Υλοποίηση Συστημάτων Βάσεων Δεδομένων

Εργασίες 1 & 2

Κωνσταντίνος Χούσος
AM: 1115202000215

Αναστάσιος-Φαίδων Σεϊτανίδης
AM: 1115202000179

ΕΚΤΕΛΕΣΗ

Για να τρέξετε το κάθε πρόγραμμα, μπορείτε να εκτελέσετε την εντολή **make** στο directory που βρίσκεται το αρχείο **Makefile**. Η επιλογή **hp** κάνει compile την main για το αρχείο σωρού, η **ht** για το αρχείο κατακερματισμού και η **sht** για το αρχείο δευτερεύοντος ευρετηρίου. Τα εκτελέσιμα βρίσκονται στον κατάλογο **build/** και πρέπει να εκτελεστούν από τον γονεϊκό κατάλογο. Άρα για παράδειγμα, για την εκτέλεση της main του αρχείου σωρού, θα πρέπει να εκτελέσετε **./build/hp_main**.

ΕΡΓΑΣΙΑ 1

Αρχείο Σωρού

Συναρτήσεις

Οι παρακάτω συναρτήσεις εκμεταλλεύονται τις συναρτήσεις επιπέδου block **BF_*** για την υλοποίησή τους, προσθέτοντας ωστόσο παραπάνω λειτουργικότητα.

- **HP_CreateFile**: Δημιουργεί το αρχείο με όνομα **filename**, δημιουργεί το block 0 κι αρχικοποιεί τις τιμές του όσον αφορά τα **HP_info** και **HP_block_info**. Κάνει το block 0 dirty και το κάνει unpin. Τέλος, κλείνει το αρχείο.
- **HP_OpenFile**: Ανοίγει το αρχείο σωρού με όνομα **filename**. Ελέγχει αν το αρχείο πρόκειται για αρχείο κατακερματισμού ή όχι. Αν όχι, συνεχίζει. Καθώς κάθε φορά που καλείται η **BF_OpenFile** το **fileDesc** αλλάζει, ενημερώνουμε κάθε φορά το **HP_info** του block 0. Κάνουμε dirty κι unpin το block 0, κρατώντας ωστόσο το **HP_info** στον σωρό κι επιστρέφοντάς το.
- **HP_CloseFile**: Βρίσκει πόσα blocks έχει το αρχείο σωρού. Σε ένα for loop κάνει το καθένα unpin. Κλείνει το αρχείο κι αποδεσμεύει την μνήμη του **HP_info** που υπάρχει στον σωρό.
- **HP_InsertEntry**: Βρίσκει το τελευταίο block του αρχείου σωρού. Ελέγχει αν υπάρχει αρκετός χώρος σε αυτό για την εγγραφή μιας πλειάδας. Αν ναι, αντιγράφει την πλειάδα σε αυτό. Αν όχι, δημιουργεί ένα νέο block στο τέλος του αρχείου σωρού, γράφει σε αυτό την πλειάδα και αρχικοποιεί τα metadata του. Έπειτα ενημερώνει το **HP_info** για το νέο block και το κάνει unpin. Τέλος, επιστρέφει τον αναγνωριστικό αριθμό του block στο οποίο έγινε τελικά η εγγραφή.
- **HP_GetAllEntries**: Η συνάρτηση επιστρέφει μία μεταβλητή **int read_blocks**. Αυτή αρχικοποιείται με -1 και ανανεώνεται όταν βρούμε ένα record με id == value. Αν επιστραφεί και δεν έχει αλλάξει, σημαίνει πως υπήρξε κάποιο σφάλμα. Διατρέχει

όλα τα blocks του αρχείου σωρού, βρίσκει για το καθένα πόσες εγγραφές υπάρχουν και τυπώνει αυτές που έχουν id == value.

