Αναφορά 2ης εργαστηριακής άσκησης ΠΛΗ202

Λεοντής Παναγιώτης ΑΜ 2018030099

Γενική εικόνα

Ο κώδικας λειτουργεί κανονικά. Κατά την εκκίνηση του προγράμματος πληκτρολογείτε ως argument το path για να ανοίξει το κατάλληλο αρχείο. Οι μέθοδοι searchByRange() είναι έτσι δομημένες ώστε να εκτυπώνουν όσους αριθμούς βρέθηκαν στο εύρος αναζήτησης.

Αρχικά έβαλα μέσα στο *interface Tree* τις 4 επιθυμητές μεθόδους που θέλαμε να εκτελούν τα δέντρα. Μέσω της κλάσης *Node* μπορούσα να δημιουργήσω αντικείμενα που θα έβαζα ως κόμβους στο *DynamicTree*. *StaticTree* είναι η κλάση για να δημιουργήσω το δέντρο με στατική παραχώρηση μνήμης. Ο constructor του *StaticTree* παίρνει ως όρισμα το πλήθος των αριθμών N για εισαγωγή, ώστε να δημιουργήσει τα array με κατάλληλο μέγεθος. Επιπλέον μέσα στο constructor καλώ την intializeStack() ώστε να αρχικοποιήσω τα 3 array(treeKey,treeLeft,treeRight). Οι constructors και των δυο υλοποιήσεων παίρνουν ως όρισμα το πλήθος N των αριθμών για να δημιουργώ ένα int array (3η υλοποίηση) το οποίο θα επιστρέφει η μέθοδος inOrder() της κάθε υλοποίησης. Επιπλέον όρισα την κλάση OrderedArray. Αυτή παίρνει ως όρισμα ένα int array στις μεθόδους της και τις υλοποιεί ώστε να κάνουμε τις απαραίτητες μετρήσεις.

Κάθε μία από τις 3 υλοποιήσεις έχει μια μεταβλητή τύπου int, compares η οποία αυξάνεται κάθε φορά που γίνεται μια σύγκριση στις μεθόδους insert(), search(), searchByRange() ή κάποια ανάθεση. Αυτή χρησιμοποιώ στις μετρήσεις.

Στην στατική παραχώρηση μνήμης προσπάθησα να υλοποιήσω τις μεθόδους της κλάσης βασισμένος στο τρόπο υλοποίησης αυτών της δυναμικής παραχώρησης μνήμης. Έτσι οι μέθοδοι δεν διαφέρουν ως προς την σκέψη υλοποίησης απλώς η κάθε μια είναι προσαρμοσμένη στα δεδομένα της κάθε κλάσης.

Επιπλέον Λειτουργία: Στην κλάση *OrderedArray* έχει υλοποιηθεί η μέθοδος printArray() για να εκτυπώνει το ταξινομημένο πεδίο. Η κλήση της βρίσκεται στην *Console* στη γραμμή 138 μέσα σε σχόλια. Μπορείτε να τα απενεργοποιήσετε και να την εκτελέσετε.

Υλοποίηση μετρήσεων

Για να παίρνω τον χρόνο χρησιμοποίησα την μέθοδο System.nanoTime() και μετά διαίρεσα με 10^6 επομένως ο τελικός χρόνος προκύπτει σε ms.

Για τις μετρήσεις δημιουργώ μια λίστα 100 τυχαίων αριθμών τύπου LinkedList intList έτσι ώστε να γίνεται σύγκριση για τους ίδιους αριθμούς σε όλες τις υλοποιήσεις κάθε φορά που τρέχει το πρόγραμμα. Έγινε χρήση του ThreadLocalRandom.current().nextInt().
https://www.geeksforgeeks.org/generating-random-numbers-in-java/

Κάθε φορά που καλώ μια μέθοδο των δέντρων που αυξάνει τα compares, όταν εκτυπωθούν οι μετρήσεις μηδενίζω τα compares για να μην καταλήξω σε λάθος αποτελέσματα.

