Αναφορά 1ης Εργαστηριακής Άσκησης

Επεξεργασία Αρχείων

Δομές Δεδομένων & Αρχείων

Μπέτσιος Θωμάς

A.M. 20103030104

Για την κλάση FileManager :

Αρχικά δημιούργησα τις μεθόδους Set και Get για τις μεταβλητές της κλάσης. Στην συνέχεια δημιούργησα τις μεθόδους που μας ζητήθηκε. Σε κάθε ερώτημα οι μέθοδοι παίρνουν τα ορίσματα και υλοποιούν την λειτουργικότητα και επιστρέφουν 0 αν απέτυχαν ή αν δεν βρέθηκε το αρχείο και 1 αν ολοκληρώθηκαν επιτυχώς. Η μόνη μέθοδος που επιστρέφει τον αριθμό των σελίδων είναι η OpenFile όπως μας ζητήθηκε. Τέλος δημιούργησα και μια μέθοδο ToByteArray για να παίρνω 1 String και 2 Integers και να τα μετατρέπω σε ενα byte array μεγέθους 128. Είχα παρατηρήσει στην DeletePage ένα θεματάκι που ενώ εχω κάνει RandomAccessFile.setLength να μειώνει το μέγεθος του αρχείου συνέχιζε να εμφανίζει τα δεδομένα της σελίδας που ειναι στο τέλος αλλα αντιγράφει κανονικά τα στοιχεία της τελευταίας σελίδας στην σελίδα που παίρνει ως όρισμα.

- **Για την εκτέλεση μπορείται να τρέξετε το FileManagerApp οπου έχω δημιουργήσει ενα menu με επιλογές για να μπορεί ο χρήστης να επιλέξει ποια μέθοδο θέλει να καλέσει .**

Για το δεύτερο κομμάτι της εργασίας :

Αρχικά να πώ οτι το υπόλοιπο κομμάτι της εργασίας είναι εντελώς ανεξάρτητο απο το 1ο κομμάτι. Το μόνο πράγμα που παίρνω απο το 1ο κομμάτι είναι ο αριθμός των bytes της σελίδας.

Για Α τρόπο οργάνωσης αρχείου :

Αρχικά δημιουργώ ενα αρχείο στο πακέτο fileCreationPackage στην κλάση CreateFile η οποία επιστρέφει εναν πίνακα ExistingKeysArray με όλα τα κλειδιά που γράφτηκαν στο αρχείο με όνομα myNewNodeFile. Έχω δημιουργήσει εναμ constructor στην κλάση Node η οποία βρίσκεται στο πακέτο fileOrganizationPackage ο οποίος παίρνει 1 κλειδί, 1 String 20 bytes και 2 Ints και έτσι δημιοργώ 4 Nodes για 2500 επαναλήψεις ωστε να έχω τελικά 10.000 εγγραφές. Χρησιμοποίησα τις μεθόδους getAlphaNumericString και randomGenerator που μας δείξατε στο φροντηστήριο. Τέλος γράφω τα δεδομένα στο αρχείο myNewNodeFile ανα 128 bytes ή 4 Nodes.

Στην συνέχεια στο πακέτο fileProcessPackage στην κλάση FileProcessA τρέχω σε ενα for loop 20 επαναλήψεων μία για κάθε RandomExistingKey διαβάζω μια σελίδα του παραπάνω αρχείου και την κάνω arraycopy σε 4 πίνακες των 32 bytes - τον τρόπο αυτόν τον βρήκα στην σελίδα (https://mkyong.com/java/java-how-to-join-and-split-byte-arrays-byte/) - και περνάω τους πίνακες ως όρισμα των constructor του Node δημιουργόντας έτσι 4 εγγραφές στις οποίες κάνω σειριακή αναζήτηση για το κλειδί της κάθε επανάληψης. Αυτο επαναλαμβάνεται μέχρι να τελειώσει το αρχείο.

