ΠΛΗ211 Δομές Δεδομένων και Αρχείων
2η Άσκηση

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

01 Μαΐου 2017

1 Σκοπός - Υποθέσεις

Η δεύτερη προγραμματιστική άσκηση αφορά ξανά την ανάπτυξη δομών αρχείου. Συγκεκριμένα, ζητείται η ανάπτυξη ενός B-tree και μίας Posting List, με σκοπό την κατασκευή προγράμματος γρήγορης εύρεσης λέξεων σε κείμενα, όπου οι δομές αυτές θα αναλάβουν το ρόλο λεξικού και ευρετηρίου αντίστοιχα. Ακολούθως, παρόμοια με την πρώτη άσκηση, ζητείται η ανάλυση της απόδοσης των δομών αυτών, τόσο κατά την εισαγωγή όσο και κατά την αναζήτηση λέξεων.

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. Το παραδοτέο δημιουργήθηκε μέσω export του Eclipse και παραδίδεται αυτούσιο με όλα τα αρχεία.

Η παρούσα υλοποίηση ακολουθεί την εξής λογική: 1) Ανάγνωση των τριών αρχείων κειμένου που δόθηκαν και αποθήκευση στη μνήμη όλων των λέξεων με το όνομα αρχείου και τη θέση στο αρχείου που βρίσκονται. 2) Δημιουργία των δυαδικών B-tree file και Posting file εισάγοντας μία-μία τις λέξεις που εντοπίσαμε στο προηγούμενο βήμα και χρησιμοποιώντας προκαθορισμένο μέγεθος σελίδας δίσκου. Σε αυτό το βήμα μετριέται ο μέσος αριθμός προσβάσεων στο δίσκο κατά την εισαγωγή στοιχείων στο B-tree (αγνοώντας τις προσβάσεις στο ευρετήριο). 3) Αναζήτηση 100 τυχαίων λέξεων από αυτές που βρίσκονται στα παραπάνω αρχεία κειμένου για τον υπολογισμό του μέσου αριθμού ανάγνωσης σελίδων δίσκου των B-tree, Posting file σε επιτυχή αναζήτηση. 4) Αναζήτηση 100 τυχαίων λέξεων, από τις οποίες κάποιες (με συγκεκριμένη συχνότητα που καθορίζεται από παράμετρο εισόδου του προγράμματος) δεν υπάρχουν στο B-tree.

Όσον αφορά τους περιορισμούς σωστής λειτουργίας του προγράμματος: 1) Το πρόγραμμα αναγνωρίζει ως ξεχωριστές δύο λέξεις που ενώνονται με παύλα. 2) Ο κώδικας υποστηρίζει την εγγραφή και αναζήτηση σε δυαδικά αρχεία οποιουδήποτε μεγέθους σελίδας του δίσκου μεγαλύτερου ή ίσου των 72 bytes (περιορισμός της ζητούμενης δομής, όχι του προγράμματος). 3) Ο κώδικας περιορίζει τον αριθμό αναζητήσεων ως ίσο ή μικρότερο του πλήθους του συνόλου (set) μοναδικών λέξεων που υπάρχουν στα κείμενα. 4) Το πρόγραμμα δεν ελέγχει την μοναδικότητα των εγγραφών στην Posting List (δεν ζητείται από την άσκηση, π.χ. αν διαβαστεί δύο φορές το ίδιο αρχείο κειμένου). 5) Το πρόγραμμα δεν ξεχωρίζει λέξεις που έχουν ίδιους πρώτους 12 χαρακτήρες. 6) Τέλος, το πρόγραμμα παράγει IOException για αρχεία που έχουν όνομα με ίδιους πρώτους 8 χαρακτήρες.

2 Υλοποίηση

Το κύριο μέρος της λειτουργίας του προγράμματος πραγματοποιείται από πέντε κλάσεις: 1) AsciiFileHandler, η οποία περιέχει τη μέθοδο ανάγνωσης των λέξεων από αρχεία κειμένου. 2-3) BTreeFile, PostingFile που υλοποιούν τις αντίστοιχες δομές στο δίσκο με τις μεθόδους εισαγωγής και αναζήτησης στοιχείων. 4) Controller, η οποία ενώνει λειτουργικά τις δύο δομές για την εισαγωγή και αναζήτηση των

αντίστοιχων πληροφοριών στο λεξικό και στο ευρετήριο. 5) MainLauncher, που περιέχει τη main, από όπου και φαίνεται η ροή του συνολικού προγράμματος.