Στατική και δυναμική παραχώρηση μνήμης

Η μέθοδος insert() των κλάσεων *DynamicTree*,*StaticTree* εκτελείται με μια επαναληπτική δομή while και ψάχνει είτε τα δεξιά είτε τα αριστερά υποδέντρα συγκρίνοντας το κλειδί εισαγωγής με αυτό του τρέχον κόμβου. Συνθήκη εξόδου είναι να φτάσουμε σε κενό κόμβο στην δυναμική παραχώρηση ή να μην υπάρχει διαθέσιμη θέση στην στατική παραχώρηση μνήμης. Στην υλοποίηση με στατική παραχώρηση μνήμης με κάθε νέα εισαγωγή εισάγω στο avail την επόμενη τιμή του array treeRight και ρυθμίζω την προηγούμενη εισαγωγή ώστε το πεδίο για το δεξί και το αριστερό παιδί να ενημερωθούν ανάλογα με την θέση της νέας εισαγωγής.

Στην κλάση *Console* ανοίγω το αρχείο που δόθηκε ως argument και μέσω των μεθόδων createBinarySearchTree() και createArrayTree() δημιουργώ τις υλοποιήσεις δυναμικής και στατικής παραχώρησης μνήμης αντίστοιχα. Υπολογίζω τον αριθμό των λέξεων για διάβασμα και μέσα σε μια while μετακινώ τον δείκτη μέσα στο αρχείο εισάγοντας κάθε φορά στα δέντρα τον τωρινό αριθμό. Πριν και μετά την while μετρώ τον χρόνο και υπολογίζοντας την διαφορά τους προκύπτει ο χρόνος για την εισαγωγή N στοιχείων. Έπειτα εκτυπώνω και τα compares διαιρεμένα με το πλήθος των αριθμών ώστε να προκύπτει ο μέσος όρος συγκρίσεων για εισαγωγή N στοιχείων.

Εξίσου και η μέθοδος search() των δύο δέντρων εκτελείται αναδρομικά ψάχνοντας είτε το δεξί είτε το αριστερό υποδέντρο ανάλογα με το αποτέλεσμα της σύγκρισης με τον τρέχον κόμβο. Συνθήκη εξόδου ή βρέθηκε το κλειδί, ή φτάσαμε σε κενό κόμβο στη δυναμική ή δεν υπάρχει άλλος διαθέσιμος κόμβος για ψάξιμο στην στατική παραχώρηση μνήμης.

Καλώ μια επανάληψη for για 100 φορές και κάθε φορά κάνω μια αναζήτηση σε κάθε υλοποίηση με έναν αριθμό από την intList. Μετράω τον χρόνο πριν και μετά την for και από την διαφορά τους προκύπτει ο χρόνος για την αναζήτηση 100 τυχαίων αριθμών. Εκτυπώνω και τα compares της κάθε υλοποίησης διαιρεμένα με το 100 και προκύπτει ο μέσος όρος συγκρίσεων.

Για την μέθοδο searchByRange() της στατικής και δυναμικής παραχώρησης μνήμης χρησιμοποίησα αυτή που μας δόθηκε στην εκφώνηση της άσκησης: https://www.geeksforgeeks.org/print-bst-keys-in-the-given-range/.

Διασχίζει αναδρομικά το δέντρο και εκτυπώνει τα κλειδιά που βρίσκονται ανάμεσα στο διάστημα k1 , k2 που είναι ορίσματα της μεθόδου.

Καλώ δύο φορές μια επανάληψη for για 100 φορές και κάθε φορά κάνω μία αναζήτηση εύρους σε κάθε υλοποίηση έχοντας το κάτω άκρο τον αριθμό από την intList και άνω άκρο τον αριθμό+100 στην πρώτη for και τον αριθμό+1000 στην δεύτερη. Αντίστοιχα υπολογίζω εκτυπώνω το μέσο όρο των compares για την κάθε αναζήτηση εύρους στην κάθε υλοποίηση.

Η μέθοδος inOrder() διασχίζει αναδρομικά διαβάζοντας πρώτα τα αριστερά υποδέντρα και έπειτα τα δεξιά προσθέτοντας κάθε φορά το κλειδί του τρέχον κόμβου σε μια λίστα. Τελικά επιστρέφει ένα int array το οποίο είναι ταξινομημένο.

