ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Άσκηση 2

1115201500001	Αθηναίος Κωνσταντίνος
1115201500013	Ασλανίδης Θεοφάνης

Η εργασία μας ανταποκρίνεται στις απαιτήσεις της άσκησης και δοκιμάστηκε και με τα τρία δοσμένα παραδείγματα. Στα ήδη υπάρχοντα αρχεία προσθέσαμε το αρχείο κεφαλίδας BplusTree.h και το πηγαίο BplusTree.c, όπου υπάρχουν δηλωμένες και υλοποιημένες βοηθητικές συναρτήσεις για την AM_InsertEntry. Για την καλύτερη δυνατή κατανόηση της λειτουργικότητας του κώδικά μας χρησιμοποιούμε σχόλια όπου αυτό κρίνεται απαραίτητο. Οι βασικές παραδοχές που έχουμε κάνει είναι οι παρακάτω:

1. Χρησιμοποιούμε το πρώτο μπλοκ κάθε αρχείου ως αναγνωριστικό, στο οποίο αποθηκεύουμε το μέγεθος των τεσσάρων μεταβλητών (attrType1, attrLength1, attrType2, attrLength2), το μέγεθος και τον τύπο του πρώτου και του δεύτερου πεδίου και του ρίζα του B+ δέντρου.
2. Το πρώτο byte σε κάθε μπλοκ ευρετηρίου έχει την τιμή «b» και το δεύτερο byte έχει τον αριθμό των κλειδιών μέσα στο μπλοκ. Έπειτα αποθηκεύεται μια μεταβλητή τύπου int που περιέχει το id του μπλοκ αυτού.
3. Το πρώτο byte σε κάθε μπλοκ δεδομένων έχει την τιμή «d» και το δεύτερο byte έχει τον αριθμό των εγγράφων μέσα στο μπλοκ. Έπειτα αποθηκεύονται δυο μεταβλητές τύπου int, όπου η πρώτη περιέχει το id του μπλοκ αυτού και η δεύτερη περιέχει το id του επόμενου μπλοκ δεδομένων. Αν δεν υπάρχει επόμενο μπλοκ δεδομένων, έχει την τιμή -1.

Παρακάτω θα δώσουμε μια συνοπτική περιγραφή για τον τρόπο λειτουργίας κάθε συνάρτησης που κληθήκαμε να υλοποιήσουμε:

AM_Init()

Στην συνάρτηση αυτή αρχικοποιούμε τους πίνακες δεικτών σε δομές για ανοιχτά αρχεία και ανοιχτές αναζητήσεις σε NULL. Ο πίνακας OPENFILES περιέχει δείκτες στην δομή AM_Index, ενώ ο πίνακας SCANFILES περιέχει δείκτες στην δομή AM_Scan.

AM_CreateIndex()

Στην συνάρτηση αυτή δημιουργούμε ένα καινούργιο αρχείο και μέσα σε αυτό δυο μπλοκ. Το πρώτο μπλοκ είναι το αναγνωριστικό μπλοκ, στο οποίο περνάμε μέσα όλες τις πληροφορίες που μας είναι απαραίτητες για την εισαγωγή και αναζήτηση δεδομένων από το αρχείο σύμφωνα με την δομή των B+ δέντρων, όπως αυτό έχει περιγραφεί στην Παραδοχή 1. Το δεύτερο μπλοκ είναι ένα κενό μπλοκ δεδομένων στο οποίο πρόκειται να γίνει η πρώτη εισαγωγή.

AM_DestroyIndex()

Στην συνάρτηση αυτή γίνεται η καταστροφή του αρχείου που ορίζει το όνομα ως παράμετρος, αφού πρώτα ελεγχθεί ότι δεν είναι ανοιχτό. Αν δεν πετύχει η καταστροφή επιστρέφεται κωδικός λάθους.

AM_OpenIndex()

Στην συνάρτηση αυτή ανοίγουμε το αρχείο που ορίζει το όνομα ως παράμετρος και δημιουργούμε μια καινούργια δομή στον πίνακα OPENFILES για αυτό. Στην δομή αυτή εισάγουμε τα δεδομένα που βρίσκονται στο αναγνωριστικό μπλοκ του αρχείου, ώστε να μπορούμε να τα χρησιμοποιήσουμε στην συνέχεια.

AM_CloseIndex()

Στην συνάρτηση αυτή κλείνουμε το αρχείο που ορίζει το όνομα ως παράμετρος, εφόσον βεβαιωθούμε ότι δεν είναι ήδη κλειστό ή δεν υπάρχει κάποια ανοιχτή σάρωση σε αυτό. Σε μια τέτοια περίπτωση εκτυπώνεται ο αντίστοιχος κωδικός λάθους. Πριν το κλείσιμο του αρχείου περνάμε στο αναγνωριστικό μπλοκ του αρχείου την πιθανή καινούργια ρίζα του B+ δέντρου και καταστρέφουμε την δομή στον πίνακα OPENFILES ώστε να ελευθερωθεί η θέση.

