Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Δομές Δεδομένων και Αλγόριθμοι

2η άσκηση

Ημερομηνία παράδοσης: 9 Ιουνίου 2024, 18:00
Ο τρόπος υποβολής θα ανακοινωθεί

Ομάδες έως 2 άτομα. Θα μπορείτε να δηλώσετε διαφορετική ομάδα από αυτήν της 1ης εργασίας.
Για τον τρόπο δήλωσης ομάδας, θα υπάρξει σχετική ανακοίνωση
Θα γίνει προφορική εξέταση της κάθε ομάδας. Θα υπάρξει σχετική ανακοίνωση για το πρόγραμμα.

Τελευταία αλλαγή στην εκφώνηση (βλέπε τέλος εγγράφου): 25/04/2024

Σκοπός της άσκησης είναι η σύγκριση διαφορετικών υλοποιήσεων σε Δομές Αναζήτησης σε δέντρα και σε γραμμικό κατακερματισμό (Linear Hashing). Παρόλο που οι υλοποιήσεις που δίνονται είναι στη RAM, μας ενδιαφέρει και η απόδοσή τους σε περίπτωση που είναι στο δίσκο. Θα εξασκηθείτε επίσης στην ανάγνωση δεδομένων (αριθμών) από αρχείο.

Σας δίνονται 4 δομές δεδομένων:

1. org.tuc.avl.AVLTree : δέντρο AVL
2. org.tuc.bst.BSTree : δέντρο BST
3. org.tuc.btree.BTree : δέντρο BTree
4. org.tuc.linearhashing.LinearHashing : υλοποίηση γραμμικού κατακερματισμού (linear hashing)

Μπορείτε να κατεβάσετε τις υλοποιήσεις απο τη διεύθυνση
https://drive.google.com/file/d/1qZhgMluY-SXyLgGnfoIfIQ635JQIUyCM/view?usp=sharing

Οι 4 υλοποιήσεις

Μετατρέψτε τις 4 δομές που σας δίνονται ώστε να υλοποιούν το παρακάτω java interface.

```
package org.tuc.interfaces;
public interface SearchInsert {
	/**
	 * Inserts key into the data structure
	 * @param key
	 */
```

```
    public void insert(int key);

    /**
     * Searches for given key.
     * @param key
     * @return true if key is found, false otherwise
     */
    public boolean searchKey(int key);
}
```

Πρακτικά θα χρειαστεί να φτιάξετε μόνο τη μέθοδο searchKey ώστε να δέχεται και να επιστρέφει ακριβώς αυτό που ζητάει το interface. Η καθ' αυτή αναζήτηση, γίνεται ήδη από τον κώδικα που σας δίνεται με τα ζητούμενα ονόματα και παραμέτρους.

Εφόσον χρησιμοποιήσετε άλλη γλώσσα προγραμματισμού, θα πρέπει να χρησιμοποιήσετε τα ίδια ονόματα μεθόδων, με τους ίδιους τύπους και ίδιο πλήθος παραμέτρων.

Πείραμα - Μετρήσεις

Έχοντας ετοιμάσει τις 4 υλοποιήσεις, θέλουμε να συγκρίνουμε την απόδοση της κάθε υλοποίησης σε συγκεκριμένες λειτουργίες της δομής. Για αυτό το σκοπό θα πάρουμε πειραματικές μετρήσεις για διάφορα μεγέθη εισόδου (πλήθος κλειδιών στη δομή). Θα καταγράψουμε κάθε φορά

- το χρόνο εκτέλεσης σε Nanoseconds για εισαγωγή κλειδιών (μέσος όρος ανά εισαγωγή)
- το χρόνο εκτέλεσης σε Nanoseconds για αναζήτηση κλειδιών (μέσος όρος ανά αναζήτηση)
- το πλήθος "επιπέδων" που προσπελαύνονται κατά την αναζήτηση κλειδιών (μέσος όρος ανά αναζήτηση)

Ως "επίπεδα" στα δέντρα ορίζουμε το πλήθος κόμβων στους οποίους γίνεται προσπέλαση κατά την αναζήτηση. Στο linear search ορίζουμε το πλήθος buckets στα οποία γίνεται προσπέλαση κατά την αναζήτηση (είτε πρωτευόντων buckets, είτε buckets υπερχείλισης).

