Υλοποίηση Συστημάτων Βάσεων Δεδομένων
Δεύτερη Εργασία

Γιάννης Δαλιάνης,1115201700027
Μιχάλης Τατάς, 1115201700161
Μιχάλης Φωτιάδης, 1115201700183

Στο αρχείο **hash_file.c** δηλώνεται ως **BUCKETS_PER_BLOCK** το πλήθος των bucket που χωράνε σε κάθε block ευρετηρίου. Δίνεται ως 127 επειδή το block δεσμεύει χώρο 512 bytes = 128*sizeof(int) και το κάθε bucket είναι ένας ακέραιος που μας οδηγεί στο πρώτο block εγγραφής που περιέχει εγγραφές με συγκεκριμένη αξία hash. Το πρώτο bucket από τα 128 κρατάει τον αριθμό του επόμενου block ευρετηρίου, αν υπάρχει.

Η συνάρτηση **hashFunctions** δέχεται ως παραμέτρους τον αριθμό του αρχείου και το id μιας εγγραφής. Παίρνει από το πρώτο μπλοκ του αρχείου τον αριθμό των buckets και επιστρέφει την hashing τιμή που προκύπτει από το id σύμφωνα με τη συνήθη πράξη **(id mod buckets).**

Το πρώτο block κάθε αρχείου κατακερματισμού περιλαμβάνει ως ειδική πληροφορία το string “**HT**”, το οποίο δηλώνει την ταυτότητα του αρχείου. Το αρχείο αρχικοποιείται με την **HT_CreateIndex**, όπου δεσμεύεται τόσο το πρώτο block, δηλαδή το block πληροφορίας όσο και τα block ευρετηρίου. Η δομή του block πληροφορίας είναι η εξής:

HT	Πλήθος εγγραφών αρχείου	Πλήθος κουβάδων αρχείου	Κενός χώρος

Τα block που δεσμεύονται αμέσως μετά είναι τα block ευρετηρίου. Η δομή ενός block ευρετηρίου είναι:

Αριθμός επόμενου block ευρετηρίου	2	3	4	. . .	. . .

Ουσιαστικά από το δύο και μετά κάθε αριθμός αντιστοιχεί σε ένα bucket. Η αρίθμηση γίνεται στο διάστημα [2, buckets+2]. Θεωρούμε ότι το 1 αντιστοιχεί στο block πληροφορίας. Για κάθε bucket αρχικοποιώ και ένα block, το οποίο θα είναι το πρώτο block εγγραφών για τη συγκεκριμένη τιμή hash.

Στη συνάρτηση **HT_InsertEntry** δίνεται ένα record. Σε τμήμα κώδικα που έχει σχολιαστεί, γίνεται ένας έλεγχος για την περίπτωση που το ευρετήριο χρειάζεται να αλλαχτεί με rehash. Η rehash δεν έχει υλοποιηθεί για την εργασία.
Λόγω της εισαγωγής record, ο αριθμός των records στο block πληροφορίας αυξάνεται κατά ένα. Από το ευρετήριο και χρησιμοποιώντας τη hashFunctions παίρνω τον αριθμό του block στο οποίο πρέπει να εισαχθεί η νέα εγγραφή. Σε κάθε block εγγραφής χωράνε μέχρι 8 εγγραφές. Δομή μπλοκ εγγραφής:

Εγγραφές block	Επόμενο block	Εγγραφή 1	Εγγραφή 2	. . .	Εγγραφή 8

Όταν κάνω εισαγωγή εγγραφής, διατρέχω την αλυσίδα των block για τη συγκεκριμένη αξία hash, μέχρι να βρω ένα block το οποίο στη θέση επόμενου block έχει το 0, κάτι που σημαίνει ότι είναι το τελευταίο block της αλυσίδας. Σε περίπτωση που χωράει στο block η εγγραφή, γράφεται μετά την τελευταία υπάρχουσα και αυξάνεται κατά 1 το πλήθος εγγραφών του block. Σε διαφορετική περίπτωση, δεσμεύεται ένα νέο block και στο αμέσως προηγούμενο block τοποθετώ τον αριθμό του καινούριου block που δεσμεύτηκε ως επόμενο block. Βάζω στο καινούριο block ως πλήθος εγγραφών το 1 και εισάγω την εγγραφή.

