ΓΕΝΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΓΙΑ ΤΗΝ ΕΡΓΑΣΙΑ

Εκτέλεση προγράμματος

Στο makefile έχουν προστεθεί κομμάτια για την εκτέλεση valgrind και clean.

Με make run εκτελείται η ht_main που έχει δοθεί.

Με make runsht εκτελείται η sht_main

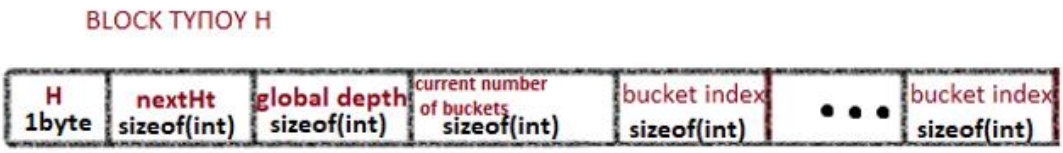
Με make myMain και μετά ./build/runner εκτελείται η my_main.c. Σκοπός της my_main είναι να επιδείξει ότι το πρόγραμμα λειτουργεί σωστά ακόμα και όταν μπαίνουν εγγραφές με το ίδιο id όπως επίσης και μεγάλο πλήθος από εγγραφές γενικότερα. Ο χρήστης μπορεί να αλλάξει το RECORD_NUM και το GLOBAL_DEPT ώστε να δοκιμάσει το πρόγραμμα και με μεγαλύτερο πλήθος εγγραφών

Το πρόγραμμα έχει ελεγχθεί και για μεγάλο πλήθος εγγραφών όπως επίσης και για μεγάλο αρχικό global depth.

Επίσης έχει ελεγχθεί για memory leaks.

Πρέπει να γίνεται πάντα make clean.

Οργάνωση των blocks-Τύπος blocks



Στην παραπάνω εικόνα φαίνεται η δομή ενός block τύπου Η.

Τα block τύπου **Η** είναι blocks που χρησιμεύουν για την αναπαράσταση του **hash table** του αρχείου.

Στο πρώτο **byte** του block αποθηκεύεται ένα **char** που δηλώνει τον τύπο του, δηλαδή στην προκειμένη περίπτωση **Η.**

Σε περίπτωση που χρειαστεί κατά την διάρκεια του προγράμματος να επεκταθεί το hash table και ένα block δεν αρκεί για την αναπαράσταση του, προστίθεται και άλλο και αποθηκεύει το **block id** στο τετραγωνάκι **nextHt** του προηγούμενου. Με αυτό τον τρόπο δημιουργούνται τόσα block τύπου **Η** όσα είναι απαραίτητα για το hash table και μέσω του ‘δείκτη΄ εξασφαλίζεται η πρόσβαση σε αυτά .

Το τρίτο τετραγωνάκι που έχει μέγεθος **sizeof(int)** αποθηκεύει το **global depth** του hash table.

Το τέταρτο τετραγωνάκι που έχει επίσης **sizeof(int)** bytes μέγεθος αποθηκεύει πόσες θέσεις υπάρχουν αυτή την στιγμή διαθέσιμες που δείχνουν σε κάποιο bucket. Για παράδειγμα αν το block τυπου **H** μπορεί να αποθηκεύσει ταυτόχρονα 124 θέσεις MAX , αλλά μόνο 64 από αυτές χρησιμοποιούνται στο **current number of buckets** αποθηκεύεται ο αριθμός 64.

Στις υπολειπόμενες θέσεις αποθηκεύονται τα id των buckets που δείχνει η κάθε θέση.

Γενικά με δεδομένο block 512 bytes, κάθε block τύπου **H** μπορεί να αποθηκεύσει 124 'δείκτες' σε buckets. Το πρόγραμμα είναι φτιαγμένο έτσι ώστε και να αλλάξουν αυτά τα δεδομένα να προσαρμοστεί αναλόγως.

BLOCK ΤΥΠΟΥ D

Στην παραπάνω εικόνα φαίνεται η δομή ενός block τύπου D.

Αυτού του είδους τα blocks είναι τα buckets. Εκεί αποθηκεύονται οι εγγραφές.

Στο πρώτο **byte** του block αποθηκεύεται ένα **char** που δηλώνει τον τύπο του, δηλαδή στην προκειμένη περίπτωση **D.**

Στην μοναδική περίπτωση που χρειαστεί να προστεθούν πολλά blocks με το **ίδιο** ID και δεν χωράνε στο ίδιο block, δημιουργείται ένα block υπερχείλισης. Μόνο σε αυτή την περίπτωση και ποτέ άλλοτε. Αν το bucket για παράδειγμα χωράει MAX 8 εγγραφές και εμείς θέλουμε να βάλουμε 9 με το ίδιο ID , οι 8 θα πάνε σε ένα bucket και η 1 σε bucket υπερχείλισης. Το id του block υπερχείλισης θα αποθηκευτεί στο τετραγωνάκι **nextD.**

Το local depth του κάθε κάδου αποθηκεύεται στο τετραγωνάκι **local depth.**

Στο επόμενο τετραγωνάκι **flags** αποθηκεύονται όσα bytes χρειάζονται για τα flags. Κάθε flag είναι 0 ή 1 ανάλογα με το αν υπάρχει εγγραφή στην αντίστοιχη θέση. Για παράδειγμα αν το flag στην θέση 0 είναι 1, σημαίνει πως η $1^{η}$ εγγραφή υπάρχει. Αν θέλουμε να σβήσουμε μια εγγραφή απλά θέτουμε το αντίστοιχο flag σε 0.

