Μπαντουβά Γεωργία 2019030006

Δομές Δεδομένων και Αρχείων

1η εργαστηριακή άσκηση

Κλάση Αρχείο

Η κλάση FileManager περιέχει μεθόδους για επεξεργασία αρχείου και συγκεκριμένα για δημιουργία/άνοιγμα/διάβασμα/γράψιμο/διαγραφή(σελίδας)/κλείσιμο αρχείου. Για την υλοποίηση των μεθόδων χρησιμοποιούνται instances της κλάσης RandomAccessFile και byte arrays στα οποία γράφονται & από τις οποίες διαβάζονται τα δεδομένα που αποθηκεύονται στα files. Συγκεκριμένα:

1) **firstPage**: η μέθοδος γράφει στην 1η σελίδα του αρχείου τις βασικές του πληροφορίες, όπως τον αριθμό των σελίδων, & το τέλος του, με τη βοήθεια ενός buffer array και χρησιμοποιείται και από άλλες μεθόδους για ενημέρωση των πληροφοριών ενός αρχείου.

2) **createFile**: δημιουργεί ένα άδειο αρχείο, θέτοντας ως mode “rw” (για ανάγνωση & εγγραφή) και καλεί τη fileHandle για να ενημερώσει τα στοιχεία του αρχείου

3) **openFile**: ανοίγει ένα αρχείο για ανάγνωση & εγγραφή

4) **readBlock/readNextBlock/readPrevBlock**: είναι μέθοδοι οι οποίες λαμβάνουν ως όρισμα τη σελίδα που θέλει ο χρήστης να διαβάσει, τοποθετούν τον «κέρσορα» στην κατάλληλη θέση πολλαπλασιάζοντας με το μέγεθος της σελίδας, και αποθηκεύουν το περιεχόμενο της σε ένα byte array.

5) **writeBlock/writeNextBlock/appendBlock**: είναι μέθοδοι για γράψιμο σε αρχείο. Βρίσκουν τη θέση στην οποία πρέπει να γράψουν με τον ίδιο τρόπο όπως πριν, και γράφουν στη θέση αυτή το περιεχόμενο ενός local buffer ο οποίος έχει οριστεί πριν το κάλεσμά τους. Γι’αυτό το λόγο δεν το λαμβάνουν ως argument. Ανανεώνουν επίσης τον συνολικό αριθμό σελίδων.

6) **deleteBlock**: με τη βοήθεια των παραπάνων μεθόδων, αντιγράφει το περιεχόμενο της τελευταίας σελίδας σε έναν buffer, και την πανωγράφει στη σελίδα που θέλουμε να διαγράψει. Ανανεώνει τον αριθμό των συνολικών bytes & των σελιδών.

7) **CloseFile**: καλεί την fileHandle για ενημέρωση των πληροφοριών του αρχείου & κλείνει το αρχείο.

Γενικά για το πρόγραμμα χρησιμοποιώ έτοιμες από φροντιστήρια τις κλάσεις ArraySearch, BinarySearch, MultiCounter, Node, SearchDataStructure.

Οργάνωση πληροφορίας σε σειριακό αρχείο και απόδοση αναζήτησης

Μέθοδος Α:

Η μέθοδος Α χρησιμοποιεί την έτοιμη μέθοδο του φροντιστηρίου **getAlphaNumericString**, για να παράγει ένα τυχαίο String συγκεκριμένου μήκους.

Έχει επιπλέον δύο μεθόδους:

1) **findKey**: Λαμβάνει ως όρισμα το key που αναζητούμε. Ανοίγουμε το αρχείο και αποθηκεύουμε στην μεταβλητή pages τον αριθμό των σελιδών του. Έτσι με ένα for loop διασχίζουμε όλες τις σελίδες, μια προς μια, αποθηκευόντας τα στοιχεία της σε ένα buffer των 128 bytes.

   //Σαν όρισμα της readBlock() έχω βάλει i+1, διότι θεωρώ ότι η 1η σελίδα κάθε //αρχείου περιέχει τις πληροφορίες του, π.χ όνομα, αριθμό σελίδων κτλ, //οπότε πρακτικά ξεκινάω να αναζητώ από την 2η και μετά.

