ΠΟΛΥΤΕΧΝΕΙΟ
ΚΡΗΤΗΣ

[ΠΛΗ_201] ΔΟΜΕΣ ΑΡΧΕΙΩΝ ΚΑΙ ΔΕΔΟΜΕΝΩΝ

2Η ΑΣΚΗΣΗ

Κατσίμπας Πέτρος

201630038

Διδάσκων: ***Ευριπίδης Πετράκης***

Υπεύθυνοι Εργαστηρίου: ***Στέφανος Καρασαββίδης ,Σπύρος Αργυρόπουλος***

B+ - TREE

Στην 3η εργαστηριακή άσκηση μας ζητήθηκε να υλοποιήσουμε πάνω στον κώδικα B+_Tree_Master από τον συντάκτη ***github.com/skskskskk*** με σκοπό να καταγράφουμε το δένδρο στον δίσκο περνώντας τους κόμβους σε ένα αρχείο ***NodeFile.bin*** , ενώ είχαμε και ένα έξτρα αρχείο ***DataFile.bin*** για την αποθήκευση της πληροφορίας που συνοδεύει κάθε κλειδί σε *LeafNode* με το *DataByteOffset. Συγκεκριμένα, οι αλλαγές που τροποποιήσαμε αναλύονται παρακάτω.*

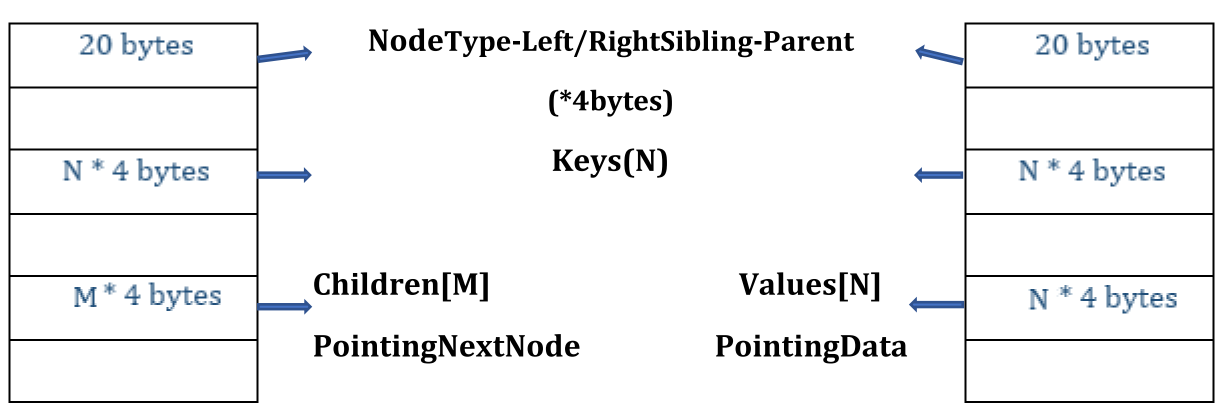
ΔΗΜΙΟΥΡΓΟΝΤΑΣ ΤΟ BPLUSTREE_PLH201_FILE_STORE

Για την επίτευξη του Project, αρχικά, συμπληρώσαμε την ***StorageCache.class***. Η λειτουργία της αφορούσε μία προσωρινή μνήμη αποθήκευσης των κόμβων καθώς και την επικοινωνία του δένδρου μας με το δίσκο/αρχείο. Οι σημαντικότερες συναρτήσεις πέρα από τις ***CacheNode()/CacheData()*** , είναι οι ***RetrieveNode() /RetrieveData()*** και οι ***FlushNode() /FlushData().***

