ΠΛΗ211 Δομές Δεδομένων και Αρχείων
1η Άσκηση

Παντουράκης Μιχαήλ ΑΜ 2015030185
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

27 Μαρτίου 2017

1 Σκοπός - Υποθέσεις

Η παρούσα προγραμματιστική άσκηση αποτελεί μία πρώτη πρακτική εξοικείωση των φοιτητών με: 1) την έννοια της χρονικής πολυπλοκότητας, 2) την οργάνωση της πληροφορίας σε σελίδες στον δίσκο, και 3) την απόδοση διαφόρων μεθόδων αναζήτησης σε αυτόν.

Για την ανάπτυξη του παρόντος προγράμματος χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java, και το εργαλείο Eclipse IDE for Java Developers, Neon Release (4.6.0), στο οποίο μεταφράστηκε και εκτελέστηκε. Πέρα από την Java SE 1.8. (και φυσικά το ίδιο το παραδοτέο project) δεν απαιτείται η εισαγωγή κάποιας άλλης βιβλιοθήκης.

Η παρούσα εφαρμογή ακολουθεί την εξής λογική: 1) Δημιουργία και αποθήκευση ενός αρχείου που περιέχει ταξινομημένους ακεραίους έως ένα προκαθορισμένο ακέραιο, ξεκινώντας από το ένα και αυξάνοντας κατά ένα. 2) Αναζήτηση κλειδιών (ακεραίων) σε αρχείο που περιέχει ταξινομημένους (με αύξουσα σειρά) ακεραίους με τέσσερις διαφορετικούς τρόπους (σειριακή, απλή δυαδική, δυαδική με ομαδοποίηση ερωτημάτων, δυαδική με χρήση προσωρινής μνήμης. 3) Για την επεξεργασία του αρχείου χρησιμοποιείται δυαδικό buffer στη μνήμη, μεγέθους μιας σελίδας δίσκου, ενώ το μέγεθος της σελίδας είναι προκαθορισμένο. 4) Κάθε είδος αναζήτησης επαναλαμβάνεται σύμφωνα με έναν ορισμένο αριθμό επαναλήψεων για εξαγωγή της μέσης τιμής προσβάσεων στο δίσκο.

Όσον αφορά τους περιορισμούς σωστής λειτουργίας του προγράμματος: 1) Ο κώδικας υποστηρίζει την εγγραφή και αναζήτηση σε δυαδικό αρχείο οποιουδήποτε συνδυασμού πλήθους ταξινομημένων ακεραίων και μεγέθους σελίδας του δίσκου. Αυτό φυσικά προϋποθέτει να δοθούν σωστά οι παράμετροι μεγέθους σελίδας *(bufferSize)* και πλήθους ακεραίων *(numberOfInt)* που χρησιμοποιήθηκαν κατά την εγγραφή του αρχείου. 2) Επιπλέον, οι ακέραιοι της εφαρμογής περιορίζονται από το primitive type *int* της Java, δηλαδή είναι 32-bit signed two's-complement. Έτσι μεγαλύτεροι ακέραιοι δεν υποστηρίζονται.

2 Υλοποίηση

Το κύριο μέρος της υλοποίησης του προγράμματος πραγματοποιείται από δύο κλάσεις, την FileHandler και την MainController. Η πρώτη περιέχει, με ξεχωριστή μέθοδο ανά ερώτημα, τη ζητούμενη λειτουργικότητα, ενώ η δεύτερη περιέχει τη main, από όπου και φαίνεται η ροή του συνολικού προγράμματος. Στις επόμενες υποενότητες παραθέτω τη γενική ιδέα επίλυσης κάθε ερωτήματος ξεχωριστά.