   Μέσα σε κάθε σελίδα, διατρέχω τις 4 εγγραφές που (μπορεί να) περιέχει, και αποθηκεύω τα δεδομένα της κάθε εγγραφής(nodeKey, nodeData) σε ένα καινούριο nodeArray τύπου StringNode. Δημιουργώ ένα Object της ArraySearch, και με τη μέθοδο search(int key) διατρέχω το nodeArray για να βρω το κλειδί.

2) **writeFile**: Πρακτικά δημιουργεί το αρχείο οργανωμένο όπως περιγράφεται στην εκφώνηση, λαμβάνοντας ως όρισμα ένα array κλειδιών το οποίο δημιουργείται στη main και γράφεται στο αρχείο. Αυτό γίνεται με τη βοήθεια ενός byte array το οποίο και «αδειάζουμε» μετά την χρήση του.

Μέθοδος Β:

Παρομοίως, χρησιμοποιεί την getAplhaNumericString και έχει ακόμα άλλες 2 μεθόδους.

1) **findKey**: Λαμβάνει ως όρισμα την τιμή του κλειδιού που αναζητείται. Η συνάρτηση επεξεργάζεται κάθε σελίδα ξεχωριστά, διαβάζοντάς την και μεταφέροντας τα δεδομένα της σε ένα buffer array. Δημιουργεί 16 pageNodes, τα οποία περιέχουν ζεύγη ακεραίων – το κλειδί και τον αριθμό της σελίδας-. Από εκεί βρίσκουμε το ζεύγος που μας ενδιαφέρει και άρα και τον αριθμό της σελίδας στην οποία περιέχεται. Μετά ανοίγουμε το 1ο αρχείο (αρχικό) και τα εντοπίζουμε εκεί.

2) **writeFile**: Δημιουργεί το 2ο αρχείο, το οποίο είναι βοηθητικό, και περιέχει τα κλειδιά και το index της σελίδας στην οποία το κλειδί βρίσκεται στο αρχικό αρχείο.

   // Σχετικά με αυτήν την μέθοδο, ενώ βρίσκεται η σελίδα του κλειδιού, υπάρχει κάποιο bug στην αναζήτηση του κλειδιού στη μνήμη (εσωτερικό for loop), μάλλον υποθέτω ότι θα μπορούσε να είναι κάποιο υπολογιστικό λάθος/παράλειψη σε index των array από nodes, το οποίο δεν μπόρεσα να βρω, ωστόσο ο υπολογισμός των προσβάσεων(disk accesses) είναι κοντά στις προβλέψιμες. Το σημείο αυτό βρίσκεται σε σχόλια στον κώδικα για να μη δημιουργεί error όταν τον τρέχουμε.

Οργάνωση πληροφορίας σε ταξινομημένο αρχείο και απόδοση αναζήτησης

Μέθοδος C:

Χρησιμοποιεί επίσης την getAlphaNumericString για τον ίδιο λόγο. Επίσης έχει:

1) **sortFile**: διαβάζει όλο το αρχείο της μεθόδου Α στη μνήμη, το μετατρέπει σε Nodes, και χρησιμοποιώντας την Arrays.sort το ταξινομεί στη μνήμη. Μετά, γράφουμε το ταξινομημένο αρχείο αυτό στο δίσκο ανά σελίδα.
2) **findKey**: χρησιμοποειται η binarysearchfile, για να βρεθεί η σελίδα στην οποία βρίσκεται το κλειδί, και έπειτα η binarysearch στη μνήμη ώστε να βρεθεί το κλειδί στη συγκεκριμένη σελίδα

Για το C χρησιμοποιείται επίσης η κλάση **BinarySearchFile**, που εντοπίζει σε ποια σελίδα βρίσκεται το κλειδί στο ταξινομημένο αυτό αρχείο. Περιγράφεται παρακάτω.