RetrieveNode/RetrieveData

Η λειτουργία των συναρτήσεων αυτών ήταν η ανάκτηση των κόμβων στον δίσκο. Για να γίνει αυτό, χρειαζόμαστε το DataPageIndex του κόμβου, το οποίο είχαμε εύκολα διαθέσιμο είτε από τον πίνακα των Children[] είτε από τα node.Siblings. Έτσι, μπορούσαμε από τον υπολογισμό DataPageIndex * DataPageSize(δηλαδή 256 όπου τα bytes κάθε σελίδας) να εισερχόμαστε στον κατάλληλο FilePointer και να διαβάζουμε ένα ArrayByte των 256 bytes. Ερχόμαστε, λοιπόν, στη νέα συνάρτηση FromByteArray() , η οποία μετατρέπει τον «διαβασμένο» πίνακα σε Node και είναι υλοποιημένη στην BTreeLeafNode.class και ελαφρώς διαφορετικά υλοποιημένη στην BTreeInnerNode.class όπως φαίνεται παρακάτω.

BTreeInnerNode.class **BTreeLeafNode.class**



Η λειτουργία της RetrieveData() είναι η ίδια με τη διαφορά ότι είναι πιο απλή καθώς περιέχει μόνο Integers.

FlushNode/FlushData

Η λειτουργία Flush που καλεί τις δύο παραπάνω συναρτήσεις απευθύνεται στα DirtyNodes(), δηλαδή τους κόμβους στους οποίους έχει γίνει κάποια αλλαγή είτε από εισαγωγή (πιθανό *Overflow()*) είτε από διαγραφή. Πάλι με τη βοήθεια του ***DataPageIndex*** ανακτούμε το σωστό ***FilePointer*** στο αρχείο των κόμβων και ενημερώνουμε- γράφουμε το αρχείο στο δίσκο. Αντίστροφα της ***FromByteArray()*** έχουμε την ***ToByteArray(),*** η οποία με ανάλογη λογική δημιουργεί ένα *ByteArray[]* σταθερού μεγέθους 256 bytes με αποθηκευμένα τα στοιχεία του κόμβου όπως φαίνεται παραπάνω.

Τέλος, αξίζει να αναφέρουμε την ***AcquireNextNode()*** , η οποία καλείται σε κάθε δημιουργία ***NewInnerNode()*** ή ***NewLeafNode()***. Εκτός ότι δημιουργεί ένα καινούργιο κόμβο στην δυναμική μνήμη βρίσκει βάσει του ***NodeFile.length()*** , το νέο ***DataPageIndex*** ώστε να αποθηκεύσουμε σωστά τον κόμβο στο αρχείο όταν τελειώσουμε με την επεξεργασία του.

ΑΠΟΔΟΣΗ Β+ ΔΕΝΔΡΟΥ

Για να μετρήσουμε την απόδοση κάναμε μία αρχικοποίηση του δένδρου με 10^5 κλειδιά όπως ζητούσε η εκφώνηση (όπου 1 < Key < 10^6). Να σημειώσουμε επίσης ότι κάθε DataPage του κόμβου, όπως προ-είπαμε, είναι 256 bytes , με αποτέλεσμα να έχουμε διαθέσιμες 29 θέσεις κλειδιών και 29 θέσεις τιμών StoreByteOffset για τα LeafNodes και αντίστοιχα 29 θέσεις κλειδιών και 30 θέσεις παιδιών για τα InnerNodes. Υποθέτουμε ότι έχουμε 69% πληρότητα κατά μέσο όρο σε κάθε κόμβο και έτσι δημιουργούμε τον εξής πίνακα.

Order Of B+ Tree: 29

69% Occupancy (Average)

Average Number of Keys = 0.69 x 29 = 20

Number of Children = 20 + 1 = 21

20 keys/21 Children per Node

	Nodes	Keys	Children
Root	**1**	**20**	**21**
Level 1:	**21**	**21*20=420**	**21*21=441**
Level 2:	**441**	**8.820**	**9.261**
Level 3/(Leaf):	**9261**	**185.220**	

Πράγματι, το project δημιουργεί αρχείοSize on disk: 1.27 MB (1,335,296 bytes) , δηλαδή αποτελείται από 5.216 κόμβους, άρα για 10^5 εισαγωγές έχουμε ύψος δένδρου ***height = 4.***