Δομή Αρχείου Σωρού

- Δομή **HP_info**

  Ένα αρχείο σωρού χαρακτηρίζεται από τα εξής στοιχεία:

  - **int fileDesc**: Ο αναγνωριστικός αριθμός ανοίγματος αρχείου από το επίπεδο block.
  - **int lastBlockDesc**: Ο αναγνωριστικός αριθμός του τελευταίου σε αριθμό block του αρχείου σωρού.
  - **char filetype[5]**: Ένα string που δηλώνει τον τύπο αρχείου. Μπορεί να είναι είτε "heap", είτε "hashtable" είτε "sec-index". Για αρχεία σωρού παίρνει την τιμή "heap".

- Δομή **HP_block_info**

  Ένα block ενός αρχείου σωρού χαρακτηρίζεται από τα εξής στοιχεία:

  - **int blockDesc**: Ο αναγνωριστικός αριθμός του block στην αρίθμηση των block του αρχείου σωρού.
  - **int recsNum**: Το πλήθος εγγεγραμμένων πλειάδων στο συγκεκριμένο block.
  - **int nextBlock**: Ο αναγνωριστικός αριθμός του επόμενου block στην αρίθμηση των block του αρχείου σωρού. Αν δεν υπάρχει τότε έχει την τιμή -1, με την οποία αρχικοποιείται.

Σημειώσεις

Το **HP_ERROR** ορίζεται σε -1, καθώς δεν έχει αρχικοποιηθεί κι αλλιώς δεν δουλεύει η συνάρτηση **CALL_BF**.

Αρχείο Κατακερματισμού

Συναρτήσεις

Οι παρακάτω συναρτήσεις εκμεταλλεύονται τις συναρτήσεις επιπέδου block **BF_*** για την υλοποίησή τους, προσθέτοντας ωστόσο παραπάνω λειτουργικότητα.

- **HT_CreateFile**: Δημιουργεί το αρχείο με όνομα **filename**, δημιουργεί το block 0 κι αρχικοποιεί τις τιμές του όσον αφορά τα **HT_info** και **HT_block_info**. Κάνει το block 0 dirty και το κάνει unpin. Τέλος, κλείνει το αρχείο.

- **HT_OpenFile**: Ανοίγει το αρχείο κατακερματισμού με όνομα **filename**. Ελέγχει αν το αρχείο πρόκειται για αρχείο κατακερματισμού ή όχι. Αν ναι, συνεχίζει. Αρχικοποιεί το hashtable που θα υπάρχει στην μνήμη. Καθώς κάθε φορά που καλείται η **BF_OpenFile** το **fileDesc** αλλάζει, ενημερώνουμε κάθε φορά το **HT_info** του block 0. Κάνουμε dirty κι unpin το block 0, κρατώντας ωστόσο το **HT_info** στον σωρό κι επιστρέφοντάς το.

- **HT_CloseFile**: Βρίσκει πόσα blocks έχει το αρχείο κατακερματισμού. Σε ένα for loop κάνει το καθένα unpin. Κλείνει το αρχείο κι αποδεσμεύει την μνήμη του **HT_info** που υπάρχει στον σωρό, καθώς και την μνήμη του hashtable.

- **HT_InsertEntry**: Βρίσκει τον κάδο στον οποίο πρέπει να καταγραφεί η εγγραφή. Αν ο κάδος δεν έχει κάποιο block, τότε δημιουργεί ένα και το συσχετίζει με τον κάδο μέσω του hashtable. Βρίσκουμε αν στο πιο πρόσφατο block του κάδου χωράει η εγγραφή. Αν ναι, την γράφει εκεί. Αν όχι, δημιουργεί ένα νέο block για τον κάδο—το οποίο γίνεται το πιο πρόσφατό του—και γράφει εκεί την εγγραφή.
- **HT_GetAllEntries**: Η συνάρτηση επιστρέφει μία μεταβλητή **int read_blocks**. Αυτή αρχικοποιείται με -1 και ανανεώνεται όταν βρούμε ένα record με id == value. Αν επιστραφεί και δεν έχει αλλάξει, σημαίνει πως υπήρξε κάποιο σφάλμα. Η συγκεκριμένη συνάρτηση βρίσκει τον κάδο που περιέχει εγγραφές με id == value και διατρέχει όλα τα blocks του, από το πιο πρόσφατο προς το πρώτο.