Μέθοδος D:

Χρησιμοποιεί επίσης την getAlphaNumericString για τον ίδιο λόγο. Επίσης έχει:

1) **findKey**:

2) **sortFile**:

// δεν πρόλαβα να τις υπολοποιήσω, πρακτικά υπάρχει παραλληλισμός ανάμεσα στις
// μεθόδους Α-Γ και Β-Δ, καθώς απλά στις μεθόδους Γ,Δ τα αρχεία πρέπει να είναι
// ταξινομημένα.

Main

Αρχικά, δημιουργώ το array με τα random keys, με τη βοήθεια της μεθόδου randomKeys (έτοιμη μέθοδος από το φροντιστήριο του μαθήματος). Στη συνέχεια, θέλω να φτιάξω έναν πίνακα 20 κλειδιών, τα οποία θα παράγονται επίσης τυχαία, ώστε να τα αναζητήσω στα αρχεία μου. Αυτό το κάνω με τη βοήθεια της java.util.Random. Έχοντας τώρα έτοιμα τόσο τα δεδομένα που θα περαστούν στα αρχεία, όσο και τα κλειδιά που θέλω να αναζητήσω, διαδοχικά για κάθε τρόπο οργάνωσης:

1) δημιουργώ ένα instance της κάθε κλάσης με σκοπό να χρησιμοποιήσω τις συναρτήσεις της.
2) καλώ την writeFile με όρισμα το keys με σκοπό την δημιουργία των αρχείων
3) με ένα for loop, και με τη βοήθεια της findkey, αναζητώ κάθε φορά 1 από τα 20 τυχαία κλειδιά που παράγονται από το keysToFind.
4) Έπειτα εκτυπώνω τα αποτελέσματα μου.
5) Χρησιμοποιώ την κλάση MultiCounter (έτοιμη από το φροντιστήριο) ώστε να απαριθμήσω τον αριθμό προσβάσεων στο δίσκο για εγγραφή/ανάγνωση. Έτσι υπολογίζω και τον μέσο όρο αυτών, διαιρώντας τον συνολικό αριθμό προσβάσεων στον δίσκο με τον συνολικό αριθμό κλειδιών που ήθελα να βρω.
6) Τέλος, κάνω reset στους counters, ώστε να μπορούν να χρησιμοποιηθούν από την αρχή για τις επόμενες μεθόδους.

Αυτά τα βήματα γίνονται 4 φορές συνολικά, μια για κάθε μέθοδο.

Άλλες βοηθητικές κλάσεις που έφτιαξα:

PageNode/StringNode:
Και οι δυό κάνουν implement το interface Node, και γι'αυτό είναι comparable, άρα μπορεί να χρησιμοποιηθεί η Arrays.sort με μέτρο σύγκρισης το κλειδί που ψάχνουμε κάθε φορά. Συγκεκριμένα:

StringNode: είναι ένα structure με δύο πεδία, ένα integer κλειδί (4Byte) και ένα String (28Byte)

PageNode: είναι ένα structure με δύο πεδία, ένα κλειδί(4Byte) και έναν δείκτη που δείχνει τον αριθμό σελίδας(4Byte)

BinarySearchFile:

Περιέχει αρχικά μια μέθοδο **search**() η οποία λαμβάνει τον αριθμό σελίδων του αρχείου καλώντας την openFile, και έπειτα καλεί την doSearch για να ξεκινήσει η αναζήτηση του κλειδιού.