AM_InsertEntry()

Σε αυτή τη συνάρτηση τοποθετούμε τις εγγραφές που δίνονται μέσα στο αντίστοιχο B+ Δένδρο. Ο αλγόριθμος που χρησιμοποιούμε είναι αυτός που μας έχει δώσει ο κ. Ιωαννίδης. Πρώτα, κάνουμε περιήγηση μέχρι το επίπεδο δεδομένων στο κατάλληλο μπλοκ, βλέπουμε αν έχει χώρο. Αν έχει χώρο, τοποθετούμε την εγγραφή στο σωστό σημείο για να είναι όλα ταξινομημένα. Αν δεν έχει χώρο, σπάμε το μπλοκ και ισομοιράζουμε τα δεδομένα στο υπάρχον και στο καινούργιο, είναι περιττός ο αριθμός, αλλιώς περνάει μία εγγραφή παραπάνω στο δεξιά. Σε κάθε περίπτωση, τα 2 αυτά μπλοκ μετά τον ισομοιρασμό είναι μισογεμάτα. Αφού γίνει το σπάσιμο του μπλοκ δεδομένων , πρέπει να τοποθετήσουμε στο από πάνω μπλοκ ευρετηρίου το μεσαίο κλειδί, δηλαδή το πρώτο κλειδί του καινούργιου/δεξιά μπλοκ. Εν συνεχεία, αν αυτό το κλειδί χωράει στο από πάνω μπλοκ το τοποθετούμε στο κατάλληλο σημείο για να είναι ταξινομημένα, αλλιώς σπάμε αναδρομικά τα μπλοκ ευρετηρίου προς τα επάνω μέχρι να βρούμε ένα μπλοκ ευρετηρίου που χωράει το κλειδί, ή να φτάσουμε στη ρίζα, που αν δεν χωράει στη ρίζα, σπάμε τη ρίζα και ορίζουμε ως ρίζα το νέο μπλοκ που θα φτιάξουμε με το μεσαίο της κλειδί.

Αναλυτικά, για να πετύχουμε την παραπάνω μεθοδολογία για την insert, φτιάξαμε μερικές δικές μου συναρτήσεις για να μας διευκολύνουν.

- Split_leaf()
- Split_interior()
- AM_Traverse()
- AM_Find_Block()
- Insert_Key()

Αυτές οι 5 συναρτήσεις σε συνδυασμό με την AM_InsertEntry οδηγούν στο επιθυμητό αποτέλεσμα. Τα ορίσματα block1, block2, block3 που χρησιμοποιούμε παρακάτω είναι ονομασμένα έτσι λόγω του split, όπου το block1 θα καταλήξει:

Split_leaf(int I, int block1_id, int block3_id)

Αυτή η συνάρτηση είναι που θα σπάσει το πλέον γεμάτο data block, σε 2 ισομοιρασμένα. Τα ορίσματα είναι int, με τα id των block που θα κάνω τον ισομοιρασμό (το allocate του καινούργιου μπλοκ έχει γίνει από την insert entry) . Οπότε μέσα σε αυτή κάνουμε BF_GetBlock(), ώστε στο τέλος της να κάνουμε Set Dirty και Unpin και να μην χάνονται οι δείκτες με τις αναδρομές και να κρατάμε τα unpin μας οργανωμένα. Τις μεταφορές και αντιγραφές δεδομένων τις κάνουμε με memcpy όταν περνάμε πολλές εγγραφές ή γενικά τα byte > 1, και memset όταν είναι 1 byte η πληροφορία που θα περάσουμε. Για την μεσαία εγγραφή κάνουμε num of records div 2, και όλα τα αριστερά της μένουν εκεί που είναι, τα δεξιά της μετακινούνται στο καινούργιο μπλοκ. Τέλος αλλάζουμε το byte με την πληροφορία του πόσες εγγραφές υπάρχουν μέσα στο μπλοκ.