Ταξινομημένο Πεδίο

Χρησιμοποίησα την μέθοδο inOrder() της DynamicTree για να δημιουργήσω το ταξινομημένο πεδίο, αν και δεν έχει διαφορά από ποια υλοποίηση θα προκύψει.

Την μέθοδο search() την υλοποίησα αναδρομικά με δυαδική αναζήτηση. Χωρίζω το array στα μισά και παίρνω είτε το δεξί μισό είτε το αριστερό ανάλογα με το κλειδί του τρέχον κόμβου. Συνθήκη εξόδου αν δεν έχει βρεθεί η θέση του στοιχείου είναι οι ακριανές θέσεις να είναι διαφέρουν κατά μία θέση.

Αντίστοιχα μέσα στη Console με μια for 100 επαναλήψεων κάνω 100 αναζητήσεις με τον τυχαίο αριθμό από την λίστα intList κάθε φορά. Μετράω τον χρόνο πριν και μετά την for και η διαφορά τους είναι ο χρόνος αναζήτησης 100 τυχαίων αριθμών. Αντίστοιχα εκτυπώνω τα compares διαιρεμένα με το 100 ώστε να προκύψει ο μέσος όρος συγκρίσεων.

Για την μέθοδο searchByRange() ακολούθησα παρόμοια λογική με την υλοποίηση στις κλάσεις *StaticTree* και *DynamicTree*. Αναδρομικά με δυαδική αναζήτηση διέσχιζα το πεδίο χωρίζοντας κάθε φορά στη μέση.

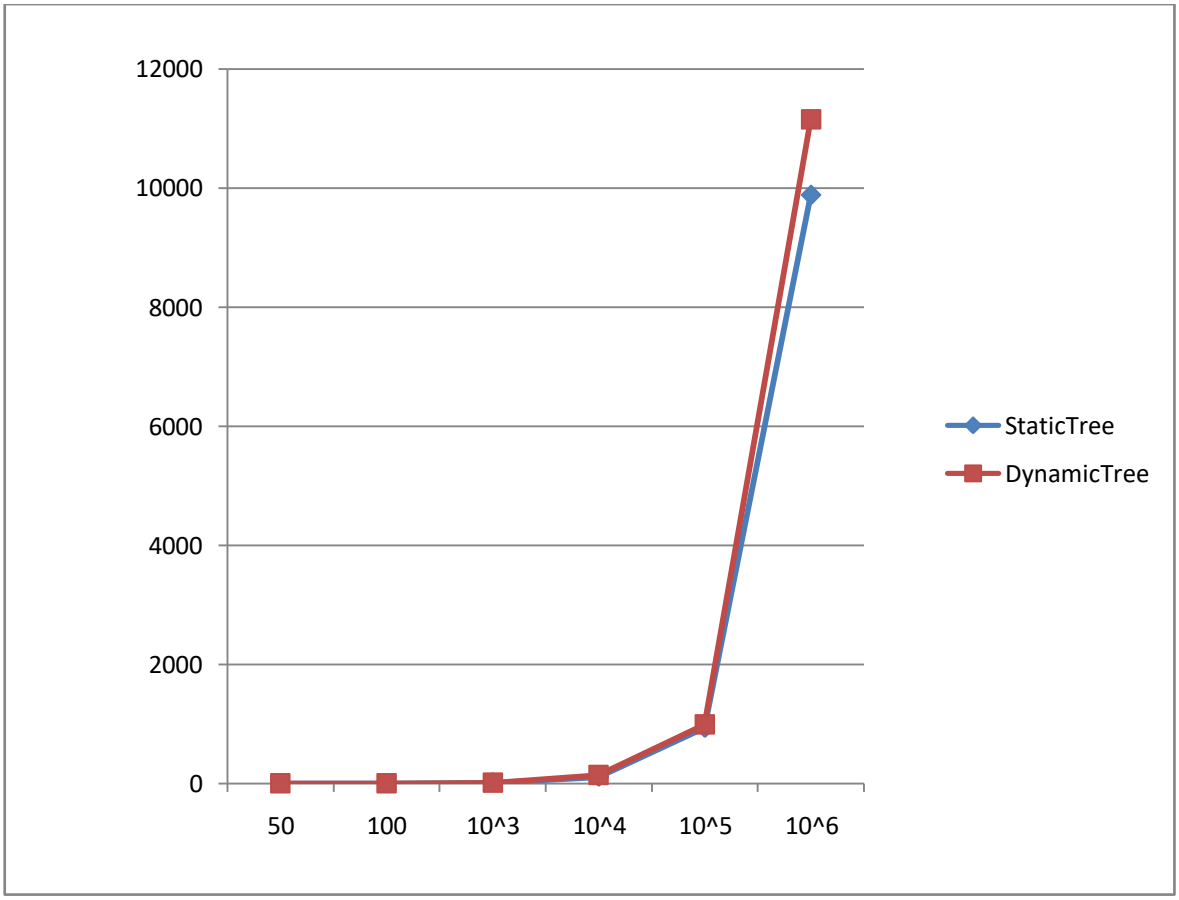
Στην Console χρησιμοποίησα 2 for μια για αναζήτηση εύρους 100 και μία για 1000. Μέσα σε κάθε for έκανα μια αναζήτηση εύρους με κάτω άκρο τον τυχαίο αριθμό από την intList και άνω τον αιρθμό+100 και τον αριθμό+1000 αντίστοιχα. Παίρνω τα compares για κάθε μία από τις 2 for τα διαιρώ με 100 και προκύπτει ο μέσος όρος συγκρίσεων.