Η **doSearch** λαμβάνει ως όρισμα τα δύο άκρα ανάμεσα στα οποία θα ψάξουμε για το κλειδί μας, και το κλειδί που θέλουμε να βρούμε. Συνθήκη τερματισμού όπως είδαμε και στο φροντιστήριο είναι η `rightIndex >= leftIndex.`
Υπολογίζει το μέσο, αυξάνει τον counter κατά ένα αφού κάνουμε ένα πρώτο διάβασμα της σελίδας που βρίσκεται στη μέση, και χωρίζει τη σελίδα σε 4 StringNodes. Επειδή είναι ταξινομημένη, προφανώς το πρώτο node είναι το nodeArray[0] και το τελευταίο το nodeArray[3], των οποίων και παίρνουμε τα κλειδιά. Τώρα ξεκινάμε τον έλεγχο, ελέγχοντας αν το κλειδί που ψάχνουμε βρίσκεται ανάμεσα στα δύο ακριανά κλειδιά. Αν βρεθεί εκεί, κλείνουμε το αρχείο και επιστρέφουμαι την μεσαία σελίδα, αλλιώς ελέγχουμε αν το κλειδί είναι μικρότερο από το κλειδί του πρώτου Node. Αν ισχύει αυτή η συνθήκη, καλούμε αναδρομικά την doSearch, αυτή τη φορά αλλάζοντας τα ορίσματα (βλ. `doSearch(leftIndex, mid - 1, key)`)
Αν δεν ισχύει αυτό, τότε προφανώς το κλειδί είναι μεγαλύτερο από το κλειδί του μεγαλύτερου Node, και αντίστοιχα καλούμε πάλι αναδρομικά την συνάρτηση με κατάλληλα άκρα (`doSearch(mid + 1, rightIndex, key)`)
Η ιδέα είναι ίδια με αυτή που παρουσιάστηκε σε φροντιστήριο του μαθήματος.

Τεκμηρίωση των Αποτελεσμάτων

Μέθοδος	Α. τρόπος οργάνωσης αρχείου	Β. τρόπος οργάνωσης αρχείου	Γ. τρόπος οργάνωσης αρχείου	Δ. τρόπος οργάνωσης αρχείου	Απόδοση Εξωτερικής Ταξινόμησης περίπτωση Γ	Απόδοση Εξωτερικής Ταξινόμησης περίπτωση Δ
Απόδοση (αριθμός προσβάσεων δίσκου)	25772 συνολικά και κατά μέσο όρο 1288	6490 συνολικά και κατά μέσο όρο 324	260 συνολικά και κατά μέσο όρο 13	-	2500 για διάβασμα και 2500 για γράψιμο	-

Παρατηρώ ότι ο Β τρόπος οργάνωσης αρχείου είναι πιο αποτελεσματικός, καθώς είναι πιο γρήγορος από τον Α. Αυτό είναι και λογικό, γιατί με την μέθοδο Α έχουμε περισσότερες σελίδες στις οποίες γίνεται η αναζήτηση συγκεκριμένα 10000/4=2500 σελίδες, ενώ στην μέθοδο Β λόγω του τρόπου οργάνωσης (16 κλειδιά ανά σελίδα) έχουμε λιγότερο αριθμό σελιδών και συγκεκριμένα 10000/16=625 σελίδες. Άρα όπως αναμενόταν, ο αριθμός προσβάσεων είναι περίπου 4 φορές μικρότερος στη μέθοδο Β, γιατί το πρόγραμμα χρειάζεται να διαβάσει 4 φορές λιγότερες σελίδες.
Παρατηρώ επίσης ότι η μέθοδος Γ είναι ακόμα περισσότερο αποτελεσματική, αφού χρειάστηκαν αριθμητικά πολύ λιγότερες προσβάσεις στον δίσκο. Για αυτό ευθύνεται η ταξινόμηση, αφού κάθε φορά σπάμε το αρχείο σε δύο κομμάτια και ψάχνουμε μόνο στο ένα από τα δύο.
//Δεν έχω προλάβει να υλοποιήσω τη μέθοδο Δ, ωστόσο και εκεί αντίστοιχα θα περίμενα πολύ μικρότερο αριθμό προσβάσεων σε σχέση με τη Β, γιατί και πάλι το αρχείο θα ήταν ταξινομημένο.

Πηγές

1) https://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html
2) https://docs.oracle.com/javase/7/docs/api/java/io/ByteArrayOutputStream.html
3) https://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html
4) https://www.tutorialspoint.com/java/io/randomaccessfile_getfilepointer.htm
5) https://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html
6) https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/
7) https://www.geeksforgeeks.org/file-createnewfile-method-in-java-with-examples/
8) www.stackoverflow.com
9) Φροντιστηριακό Υλικό & Διαλέξεις μαθήματος