Στον υπολειπόμενο χώρο του block μπαίνουν οι εγγραφές.

BLOCK ΤΥΠΟΥ M

Type (M) 1byte	nextMBlock sizeof(int)	currentNumberOfStatistics sizeof(int)	struct_Statistics sizeof(Statistics)	. . .	struct_Statistics sizeof(Statistics)

Στην παραπάνω εικόνα φαίνεται η δομή ενός block τύπου M.

Το πρώτο block του κάθε αρχείου κρατάει στατιστικά στοιχεία για κάθε bucket του αρχείου. Προφανώς τα buckets μπορεί να είναι περισσότερα από όσα struct Statistics χωράνε σε ένα block τύπου M, οπότε διατηρούμε όσα απαιτούνται, διάσπαρτα μέσα στο αρχείο.

Στη περίπτωση που κατά τη διάρκεια εκτέλεσης του προγράμματος χρειαστούν περισσότερα του ενός MBlocks, τότε ο "δείκτης" για το επόμενο MBlock (μορφή λίστας) βρίσκεται στο πεδίο nextMBlock.

Το τρίτο τετραγωνάκι, περιέχει το πλήθος από structs Statistics που υπάρχουν στο συγκεκριμένο MBlock. Στη συνέχεια, ανάλογα με το στιγμιότυπο του προγράμματος, προσθέτουμε structs Statistics για κάθε νέο Data Block που δημιουργείται (εκτενέστερη ανάλυση θα γίνει στη παράγραφο για τη συνάρτηση διαχείρισης των MBlock).

(Έχει ακολουθηθεί η ίδια λογική με τα block τύπου M, της πρώτης εργασίας)

BLOCK ΤΥΠΟΥ s

Type (M) 1byte	nextMBlock sizeof(int)	currentSizeOfStatistics sizeof(int)	lastMBlock sizeof(int)	attrName sizeof(char)	primaryFileName 20*sizeof(char)	struct_Statistics sizeof(Statistics)	. . .	struct_Statistics sizeof(Statistics)

Το 1ο block, λοιπόν, έχει αυτή τη δομή απλά υπάρχει και ένα έξτρα πεδίο μετά το currentNumberOfStatistics και το πρώτο struct, που είναι ένας int, που μας δείχνει το τελευταίο MBlock του αρχείου. Επίσης, υπάρχει 1 byte που διατηρεί τιμή City ή Surname (διαχειρίζεται μέσω enum) και άλλα 20 bytes, που διατηρούν το όνομα (ή έστω ένα μέρος αυτού), από το πρωτεύον αρχείο. Τέλος, στο πρώτο byte του block αποθηκεύεται κι ένα char που δηλώνει τον τύπο του, δηλαδή στη προκειμένη περίπτωση s.

Ο διαχωρισμός ότι το αρχείο είναι αρχείο extensible secondary hash table μπορεί να γίνει επίσης διαβάζοντας τον τύπο του πρώτου block του αρχείου. Αν είναι τύπου **s** πρόκειται για αρχείο extensible secondary hash table.

Ο διαχωρισμός ότι το αρχείο είναι αρχείο secondary hash table, μπορεί να γίνει διαβάζοντας τον τύπο του πρώτου block του αρχείου. Αν είναι τύπου **s** πρόκειται για αρχείο secondary hash table. Για να πάρει κάποιος το όνομα του primary hash table πρέπει να προσπελάσει το block τύπου **s** και να βρει το πεδίο primaryFileName που βρίσκεται μετά το byte: 1+3*sizeof(int)+sizeof(char).

Συναρτήσεις-Περιγραφή

ΣΥΝΑΡΤΗΣΕΙΣ HT

HT_ErrorCode HT_Init()

- Επιλέχτηκε η αποθήκευση διαφόρων μεταβλητών στο στατικό χώρο, επομένως η συγκεκριμένη συνάρτηση δε χρειάζεται να δεσμεύσει μνήμη στο σωρό προκειμένου να λειτουργήσουν άλλες συναρτήσεις όπως η HT_CreateIndex.

HT_ErrorCode HT_CreateIndex(const char *filename, int depth)

Δημιουργία και αρχικοποίηση ενός νέου αρχείου κατακερματισμού:

- Δημιουργία hash μπλοκ και bucket block που αντιστοιχούν στο hash block
- Αποθήκευση local depth στα bucket block
- Αποθήκευση id από κάθε bucket block στο hash block
- Αποθήκευση στο συγκεκριμένο hash block του αριθμού των bucket blocks που αντιστοιχούν στο συγκεκριμένο hash block
- Αν δεν αρκεί ένα hash block (σε περίπτωση που το global depth είναι πολύ μεγάλο), δημιουργία επιπλέον hash blocks και bucket blocks
- Αποθήκευση σε κάθε hash block τον αριθμό (id) του επόμενου hash block, αν υπάρχει
- Δημιουργία m-block

Δηλαδή αυτή η συνάρτηση παίρνει το αρχικό global depth και φτιάχνει τα κατάλληλα buckets(αν global depth =n,τότε αρχικά buckets = 2^n). Αν δεν χωράει ένα block για να αναπαραστήσει την δομή του hash table φτιάχνεται και άλλο και συνδέεται μέσω του αναγνωριστικού του αριθμού με το προηγούμενο του. Επίσης στην συνάρτηση γίνεται η αρχικοποίηση της δομής του m-block.