Τα επίπεδα μας ενδιαφέρουν για την περίπτωση που κάθε υλοποίηση θα ήταν στο δίσκο, όπου κάθε κόμβος ή κάθε bucket θα αντιστοιχούσε σε μία πρόσβαση στο δίσκο (ένα data page). Αντιστοιχεί δηλαδή στο πλήθος των αναγνώσεων από το δίσκο που θα απαιτηθεί για μία αναζήτηση.

Για το btree και το linear hashing θα έχετε 2 στιγμιότυπα κάθε φορά, με διαφορετικές παραμέτρους. Άρα συνολικά θα έχετε 6 στιγμιότυπα στα οποία θα κάνετε εισαγωγή και αναζήτηση κλειδιών!

Σενάριο πειράματος

Το πείραμα αποτελείται από τα παρακάτω βήματα, τα οποία θα επαναληφθούν για πλήθος κλειδιών N=20, 50, 100, 200, 1000, 2500, 5000, 10000, 20000, 40000, 60000, 80000, 100000, 200000, 1000000, 1500000, 2000000, 2500000, 3000000

Τα κλειδιά που θα εισάγετε σας δίνονται σε 19 αρχεία (ένα για κάθε N) στη διεύθυνση https://drive.google.com/file/d/1ALg0YjD8-J0MhmJwcmKxsu5W8gUMEoaH/view?usp=sharing . Τα αρχεία περιέχουν N ακεραίους με πρόσημο (signed integers), 4 byte ανά ακέραιο, αποθηκευμένοι ως Little Endian. Οι πρώτοι αριθμοί (70 στο μεγαλύτερο αρχείο) έχουν συνεχόμενες τιμές ξεκινώντας από το 1, ενώ οι υπόλοιποι είναι ομοιόμορφα κατανεμημένοι.

1) Ανάγνωση κλειδιών
    a) διαβάζετε τα κλειδιά από το αρχείο που αντιστοιχεί στο N, και τα αποθηκεύεται σε ένα array στη μνήμη. Από αυτό το array θα κάνετε τις εισαγωγές κλειδιών. Μπορείτε να διαβάσετε και όλα τα αρχεία μαζεμένα στην αρχή.
2) Προετοιμασία
    a) δημιουργείτε 6 στιγμιότυπα από τις 4 υλοποιήσεις ως εξής
        i) ένα AVLTree
        ii) ένα BSTree
        iii) ένα BTree με order 100[1]
        iv) ένα BTree με order 600
        v) ένα LinearHashing με bucket size 40 και 500 αρχικά buckets
        vi) ένα LinearHashing με bucket size 1000 και 500 αρχικά buckets
    b) εισάγετε σε κάθε δομή N κλειδιά από το αντίστοιχο αρχείο κλειδιών (τα οποία έχετε ήδη διαβάσει από το βήμα 1)
3) Μετρήσεις
    a) παράγετε K* τυχαίους, μοναδικούς αριθμούς από 1 έως 3*N. Κάντε εισαγωγή κάθε κλειδιού στις 6 δομές σας. Καταγράψτε κάθε φορά το χρόνο εκτέλεσης σε nanoseconds. Από τις K εκτελέσεις εισαγωγής, θα πάρετε **το μέσο όρο του χρόνου εκτέλεσης** για μία εισαγωγή. Χρησιμοποιήστε 2 δεκαδικά ψηφία για τους μέσους όρους.
    b) παράγετε άλλους K τυχαίους, μοναδικούς αριθμούς από 1 έως 3*N. Κάντε αναζήτηση κάθε κλειδιού στις 6 δομές σας. Καταγράψτε κάθε φορά το χρόνο εκτέλεσης σε nanoseconds **ΚΑΙ** το πλήθος επιπέδων που προσπελαύνονται κάθε φορά. Από τις K εκτελέσεις αναζήτησης, θα πάρετε **το μέσο όρο του χρόνου εκτέλεσης και το μέσο όρο επιπέδων** για μία αναζήτηση. Χρησιμοποιήστε 2 δεκαδικά ψηφία για τους μέσους όρους.

