ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΑΡΧΕΙΩΝ

2Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Ονοματεπώνυμο & Αριθμός Μητρώου: Μπαντουβά Γεωργία 2019030006

Θα περιγράψω τη λειτουργία των βασικών συναρτήσεων που υλοποιήθηκαν, καθώς έχω βάλει αναλυτικά σχόλια υπο μορφή Javadoc στον κώδικα που εξηγούν κάθε κλάση και μέθοδο.

Αρχικά, οι δύο βασικές κλάσεις «BinarySearchTree» και «ThreadedBinarySearchTree» κάνουν implement το interface «BST» που βασίζεται σε κώδικα του φροντιστηρίου.

Δομή δέντρων

Για την υλοποίηση των δέντρων χρησιμοποιήθηκε έτοιμος κώδικας από το www.geeksforgeeks.org που όμως έπρεπε να διαφοροποιηθεί στα πλαίσια της εκφώνησης. Συγκεκριμένα, επειδή η άσκηση ζητούσε στατική καταχώρηση μνήμης για την υλοποίηση του BinarySearchTree δημιουργήθηκε ένας δυσδιάστατος πίνακας ακεραίων διαστάσεων N x 3, ενώ για το ThreadedBinarySearchTree ένας πίνακας N x 5. Οι δύο τελευταίες στήλες του ThreadedBinarySearchTree κανονικά είναι τύπου boolean, υλοποιήθηκαν ωστόσο σαν integers και έχουν την τιμή 0 στο αντίστοιχο πεδίο left/right thread όταν έχουν αντίστοιχα αριστερό/δεξί παιδί, ενώ όταν δεν έχουν λαμβάνουν την τιμή 1 και δείχνουν στο inorder predecessor/successor αντίστοιχα.

Μέθοδοι του interface BST που χρησιμοποιούνται για τη βασική λειτουργία του προγράμματος και στις δύο άλλες κλάσεις:

1. public void insert(int key);
   Η μέθοδος αυτή εισάγει ένα συγκεκριμένο κλειδί το οποίο και λαμβάνει ως όρισμα. Καλείται από την συνάρτηση insertKeys που θα περιγραφεί παρακάτω.
2. public int search(int root, int key);
   Η μέθοδος αυτή λαμβάνει ως όρισμα το root και το key που θέλουμε να ψάξουμε μέσα στο δέντρο. Η λογική είναι η εξής:
   Για το BinarySearchTree:
   Ελέγχει εάν το δέντρο είναι άδειο, ή αν το κλειδί βρίσκεται στη ρίζα του δέντρου. Έπειτα, ανάλογα με το εάν το κλειδί που ψάχνουμε είναι μικρότερο ή μεγαλύτερο από το key της ρίζας, καλούμε αναδρομικά τη συνάρτηση μας με κατάλληλο όρισμα τον δεξιό ή τον αριστερό κόμβο του δέντρου.
   Για το ThreadedBinarySearchTree:

Ακολουθούμε την ίδια λογική, με μια διαφορά. Ελέγχουμε επιπλέον αν το left και right δείχνουν σε κάποιο child ή στο inorder predecessor/successor και αντιμετωπίζουμε τις περιπτώσεις ανάλογα.

3. public void insertKeys(int[] integers);
   Η μέθοδος αυτή λαμβάνει ως όρισμα έναν πίνακα ακεραίων που παράγεται στη main, και έπειτα με ένα for loop διασχίζει το μήκος του πίνακα και εισάγει έναν έναν τους αριθμούς καλώντας την συνάρτηση insert.
4. public void findInRange(int root, int kMin, int kMax);
   Λαμβάνει ως όρισμα το root, και δύο ακεραίους που ορίζουν ένα range τιμών. Η συνάρτηση αυτή βρίσκει τα κλειδιά που έχουν τιμή μεταξύ αυτού του range.
5. Public void randomSearches(int numOfSearches)
   Η μέθοδος αυτή παράγει έναν πίνακα τυχαίων ακεραίων. Έπειτα, κάνει τόσες αναζητήσεις όσες έχουν δοθεί σαν όρισμα και ψάχνει τους τυχαίους αυτούς αριθμούς που έχουν παραχθεί. Τέλος, εκτυπώνει τον μέσο όρο των συγκρίσεων/αναθέσεων που έγιναν ανά κάθε αναζήτηση.
6. Public void rangeRandomSearching(int range, int numOfSearches)
   Η μέθοδος αυτή έχει παρόμοια λογική με την παραπάνω, με τη διαφορά ότι εδώ παράγεται ένα τυχαίο διάστημα συγκεκριμένου εύρους και καλώντας τη συνάρτηση findInRange αναζητούνται τα κλειδιά που έχουν τιμή μεταξύ αυτού του εύρους.

Κοινές βασικές μέθοδοι μεταξύ BinarySearchTree/ThreadedBinarySearchTree

1. Private int getNode()

Παίρνει την αμέσως επόμενη διαθέσιμη θέση από τη μεταβλητή avail και αποθηκεύει στη μεταβλητή avail το απότελεσμα της getRight(pos) όπου pos η διαθέσιμη αυτή θέση.