Η συνάρτηση **HT_PrintAllEntries,** αν έχει δοθεί ως id το NULL εκτυπώνει όλες τις εγγραφές του αρχείου καλώντας επαναληπτικά για κάθε bucket του ευρετηρίου τη συνάρτηση **HT_PrintBlockChain,** η οποία δέχεται τον αριθμό id του αρχείου, τον αριθμό του block εγγραφών και το id της εγγραφής που θέλω να εκτυπώσω ή το NULL για να εκτυπώσω όλες τις εγγραφές. Η **HT_PrintBlockChain** είναι μια αναδρομική συνάρτηση που εκτυπώνει τις εγγραφές ξεκινώντας από αυτές του τελευταίου block εγγραφών της αλυσίδας πηγαίνοντας προς το πρώτο κάνοντας χρήση της **HT_PrintRecord** η οποία εκτυπώνει το περιεχόμενο κάθε record με ένα συγκεκριμένο τρόπο.
Αν στην **HT_PrintAllEntries** δοθεί κάποιο id, βρίσκει με με τη hashFunctions το block στο οποίο θα έπρεπε να βρίσκεται και αν υπάρχει, η HT_Print_BlockChain εκτυπώνει την εγγραφή με αυτό το id.

Στη συνάρτηση **HT_DeleteEntry** δίνεται ως όρισμα το id της εγγραφής που θέλουμε να διαγράψουμε. Βρίσκει το μπλοκ του αρχείου στο οποίο θα πρέπει να υπάρχει η εγγραφή. Βρίσκει το τελευταίο block του συγκεκριμένου blockchain εγγραφών και κρατάει το τελευταίο του record. Αν αυτό είναι το record που θέλουμε να διαγράψουμε, τότε το αντικαθιστούμε όλο με μηδενικά και μειώνουμε στο block του το πλήθος εγγραφών που κρατάει. Αν δεν είναι το τελευταίο block αυτό που θέλουμε να διαγράψουμε, τότε ψάχνουμε σε όλο το blockchain την εγγραφή που αναζητούμε. Αν τη βρούμε, την αντικαθιστούμε με μηδενικά. Στη συνέχεια, διατρέχουμε το block στο οποίο βρέθηκε η εγγραφή και βάζουμε στη θέση της την τελευταία εγγραφή του blockchain. Αυτή την αντικαθιστούμε με μηδενικά και μειώνουμε τον αριθμό εγγραφών στο block της.
Προφανώς αν στο αρχείο δεν υπάρχει εγγραφή με το συγκεκριμένο id, τότε δε συμβαίνει τίποτα.

Υπάρχουν 3 main για να δοκιμαστεί η λειτουργία του προγράμματος. Η ht_main.c δόθηκε εξ’ αρχής και είναι ίδια.

Στην ht_main2.c έχει προστεθεί η **MAX_OPEN_FILES** με τιμή 20. Δίνεται ως όρισμα γραμμής εντολών το πλήθος αρχείων ΗΤ που θέλουμε να ανοίξουμε και να επεξεργαστούμε. Δεσμεύεται ένας πίνακας δεικτών σε δομή που κρατάει για κάθε αρχείο τον αριθμό του, το όνομά του και ένα δείκτη σε block. Με μια επανάληψη δημιουργούμε και εισάγουμε εγγραφές μέσα σε κάθε αρχείο, στη συνέχεια εκτυπώνουμε τα περιεχόμενα κάθε αρχείου και κάνουμε απόπειρα διαγραφής από κάθε αρχείο εγγραφής με ίδιο κάθε φορά id, που προφανώς υπάρχει μόνο σε ένα από τα αρχεία. Ο δείκτης σε block της δομής δε χρησιμοποιείται στην υλοποίηση. Αν ήταν άλλη η μορφή των αρχείων ΗΤ, αυτός ο δείκτης θα μπορούσε να έχει πρόσβαση στο block πληροφορίας ενός αρχείου, αν στο αρχείο περιέχονται μόνο block ευρετηρίου και block εγγραφών.

Η ht_main3.c χρησιμοποιεί έναν πίνακα όπως η ht_main2.c, μόνο που η κάθε του θέση κρατάει μία δομή για το ίδιο κάθε φορά αρχείο. Ουσιαστικά η συνάρτηση αυτή εκτελεί λειτουργίες στο ίδιο αρχείο το οποίο είναι πολλές φορές ανοιχτό.

Στη Makefile υπάρχουν εντολές για compile της κάθε main.

Εκτέλεση με: ./build/runner

Ορίσματα χρειάζεται η ht_main2 και η ht_main3.

Μετά από εκτέλεση κάθε main πρέπει να διαγραφούν τα αρχεία data με make clean.