*

[1] Η υλοποίηση που σας δίνεται για το BTree, χρησιμοποιεί διαφορετικό ορισμό για την τάξη ενός δέντρου (order) από αυτόν που σας δίνεται στο μάθημα. Συγκεκριμένα, στην υλοποίηση θέτοντας order K, ο κάθε κόμβος μπορεί να περιέχει έως 2xK κλειδιά. Ο ορισμός που χρησιμοποιούμε στο μάθημα είναι ότι όταν έχουμε BTree τάξης K, ο κάθε κόμβος μπορεί να περιέχει το πολύ K κλειδιά. Στην άσκηση που σας ζητάμε να χρησιμοποιήσετε order 100 και 600, εννοούμε το order όπως το ορίσαμε στο μάθημα. Θα πρέπει να προσαρμόσετε τη σχετική παράμετρο στον constructor των δομών.

K=10 όταν N < 201
K=50 όταν N > 200 και N < 1001
K=100 όταν N > 1000

Στο τέλος θα έχετε για κάθε πλήθος N κλειδιών, και για κάθε δομή, το μέσο χρόνο εισαγωγής, το μέσο χρόνο αναζήτησης και το μέσο πλήθος επιπέδων αναζήτησης (3 μετρήσεις για κάθε δομή και για κάθε N).

Κατά την εκτέλεση της εφαρμογής σας, θα πρέπει, χωρίς καμία ερώτηση στο χρήστη, να γίνονται όλες οι παραπάνω μετρήσεις, και να τυπώνονται στην οθόνη τα παρακάτω στοιχεία σε έναν ευανάγνωστο πίνακα. Δε μας ενδιαφέρει εάν είναι 3 πίνακες ή ένας ενιαίος πίνακας με όλες τις μετρήσεις. Μπορείτε επίσης να έχετε ανάποδα τις στήλες/γραμμές. Είναι σημαντικό να είναι ευανάγνωστος!

Χρόνοι εισαγωγής

	BST	AVL	BTree100	BTRee600	Linear40	Linear1000
20	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής
50	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής
100	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής
1000	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής
….	…	…	…	…	…	…
3.000.000	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής	χρόνος εισαγωγής

Χρόνοι αναζήτησης

	BST	AVL	BTree100	BTRee600	Linear40	Linear1000
20	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης
50	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης
100	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης

1000	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης
….	…	…	…	…	…	…
3.000.000	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης	χρόνος αναζήτησης

Επίπεδα πρόσβασης για αναζήτηση

	BST	AVL	BTree100	BTRee600	Linear40	Linear1000
20	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης
50	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης
100	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης
1000	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης
….	…	…	…	…	…	…
3.000.000	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης	επίπεδα πρόσβασης

Αναφορά

1) Κάντε μία σύντομη θεωρητική ανάλυση για την πολυπλοκότητα χειρότερης περίπτωσης (worst case) των αλγορίθμων εισαγωγής και αναζήτησης για κάθε περίπτωση. Συγκεκριμένα, συμπληρώστε τον παρακάτω πίνακα, και για κάθε κελί δώστε μία σύντομη τεκμηρίωση.

	BST	AVL	BTree100	BTRee600	Linear40	Linear1000
insert()	O(??)	O(??)	O(??)	O(??)	O(??)	O(??)
search time()	O(??)	O(??)	O(??)	O(??)	O(??)	O(??)
search levels()	O(??)	O(??)	O(??)	O(??)	O(??)	O(??)