Πέραν τούτων, χρησιμοποιήθηκαν άλλες τέσσερις βοηθητικές κλάσεις και μία διεπαφή (interface). Πιο συγκεκριμένα, χρησιμοποίησα τις Queue (διεπαφή), ArrayQueue, και QueueEndReachedException για την υλοποίηση της ουράς ως δομή προσωρινής μνήμης, τη DiskPage για τη αποθήκευση του buffer μαζί με τη σελίδα του αρχείου στην οποία αντιστοιχεί, και την Assert για τον έλεγχο boolean παραμέτρων (δείτε ενότητα 4 για τις πηγές και τροποποιήσεις).

2.1 Εισαγωγή και Ανάγνωση Στοιχείων

Η δημιουργία του ζητούμενου αρχείου βασίζεται στη μέθοδο create. Όλοι οι ακέραιοι εισάγονται αρχικά με αύξουσα σειρά σε ένα array, από το οποίο και αντιγράφονται στο buffer (και κατ΄ επέκταση στο αρχείο) ακέραιοι πλήθους μίας σελίδας, έως ότου γραφτούν όλοι οι ακέραιοι με σειριακό τρόπο (σε ένα for loop). Για τη λειτουργικότητα seek και write χρησιμοποιήθηκε η κλάση RandomAccessFile, ενώ οι κλάσεις ByteBuffer, IntBuffer ανέλαβαν τη μετατροπή του int array σε byte array. Αντίστροφη διαδικασία χρησιμοποιείται με τη μέθοδο readPage σε όλες τις αναζητήσεις για την ανάγνωση του αρχείου, με τη μέθοδο να επιστρέφει ένα αντικείμενο της κλάσης DiskPage.

Άξια αναφοράς αποτελεί η προσθήκη του Integer.*MAX_VALUE* (0x7FFFFFFF) σε πιθανές κενές θέσεις της τελευταίας σελίδας του αρχείου. Επέλεξα αυτή τη μέθοδο ώστε το buffer της τελευταίας σελίδας να παραμένει σε κάθε περίπτωση ταξινομημένο, εξασφαλίζοντας έτσι τη σωστή λειτουργία της έτοιμης μεθόδου Arrays.binarySearch. Φυσικά για την ειδική περίπτωση των τιμών της άσκησης δεν υπάρχουν κενές θέσεις.

2.2 Σειριακή Αναζήτηση στο Αρχείο για Τυχαίο Κλειδί

Για την σειριακή αναζήτηση αρκεί η χρήση ενός for loop, το οποίο αντιστοιχεί στον αριθμό των ερωτημάτων, και ενός φωλιασμένου do while, για την σειριακή ανάγνωση των σελίδων έως ότου βρει τον ζητούμενο ακέραιο ή αποτύχει φτάνοντας στο τέλος του αρχείου. Η ανάγνωση των σελιδών γίνεται με τη χρήση ενός *filePointer*, που αλλάζει σύμφωνα με τη σχέση $filePointer = (currentPage - 1) \cdot bufferSize$. Φυσικά, σε κάθε ανάγνωση σελίδας η μεταβλητή-μετρητής *diskAccesses* αυξάνεται κατά ένα.

Για την εξαγωγή τυχαίων κλειδιών (μέσα στο εύρος τιμών του αρχείου), ανέπτυξα τη μέθοδο getRandomIntegers, η οποία και κάνει χρήση της κλάσης ψευδοτυχαίων αριθμών Random. Τέλος, για την δυαδική αναζήτηση μέσα στο buffer, χρησιμοποιήθηκε η μέθοδος Arrays.binarySearch.

2.3 Δυαδική Αναζήτηση στο Αρχείο για Τυχαίο Κλειδί

Η δυαδική αναζήτηση στο αρχείο διαφέρει από τη σειριακή αναζήτηση μόνο στη σειρά διάσχισης των σελίδων του αρχείου. Έτσι ψάχνουμε τον ακέραιο ξεκινώντας από τη μεσαία σελίδα του υπολειπόμενου αρχείου, και ανάλογα με την τιμή που επιστρέφει η μέθοδος Arrays.binarySearch (είτε -1 αν είναι σε προηγούμενη σελίδα, είτε -1-πλήθος ακεραίων στο buffer αν είναι σε επόμενη σελίδα, ή η θέση στο buffer αν βρεθεί) συνεχίζουμε τον έλεγχο στη μεσαία σελίδα του δεξιού ή αριστερού μισού του υπολειπόμενου αρχείου. Η διαδικασία αυτή επαναλαμβάνεται έως ότου μηδενιστεί το εύρος των πιθανών σελίδων ή βρεθεί ο αριθμός.