Δομή Αρχείου Κατακερματισμού

- Δομή **HT_info**

  Ένα αρχείο κατακερματισμού χαρακτηρίζεται από τα εξής στοιχεία:

  - **int fileDesc**: Ο αναγνωριστικός αριθμός ανοίγματος αρχείου από το επίπεδο block.
  - **int lastBlockDesc**: Ο αναγνωριστικός αριθμός του τελευταίου σε αριθμό block του αρχείου κατακερματισμού.
  - **char filetype[10]**: Ένα string που δηλώνει τον τύπο αρχείου. Μπορεί να είναι είτε “heap”, είτε “hashtable” είτε “sec-index”. Για αρχεία κατακερματισμού παίρνει την τιμή “hashtable”.
  - **long int numBuckets**: Το πλήθος των κάδων που θα έχει το αρχείο κατακερματισμού.
  - **int *hashtable**: Ένας πίνακας μεταβλητού μεγέθους σε int. Το hashtable του αρχείου κατακερματισμού αναπαριστάται ως εξής: Η κάθε θέση του πίνακα hashtable αναπαριστά έναν κάδο. Δηλαδή για παράδειγμα, το **hashtable[2]** δηλώνει τον κάδο 2. Το περιεχόμενο της κάθε θέσης όμως είναι ένας int, ο οποίος ταυτίζεται με το **blockDesc** του block του κάδου που προστέθηκε πιο πρόσφατα σε αυτόν. Ένα παράδειγμα της μοντελοποίησης αυτής υπάρχει στο σχήμα 1.

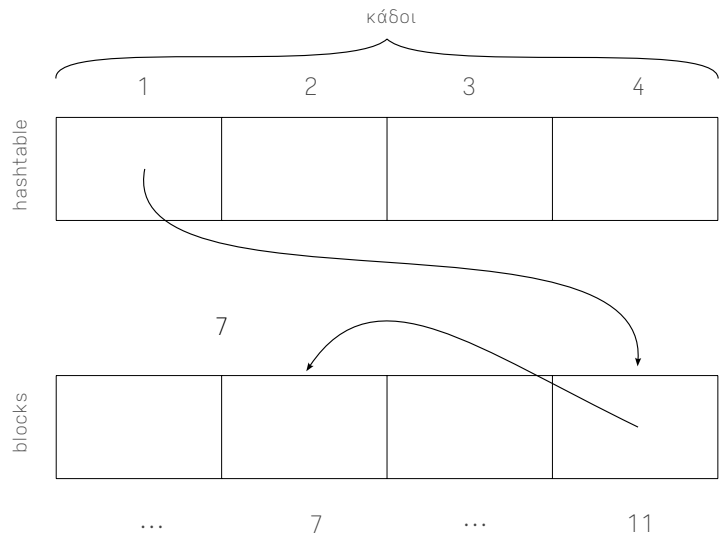
- Δομή **HT_block_info**

  Ένα block ενός αρχείου κατακερματισμού χαρακτηρίζεται από τα εξής στοιχεία:

  - **int blockDesc**: Ο αναγνωριστικός αριθμός του block στην αρίθμηση των block του αρχείου κατακερματισμού.
  - **int prevBlockDesc**: Ο αναγνωριστικός αριθμός του προηγούμενου block του κάδου. Αρχικοποιείται με -1 εφόσον δεν υπάρχει προηγούμενο block. Στο παράδειγμα του σχήματος 1 το **prevBlockDesc** του block 11 θα είναι ο αριθμός 7, ενώ το **prevBlockDesc** του block 7 θα είναι -1.
  - **int recsNum**: Το πλήθος εγγεγραμμένων πλειάδων στο συγκεκριμένο block.