Πέραν τούτων, χρησιμοποιήθηκαν άλλες πέντε βοηθητικές κλάσεις. Πιο συγκεκριμένα, χρησιμοποίησα τις BTreeNode, PostingFileNode, για τη δημιουργία/επεξεργασία των κόμβων των αντίστοιχων δομών, τις Word, BTreeSearchResult, DiskAccessNum για την ομαδοποίηση επιστρεφόμενων τιμών των λειτουργιών ανάγνωσης κειμένου, εισαγωγής και αναζήτησης κλειδιών, και την Assert (από το υλικό του μαθήματος) για τον έλεγχο της ύπαρξης του αρχείου B-tree πριν από αναζήτηση. Στις υποενότητες που ακολουθούν παραθέτω τη γενική ιδέα επίλυσης κάθε ερωτήματος ξεχωριστά.

2.1 Ανάγνωση Λέξεων από Αρχεία Κειμένου

Για την ανάγνωση και διαχωρισμό των λέξεων που περιέχονται στα αρχεία κειμένου χρησιμοποιήθηκε η μέθοδος readAsciiFile, παραλλαγή του αντίστοιχου κώδικα ReadFromAsciiFile από το υλικό του μαθήματος. Ως delimiters προστέθηκαν ακόμη όσες περιπτώσεις χρειάζονταν ώστε κανένα token από τα δοθέντα αρχεία κειμένου να μην περιέχει το regular expression '\w*\W\w*|\w*[0-9_]\w*'(ώστε να μην εμφανίζονται λέξεις ενωμένες με σύμβολα ή ψηφία στα τρία αρχεία που δόθηκαν). Ως έξοδος της μεθόδου δίνεται η λίστα των λέξεων με τις θέσεις που βρέθηκαν στο αρχείο.

2.2 B-Tree στον Δίσκο

Πέρα από το υλικό του μαθήματος, για την υλοποίηση του B-Tree στο δίσκο στηρίχθηκα στο υλικό (περιγραφή και ψευδοκώδικας) του συνδέσμου [1], το οποίο και προσάρμοσα στα δεδομένα της άσκησης και της Java.

Διαχείριση κόμβων: Για την αποθήκευση/ανάκτηση των κόμβων και των δύο δομών σε/από δυαδικά αρχεία, ανέπτυξα αντίστοιχες μεθόδους serialize, deserialize.

Ρίζα: Ο παρών κώδικας φροντίζει ώστε πάντα η ρίζα του δέντρου να βρίσκεται στην αρχή του B-Tree αρχείου, αλλάζοντας την υπόθεση του [1] ότι η ρίζα υπάρχει πάντα στη μνήμη.

Τάξη: Για τον καθορισμό της τάξης του B-Tree, στον constructor της κλάσης χρησιμοποιώ τον τύπο $4 \cdot \text{order} + 16 \cdot (\text{order} - 1) + 8 \leq \text{pageSize} \implies \text{order} = 2 \cdot \lfloor \lfloor (\text{pageSize} + 8)/20 \rfloor / 2 \rfloor$, ο οποίος εξασφαλίζει τόσο το μέγιστο δυνατό αριθμό κλειδιών ανά κόμβο (συμπεριλαμβανομένου των ζητούμενων πεδίων), όσο και την προς τα κάτω στρογγυλοποίηση της τάξης στον πλησιέστερο άρτιο. Αυτός ο περιορισμός επιτρέπει τον τρόπο διάσπασης πλήρους κόμβου πριν από την εισαγωγή όπως περιγράφεται στη συνέχεια.

Διαχείριση μεγέθους λέξεων: Για την αποθήκευσή τους χρησιμοποιήθηκε String array 12 bytes. Επομένως, όσες λέξεις είχαν λιγότερους από 12 χαρακτήρες, συμπληρώθηκαν κατά την εισαγωγή τους στη δομή με κενά. Αυτό διευκόλυνε και την άμεση χρήση των τελεστών σύγκρισης πάνω στα κλειδιά.