HT_ErrorCode HT_OpenIndex():

- Άνοιγμα αρχείου αν δεν είναι ήδη ανοικτό και αν δεν υπερβαίνουμε το MAX_OPEN_FILES
- Αποθήκευση γενικών πληροφοριών του αρχείου στον πίνακα openFiles, όπως:
  - το όνομα του αρχείου (αποθήκευση στο σωρό)
  - το file descriptor του αρχείου
- Άυξηση του openFilesCount

HT_ErrorCode HT_CloseFile(int indexDesc)

- Κλείσιμο του αρχείου
- Απελευθέρωση πόρων μνήμης που περιέχει πληροφορίες για το αρχείο
- Μείωση του openFilesCount

ΣΥΝΑΡΤΗΣΕΙΣ SHT

HT_ErrorCode SHT_Init

- Αρχικοποίηση πίνακα ανοικτών αρχείων δευτερευοντος ευρετηρίου

HT_ErrorCode SHT_CreateSecondaryIndex

- Άνοιγμα αρχείου πρωτεύοντος ευρετηρίου, αν δεν είναι ήδη ανοικτό
- Δημιουργία νέου αρχείου για το δευτερεύον ευρετήριο
- Δημιουργία του 1ου block (block τύπου s) για το δευτερεύον
- Δημιουργία hash blocks (τύπου H) για το δευτερεύον ευρετήριο
- Δημιουργία bucket blocks (τύπου D) που αντιστοιχούν στα hash blocks
- Αποθήκευση local depth στα bucket block
- Αποθήκευση id από κάθε bucket block στο hash block
- Αποθήκευση στο συγκεκριμένο hash block του αριθμού των bucket blocks που αντιστοιχούν στο συγκεκριμένο hash block
- Αν δεν αρκεί ένα hash block (σε περίπτωση που το global depth είναι πολύ μεγάλο), δημιουργία επιπλέον hash blocks και bucket blocks
- Αποθήκευση σε κάθε hash block τον αριθμό (id) του επόμενου hash block, αν υπάρχει
- Έλεγχος όλων των bucket blocks (τύπου D) στο πρωτεύον αρχείο
- Υπολογισμός tupleID για κάθε εγγραφή από το πρωτεύον και δημιουργία αντίστοιχου SecondaryRecord
- Εισαγωγή εγγραφών τύπου SecondaryRecord στο δευτερεύον ευρετήριο, μέσω κλήσης στην SHT_SecondaryInsertEntry

- Προσωρινό άνοιγμα του αρχείου που δημιουργείται (δηλαδή προσθήκη του στον πίνακα ανοικτών αρχείων δευτερευοντος ευρετηρίου) ώστε να υπάρχει κάποιο indexDesc που θα δωθεί σαν όρισμα στην SHT_SecondaryInsertEntry

HT_ErrorCode SHT_OpenSecondaryIndex

- Άνοιγμα αρχείου αν δεν είναι ήδη ανοικτό και αν δεν υπερβαίνουμε το MAX_OPEN_FILES

- Έλεγχος του πρώτου μπλοκ του αρχείου (τύπου s) με σκοπό τον προσδιορισμό του ονόματος του αρχείου πρωτεύοντος ευρετηρίου
- Έλεγχος για το αν το πρωτεύον ευρετήριο είναι ανοικτό. Αν όχι, κλείσιμο αρχείου δευτερεύοντος ευρετηρίου και επιστροφή κωδικού λάθους
- Αποθήκευση γενικών πληροφοριών του αρχείου στον πίνακα openSHTFiles, όπως:
- το όνομα του αρχείου (αποθήκευση στο σωρό)
- το όνομα του πρωτεύοντος αρχείου (αποθήκευση στο σωρό)
- το file descriptor του αρχείου
- Άυξηση του openSHTFilesCount

HT_ErrorCode SHT_CloseSecondaryIndex(int indexDesc)

- Κλείσιμο του αρχείου
- Απελευθέρωση πόρων μνήμης που περιέχει πληροφορίες για το αρχείο
- Μείωση του openSHTFilesCount

Σημειώσεις

Υπάρχουν ξεχωριστοί πίνακες για τα ανοιχτά αρχεία του πρωτευοντος και του δευτερεύοντος ευρετηρίου.

Συναρτήσεις HT

insertRecord(): Αυτή η συνάρτηση παίρνει ένα record και το id του bucket που θα βάλει την εγγραφή και την προσθέτει αν βρει χώρο. Σε περίπτωση που **ΔΕΝ** έχει χώρο **ΚΑΙ** όλες οι εγγραφές στο bucket έχουν το ίδιο id, τότε και μόνο τότε φτιάχνεται block υπερχείλισης.