Πίνακας Μετρήσεων και Γραφικές Παραστάσεις

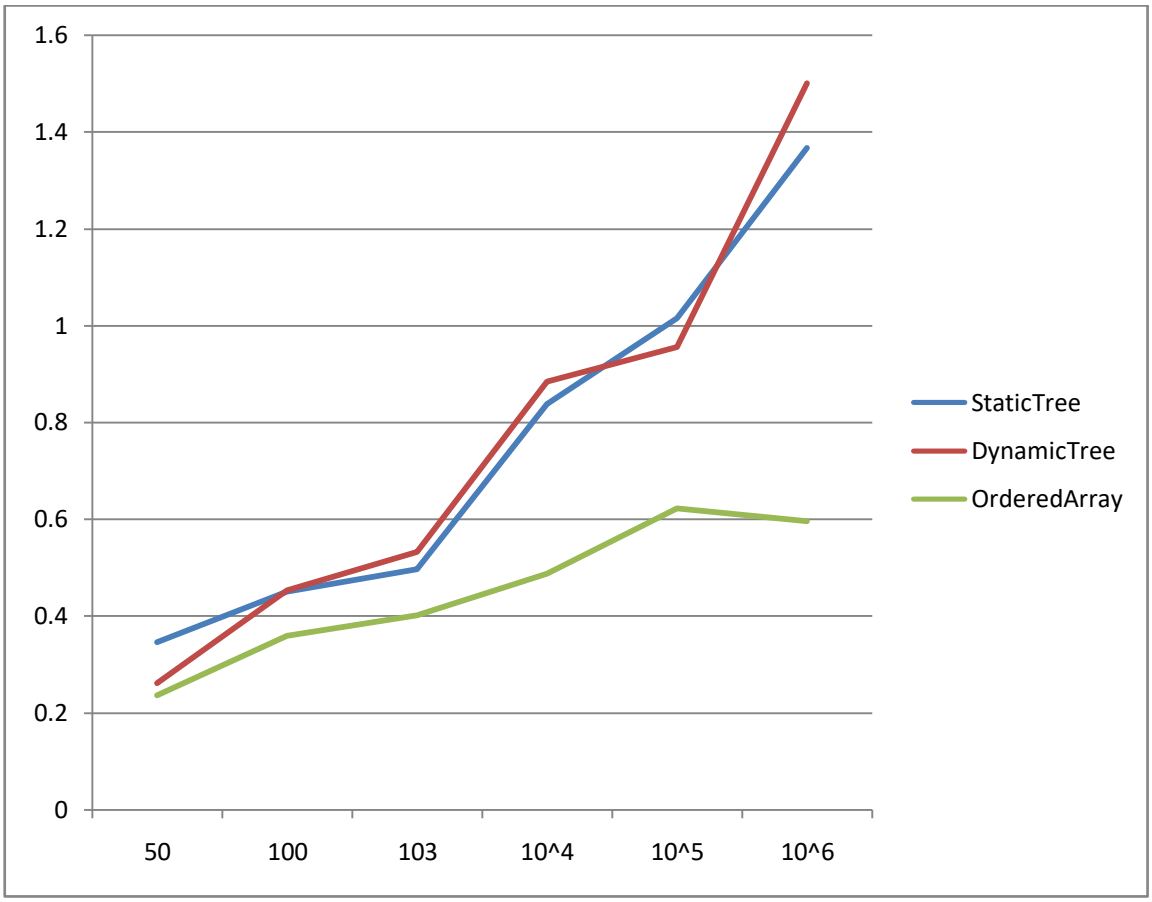
Παρατίθενται οι μετρήσεις για το αρχείο μεγέθους 10^6