Συνάρτηση Κατακερματισμού

Η συνάρτηση κατακερματισμού που χρησιμοποιήθηκε στη συγκεκριμένη υλοποίηση είναι η $h(K) = K \% M$ όπου K είναι το κλειδί—στη συγκεκριμένη περίπτωση το id—, και M το πλήθος των κάδων του αρχείου κατακερματισμού.



Σχήμα 1: Παράδειγμα μοντελοποίησης hashtable. Αν ο κάδος 1 αποτελείται από το block 7, τότε **hashtable[1] = 7**. Αν το block 7 κάποια στιγμή γεμίσει, τότε για να προστεθεί μια νέα εγγραφή στον κάδο 1 θα δημιουργηθεί ένα νέο block στο τέλος του αρχείου. Έστω ότι το νέο block έχει αριθμό—κι άρα **blockDesc**—11. Πλέον το **hashtable[1]** θα ενημερωθεί και θα αποκτήσει την τιμή 11, τον αριθμό δηλαδή του πιο πρόσφατου block του. Στο **HT_block_info** του block 11 ο αναγνωριστικός αριθμός **prevBlockDesc** θα αλλάξει από -1 και θα γίνει 7, έτσι ώστε να δείχνει στο προηγούμενο στη σειρά block. Άρα για κάθε κάδο έχουμε πρόσβαση σε όλα τα blocks του σειριακά.

ΕΡΓΑΣΙΑ 2

Δευτερεύον Ευρετήριο Κατακερματισμού

Συναρτήσεις

Οι παρακάτω συναρτήσεις εκμεταλλεύονται τις συναρτήσεις επιπέδου block **BF_*** για την υλοποίησή τους, προσθέτοντας ωστόσο παραπάνω λειτουργικότητα.

- **SHT_CreateSecondaryIndex**: Δημιουργεί το αρχείο δευτερεύοντος ευρετηρίου με όνομα **sfileName**, δημιουργεί το block 0 κι αρχικοποιεί τις τιμές του όσον αφορά τα **SHT_info** και **SHT_block_info**. Κάνει το block 0 dirty και το κάνει unpin. Τέλος, κλείνει το αρχείο.
- **SHT_OpenSecondaryIndex**: Ανοίγει το αρχείο δευτερεύοντος ευρετηρίου με όνομα **filename**. Ελέγχει αν το αρχείο πρόκειται για αρχείο κατακερματισμού ή όχι. Αν ναι, συνεχίζει. Αρχικοποιεί το sht_hashtable που θα υπάρχει στην μνήμη. Καθώς κάθε φορά που καλείται η **BF_OpenFile** το **fileDesc** αλλάζει, ενημερώνουμε κάθε φορά το **SHT_info** του block 0. Κάνουμε dirty κι unpin το block 0, κρατώντας ωστόσο το **SHT_info** στον σωρό κι επιστρέφοντάς το.
- **SHT_CloseSecondaryIndex**: Βρίσκει πόσα blocks έχει το αρχείο δευτερεύοντος ευρετηρίου. Σε ένα for loop κάνει το καθένα unpin. Κλείνει το αρχείο κι αποδεσμεύει την μνήμη του **SHT_info** που υπάρχει στον σωρό, καθώς και την μνήμη του sht_hashtable.
- **SHT_SecondaryInsertEntry**: Δημιουργεί ένα νέο **SHT_Record** από το **name** και το **blockid**. Βρίσκει τον κάδο του δευτερεύοντος ευρετηρίου στον οποίο πρέπει να καταγραφεί το **SHT_Record**, όπου σε περίπτωση λάθους επιστρέφει με τιμή -1. Αν ο κάδος δεν

έχει ακόμα κάποιο κατανεμημένο σε αυτό block, τότε δημιουργεί ένα. Εισχωρεί το **SHT_Record** στο πιο πρόσφατο block του κάδου, αν χωράει σε αυτό. Αν όχι, τότε δημιουργεί ένα νέο block για τον κάδο και εισχωρεί εκεί την εγγραφή.