Σε περίπτωση που δεν βρεθεί χώρος να μπει η εγγραφή επιστρέφεται HT_ERROR και ενεργοποιούνται οι υπόλοιπες διαδικασίες της InsertEntry για την εισαγωγή της.

Βασικά σημεία:

- Αν δεν υπάρχει χώρος στο block να μπει η εγγραφή και **όλες** οι υπάρχουσες εγγραφές έχουν το ίδιο id, αυτό σημαίνει πως θα πρέπει να φτιάξω block υπερχείλισης. Όταν συμβαίνει αυτό κάνω allocate ένα καινούριο block του αρχικοποιώ σε block τύπου 'D' και αποθηκεύω το id του στο παλιό block,ώστε να μπορεί να είναι προσβάσιμο.

arrangeBuckets(): Η arrange buckets είναι η συνάρτηση που χρησιμοποιείται όταν δεν χωράει η εγγραφή στον κατάλληλο κουβά και το local depth < global depth. Αρχικά η arrange buckets δημιουργεί τον νέο κουβά. Ύστερα ψάχνει την θέση στο hash table που τα φιλαράκια αρχίζουν να δείχνουν σε αυτόν τον κουβά. Αφού βρεί ποια είναι αυτή η θέση, παίρνει τα πρώτα μισά φιλαράκια και τα κάνει να ΄δείχνουν΄ στον νέο κουβά. Τα υπόλοιπα μισά δείχνουν στον παλιό κουβά. Τέλος για κάθε εγγραφή στον παλιό κουβά καλεί την insert entry όπως επίσης και για την καινούρια εγγραφή.

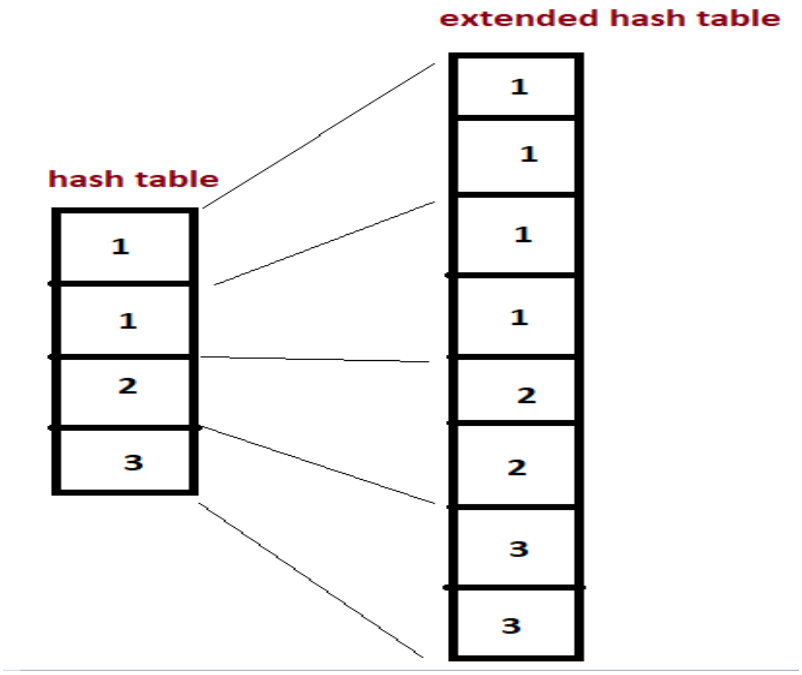
Βασικά σημεία:

- Για να βρω από πού αρχίζουν τα buddies, αρχικά βρίσκω μέσω της find_hash_table_block() ποιο είναι το bucket id που αντιστοιχεί στο κλειδί. Αφού το βρω κάνω iterate το hash table και στο πρώτο block που έχει αυτό το id σταματάω. Επιπλέον διασφαλίζω ότι ψάχνω όλα τα hash tables(επειδή το table μπορεί να εκτείνεται σε πολλά blocks).
- Οι **πρώτα** μισά φιλαράκια δείχνουν στο νέο bucket και **όχι** τελευταία μισά
- Όταν αφαιρώ τις εγγραφές από το παλιό bucket, απλά θέτω το αντίστοιχο flag τους σε 0
- Αφαιρώ **και** τις εγγραφές από το block υπερχείλισης αν υπάρχει

extendHashTable(): Η συνάρτηση αυτή χρησιμοποιείται όταν το local depth = global depth. Ουσιαστικά διπλασιάζει το μέγεθος του hash table και αυξάνει το global depth του κατά ένα.

Βασικά σημεία:

- Επειδή το hash table διπλασιάζεται φτιάχνουμε προσωρινά έναν δυναμικά δεσμευμένο πίνακα διπλάσιου μεγέθους και αντιγράφουμε όλα τα παλιά bucket ids εκεί στην κατάλληλη θέση, όπως αυτά θα είναι στο τελικό hash table. Ένα παράδειγμα φαίνεται στην παρακάτω εικόνα όπου για κάθε θέση στο παλιό hash table, αντιστοιχούν δύο θέσεις στο καινούριο hash table.



Δηλαδή η πρώτη θέση με id=1 στο παλιό hash table θα αντιγραφεί στις 2 διαδοχικές θέσεις στον καινούριο. Η επόμενη θέση με id=1 θα αντιγραφεί στις επόμενες 2 διαδοχικές και ου το καθεξής.