2.4 Δυαδική Αναζήτηση με Ομαδοποίηση των Ερωτήσεων

Σε αυτό το ερώτημα στην ουσία έρχεται να προστεθεί η μέθοδος Arrays.sort, η οποία ταξινομεί τα κλειδιά κατά αύξουσα σειρά. Στη συνέχεια εκτελούμε δυαδική αναζήτηση (όπως προηγουμένως) για το μικρότερο κλειδί. Έχοντας τώρα προσωρινά αποθηκευμένο στο buffer την σελίδα που περιείχε το προηγούμενο κλειδί, ελέγχουμε αν και το επόμενο υπάρχει στο buffer, γλυτώνοντας έτσι περιττές προσβάσεις στο δίσκο. Διαφορετικά, συνεχίζουμε κανονικά με δυαδική αναζήτηση.

2.5 Δυαδική Αναζήτηση με Χρήση Προσωρινής Μνήμης

Η λύση εδώ χρησιμοποιεί και επεκτείνει τη λογική της αναζήτησης πρώτα στην προσωρινή μνήμη, και μόνο αν αυτή αποτύχει, έπειτα στο δίσκο. Έτσι, αξιοποιεί την κυκλική λίστα ArrayQueue ως δομή μνήμης που περιέχει τις πιο πρόσφατες σελίδες. Στην πράξη, η ArrayQueue συμπληρώνεται με σελίδες κατά τη δυαδική αναζήτηση του πρώτου κλειδιού. Συνεπώς, για τις δυαδικές αναζητήσεις των υπόλοιπων κλειδιών όποια σελίδα υπάρχει ήδη στην ArrayQueue, δεν χρειάζεται να ξαναδιαβαστεί από το δίσκο.

3 Τεκμηρίωση Αποτελεσμάτων

Τα αποτελέσματα του πίνακα 1 επιβεβαιώνουν τις θεωρητικά υπολογιζόμενες χρονικές πολυπλοκότητες (ως προς τον αριθμό προσβάσεων) της σειριακής και δυαδικής αναζήτησης. Συγκεκριμένα, το αρχείο για τις τιμές της άσκησης αποτελείται από $N = 78125$ σελίδες. Επομένως, η σειριακή αναζήτηση είναι και η πιο αργή αφού παρουσιάζει στη μέση περίπτωση χρονική πολυπλοκότητα $(N + 1)/2 \in \mathcal{O}(N)$, κάτι που

Μέθοδος	**2.2**	**2.3**	**2.4**	**2.5** ($K = 1$)	**2.5** ($K = 50$)	**2.5** ($K = 100$)
Απόδοση ($M = 10^4$)	39000,3	15,341	14,451	15,322	12,913	11,902
Απόδοση ($M = 10^5$)	-	15,331	8,641	15,327	12,917	11,882
Απόδοση ($M = 10^3$)	-	15,41	15,28	15,353	12,928	11,931

Πίνακας 1: Πίνακας μέσου αριθμού προσβάσεων στο δίσκο για M αναζητήσεις κάθε μεθόδου.

επιβεβαιώνει ο μέσος αριθμός προσβάσεων της δοκιμασίας (με την αναμενόμενη στατιστική διακύμανση). Αντίθετα, η απλή δυαδική αναζήτηση παρουσιάζει πολυπλοκότητα $\mathcal{O}(\log N)$, δικαιολογώντας τη μεγάλη διαφορά στο μέσο αριθμό προσβάσεων ($\log 78125 \approx 16,253$).