Split_Interior(int I,int block1_id,int block3_id)

Αυτή είναι μια συνάρτηση παρόμοια με την split_leaf, αλλά σπάει μπλοκ ευρετηρίου, σε αντίθεση με την split leaf που σπάει μπλοκ δεδομένων. Λειτουργεί με τον ίδιο τρόπο, μόνο που το αρχικό μπλοκ της μορφής (δείκτης){κλειδί}(δείκτης){κλειδί}(δείκτης){κλειδί}…..(δείκτης) σπάει ανομοιόμορφα. Δηλαδή, στο αριστερό μπλοκ, στο τέλος αφήνουμε μόνο {κλειδί} το οποίο θα το πάρει η insert key για να το βάλει πάνω. Το βάζουμε από την αριστερή μεριά, ώστε η μετακίνηση του αυτή πάνω να είναι εύκολη, γιατί αν ήταν πρώτο από δεξιά θα έπρεπε να κάνουμε ολίσθηση τα δεδομένα προς τα δεξιά μετά τη διαγραφή του από το μπλοκ. Οπότε η τοποθέτηση της μεσαίας τιμής στο τέλος του αριστερού μπλοκ, χωρίς κάποιον δείκτη στα δεξιότερα μας διευκολύνει πάρα πολύ στην τοποθέτηση του κλειδιού στο παραπάνω μπλοκ.

Insert_key(int I, int blockL_id, int blockR_id)

Αυτή η συνάρτηση, καλείται μετά την split_leaf ή split_interior, είναι αυτή που θα πάρει το αντίστοιχο-μεσαίο κλειδί και θα το τοποθετήσει στο από πάνω μπλοκ ευρετηρίου. Τα ορίσματα της επίσης είναι int, ώστε να κάνουμε getblock μέσα στη συνάρτηση και να είναι όλα τα unpin μου οργανωμένα. Ξεδιαλύνεται σε κάποια case. Αναλυτικά, το πρώτο case είναι, τα blockL μας να είναι η ρίζα, οπότε, αυτό σημαίνει ότι η παλιά ρίζα έσπασε, και το καινούργιο μπλοκ που θα δημιουργηθεί, θα είναι η νέα μας ρίζα, γι’ αυτό κάνουμε και την αντίστοιχη ανάθεση στο struct μας που κρατάει την πληροφορία της ρίζας. Σε αυτήν την περίπτωση όπως και στην περίπτωση το blockL να είναι οποιοδήποτε μπλοκ ευρετηρίου, το μεσαίο κλειδί που θα ανέβει πάνω όπως αναφέραμε στην split interior είναι το τελευταίο από το blockL. Αν το blockL είναι μπλοκ δεδομένων(δηλαδή και το blockR αφού έχουν το ίδιο βάθος), τότε παίρνουμε ως κλειδί να ανεβάσουμε πάνω, το πρώτο κλειδί από το δεξί μπλοκ blockR. Αν το κλειδί που θα πάμε να ανεβάσουμε πάνω στο UpperBlock δεν χωράει μαζί με το δείκτη, το UpperBlock θα κάνει split interior και θα ξανακληθεί αναδρομικά γι’αυτό η insert_key, μέχρι το κλειδί να χωρέσει κάπου. Στην τοποθέτηση του κλειδιού στο UpperBlock γίνονται συγκρίσεις, και μετά ολίσθηση δεδομένων ώστε να μπει ταξινομημένο. Για την σύγκριση αυτή, όπως και για κάθε άλλη σύγκριση χρησιμοποιούμε memcmp(). Αυτή η συνάρτηση για τα float δεν δουλεύει σωστά, οπότε αναγκαστήκαμε να κάνουμε cast, τα void* σε float* και μετά να περάσουμε την τιμή σε μεταβλητή για να κάνουμε τη σύγκριση. Τέλος, αυξάνει τον αριθμό κλειδιών στο μπλοκ ευρετηρίου, και αν το blockL είναι μπλοκ ευρετηρίου εξίσου, θα του μειώσει τα κλειδιά κατά ένα.

AM_Traverse(int I,int block1, void* value1)

Σε αυτή τη συνάρτηση περιηγούμαστε στο δέντρο με βάση το κλειδί value1, ξεκινώντας κάθε φορά(επειδή είναι αναδρομική) από το block1, για να φτάσουμε στο μπλοκ δεδομένων που πρέπει να τοποθετηθεί και επιστρέφω το blockid του. Επίσης δεν δουλεύουμε με δείκτες μπλοκ σαν όρισμα για να έχω οργανωμένα τα block init και unpin.

AM_Find_Block(int I ,int root, int block_id, int *Destination)

Σε αυτή τη συνάρτηση ψάχνουμε για εκείνο το μπλοκ που έχει μέσα δείκτη(έναν int δηλαδή με το αναγνωριστικό) στο block_id. Αυτή τη συνάρτηση τη χρησιμοποιούμε για να ανέβουμε επίπεδο(βάθος) στην Insert Key όπου θέλουμε να βάλουμε το κλειδί πάνω, αλλά δεν ξέρουμε σε ποιο μπλοκ. Δουλεύει αναδρομικά και τρέχει ένα ένα τους "δείκτες", αν βρει το block_id σταματάει και γυρνάει, αλλιώς καλεί πάλι am_find_block γι' αυτόν τον id. Το αποτέλεσμα το γυρνάει στο *Destination, όπου από εκεί θα χρησιμοποιηθεί από την insert key.