- Είναι πιθανό επειδή η χωρητικότητα ενός block είναι περιορισμένη να χρειαστεί να δεσμεύσουμε χώρο και για άλλο όταν μεγαλώνει το hash table. Για τον λόγο αυτό όταν δεσμεύουμε νέο block τύπου **H** αντιγράφουμε το block id στην κατάλληλη θέση **nextHt** στο παλιό block τύπου H, ώστε να μπορούμε να το προσπελάσουμε.

InsertEntry(): Είναι η πρώτη συνάρτηση που καλείται όταν είναι ώρα να μπει μια νέα εγγραφή. Κάνει αρχικά το hash του id και μετά προσπαθεί να βάλει την εγγραφή μέσω της insertRecord().Αν επιστρέψει HT_ERROR σημαίνει ότι η εγγραφή δεν χώραγε να μπει. Τότε εξετάζεται το local depth και το global depth και αν το local depth είναι μικρότερο του global depth καλείται η arrange buckets, αλλιώς καλείται η extend hash table και μετά γίνεται ξανά η κλήση της insertEntry() ώστε να αποφασιστεί που θα μπει ή νέα εγγραφή.

Γενικές παραδοχές:

- Όταν γράφουμε block υπερχείλισης εννοούμε τα blocks που προέκυψαν επειδή έγινε πολλές φορές insert εγγραφες με το ίδιο id. Επειδή υπάρχει ανάγκη να καλύπτεται και αυτή η περίπτωση αν οι εγγραφές με το ίδιο id είναι περισσότερες από την χωρητικότητα του block τότε και μόνο τότε δημιουργούνται block υπερχείλισης που αποθηκεύουν αυτές τις εγγραφές. Επίσης block υπερχείλισης δημιουργείται **αν και μόνο αν** όλες οι υπάρχουσες εγγραφές στο block έχουν το ίδιο id και συνεπώς δεν μπορεί να δημιουργηθεί χώρος να μπει η καινούρια με την arrange_buckets()
- Σε κάποιο block υπερχείλισης μπορούμε να αποθηκεύσουμε και εγγραφές και με άλλα id. Εφόσον δημιουργήθηκε δεν πρέπει να μείνει ανεκμετάλλευτο.
- Αν ένα block τύπου D συνδεόταν με κάποιο block υπερχείλισης και χρειάστηκε μέσω της arrangeBuckets() να φύγουν όλες οι εγγραφές από εκεί και να πάνε σε κάποιο άλλο block, ο δείκτης στο παλιό block υπερχείλισης **ΔΕΝ** θα σβηστεί, οι εγγραφές θα φύγουν απλά. Αυτό διότι λόγω της φύσης των συναρτήσεων που χειρίζονται block, δεν δίνεται η δυνατότητα διαγραφής ενός block από το αρχείο. Άρα ο δείκτης θα παραμείνει και απλά αν χρειαστεί θα χρησιμοποιηθεί ξανά το block υπερχείλισης.

Συναρτήσεις SHT

Οι συναρτήσεις για το secondary hash table είναι οι ίδιες με αυτές με το primary hash table με την διαφορά ότι αντί για sizeof(Record) έχω sizeof(SecondaryRecord).

Επίσης έχουν το πρόθεμα SHT μπροστά για να ξεχωρίζουν από τις συναρτήσεις του primary hash table

Επιπλέον συναρτήσεις που δημιουργήθηκαν είναι η:

validateInsertion(int indexDesc, SecondaryRecord record)

Η συνάρτηση αυτή καλέιται πριν από κάθε insertion στο secondary hash table. Σκοπός είναι να διασφαλίσει ότι δεν μπαίνει εγγραφή στο secondary hash table που δεν υπάρχει στο primary hash table.

Ελέγχει τις εξής περιπτώσεις:

a. Αν το tupleId του πρωτεύοντος ευρετηρίου είναι NULL, σημαίνει πως η εγγραφή που πάμε να εισάγουμε στο δευτερεύον ευρετήριο είναι λάθος. Άρα επιστρέφουμε μήνυμα λάθους.
b. Αν δεν είναι NULL το tupleId, πρέπει να εξεταστεί αν υπάρχει ήδη record με αυτό το tupleId στο δευτερεύον ευρετήριο. Αν υπάρχει επιστρέφω λάθος.

Οι παραπάνω περιπτώσεις είναι ικανές να εξασφαλίσουν ότι δεν εισάγουμε λανθασμένη εγγραφή στο secondary hash table.

ΣΥΝΑΡΤΗΣΕΙΣ HT

HT_PrintAllEntries(int indexDesc, int* id)

Στη συνάρτηση αυτή δεχόμαστε το index του αρχείου και ένα id από record. Αν το id είναι null, τότε η λειτουργία της συνάρτησης είναι αρκετά απλή. Παίρνει τον αριθμό των blocks του αρχείου και πηγαίνει να τα ελέγξει ένα-ένα. Όσα δεν είναι block με δεδομένα-records, τα αγνοεί. Για τα υπόλοιπα, εκτυπώνει όλα τα records που βρίσκονται στο συγκεκριμένο block.

