπορείτε στον φάκελο εκεί που βρίσκεται το file.txt να βάλετε το δικό σας txt και από το πρόγραμμα θα σας ζητήσει να βάλετε το όνομα **χωρίς την επέκταστη .txt**.

IDE: Intellij (της JetBrain).

Για οποιοδήποτε θέμα ή απορία μπορείτε να επικοινωνήσετε στα ακαδημαϊκά μας email,

p3190013@aueb.gr [Ιωάννης Αποστόλου] και p3190068@aueb.gr [Ηλίας Καλαντζής]