<u>Άλλες βασικές μέθοδοι στη κλάση BinarySearchTree:</u>

1. private int insertRec(int root, int key)
   Είναι αναδρομική. Αν δεν υπάρχει δέντρο δημιουργεί καινούριο κόμβο και αλλάζει το root, ενώ αν υπάρχει ελέγχει έαν το κλειδί της ρίζας είναι μικρότερο ή μεγαλύτερο και ανάλογα καλεί αναδρομικά τον εαυτό της με το κατάλληλο όρισμα του αριστερού/δεξιού υποδέντρου.

<u>Άλλες βασικές μέθοδοι στη κλάση ThreadedBinarySearchTree:</u>

1. Private int insertMethod(int root, int key)
   Είναι iterative μέθοδος αντί για αναδρομική όπως το BinarySearchTree, χρησιμοποιούμε while loop για να εντοπίσουμε που πρέπει να εισαχθεί το

κλειδί και κάνουμε τους απαραίτητους ελέγχους & αρχικοποιήσεις για τα threads.

2. Private int successor(int pointer) & private int predecessor(int pointer)
   Βρίσκουμε το successor/predecessor αντίστοιχα σύμφωνα με την εκτύπωση δέντρου inorder.

*Οι υπόλοιπες συναρτήσεις που χρησιμοποιήθηκαν είναι κυρίως getters & setters, ωστόσο μπορείτε να ανατρέξετε και στα σχόλια στον κώδικα για την τεκμηρίωσή τους.

BinarySearch

1. Private int doSearch(int leftIndex, int rightIndex, int key)
   Μέθοδος που εκτελεί δυαδική αναζήτηση σε ταξινομημένο array χρησιμοποιώντας αναδρομή, βασίζεται σε κώδικα φροντιστηρίου. Επιστρέφει Integer.MIN_VALUE αν δεν βρεθεί το κλειδί, αλλιώς το position στο οποίο θα βρεθεί.

2. Public int rangeSearch(int leftIndex, int rightIndex, int startRange, int endRange)
   Είναι μία αναδρομική συνάρτηση που «βρίσκει» κλειδιά σε συγκεκριμένο εύρος. Πρώτα θέλουμε να βρούμε το σημείο(mid) από το οποίο θα ξεκινήσει η αναζήτηση εύρους στο οποίο τηρούνται οι εξής προυποθέσεις :
   data[mid] <= startRange && data[mid+1] > startRange
   και επειτα «βρίσκουμε» αυξάνοντας το position(position=mid) κάθε κλειδί για το οποίο ισχυεί η προυπόθεση:
   data[position] <= endRange (*)
   Παρατηρείται το παράλογο αποτελέσμα με τις μετρήσεις στο range 1000 να είναι λιγότερες από το range 100. Μάλλον ευθύνεται κάποια παράβλεψη στον κώδικα που λόγω χρονικών περιορισμών δεν βρέθηκε. Δεν την έβαλα σε σχόλια γιατί δεν δημιουργεί error.

Επεξεργασία Αποτελεσμάτων

Χρησιμοποιήθηκε η κλάση του φροντιστηρίου MultiCounter και έτσι μετρήθηκαν οι συγκρίσεις και οι αναθέσεις που απαιτεί η άσκηση. Συγκεκριμένα χρησιμοποιήθηκαν διαφορετικοί μετρητές για κάθε λειτουργία ως εξής:

Counter1: Comparisons για την εισαγωγή κλειδιού
Counter2: Comparisons για την τυχαία αναζήτηση κλειδιού

Counter3: Comparisons για την αναζήτηση εύρους κλειδιού

Για την άσκηση, έγιναν 100 τυχαίες αναζητήσεις και 100 αναζητήσεις τυχαίου εύρους μήκους 100 και 1000 και βρέθηκε ο μέσος όρος των συγκρίσεων & αναθέσεων που χρειάστηκαν να γίνουν κάθε φορά.

Συγκεκριμένα:

	Συγκρίσεις ανά εισαγωγή	Συγκρίσεις ανά τυχαία αναζήτηση	Συγκρίσεις ανά αναζήτηση εύρους 100	Συγκρίσεις ανά αναζήτηση εύρους 1000
BinarySearchTree	115	103	284	1386
Threaded BST	134	107	253	1348
Sorted	-	117	97	71

Είναι αναμενόμενο οι συγκρίσεις στο threaded να είναι περισσότερες στην εισαγωγή αφού χρειάζονται περισσότερες αναθέσεις λόγω των threads. Σχετικά με τις τυχαίες αναζητήσεις, θα περίμενα οι μετρήσεις του threaded να είναι εμφανώς λιγότερες σε σχέση με το BinarySearchTree, αλλά ίσως έχει συμβεί κάποια παράβλεψη με αποτέλεσμα αυτό να μην είναι εντελώς εμφανές. Τέλος, περίμενα, το sorted array να έχει λιγότερες συγκρίσεις σε αναζήτηση μεγάλου εύρους λόγω της προυπόθεσης (*) της rangeSearch() σε σχέση με το BST & TBST. Ωστόσο όπως αναφέρθηκε και πάνω, έχει γίνει κάποιο λάθος με τους μετρητές που αποτυπώνεται και στα αποτελεσμάτα των μετρήσεων στην αναζήτηση εύρους.

<u>Πηγές</u>

https://www.geeksforgeeks.org/threaded-binary-tree/?ref=rp

https://iq.opengenus.org/operations-in-threaded-binary-tree/

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

Διαλέξεις & Φροντιστήρια μαθήματος