Αν για id δοθεί κάποια τιμή (και όχι null) τότε χρησιμοποιώντας τη hash_function, βρίσκουμε το bucket που θα περιέχεται το συγκεκριμένο id. Πηγαίνουμε σε αυτό το block και σειριακά εκτυπώνουμε όλα τα records που έχουν id ίδιο με το δοσμένο. Επίσης, ελέγχουμε (κι αν χρειαστεί εκτυπώνουμε) τυχόν records που υπάρχουν στο/στα overflow block (χρησιμεύουν στη περίπτωση που έχουμε παραπάνω records από όσα χωράνε σε ένα block με το ίδιο ακριβώς id).

HashStatistics(char* filename)

Στη συνάρτηση αυτή δίνεται μόνο το όνομα του αρχείου ως όρισμα. Αρχικά, ψάχνουμε το αρχείο στον πίνακα με τα ανοιχτά αρχεία και παίρνουμε το index του. Στη συνέχεια, εκτυπώνουμε το συνολικό πλήθος block που περιέχονται σε αυτό. Ύστερα, κοιτάμε σειριακά ένα-ένα όλα τα τύπου M block που υπάρχουν στο αρχείο (αυτό είναι εφικτό αφού τα τύπου M block συνδέονται μεταξύ τους με μορφή λίστας), τα οποία περιέχουν μερικά structs Statistics, που διατηρούν τα στατιστικά στοιχεία για κάθε block τύπου D του αρχείου. Τότε, παίρνοντας ένα-ένα όλα τα structs αυτά, εκτυπώνουμε τα στοιχεία που περιέχουν για το αντίστοιχο block τύπου D.

BlockHeaderInit(BF_Block* block, char type)

Στη συνάρτηση αυτή (βοηθητική) δεχόμαστε ένα block και ένα γράμμα που αντιπροσωπεύει τον τύπο του. Η λειτουργία αυτής της συνάρτησης είναι να αρχικοποιεί κατάλληλα το header ενός block. Με βάση, λοιπόν, τις δομές που περιγράφουμε στο παρόν README, αρχικοποιούμε τα βασικά μέρη ενός block. Για παράδειγμα, στη συνάρτηση αυτή, γράφουμε στο πρώτο byte όλων των block, τον τύπο του block (H, M, m, D). Τα υπόλοιπα πεδία είναι κυρίως ints τα οποία πρέπει να αρχικοποιηθούν με 0. Κάποια από τα πεδία αυτά παραμένουν όπως είναι για πάντα, ενώ κάποια άλλα αλλάζουν κάθε φορά που αλλάζουν τα δεδομένα του συγκεκριμένου block.

Στον κώδικα φαίνονται ξεκάθαρα οι περιπτώσεις για κάθε block.

BlockHeaderUpdate(BF_Block* block, int flagPosition, char value)

Η συνάρτηση αυτή (βοηθητική) δέχεται ως όρισμα ένα block, τη θέση ενός flag και τη τιμή value που θα μετατραπεί αυτό το flag. Η λειτουργία της είναι αρκετά απλή. Ουσιαστικά πηγαίνει στο block που της δόθηκε και μετατρέπει τη τιμή του συγκεκριμένου flag από τη τιμή που είχε, στη τιμή value. Δηλαδή, αλλάζει το flag (στη συγκεκριμένη θέση) που δηλώνει το αν υπάρχει το αντίστοιχο record μέσα στο block τύπου D ή όχι. Καλείται μέσα στην InsertEntry, κάθε φορά που προστίθεται ή αφαιρείται μια εγγραφή από κάποιο block. Η συνάρτηση αυτή, δηλαδή, κάνει την αντίστοιχη ενημέρωση και δε πειράζει τίποτα άλλο στο υπόλοιπο block.

BucketStatsInit(int indexDesc, int id)

Στη συνάρτηση αυτή (βοηθητική) δεχόμαστε σαν όρισμα το index ενός αρχείου και το id ενός block τύπου D. Καλείται από διάφορα σημεία του hash_file.c, κάθε φορά που δημιουργείται ένα νέο block τύπου D. Η λειτουργία της είναι να πάει, αρχικά στο 1^{ο} block του αρχείου, που είναι τύπου m. Σε αυτό αποθηκεύουμε έναν int που έχει τον αριθμό_block για το τελευταίο block τύπου M (ή του ίδιου του 1^{ου} εφόσον είναι το τελευταίο). Πηγαίνουμε σε αυτό το block και ελέγχουμε αν χωράει κι άλλο struct Statistics να δημιουργηθεί για αυτό το νέο block τύπου D. Αν χωράει απλά φτιάχνουμε ένα νέο struct και το προσθέτουμε (με αρχικοποιημένες όλες τις τιμές του) στο block τύπου M που βρισκόμαστε. Αν δεν χωράει, δημιουργούμε ένα νέο block τύπου M, αλλάζουμε κατάλληλα τους ints του 1^{ου} m block και του προηγούμενου τελευταίου M block, ώστε να μπορούμε να βρούμε και την επόμενη φορά το τελευταίο block τύπου M σε O(1) και να διατηρηθεί η “δομή της λίστας” μεταξύ των block τύπου M. Ύστερα,

προσθέτουμε το νέο struct (με αρχικοποιημένες όλες τις τιμές του) στο νέο αυτό block τύπου M.