2) Δημιουργήστε ένα γράφημα με τους χρόνους εισαγωγής για όλες τις δομές (4 δομές, 6 καμπύλες) και ένα αντίστοιχο ξεχωριστό γράφημα για τους χρόνους αναζήτησης, όπου στο X άξονα έχετε τις διάφορες τιμές του N. Μπορείτε να ομαλοποιήσετε τυχόν ακραίες τιμές, ειδικά για μικρά N.

Τί μορφή έχουν οι καμπύλες (σταθερή, λογαριθμική, γραμμική, εκθετική κλπ); Συμφωνούν οι μορφές με τη θεωρητική ανάλυση που έγινε στο (1); Αν όχι, μπορείτε να αιτιολογήσετε τις διαφορές;

3) Δημιουργήστε ένα γράφημα με τα επίπεδα ανά αναζήτηση για όλες τις δομές (4 δομές, 6 καμπύλες), όπου στο X άξονα έχετε τις διάφορες τιμές του N.

Τί μορφή έχουν οι καμπύλες (σταθερή, λογαριθμική, γραμμική, εκθετική κλπ); Είναι οι καμπύλες έτσι όπως θα τις αναμένατε; Αιτιολογήστε σύντομα την απάντησή σας.

4) Τι διαφορά βλέπετε (αν βλέπετε), μεταξύ του AVL δέντρου και του BST δέντρου (είτε σε χρόνο είτε σε επίπεδα); Αιτιολογήστε τη διαφορά (αν υπάρχει).

5) Τι διαφορά βλέπετε (αν βλέπετε), μεταξύ του BTree 100 και Btree 600 (είτε σε χρόνο είτε σε επίπεδα); Αιτιολογήστε τη διαφορά (αν υπάρχει).

6) Τι διαφορά βλέπετε (αν βλέπετε), μεταξύ του Linear Hashing 40 και Linear Hashing 1000 (είτε σε χρόνο είτε σε επίπεδα); Αιτιολογήστε τη διαφορά (αν υπάρχει).

7) Σε ποια περίπτωση (αν υπάρχει) θα προτιμούσατε τη χρήση του BST δέντρου έναντι των άλλων δομών (εισαγωγή και αναζήτηση στη RAM); Αιτιολογείστε την απάντησή σας βάσει των μετρήσεων που πήρατε.

8) Αν αντί στη RAM, χρειάζεται να υλοποιήσετε τις ίδιες δομές σε σκληρό δίσκο, όπου κάθε κόμβος/bucket αντιστοιχεί σε ένα Data Page στο δίσκο, ποια δομή θα προτιμήσετε για αναζήτηση τυχαίου κλειδιού; Αιτιολογείστε την απάντησή σας βάσει των μετρήσεων που πήρατε.

9) Αν χρειαζόμασταν ερωτήσεις εύρους τιμών (όλα τα κλειδιά μεταξύ 2 τιμών), ποια (ή ποιές) δομή(ές) θα προτιμούσατε για τη RAM, και ποια (ή ποιες) για το δίσκο;

Παραδοτέα

Ένα συμπιεσμένο zip αρχείο που περιέχει τον πηγαίο κώδικα (φάκελος src) και τη ζητούμενη αναφορά σε pdf (όχι zip μέσα στο zip!).

Γενικοί κανόνες για τον κώδικα και την αναφορά:

- Ο κώδικας περιέχει συνοπτικά σχόλια που εξηγούν την υλοποίηση.
- Εφόσον χρησιμοποιήσετε κώδικα ή στοιχεία από αλλού, αναφέρετε τις πηγές σας (πλήρης διεύθυνση σελίδας διαδικτύου, βιβλίο και συγκεκριμένο κεφάλαιο, κλπ). Εννοείται ότι πηγή σας δεν μπορεί να είναι η εργασία από συμφοιτητή σας.
- Οι αντιγραφές μηδενίζονται.
- Δώστε μεγάλη σημασία στην ποιότητα του κώδικα που θα στείλετε. Θα πρέπει να ακολουθεί τουλάχιστον τους παρακάτω κανόνες:
    - Χρησιμοποιείστε εύστοχα ονόματα μεταβλητών και μεθόδων έτσι ώστε να γίνεται εύκολα αντιληπτός ο λόγος και ο τρόπος χρήσης τους.
    - Γράψτε απλό και δομημένο κώδικα ώστε αυτός που τον διαβάζει να μπορεί να καταλάβει τα βήματα που ακολουθούνται για την υλοποίηση του κάθε προβλήματος.
    - Βάλτε εύστοχα σχόλια όπου χρειάζεται.
    - Κάντε το κώδικά σας παραμετρικό με χρήση μεταβλητών μέσω των οποίων θα μπορείτε να αλλάζετε κάποια μεγέθη (π.χ. τα μεγέθη για N, M κ.ο.κ.) χωρίς να χρειάζεται να αλλάζετε κάθε φορά και την αντίστοιχη υλοποίηση.

Χρήσιμα

Παραγωγή τυχαίων μοναδικών και μη μοναδικών ακεραίων

Για την παραγωγή μοναδικών numberOfNumbers ακεραίων μεταξύ minIntNumber και maxIntNumber μπορείτε να χρησιμοποιήσετε τον παρακάτω κώδικα.

```
java.util.Random randomGenerator = new java.util.Random();
int[] randomInts = randomGenerator.ints(minIntNumber,
maxIntNumber+1).distinct().limit(numberOfNumbers).toArray();
```

Για την παραγωγή μη μοναδικών numberOfNumbers ακεραίων μεταξύ minIntNumber και maxIntNumber μπορείτε να χρησιμοποιήσετε τον παρακάτω κώδικα.

```
java.util.Random randomGenerator = new java.util.Random();
int[] randomInts = randomGenerator.ints(minIntNumber,
maxIntNumber+1).limit(numberOfNumbers).toArray();
```

Για τη μέτρηση του χρόνου

Τρέχουσα ώρα σε nanosecond (long).

```
System.nanoTime()
```

Οπότε αρκεί να καλέσετε τη μέθοδο πριν την έναρξη της λειτουργίας που θέλετε να μετρήσετε το χρόνο, και να την ξανακαλέσετε μετά τον τερματισμό και να αφαιρέσετε τις τιμές μεταξύ τους.

Παραγωγή μέσων όρων

Στους μέσους όρους των μετρήσεων (χρόνου/πράξεων), θέλουμε αποτέλεσμα με δεκαδικά ψηφία. Στην java, όταν διαιρείται ένας ακέραιος με έναν άλλο ακέραιο, το αποτέλεσμα είναι πάλι ακέραιος, που δεν είναι το επιθυμητό. Για να παραχθεί αριθμός με δεκαδικά, αρκεί να γίνει cast ο αριθμητής σε τύπο με δεκαδικά. Π.χ.

```
long totalTime = 2548788751;
int numberOfRepetitions = 100;
float meanTimePerRepetition = (double) totalTime /
numberOfRepetitions;
```

Ανάγνωση ακεραίων από αρχείο

Για την ανάγνωση ακεραίων από τα αρχεία, μπορείτε να χρησιμοποιήσετε τη μέθοδο

```
private int[] readInts(String fileName)
```

που θα βρείτε στη διεύθυνση
https://drive.google.com/file/d/1zWtjSe-TV6VkoP4ctrkAPgRNJdUBTMOV/view?usp=sharing

Αλλαγές στην εκφώνηση

Σε σχέση με το αρχικό κείμενο της εκφώνησης, έγιναν οι εξής αλλαγές:

24/10/2024: προστέθηκαν σύνδεσμοι προς τα αρχεία που περιέχουν τους αριθμούς, και κώδικας για ανάγνωση των αρχείων