AM_OpenIndexScan()

Στην συνάρτηση αυτή δημιουργούμε μια καινούργια δομή στον πίνακα SCANFILES, ώστε να εκτελεστεί μια σάρωση πάνω στο ανοιχτό αρχείο fileDesc με τελεστή σύγκρισης op και μεταβλητή σύγκρισης value. Αν κάποια από αυτές τις παραμέτρους έχει κάποιο λάθος εκτυπώνεται το αντίστοιχο μήνυμα. Αν όλα είναι σωστά, η δομή αρχικοποιείται και περνάνε τιμές στα πεδία της, ώστε να ξεκινήσει η σάρωση του αρχείου.

AM_FindNextEntry()

Στην συνάρτηση αυτή αναζητούμε την επόμενη εγγραφή μέσα στο αρχείο που ικανοποιεί τον τελεστή σύγκρισης με την δοσμένη τιμή για σύγκριση. Αρχικά δημιουργούμε ένα μπλοκ στο οποίο κρατάμε τα δεδομένα του τελευταίου μπλοκ που σαρώσαμε και δημιουργούμε και το ανάλογο offset, ώστε να φτάσουμε στην αμέσως επόμενη εγγραφή από την τελευταία που ελέγξαμε. Έπειτα, με βάση τον κατάλληλο τελεστή σύγκρισης μπαίνουμε και στο ανάλογο case του switch. Σε όλες τις περιπτώσεις σύγκρισης, υπάρχει και μια μέθοδος ώστε να κατέβουμε από την ρίζα του B+ δέντρου στο επίπεδο των μπλοκ δεδομένων. Στις περιπτώσεις των EQUAL, GREATER_THAN και GREATER_THAN_OR_EQUAL φροντίζουμε να κατέβουμε στο αντίστοιχο μπλοκ που περιέχει την πρώτη εγγραφή που θέλουμε να επιστρέψουμε. Στις περιπτώσεις των NOT_EQUAL, LESS_THAN και LESS_THAN_OR_EQUAL κατεβαίνουμε στο πρώτο μπλοκ δεδομένων από αριστερά. Αφού φτάσουμε στο επίπεδο των μπλοκ δεδομένων, ο κώδικας σε όλα τα case είναι σχεδόν πανομοιότυπος με ελάχιστες διαφοροποιήσεις που απαιτούνται λόγω του διαφορετικού τελεστή σύγκρισης. Ξεκινάμε ελέγχοντας τον τύπο του κλειδιού και κάνοντας το κατάλληλο cast ώστε να μπορούμε να συγκρίνουμε τα δεδομένα μας με την value. Αν ο έλεγχος είναι επιτυχής, φροντίζουμε ώστε να επιστραφεί η αντίστοιχη τιμή, αλλιώς προχωράμε προοδευτικά αλλάζοντας το offset μέχρις ότου να μην υπάρχει επόμενο μπλοκ δεδομένων.

AM_CloseIndexScan()

Στην συνάρτηση αυτή κλείνουμε την σάρωση που ορίζει η παράμετρος, εφόσον βεβαιωθούμε ότι είναι πράγματι ανοιχτή. Έπειτα καταστρέφουμε την δομή και ελευθερώνουμε την θέση του πίνακα SCANFILES.

AM_Close()

Στην συνάρτηση αυτή απλά καλούμε την BF_Close().

Κλείνοντας, θέλουμε να αναφέρουμε ότι εντοπίσαμε ένα λάθος στις εκτυπώσεις του πρώτου δείγματος am_main1, όπου στο δεύτερο υποερώτημα χάνονται 7 εγγραφές που βρίσκονται όλες στο ίδιο μπλοκ. Δοκιμάσαμε να μην κλείσουμε το αρχείο και είδαμε ότι οι εκτυπώσεις έβγαιναν σωστές. Ελέγξαμε τα Set_Dirty και Unpin σε όλα τα μπλοκ της εισαγωγής μας και δεν βρήκαμε κάποιο λάθος. Μετά από πολλούς ελέγχους μπορούμε απλά να υποθέσουμε ότι αυτό ίσως να οφείλεται στο γεγονός ότι το κλειδί σύγκρισης είναι τύπου float, καθώς στα αρχεία με κλειδί τύπου char ή int δεν αντιμετωπίσαμε παρόμοιο πρόβλημα.