BucketStatsUpdate(int indexDesc, int id)

Στη συνάρτηση αυτή (βοηθητική) δεχόμαστε σαν όρισμα το index ενός αρχείου και το id ενός block τύπου D. Καλείται κάθε φορά που προστίθεται ή αφαιρείται κάποια εγγραφή από το block τύπου D με το δοσμένο id. Ελέγχοντας σειριακά όλα τα block τύπου M, βρίσκουμε αυτό που περιέχει το struct για το συγκεκριμένο id του D block. Για αυτό το D block, καλούμε την count_flags(), η οποία υπολογίζει το πλήθος των flags (δηλαδή το πλήθος των records) που υπάρχουν στο D block εκείνη τη στιγμή. Με τον αριθμό που μας επέστρεψε, ενημερώνουμε όλα τα στοιχεία του struct, με τις κατάλληλες νέες τιμές, ώστε να γραφούν ενημερωμένες στον δίσκο και να εκτυπωθούν σωστά στη HashStatistics().

ΣΥΝΑΡΤΗΣΕΙΣ SHT

SHT_PrintAllEntries(int indexDesc, int* id)

Στη συνάρτηση αυτή δεχόμαστε το index του αρχείου και ένα id από record. Αν το id είναι null, τότε η λειτουργία της συνάρτησης είναι αρκετά απλή. Παίρνει τον αριθμό των blocks του αρχείου και πηγαίνει να τα ελέγξει ένα-ένα. Όσα δεν είναι block με δεδομένα-records, τα αγνοεί. Για τα υπόλοιπα, εκτυπώνει όλα τα records που βρίσκονται στο συγκεκριμένο block.

Αν για id δοθεί κάποια τιμή (και όχι null) τότε χρησιμοποιώντας τη hash_function, βρίσκουμε το bucket που θα περιέχεται το συγκεκριμένο id. Πηγαίνουμε σε αυτό το block και σειριακά εκτυπώνουμε όλα τα records που έχουν id ίδιο με το δοσμένο. Επίσης, ελέγχουμε (κι αν χρειαστεί εκτυπώνουμε) τυχόν records που υπάρχουν στο/στα overflow block (χρησιμεύουν στη περίπτωση που έχουμε παραπάνω records από όσα χωράνε σε ένα block με το ίδιο ακριβώς id).

SHT_HashStatistics(char* filename)

Στη συνάρτηση αυτή δίνεται μόνο το όνομα του αρχείου ως όρισμα. Αρχικά, ψάχνουμε το αρχείο στον πίνακα με τα ανοιχτά αρχεία και παίρνουμε το index του. Στη συνέχεια, εκτυπώνουμε το συνολικό πλήθος block που περιέχονται σε αυτό. Ύστερα, κοιτάμε σειριακά ένα-ένα όλα τα τύπου M block που υπάρχουν στο αρχείο (αυτό είναι εφικτό

αφού τα τύπου M block συνδέονται μεταξύ τους με μορφή λίστας), τα οποία περιέχουν μερικά structs Statistics, που διατηρούν τα στατιστικά στοιχεία για κάθε block τύπου D του αρχείου. Τότε, παίρνοντας ένα-ένα όλα τα structs αυτά, εκτυπώνουμε τα στοιχεία που περιέχουν για το αντίστοιχο block τύπου D.

SHT_InnerJoin(int sindexDesc1, int sindexDesc2, char* index_key)

Η συνάρτηση αυτή σκοπό έχει να "ενώσει" τα κοινά πεδία 2 αρχείων secondary_hash_table. Άρα, αυτό που θέλει να πετύχει είναι να εκτυπωθούν στην οθόνη όλα τα records των 2 αρχείων που έχουν ίδιο index_key (το οποίο είναι η τιμή για το πεδίο City ή Surname).

Στη περίπτωση που το index_key έχει κάποια τιμή και δεν είναι NULL, τότε με βάση τη hash_function, ψάχνουμε και βρίσκουμε το block που περιέχει όλες τις εγγραφές του αρχείου με attrName ίσο με τη τιμή του index_key (η εκφώνηση μας βεβαιώνει ότι όλες αυτές οι εγγραφές βρίσκονται σε ένα block). Την ίδια διαδικασία κάνουμε και για το 2^{ο} αρχείο και στο τέλος εκτυπώνουμε, κοινά, τις εγγραφές που βρήκαμε.

Αν το index_key είναι NULL, τότε πρέπει να εκτυπωθούν όλες οι εγγραφές που έχουν ίδιες τιμές σε αυτό το πεδίο. Αυτό που υλοποιήθηκε είναι, για το 1^{ο} αρχείο να ελέγχονται 1-προς-1 τα block του. Για κάθε block, ελέγχουμε (με τη βοήθεια χρήσης ενός array) ότι τα παρακάτω βήματα εκτελούνται μόνο 1 φορά για κάθε διαφορετικό index_key του block. Για κάθε διαφορετικό index_key που έχουμε, εκτυπώνουμε όλες τις εγγραφές του block του 1^{ου} αρχείου που βρισκόμαστε και έχουν τέτοιο index_key και με τη βοήθεια της hash_function βρίσκουμε το block του 2^{ου} αρχείου που περιέχει τις εγγραφές, οι οποίες έχουν ίδιο index_key και τις εκτυπώνουμε κι αυτές. Ύστερα, συνεχίζουμε τα loops για τις υπόλοιπες εγγραφές του block και για τα υπόλοιπα blocks του αρχείου.

