ΑΝΑΦΟΡΑ 1ης ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Δομές δεδομένων και αρχείων

Μιχάλης Γαλάνης
2016030036

ΣΚΟΠΟΣ

Σκοπός της πρώτης εργαστηριακής άσκησης είναι η εξοικείωσή μας με τη χρήση αρχείων στο δίσκο (δευτερεύουσα μνήμη), η κατανόησή μας για τον τρόπο επεξεργασίας πληροφοριών με χρήση σελίδων δίσκου και buffers και η σύγκριση ταχύτητας του αλγορίθμου ανάλογα τον αριθμό των σελίδων.

ΕΙΣΑΓΩΓΗ

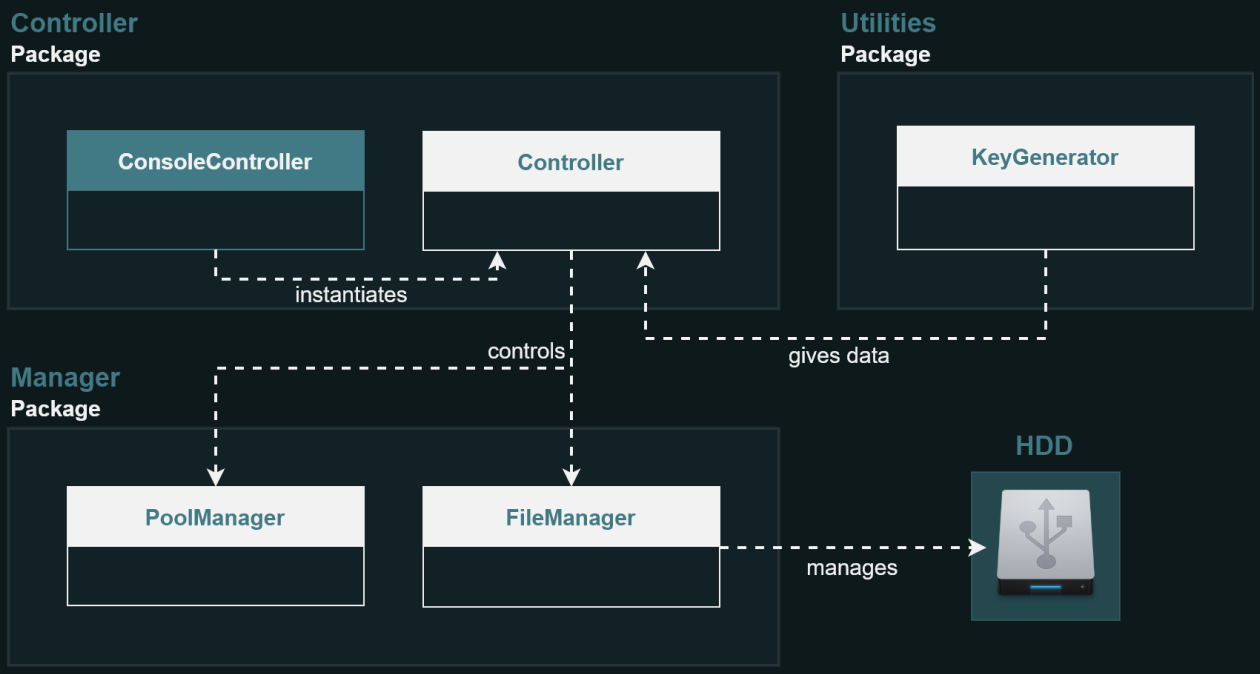
Η πληροφορία γνωρίζουμε από τη θεωρία οτι ταξινομείται σε **σελίδες δίσκου** και η επεξεργασία αρχείων περιλαμβάνει τη μεταφορά κάθε σελίδας μεταξύ κύριας και δευτερεύουσας μνήμης. Το ουσιαστικό **bottleneck** που δημιουργείται στη διαδικασία αυτή, είναι ο αριθμός των προσβάσεων στις σελίδες του δίσκου, καθώς η κύρια μνήμη του υπολογιστή είναι χιλιάδες φορές ταχύτερη στην επεξερασία πληροφοριών.