Μέθοδος	Μέσος αριθμός συγκρίσεων/ εισαγωγή	Συνολικός χρόνος για N Εισαγωγές	Μέσος αριθμός συγκρίσεων / τυχαία αναζήτηση	Συνολικός χρόνος για 100 αναζητήσεις	Μέσος αριθμός συγκρίσεων / αναζήτηση εύρους(K=100)	Μέσος αριθμός συγκρίσεων / αναζήτηση εύρους(K=1.000)
ΔΔΕ με δυναμική παραχώρηση μνήμης	104	11160.11 ms	53	1.5006 ms	105	107
ΔΔΕ με array	107	9890.274 ms	53	1.3671 ms	105	107
Ταξινομημένο πεδίο			29	0.5971 ms	80	81

Γραφική παράσταση για το συνολικό χρόνο εισαγωγής N στοιχείων ($N=50,10^2,10^3,10^4,10^5,10^6$). Χρόνος σε ms.



Γραφική παράσταση για το συνολικό χρόνο για 100 τυχαίες εισαγωγές σε δέντρο μεγέθους N στοιχείων(N=50,10^2,10^3,10^4,10^5,10^6). Χρόνος σε ms.



Σχολιασμός:

Όπως παρατηρούμε στην πρώτη γραφική παράσταση όσο μεγαλώνει το μέγεθος του αρχείου μεγαλώνει και η διαφορά του χρόνου εισαγωγής N στοιχείων των δύο υλοποιήσεων. Όπως είχε αναφερθεί περιμέναμε η υλοποίηση της στατικής παραχώρησης μνήμης να είναι γρηγορότερη καθώς δεσμεύει την κατάλληλη μνήμη από την αρχή. Σε αντίθεση η δυναμική παραχώρησης μνήμης πρέπει να δεσμεύει μνήμη για την εισαγωγή κάθε ξεχωριστού κόμβου τη φορά.

Στην δεύτερη γραφική παράσταση τα αποτελέσματα δεν είναι τόσο ευανάγνωστα όπως στην πρώτη. Παρατηρείται μια εναλλαγή των χρόνων για 100 τυχαίες αναζητήσεις μεταξύ της στατικής και της δυναμικής παραχώρησης μνήμης. Είχε συζητηθεί στο φροντιστήριο ότι αναμένουμε την στατική να είναι γρηγορότερη όμως υπάρχουν πολλοί παράγοντες που μπορούν να επηρεάσουν το τελικό αποτέλεσμα (πχ υλοποίηση μεθόδου, η δομή της Java). Εξίσου μη αναμενόμενο ήταν το ταξινομημένο πεδίο να είναι γρηγορότερο από τις δυο υλοποιήσεις. Σε όλα τα μεγέθη αρχείου είχε τις λιγότερες συγκρίσεις και είναι ταχύτερο στην αναζήτηση σε σύγκριση με τα δυο δέντρα.

Παρατήρηση:

Παρατήρησα ότι εάν αλλάξουμε την σειρά εκτέλεσης των εντολών μπορεί να προκύψουν εντελώς διαφορετικά αποτελέσματα. Στο υπάρχων πρόγραμμα δημιουργώ πρώτα το δέντρο με στατική παραχώρηση μνήμης και μετά με τη δυναμική. Εάν αλλάξω την σειρά εκτέλεσης των αντίστοιχων εντολών προκύπτουν αντίθετα αποτελέσματα από όσα καταγράφηκαν.