Για να μετρήσουμε, λοιπόν, την απόδοση του δένδρου έρευνας χρησιμοποιήσαμε 20 εισαγωγές, 20 αναζητήσεις, 20 αναζητήσεις με εύρος τιμών για K = 10 και K = 1000 καθώς και 20 διαγραφές. Οι μετρήσεις είναι οι εξής:

Μέσος αριθμός προσβάσεων δίσκου ανά εισαγωγή	Μέσος αριθμός προσβάσεων δίσκου ανά τυχαία αναζήτηση	Μέσος αριθμός προσβάσεων δίσκου ανά διαγραφή	Μέσος αριθμός προσβάσεων δίσκου για διάστημα K τιμών (K=10)	Μέσος αριθμός προσβάσεων δίσκου για διάστημα K τιμών (K=1000)
8	**4**	**7**	**4**	**5**

Αναλύοντας τα αποτελέσματα, το B+ δένδρο είναι, όπως και το ταξινομημένο πεδίο, ένα πολύ αργό δένδρο όσο αναφορά την δημιουργία του (**(!!!)** *μπορεί να χρειαστεί και 5 λεπτά για την δημιουργία του*). Ωστόσο, οι μετρήσεις-προσβάσεις στον δίσκο είναι πάρα πολύ χαμηλές και γρήγορες. Αυτό συμβαίνει, καθώς, για να φτάσει στον σωστό κόμβο(Leaf) αρκούν μόνο 4 προσβάσεις δίσκου. Από κει και πέρα, στην εισαγωγή θα χρειαστεί +2 προσβάσεις δίσκου για την ενημέρωση αρχείου NodeFile και DataFile και ίσως και +2*i όπου i κάποιο πιθανό Overflow (+1 για τη δημιουργία και +1 για την ενημέρωση του ParentNode). Για την αναζήτηση μας αρκεί να φτάσουμε σε κόμβο Leaf και να εξετάσουμε και τα 29(maximum capacity) κλειδιά. Στο παρόν κόμβο είτε θα έχουμε επιτυχή αναζήτηση αλλιώς αποκλείεται να βρίσκεται το αναζητήσιμο κλειδί σε άλλον κόμβο. Παρόμοιας λογικής είναι και η αναζήτηση με εύρος αφού η μόνη διαφορά βρίσκεται στο γεγονός της αναζήτησης και του high value όπου high value = randomValue + K (η υλοποίηση βρίσκεται μέσα στη Custom Search Range), κάτι το οποίο μπορεί να χρειαστεί ανάκτηση κόμβου από το δίσκο (RetrieveNode), συνεπώς , κάποια/-ες έξτρα προσβάσεις του δίσκου. Αναμενόμενο, λοιπόν, όσο μεγαλώνει το K να προχωράμε προς το RightSilbing του κάθε LeafNode. Τέλος, έχουμε τις μετρήσεις της διαγραφής, οι οποίες είναι λογικό να

πλησιάζουν τις προσβάσεις της αναζήτησης στο δίσκο αφού η λειτουργίας ανάκτησης-ενημέρωσης συμπίπτουν. (Η διαγραφή δυσλειτουργεί στη συνάρτηση της FlushData καθώς δεν ενημερώνει σωστά το RetrievedData.KeySet. Για αυτό έχω προσθέσει ένα τυπικό +2 στον counter προσβάσεων για το ενημερωμένου κόμβου και δεδομένων με την προϋπόθεση ότι δεν έχει δημιουργηθεί κάποιο Merge-Fusion κατά την διαγραφή).

ΕΥΧΑΡΙΣΤΩ ΓΙΑ ΤΟΝ ΧΡΟΝΟ ΣΑΣ!!!

(ΥΓ. Σε περίπτωση εκτέλεσης του προγράμματος- αν και έχω συνάρτηση διαγραφής των αρχείων – καλό θα ήταν να τα διαγράψετε χειροκίνητα. Διαφορετικά θα ξεκινήσει από λάθος Offset η αρχικοποίηση)