Μπορούμε να μειώσουμε τον αριθμό των προσβάσεων αυξάνοντας τη χωρητικότητας της σελίδας ή ακόμα και χρησιμοποιώντας πολλαπλές σελίδες με **trade-off** την αυξημένη κατανάλωση χώρου στην κύρια μνήμη. Το μέγεθος σελίδας που συμφέρει εξαρτάται καθαρά απο τις ανάγκες της κάθε εφαρμογής, και για την άσκηση αυτή επιλέγουμε το **page_size** ίσο με **512 bytes.**

Η επιλογή γλώσσας προγραμματισμούς ήταν η **Java,** καθώς είναι **object oriented** (βοηθάει στην οργάνωση του προγράμματος) και έχει πλούσια και δυνατά εργαλεία για επεξεργασία αρχείων.

ΔΟΜΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Το πρόγραμμα αποτελείται απο 3 πακέτα και 5 κλάσεις οργανωμένα με τον παρακάτω τρόπο:



ΕΠΕΞΗΓΗΣΗ ΕΡΩΤΗΜΑΤΩΝ

ΕΡΩΤΗΜΑ Α – Δημιουργία, Ανάγνωση, Εγγραφή και Διαγραφή αρχείων

Η κλάση **FileManager** αποτελείται από 11 μεθόδους επεξεργασίας αρχείων (δε συμπεριλαμβάνονται **setters, getters** και ο **constructor** της κλάσης). Η μέθοδος **createFile** με τη βοήθεια της java.io δημιουργεί ένα άδειο αρχείο και ύστερα καλεί την **fileHandle** για να εγγράψει τα βασικά χαρακτηριστικά του αρχείου στη πρώτη σελίδα του αρχείου. Η μέθοδος **openFile** ανοίγει το αρχείο με **RandomAccessFile** (λειτουργία **rw**) για να έχουμε τη δυνατότητα για ανάγνωση και εγγραφή πληροφορίας σε οποιοδήποτε σημείο του αρχείου. Η **closeFile** με τον ίδιο τρόπο έκλεινε το αρχείο.

Για τις υπόλοιπες μεθόδους, χρειαζόμαστε δυο μεταβλητές: byte[512] ***page***, int[128] ***buffer.*** Για την εγγραφή πληροφορίας (μεδόδους **writeBlock, writeNextBlock, writePreviousBlock, appendBlock**), το buffer γεμίζει αρχικά από τη γεννήτρια τυχαίων αριθμών (128 αριθμοί), κάνουμε ύστερα seek στο αρχείο σε επιθυμητό σημείο, μετατρέπεται η λίστα ακεραίων σε **page** (σελίδα δίσκου - λίστα τύπου byte) και αυτή εγγράφεται στο δίσκο. Η μετατροπή του **buffer** σε **page** επιτυγχάνεται με τη μέθοδο **convertBufferToPage.** Η διαδικασία για την ανάγνωση αντικειμένου (μεδόδους **readBlock, readNextBlock, readPreviousBlock**), είναι ακριβώς η ίδια με την διαφορά ότι αυτή τη φορά γεμίζει αρχικά το **page** και μετατρέπεται αργότερα σε **buffer** με τη βοήθεια της **convertPageToBuffer.**

Το πρώτο ερώτημα συμπεριλαμβάνει και την εισαγωγή στοιχείων. Η κλάση **Controller** δημιουργεί αρχικά ένα αρχείο **RandomGenFile.txt** με τη βοήθεια των μεθόδων της κλάσης **FileManager** και εισάγει 10^6 ακέραιους τυχαίους αριθμούς (κλειδιά) που παράγονται από τη μέθοδο της κλάσης **KeyGenerator.**

Υπάρχει ένας βασικός έλεγχος για λάθη σε όλες τις μεθόδους της κλάσης, αλλά δε χρησιμοποιήθηκαν **exceptions** καθώς η ένταξή τους θα ήταν χρονοβόρα και δεν αποτελεί αντικείμενο εξέτασης σε αυτή την άσκηση.

ΕΡΩΤΗΜΑ Β – Αναζήτηση κλειδιών στο αρχείο με χρήση μονοσέλιδου buffer

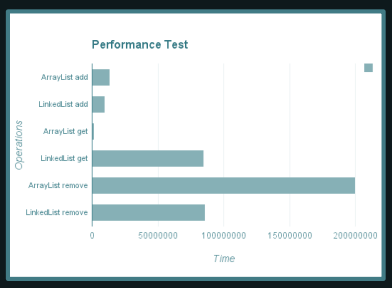
Η μέθοδος **searchWithoutPool** της κλάσης **Controller** ήταν υπεύθυνη για την αναζήτηση στο αρχείο **RandomGenFile.txt** για τυχαία κλειδιά. Πιο συγκεκριμένα, για μια επανάληψη 10.000 φορών (αριθμός αναζητήσεων): παραγόταν ένας αριθμός από τη μέθοδο **generateRandomKey** της **KeyGenerator.** Αμέσως μετά, μια άλλη επανάληψη (while) έλεγχε για κάθε block του αρχείου για τη ταυτοποίηση του κλειδιού με τη βοήθεια των μεθόδων **readNextBlock** και **isInBuffer.** Σε κάθε περίπτωση μια μεταβλητή μετρούσε τα disk accesses (ισχύει 1 disk access ανά 1 block) και στο τέλος υπολογιζόταν η μέση τιμή των disk accesses για όλες τις αναζητήσεις.