Για τον Β τρόπο οργάνωσης αρχείου :

Για τον δέυτερο τρόπο οργάνωσης χρησιμοποίησα την ίδια λογική. Αρχικα διαβάζω μια μια τις σελίδες του παραπάνω αρχείου και δημιουργώ 4 IndexNodes και η κλάση βρίσκεται στο πακέτο fileOrganization, και στην συνέχεια με την αντίστροφη διαδικάσία πάιρνω τα δεδομένα απο τα 4 IndexNodes και καλώ την ToByteArray μια μέθοδο της κλάσης IndexNode η οποία δημιουργεί ενα byte array των 8 και κάθε 16 τέτοιες εγγραφές που βάζω σε εναν writeBuff τα γράφω στο αρχείο IndexFile. Προσέχω το κάθε ζεύγος (κλειδί- σελίδα) να έχει την σωστή σελίδα πριν γραφτεί στο αρχείο.

Στην συνέχεια στο πακέτο fileProcessPackage στην κλάση FileProcessB με όμοιο τρόπο με την κλάση FileProcessB παίρνω μια σελίδα απο το IndexFile αυτή την φορά την μετατρέπω σε 16 εγγραφές ψάχνω το κλειδί και επιστρέφει το IndexNode. Στην συνέχεια ψάχνω στην σελίδα που βρίσκεται το συγκεκριμένο κλειδί και την μετατρέπω σε 4 Nodes και ψάχνω εκ νέου το ίδιο κλειδί.

Για τον Γ τρόπο οργάνωσης αρχείου :

Στο πακέτο fileCreationPackage στην κλάση SortedNodeFileCreation δημιουργώ το sorted αρχείο SortedNodeFile που προκύπτει απο την ταξινόμηση του πρώτου αρχείου. Διαβάζω σειριακά τις σελίδες του πρώτου αρχείου, δημιουργώ ενα listOfNodes εναν πίνακα με Nodes και κάθε 4 εγγραφές τα βάζω στην παρπανω λιστα. Μετα χρησιμοποιώντας την Arrays.sort ταξινόμησα την λίστα με βάση το κλειδί γιατι η κλάση Node implements το Comparable<Node> και υλοποιεί μια μέθοδο την compareTo η οποία συγκρίνει 2 κόμβους με βάση το κλειδί (αυτο το βρήκα στο ιντερνετ αλλα δεν κράτησα την ιστοσελίδα). Επίσης χρησιμοποιώ έναν MultiCounter(4) για να μετράω τις προσβάσεις στον δίσκο για την ταξινόμηση.

Στην συνέχεια παιρνω απο τον πίνακα listOfNodes καθε 4 Nodes και τα γράφω σε ένα καινούριο αρχείο το SortedNodeFile.

Για τον Δ τρόπο οργάνωσης αρχείου :

Δεν πρόλαβα να το υλοποιήσω.

- **Για το τρέξιμο του δεύτερου κομματιου της άσκησης χρησιμοποιώ την κλάση Application που περιέχει την μέθοδο main και είναι στο πακέτο appPackage. Με την υλοποίηση που έκανα χρειάζεται να δημιουργεί καινούριο αρχείο myNewNodeFile σε κάθε εκτέλεση του κώδικα γιατι δεν προλάβαινα να τα διαχωρίσω, αλλα και τα άλλα δύο αρχεία είναι άμεσα εξαρτημένα απο το πρώτο γιατι παράγονται απο την επεξεργασία αυτού.**
- **Στις κλάσεις ProcA,ProcB έχω έναν μετρτή που αυξάνει σε κάθε πρόσβαση στον δίσκο, ειναι οι MultiCounter(1) & MultiCounter(2). Στον Γ τρόπο οργάνωσης του αρχείου έχω εναν MultiCounter (3) στην μέθοδο FileBinarySearch που υπάρχει στο πακέτο searchPackage στην κλάση BinarySearch.**

Για την τεκμηρίωση των αποτελεσμάτων :

Ενδεικτικά κάποιες τιμές απο τον κώδικα με μέσο όρο προσβάσεων στο δίσκο για 20 αναζητήσεις και προσβάσεις στον δίσκο για την ταξινόμηση.

