Δομές Δεδομένων: Εργασία 3

Μέθοδοι του RandomizedBST

απο Βασιλης Παπαδημας (3220150) & Μαριος Ματσα (3220120)

void insert(LargeDepositor item)

Χρησιμοποιούμε την TreeNode RandomizedBST.foundByAFM(int AFM) για να βρούμε αν υπάρχει ήδη κάποιο TreeNode στο δέντρο με ίδιο ΑΦΜ με του item. Αν ναι (curr != null) τότε εκτυπώνουμε το μήνυμα, αλλιώς θέτουμε root = insertAsRoot(root, item) και αυξάνουμε το N του root. Η TreeNode insertAsRoot(TreeNode h, LargeDepositor x) και η TreeNode RandomizedBST.insertT(TreeNode h, LargeDepositor x) που χρησιμοποιεί (και οι TreeNode RandomizedBST.rotR(TreeNode h) και TreeNode RandomizedBST.rotL(TreeNode h) που χρησιμοποιεί εκείνη) έχουν υλοποιηθεί όλες σύμφωνα με τα παραδείγματα που έχουν δοθεί στα πλάισια του μαθήματος.

void load(String filename)

Διαβάζουμε σειριακά το αρχείο που ορίζει το filename, σπάμε την κάθε γραμμή σε substrings με spacer τα κενά μέσω regex, δημιοργούμε αντικείμενα LargeDepositor με τα κατάλληλα δεδομένα από τα strings (που όταν χρειάζεται μετατρέπονται σε άλλες δομές δεδομένων) και καλούμε την insert με αυτά ως όρισμα. Αν προκύψει κάποιο Exception σε αυτή την διαδικασία, κάνουμε throw RuntimeException.

List searchByLastName(String last_name)

Δημιουργούμε μια List για να αποθηκεύσουμε το αποτέλεσμα της αναζήτησης. Η υλοποιήση της List είναι βασισμένη στον σχετικό κώδικα που γράψαμε για μια προηγούμενη εργασία του μαθήματος, με ελαφρές αλλαγές. Η αναζήτηση γίνεται με αναδρομικές κλήσεις της void traverseAndBuild(TreeNode node, List ls), η οποία κάνει inorder traversal του BST και προσθέτει τον κόμβο στον οποίο βρίσκεται στην List αν έχει ίδιο επώνυμο με το όρισμα last_name (το οποίο για ευκολία το αποθηκεύουμε σε ένα ειδικό πεδίο της List). Οταν τελειώσει η traverseAndBuild, ελέγχουμε πόσα στοιχεία έχει η ls (με το πεδίο της N) και αν είναι μεταξύ του 0 και του 5 τα εκτυπώνουμε πριν την επιστρέψουμε.

double getMeanSavings()

Επιστρέφουμε το αποτέλεσμα της κλήσης της double traverseAndSum(TreeNode node), η οποία με αναδρομικές κλήσεις κάνει inorder traversal του BST (φυσικά δεν έχει σημασία η σειρά διασχίσης) και αθροίζει και επιστρέφει τα savings όλων των LargeDepositor που υπάρχουν στο δέντρο, διαιρεμένο με το πεδίο N του BST.

void printTopLargeDepositors(int k)

Δημιουργούμε ενα priority queue **pq** με capacity k (η υλοποίηση της κλάσης PQ είναι από προηγούμενη εργασία, με την διαφορά ότι κάναμε το capacity σταθερό για διευκολύνση στην χρήση της σε αυτό το πλαίσιο) και μια List που θα χρειαστούμε στην εκτύπωση των αποτελεσμάτων.

Καλούμε την void traverseAndRank(TreeNode node, PQ pq), η οποία αναδρομικά διασχίζει το δέντρο (πάλι in-order χωρίς να κάνει διαφορά η σειρά) και προσθέτει το LargeDepositor που είναι αποθηκευμένο στο TreeNode που βρίσκεται σε κάθε κλήση της, εφόσον η pq δεν έχει γεμίσει, ή διαφορετικά εάν το score του τρέχοντος LargeDepositor είναι υψηλότερο από το score του min της pq (ο comparator της τοποθετεί σε αυτή την θέση το LargeDepositor με το μικρότερο score από αυτά της pq), στην οποία περίπτωση πρώτα αφαιρεί το ελάχιστο στοιχείο από την pq ώστε να αποφευχθεί το overflow της.

Μολίς τελειώσει η traverseAndRank, με ένα loop αποθηκεύουμε τα LargeDepositor του pq στην List, χρησιμοποιόντας την ουσιαστικά σαν ένα stack, αφού κάνουμε insertAtFront τα στοιχεία με αύξουσα σειρά κατά score, και μέτα τα κάνουμε removeFromFront με φθίνουσα σειρά, παράλληλα εκτυπώνοντας τα.