ΕΡΩΤΗΜΑ Γ – Χρήση πολλαπλών σελίδων buffer

Σε αυτό το ερώτημα, έπρεπε να κατασκευάσουμε την κλάση **PoolManager** με 5 απλές μεθόδους διαχείρηση λίστας συμπεριλαμβανομένου: δημιουργίας, εισαγωγής, αναζήτησης, διαγραφής και καθαρισμού της λίστας με γνωστή ύλη από το προηγούμενο εξάμηνο . Αυτή τη φορά ο **buffer** ήταν **ArrayList()** με elements τύπου int[128].

Σημείωση: Ο λόγος που προτίμησα **ArrayList()** αντί για παράδειγμα **LinkedList()** ήταν στη διαφορά της ταχύτητας των μεθόδων τους (**get, add, remove**). Για τις ανάγκες της άσκησης αυτής, μας ενδιαφέρει η **get** πολύ περισσότερο από την add και την remove.

	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1) amortized
remove()	O(n)	O(n)



Για τα 2 διαφορετικά **buffer pools** που θα χρησιμοποιήσουμε στην αναζήτηση **Δ**, δημιουργήθηκαν 2 objects του **PoolManager**, με σελίδες **K = 500**, **K = 1000** αντίστοιχα.

ΕΡΩΤΗΜΑ Δ - Αναζήτηση κλειδιών στο αρχείο με χρήση πολλαπλών σελίδων buffer

Η μέθοδος **searchWithPool** της κλάσης **Controller** είναι παρόμοια με αυτή του **B** ερωτήματος, με τη διαφορά ότι αυτή τη φορά εκμεταλλευόμαστε το **buffer pool.** Εδώ, 1 disk access χρεώνεται κάθε φορά που ξαναγεμίζουμε τη λίστα σελίδων του buffer εφόσον δεν έχει βρεθεί το ζητούμενο κλειδί σε προηγούμενο pool. Η παραπάνω μέθοδος εκτελείται 2 φορές, μία για κάθε διαφορετικού μεγέθους **buffer pool.**

ΕΡΩΤΗΜΑ Ε – Summary (Σύγκριση αποτελεσμάτων)

Περιμένουμε λοιπόν με τη χρήση του Buffer Pool να μειωθούν δραματικά τα disk Accesses και να καταλήξουμε σε έναν πολύ πιο αποτελεσματικό αλγόριθμο. Εκτελώντας το πρόγραμμα, η κονσόλα παράγει τις παρακάτω γραμμές:

```
Attempting to generate a random key file.
RandomGenFile.txt successfully created!
----------------------------------------
Number of pages: 7814
File size: 4000256 bytes (4 MB).
```

```
-------------- METHOD B (1-page buffer) --------------
Numbers found: 3973/10000 (39.73%)
Average number of disk accesses for all searches: 5420
```

```
-------------------- METHOD D (K= 500) --------------------
Numbers found: 3927/10000 (39.27%)
Average number of disk accesses for all searches: 11
```

```
-------------------- METHOD D (K= 1000) --------------------
Numbers found: 3912/10000 (39.12%)
Average number of disk accesses for all searches: 5
```

Παρόλο το κέρδος της ταχύτητας με τη μέθοδο **Δ,** αυτό είναι ένα κλασικό παράδειγμα trade-off ταχύτητας – χώρου, καθώς το **buffer pool** ενάντια στο **1-page buffer** καταναλώνει 500 ή και 1000 φορές αντίστοιχα περισσότερο χώρο στη κύρια μνήμη.

ΠΗΓΕΣ

- Βοήθεια για τη κλάση **ByteBuffer** της Java που επιτρέπει τη μετατροπή των **integers** σε **byte arrays** και αντιστρόφως: **link**
- Βοήθεια για την επιλογή κατάλληλου List Collection και σύγκριση ταχύτητας: **link**