Όσον αφορά τις τεχνικές ομαδοποίησης και προσωρινής μνήμης, παρατηρούμε ότι αν και δεν διαφέρουν στην τάξη μεγέθους της πολυπλοκότητας δυαδικής αναζήτησης, καταφέρνουν να μειώσουν ουσιαστικά τις σταθερές της. Η ομαδοποίηση ερωτημάτων συντελεί ώστε κλειδιά που βρίσκονται στην ίδια σελίδα με το αμέσως προηγούμενό τους να εντοπίζονται σε σταθερό χρόνο ($\mathcal{O}(1)$).Συνεπώς, όταν αυτό συμβαίνει συχνά, δηλαδή όταν η αναλογία αριθμός σελίδων - αριθμός κλειδιών είναι μικρή, βλέπουμε ότι αυτή η μέθοδος πλεονεκτεί (χρονικά). Από την άλλη πλευρά, παρατηρούμε ότι η χρήση προσωρινής μνήμης αποτελεί μία εύρωστη τεχνική, καθώς η απόδοσή της δεν επηρεάζεται από αυτή την αναλογία. Αναμενόμενα, όσο μεγαλύτερη μνήμη χρησιμοποιεί η μέθοδος, τόσο λιγότερες προσβάσεις δίσκου θα χρειαστεί, ενώ για $K = 1$ είναι προφανές ότι θα βοηθήσει μόνο τις ελάχιστες φορές που το κλειδί που αναζητείται βρίσκεται στη μεσαία σελίδα.

Τέλος, σχετικά με την εναλλακτική υλοποίηση της μνήμης, μια απλή και αποτελεσματική λύση θα ήταν η χρήση δομής min binary heap, με κλειδί την πιθανότητα χρήσης κάθε σελίδας (συνοδευόμενο από ένα hash table ώστε να γνωρίζουμε σε σταθερό χρόνο το που βρίσκεται μέσα στη δομή η κάθε σελίδα). Στην συγκεκριμένη περίπτωση αναμένουμε βελτίωση στο μέσο αριθμό προσβάσεων σε σύγκριση με τα αποτελέσματα της κυκλική ουράς, καθώς οι σελίδες δεν έχουν ίση πιθανότητα χρήσης p_i, αλλά αυτή εξαρτάται από το βάθος που βρίσκεται η σελίδα στο αντίστοιχο της αναζήτησης δυαδικό δένδρο (BST), δηλαδή $p_i = 1/2^{d_i}$ (π.χ. η μεσαία σελίδα χρειάζεται πάντα, η μεσαία του δεξιού/αριστερού μισού έχει $p_i = 0,5$, κ.ο.κ.). Επομένως, ακόμη και αν δεν προ-αποθηκευτούν οι θεωρητικά συχνότερα χρησιμοποιούμενες σελίδες, αναμένουμε σε εύλογο διάστημα χρήσης η κατανομή αυτή να προσεγγιστεί και οι αριθμοί πρόσβασης στο δίσκο να μειωθούν.

4 Πηγές και Συνεργασίες

Για την υλοποίηση του προγράμματος υιοθετήθηκαν οι εξής ενδεικτικοί κώδικες του μαθήματος: 1) της διεπαφής Queue, και της αντίστοιχης κλάσης AQueue, στην οποία και πρόσθεσα μεθόδους για τη σειριακή διάσχιση όλων των στοιχείων της ουράς, φτιάχνοντας έτσι την κλάση ArrayQueue και το Exception QueueEndReachedException του παραδοτέου. 2) Επιπλέον για λόγους ευκολίας χρησιμοποίησα και την Assert.

Κατά τη διεκπεραίωση της άσκησης συνεργάστηκα με τον συμφοιτητή μου Πανταζή Θεόδωρο (ΑΜ 2005030004), τον οποίο και βοήθησα εξηγώντας του τα ζητούμενα της άσκησης, το γενικό τρόπο αντιμετώπισης τέτοιων ασκήσεων, και βασικά τμήματα του κώδικά μου.