Μέθοδος	Α. τρόπος οργάνωσης αρχείου	Β. τρόπος οργάνωσης αρχείου	Γ. τρόπος οργάνωσης	Απόδοση Ταξινόμησης περίπτωση Γ
Απόδοση (αριθμός προσβάσεων δίσκου)	1391	448	10	5000

Παρατηρόντας τα αποτελέσματα του παραπάνω πίνακα παρατηρούμε αρχικά οτι για τους δύο πρώτους τρόπους οργάνωσης αρχείου χρησιμοποιούμε σηριακή αναζήτηση που στην γενική περίπτωση έχει πολυπλοκότητα O(N). **Συγκρίνοντας τους δύο τρόπους για την αναζήτηση 20 κλειδιών βλέπουμε οτι επειδή το αρχείο των κλειδιών είναι κατα 4 φορές μικρότερο απο το αρχείο του τρόπου οργάνωσης Α χρειάζονται πολύ λιγότερες προσβάσεις δίσκου (448 vs 1391 Α τρόπος) κατά την αναζήτηση και άρα επιτυγχάνει καλύτερη απόδοση σε αναζητήσεις**. (Επίσης αν έχουμε κλειδιά που έχουν μεγαλύτερη πιθανότητα να βρεθούν κοντά στην αρχή του αρχείου αυτή η μέθοδος μπορει να έχει πολυπλοκότητα O(1).)

Όσον αφορά τον Γ τρόπο οργάνωσης **αρχείου παρατηρούμε οτι αν έχουμε ταξινομημένο αρχείο πετυχαίνουμε με μεγάλη διαφορά καλύτερη απόδοση σε αναζήτηση σε σχέση με την σηριακή αναζήτηση (10 προσβάσεις στον δίσκο κατα μέσο όρο για κάθε κλειδί), με την μέθοδο της δυαδικής αναζήτησης που έχει πολυπλοκότητα O(logN)**. Άν όμως δεν έχουμε ταξινομημένο αρχείο **βλέπουμε οτι δεν μας συμφέρει να το ταξινομήσουμε και μετα να εφαρμόσουμε δυαδική αναζήτηση** γιατι σύμφωνα με τα αποτελέσματα θέλουμε 2 προσβάσεις στον δίσκο για κάθε κλειδι μόνο για την ταξινόμηση **σύνολο 5000 και άλλες 10 για την αναζήτηση.**

Αρα σαν συμπέρασμα βγαίνει οτι αν έχουμε ταξινομημένο αρχείο θα ήταν καλύτερος ο Δ τρόπος οργάνωσης του αρχείου σε σχέση με τον Γ, γιατί το αρχείο σε αυτήν την περίπτωση έιναι πολυ πιο μικρό και θα είχαμε ακόμα καλύτερη απόδοση (δεν φαίνεται γιατι δεν υλοποίησα το Δ). Και αν δεν έχουμε ταξινομημένο αρχείο καλύτερος τρόπος οργάνωσης αρχείου είναι ο Β σε σχέση με τον Α γιατί έχει πολύ μικρότερο μέγεθος.΄

Τέλος στην εκφώνηση δεν ήταν προφανές το αν ήθελε στον Β τρόπο οργάνωσης να μετράμε τις προσβάσεις στον δίσκο και κατα την δημιουργία του αρχείου κλειδιών. Σ’αυτήν την περίπτωση δεν ξέρω ποιά θα ήταν τα αποτελέσματα αλλα δεν το έχω υλοποιήσει. (Θα χρειαζόταν απλα να προσθέσω στον Multicounter(2) και τις προσβάσεις κατα την δημιουργία).