# *3η Άσκηση - B+Δέντρα*

*Βαγγέλης Αθανασάκης 2019030118*

Σκοπός της τρίτης εργαστηριακής άσκησης ήταν η εξοικείωση με τα B+ δέντρα και τις μεθόδους τους (εισαγωγή στοιχείων, αναζήτηση στοιχείων, διαγραφή στοιχείων και αναζήτηση εύρους στοιχείων) και η διερεύνηση της αποδοτικότητας κάθε μιας από αυτές τις μεθόδους στην περίπτωση που οι κόμβοι του δέντρου αλλά και τα δεδομένα βρίσκονται στον δίσκο.

- Η τάξη του δέντρου υπολογίστηκε 29 εφόσον κάθε DataPage πρέπει να είναι 256 bytes. Από τα 256 bytes τα 20 είναι δεσμευμένα από τα 2 παιδία, τον πατέρα, τον αριθμό των κλειδιών και από τον τύπο του κόμβου. Έτσι μένουν 236 bytes για τα παιδιά και τα κλειδιά στους εσωτερικούς κόμβους και για τα values και τα κλειδιά αντίστοιχα για τους κόμβους φύλλα. Έτσι υπολογίζεται ότι 236=4*(αριθμό κλειδιών)+4*(αριθμό κλειδιών+1)=> αριθμός κλειδιών= τάξη δέντρου = 29.
- Για την υλοποίηση του B+ δέντρου χρησιμοποιούνται δύο ειδών κόμβοι: Οι κόμβοι που είναι εσωτερικά του δέντρου οι οποίοι περιέχουν τα κλειδιά, δείκτες στα αδέρφια και τον πατέρα τους καθώς και δείκτες προς τα παιδία τους (Υλοποιούνται από την κλάση BTreeInnerNode) και κόμβοι που είναι στα φύλλα του δέντρου οι οποίοι περιέχουν και αυτοί δείκτες προς τα αδέλφια και τον πατέρα τους και values που δείχνουν για κάθε κλειδί σε ποιο σημείο του αρχείου των δεδομένων βρίσκεται η πληροφορία του (Υλοποιούνται από την κλάση BTreeLeafNode).
- Η κλάση που υλοποιεί το δέντρο είναι η <BTree> η οποία έχει μεθόδους εισαγωγής στοιχείου (Insert) η οποία αρχικά υπολογίζει το σημείο μέσα στο DataFile όπου θα γίνει η εισαγωγή της νέας τιμής και στην συνέχεια μέσω της μεθόδου <findLeafShouldContainKey> βρίσκει το φύλλο στο οποίο θα πρέπει να γίνει η εισαγωγή του νέου κλειδιού στο δέντρο. Αν αυτή η εισαγωγή προκαλέσει overflow σε κάποιον κόμβο τότε μέσω της μεθόδου <dealOverFlow> γίνεται split του κόμβου που υπερχειλίζει δημιουργώντας δύο νέους κόμβους, ένα για τον καινούργιο πατέρα και έναν για τον καινούργιο αδελφό.
- Μία άλλη μέθοδος είναι η <search> η οποία ψάχνει ένα κλειδί μέσα στο δέντρο. Αρχικά, μέσω της μεθόδου findLeafShouldContainKey βρίσκει το φύλλο του δέντρου στο οποίο θα πρέπει να βρίσκεται το κλειδί αν υπάρχει. Στην συνέχεια, μέσω της μεθόδου search της κλάσης BTreeLeafNode ψάχνει το κλειδί στο συγκεκριμένο φύλλο και αν το βρει επιστρέφει τα δεδομένα που αποθηκεύει αυτό το κλειδί.
- Τέλος, υπάρχει η μέθοδος delete η οποία σβήνει ένα κλειδί από το δέντρο. Αρχικά, βρίσκει μέσω της findLeafShouldContainKey το φύλλο που θα πρέπει

να υπάρχει το κλειδί. Αφού διαγράψει το κλειδί, ελέγχει αν το κλειδί είναι underflow. Αν είναι, ελέγχει αν μπορεί να δανειστεί κλειδί από κάποιον αδελφό του. Αν μπορεί τότε δανείζεται διαφορετικά μέσω της μεθόδου processChildFusion συγχωνεύεται με κάποιον αδερφό του.