Εισαγωγή: Η εισαγωγή στοιχείων στο δέντρο εκτελείται από τρεις ρουτίνες, την insert, την insertNonFull και την split, όπως αυτές περιγράφονται στο σύνδεσμο [1]. Βασικό χαρακτηριστικό της παρούσας υλοποίησης εισαγωγής είναι το προκαταβολικό split κάθε φορά που συναντά ένα γεμάτο κόμβο κατά την πορεία προς το υποψήφιο φύλλο εισαγωγής. Με αυτό τον τρόπο αποφεύγεται η πιθανή επανάκτηση από τον δίσκο (και άρα το επιπλέον κόστος ανάγνωσης) των γονεϊκών κόμβων όταν το φύλλο εισαγωγής είναι ήδη γεμάτο.

Μιας και η εκφώνηση της άσκησης απαιτούσε την αποθήκευση δείκτη στο γονέα, για την εφαρμογή της συγκεκριμένης παραλλαγής ανέπτυξα την μέθοδο updatePointerOfChildren. Η μέθοδος αυτή είναι αναγκαία καθώς μετά από κάθε διάσπαση κόμβου είναι απαραίτητη η ανανέωση του δείκτη πατέρα των παιδιών του νέου κόμβου. Το ίδιο ισχύει και στην περίπτωση διάσπασης της ρίζας, όπου η παλιά ρίζα μετακινείται στο τέλος του αρχείου. Φυσικά, με την παρούσα μέθοδο εισαγωγής στοιχείων ο δείκτης προς τον πατέρα-κόμβο θα μπορούσε να παραλειφθεί.

Τέλος, σημαντική είναι η εξασφάλιση της μοναδικότητας των κλειδιών του B-tree, η οποία και πραγματοποιείται πριν την εισαγωγή (εξωτερικά στον Controller) με αναζήτηση του κλειδιού, ώστε αν υπάρχει να μην επανεισαχθεί.

Αναζήτηση: Αποτελεί απλή επέκταση της αναζήτησης σε δυαδικό δένδρο. Για την αναζήτηση εσωτερικά των κόμβων χρησιμοποιώ δυαδική αναζήτηση (Arrays.binarySearch). Τέλος, οι γραμμές κώδικα που εκτυπώνουν στην οθόνη τα σημεία εντοπισμού της κάθε λέξης, έχουν αφαιρεθεί ως σχόλια για

	Εισαγωγή στο B-tree	Επιτυχής Αναζήτηση (100)		Τυχαία Αναζήτηση (100)	
Μέθοδος	**Μέσος αριθμός προσβάσεων δίσκου**	**Μέσος αριθμός προσβάσεων δίσκου στο B-tree**	**Μέσος αριθμός προσβάσεων δίσκου στο Ευρετήριο**	**Μέσος αριθμός προσβάσεων δίσκου στο B-tree**	**Μέσος αριθμός προσβάσεων δίσκου στο Ευρετήριο**
Μέτρηση	με διπλοεγγραφές: 6,60 χωρίς: 11,90	5,66	1,02	$p = 0,5$: 5,78 $p = 1$: 6	$p = 0,5$: 0,50 $p = 1$: 0

Πίνακας 1: Πίνακας μέσου αριθμού προσβάσεων στο δίσκο για τις εισαγωγές στο B-tree και 100 επιτυχείς και τυχαίες αναζητήσεις. Στη μέτρηση για τις εισαγωγές δεν υπολογίζονται οι προσβάσεις στο ευρετήριο. Το p συμβολίζει την πιθανότητα μη ύπαρξης της επιλεγμένης λέξης στο στο B-tree.

ευκολότερη εξαγωγή των αποτελεσμάτων.

2.3 Ευρετήριο στον Δίσκο

Για τη δημιουργία του ζητούμενου ευρετηρίου χρησιμοποιείται η μέθοδος της PostingFile, insert. Σε αυτή διαχωρίζονται επίσης οι περιπτώσεις όπου δημιουργείται μία νέα σελίδα για μία νέα λέξη, ή μία νέα σελίδα όταν η υπάρχουσα σελίδα μίας λέξης έχει γεμίσει από αναφορές της.