(Η παραπάνω λογική στηρίζεται στην *υπόδειξη* της εκφώνησης, που αναφέρει ότι " υποθέστε ότι οι εγγραφές με ίδιο index_key, βρίσκονται στο ίδιο block")

SHT_BlockHeaderInit(BF_Block* block, char type)

Στη συνάρτηση αυτή (βοηθητική) δεχόμαστε ένα block και ένα γράμμα που αντιπροσωπεύει τον τύπο του. Η λειτουργία αυτής της συνάρτησης είναι να αρχικοποιεί κατάλληλα το header ενός block. Με βάση, λοιπόν, τις δομές που περιγράφουμε στο παρόν README, αρχικοποιούμε τα βασικά μέρη ενός block. Για παράδειγμα, στη συνάρτηση αυτή, γράφουμε στο πρώτο byte όλων των block, τον τύπο του block (H, M,

s, D). Τα υπόλοιπα πεδία είναι κυρίως ints τα οποία πρέπει να αρχικοποιηθούν με 0. Κάποια από τα πεδία αυτά παραμένουν όπως είναι για πάντα, ενώ κάποια άλλα αλλάζουν κάθε φορά που αλλάζουν τα δεδομένα του συγκεκριμένου block.

Στον κώδικα φαίνονται ξεκάθαρα οι περιπτώσεις για κάθε block.

SHT_BlockHeaderUpdate(BF_Block* block, int flagPosition, char value)

Η συνάρτηση αυτή (βοηθητική) δέχεται ως όρισμα ένα block, τη θέση ενός flag και τη τιμή value που θα μετατραπεί αυτό το flag. Η λειτουργία της είναι αρκετά απλή. Ουσιαστικά πηγαίνει στο block που της δόθηκε και μετατρέπει τη τιμή του συγκεκριμένου flag από τη τιμή που είχε, στη τιμή value. Δηλαδή, αλλάζει το flag (στη συγκεκριμένη θέση) που δηλώνει το αν υπάρχει το αντίστοιχο record μέσα στο block τύπου D ή όχι. Καλείται μέσα στην InsertEntry, κάθε φορά που προστίθεται ή αφαιρείται μια εγγραφή από κάποιο block. Η συνάρτηση αυτή, δηλαδή, κάνει την αντίστοιχη ενημέρωση και δε πειράζει τίποτα άλλο στο υπόλοιπο block.

SHT_BucketStatsInit(int indexDesc, int id)

Στη συνάρτηση αυτή (βοηθητική) δεχόμαστε σαν όρισμα το index ενός αρχείου και το id ενός block τύπου D. Καλείται από διάφορα σημεία του sht_file.c, κάθε φορά που δημιουργείται ένα νέο block τύπου D. Η λειτουργία της είναι να πάει, αρχικά στο 1^{ο} block του αρχείου, που είναι τύπου s. Σε αυτό αποθηκεύουμε έναν int που έχει τον αριθμό_block για το τελευταίο block τύπου M (ή του ίδιου του 1^{ου} εφόσον είναι το τελευταίο). Πηγαίνουμε σε αυτό το block και ελέγχουμε αν χωράει κι άλλο struct Statistics να δημιουργηθεί για αυτό το νέο block τύπου D. Αν χωράει απλά φτιάχνουμε ένα νέο struct και το προσθέτουμε (με αρχικοποιημένες όλες τις τιμές του) στο block τύπου M που βρισκόμαστε. Αν δεν χωράει, δημιουργούμε ένα νέο block τύπου M, αλλάζουμε κατάλληλα τους ints του 1^{ου} m block και του προηγούμενου τελευταίου M block, ώστε να μπορούμε να βρούμε και την επόμενη φορά το τελευταίο block τύπου M σε O(1) και να διατηρηθεί η “δομή της λίστας” μεταξύ των block τύπου M. Ύστερα, προσθέτουμε το νέο struct (με αρχικοποιημένες όλες τις τιμές του) στο νέο αυτό block τύπου M.

SHT_BucketStatsUpdate(int indexDesc, int id)

Στη συνάρτηση αυτή (βοηθητική) δεχόμαστε σαν όρισμα το index ενός αρχείου και το id ενός block τύπου D. Καλείται κάθε φορά που προστίθεται ή αφαιρείται κάποια εγγραφή από το block τύπου D με το δοσμένο id. Ελέγχοντας σειριακά όλα τα block τύπου M, βρίσκουμε αυτό που περιέχει το struct για το συγκεκριμένο id του D block. Για αυτό το D block, καλούμε την count_flags(), η οποία υπολογίζει το πλήθος των flags (δηλαδή το πλήθος των records) που υπάρχουν στο D block εκείνη τη στιγμή. Με τον αριθμό που μας επέστρεψε, ενημερώνουμε όλα τα στοιχεία του struct, με τις κατάλληλες νέες τιμές, ώστε να γραφούν ενημερωμένες στον δίσκο και να εκτυπωθούν σωστά στη SHT_HashStatistics().