- Τα δεδομένα υλοποιούνται από την κλάση <Data> η οποία έχει μεθόδους μετατροπής ενός Data Instance σε array από bytes και αντίστροφα έτσι ώστε να είναι δυνατή η αποθήκευσή τους σε αρχείο αλλά και το διάβασμα τους από αυτό.
- Η κλάση StorageCache χρησιμοποιείται για την ανάκτηση κόμβων και δεδομένων από την μνήμη και για την αποθήκευσή τους στα αρχεία. Η κλάση χρησιμοποιεί hash maps για να αποθηκεύει τους κόμβους και τα δεδομένα που δεν έχουν αποθηκευτεί ακόμη στα αρχεία ή εκείνα που είναι αποθηκευμένα στα αρχεία αλλά μετά την μεταφορά τους στην κεντρική μνήμη έχουν υποστεί επεξεργασία και πρέπει να αποθηκευτούν εκ νέου στον δίσκο. Αυτοί οι κόμβοι που χρειάζεται να αποθηκευτούν ξανά στον δίσκο «σημειώνονται» μέσω της μεταβλητής dirty.
- Πιο συγκεκριμένα, μέσω των μεθόδων cacheNode και cacheData τοποθετεί έναν κόμβο ή δεδομένα στο hash map και μέσω των μεθόδων getNodeFromCache-getDataFromCache τα ανακτάει από αυτό.
- Οι μέθοδοι flushNodes-flushData ελέγχουν αν κάποιος κόμβος η δεδομένα έχουν «σημειωθεί» από την μεταβλητή dirty. Αν έχουν σημειωθεί, τότε τους αποθηκεύουν ξανά στον δίσκο.
- Οι μέθοδοι retrieveNode-retrieveData ανακτούν κόμβους και δεδομένα από την Cache η τον δίσκο. Συγκεκριμένα, παίρνουν ως όρισμα την σελίδα του δίσκου που είναι αποθηκευμένος ο κόμβος και το offset στο DataFile των δεδομένων αντίστοιχα και αν οι κόμβοι – δεδομένα είναι στην cache τότε τους ανακτάει από εκεί διαφορετικά διαβάζει από το αντίστοιχο αρχείο για να κάνει την ανάκτηση.
- Τέλος, οι μέθοδοι BTreeInnerNode-BTreeLeafNode δεσμεύουν χώρο στο αρχείο για την αποθήκευση των κόμβων μέσω της aquireNodeStorage και επιστρέφουν Instances καινούργιων κόμβων και η μέθοδος newData αντίστοιχα βρίσκει το offset στο οποίο θα αποθηκευτεί ένα καινούργιο Data Instance.

## Ερμηνεία Αποτελεσμάτων

Μετά από μετρήσεις προσβάσεων στον δίσκο για κάθε λειτουργία του δέντρου προέκυψαν τα εξής αποτελέσματά:

| Μέσος Αριθμός προσβάσεων δίσκου ανά εισαγωγή | Μέσος Αριθμός προσβάσεων δίσκου ανά τυχαία αναζήτηση | Μέσος Αριθμός προσβάσεων δίσκου ανά διαγραφή |
|---|---|---|
| 8 | 1 | 11 |

- Παρατηρούμε ότι η πιο αποδοτική μέθοδος είναι αυτή της τυχαίας αναζήτησης καθώς αυτή πετυχαίνει λιγότερες προσβάσεις στον δίσκο ενώ η λιγότερο αποδοτική είναι αυτή της διαγραφής στοιχείου καθώς έχει τις περισσότερες προσβάσεις στον δίσκο.
- Το γεγονός ότι η εισαγωγή είναι πιο αποδοτική από την διαγραφή είναι λογικό καθώς η αναζήτηση είναι μέρος της διαδικασίας της διαγραφής (πρώτα βρίσκουμε το κλειδί και στην συνέχεια στο διαγράφουμε).
- Επίσης, είναι λογικό που η αποδοτικότητα της εισαγωγής είναι παρόμοια με της διαγραφής καθώς και οι δύο μέθοδοι μετά την εισαγωγή-διαγραφή αντίστοιχα χρειάζεται να εκτελέσουν διεργασίες που απαιτούν περίπου τις ίδιες προσβάσεις στον δίσκο πχ. Στις ακραίες περιπτώσεις όπως overflow-underflow όπου στην πρώτη περίπτωση πρέπει να γίνει split και να δημιουργηθούν 2 νέοι κόμβοι και στην δεύτερη πρέπει είτε να γίνει δανεισμός ή συγχώνευση.
- Ωστόσο, το B+ Tree είναι πολύ πιο αποδοτικό σε σχέση με άλλες δομές δεδομένων στην εκτύπωση εύρους καθώς τα κλειδιά στα φύλλα είναι τοποθετημένα με αύξουσα σειρά και αυτά συνδέονται μεταξύ τους με δείκτες οπότε η διάσχιση μπορεί να γίνει από το ένα κατευθείαν στο άλλο χωρίς να εμπλέκονται οι εσωτερικοί κόμβοι.