Στην παρούσα υλοποίηση δεν τηρώ κάποια ταξινόμηση των δεδομένων. Επομένως, ως λίστα οποιαδήποτε εσωτερική αναζήτηση γίνεται σειριακά. Παρόλα αυτά, σε κάθε κόμβο διατηρώ επιπλέον πληροφορία για τον αριθμό των αποθηκευμένων στοιχείων, κάτι που επιταχύνει τη διαδικασία εισαγωγής (και για το παράδειγμα της σελίδας 128 bytes δεν κοστίζει σε χώρο).

3 Ανάλυση Απόδοσης

Ο πίνακας 1 παρουσιάζει τα ζητούμενα αποτελέσματα για τα τρία αρχεία κειμένου που μας δόθηκαν και για μέγεθος σελίδας δίσκου 128 bytes, και οι προκύπτουσες τιμές είναι οι αναμενόμενες. Πιο συγκεκριμένα, το B-tree σχημάτισε συνολικά 458 κόμβους με βάθος 5. Αναλογιζόμενοι ότι στη χειρότερη περίπτωση κάθε κόμβος έχει 3 παιδιά, τότε με βάθος 5 αναμένουμε τουλάχιστον $3^5 + 1 = 244$ κόμβους, κάτι που επαληθεύεται εδώ.

Το βάθος του δένδρου γίνεται αμέσως φανερό από το μέσο αριθμό προσβάσεων δίσκου κατά τις αναζητήσεις στο B-tree. Έτσι, αν όλες οι λέξεις που αναζητήσουμε δεν υπάρχουν στο λεξικό ($p = 1$), τότε όλες οι αναζητήσεις τερματίζουν στα φύλλα του δένδρου, φτάνοντας εκεί πάντα με 6 προσβάσεις (βάθος 5 συν την ανάγνωση της ρίζας). Αντίθετα αν όλες οι αναζητήσεις είναι επιτυχημένες, κάποιες από αυτές θα βρουν το κλειδί πριν φτάσουν στα φύλλα, και έτσι ο μέσος αριθμός πέφτει (5,66), ενώ για $p < 1$ στις τυχαίες αναζητήσεις παίρνουμε αναμενόμενα ενδιάμεσες τιμές (5,78). Αντίστροφη είναι η πορεία του μέσου αριθμού ανάγνωσης σελίδων του Ευρετηρίου, καθώς αν δεν υπάρχει η λέξη στο B-tree δεν ψάχνουμε στο Ευρετήριο, ενώ στις επιτυχείς αναζητήσεις βλέπουμε ότι στην πλειοψηφία τους κάθε λέξη εμφανίζεται έως 10 φορές και δεσμεύει χώρο μίας σελίδας μόνο (λογικό αφού με μοναδικές λέξεις 1496 δημιουργήθηκαν 1752 σελίδες στο Posting File).

Τέλος, για τη δημιουργία του B-tree με εισαγωγή μίας-μίας λέξης χρειάστηκαν κατά μέσο όρο 11,90 προσβάσεις στο δίσκο, δηλαδή κοντά στο διπλάσιο από αυτό μιας αναζήτησης μέχρι τα φύλλα. Σε αυτές περιλαμβάνεται τόσο το κόστος αναζήτησης μίας λέξης που δεν υπάρχει ήδη (πάντα 6 προσβάσεις) και το κόστος της περαιτέρω εισαγωγής της. Από την άλλη πλευρά, αν προσθέσουμε στη μέτρηση και τις περιπτώσεις απόρριψης κλειδιών που υπάρχουν ήδη (διπλοεγγραφών), τότε βλέπουμε ότι το μέσο κόστος μειώνεται κατά πολύ (6,60), καθώς εκεί οι προσβάσεις περιορίζονται ξανά μόνο στο κόστος της αναζήτησης (≤ 6 προσβάσεις).

4 Πηγές εκτός μαθήματος

[1] T. H. Cormen, C. E. Leiserson, and C. E. Rivest. *Introduction to Algorithms, CHAPTER 19: B-TREES*. [Online; accessed 30-April-2017]. 1990. URL: http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap19.htm.