- **SHT_SecondaryGetAllEntries**: Η συνάρτηση επιστρέφει μία μεταβλητή **int read_blocks**. Αυτή αρχικοποιείται με -1 και ανανεώνεται όταν βρούμε ένα record με id == value. Αν επιστραφεί και δεν έχει αλλάξει, σημαίνει πως υπήρξε κάποιο σφάλμα. Η συγκεκριμένη συνάρτηση βρίσκει τον κάδο του δευτερεύοντος ευρετηρίου που μπορεί να περιέχει εγγραφές με hash value == **hash(name)** της εγγραφής και διατρέχει τα blocks του. Αν βρει μια τέτοια εγγραφή, τότε φορτώνει το block του **data.db** στο οποίο δείχνει η εγγραφή. Ύστερα, το διατρέχει ώσπου να βρει την εγγραφή με όνομα **name**.

Δομή Αρχείου Κατακερματισμού

- Δομή **SHT_Record**

  Αποτελεί μια εγγραφή του αρχείου δευτερεύοντος ευρετηρίου. Περιέχει:

  - **char name[15]**: Το όνομα της εγγραφής. Έχει μέγιστο μήκος 15, όμοια με το **name** του struct **Record**.
  - **int blockDesc**: Ο αναγνωριστικός αριθμός του block του αρχείου κατακερματισμού στο οποίο βρίσκεται η τιμή.

- Δομή **SHT_info**

  Ένα αρχείο δευτερεύοντος ευρετηρίου χαρακτηρίζεται από τα εξής στοιχεία:

  - **int fileDesc**: Ο αναγνωριστικός αριθμός ανοίγματος αρχείου από το επίπεδο block.
  - **int lastBlockDesc**: Ο αναγνωριστικός αριθμός του τελευταίου σε αριθμό block του αρχείου δευτερεύοντος ευρετηρίου.
  - **char filetype[10]**: Ένα string που δηλώνει τον τύπο αρχείου. Μπορεί να είναι είτε "heap", είτε "hashtable" είτε "sec-index". Για αρχεία δευτερεύοντος ευρετηρίου παίρνει την τιμή "sec-index".
  - **long int numBuckets**: Το πλήθος των κάδων που θα έχει το αρχείο δευτερεύοντος ευρετηρίου.
  - **int *sht_hashtable**: Ένας πίνακας μεταβλητού μεγέθους σε int, όμοιος με τον πίνακα **hashtable** του αρχείου κατακερματισμού.

- Δομή **SHT_block_info**

  Ένα block ενός αρχείου δευτερεύοντος ευρετηρίου χαρακτηρίζεται από τα εξής στοιχεία:

  - **int blockDesc**: Ο αναγνωριστικός αριθμός του block στην αρίθμηση των block του αρχείου δευτερεύοντος ευρετηρίου.
  - **int prevBlockDesc**: Ο αναγνωριστικός αριθμός του προηγούμενου block του κάδου. Αρχικοποιείται με -1 εφόσον δεν υπάρχει προηγούμενο block.
  - **int recsNum**: Το πλήθος εγγεγραμμένων πλειάδων στο συγκεκριμένο block.

Συνάρτηση Κατακερματισμού

Η συνάρτηση κατακερματισμού του αρχείου δευτερεύοντος ευρετηρίου ορίζεται στην `SHT_Hash`. Παίρνει ως όρισμα το όνομα της εγγραφής και το πλήθος των κάδων του αρχείου. Επιστρέφει τον κάδο στον οποίο πρέπει να καταγραφεί η εγγραφή.

Ο αλγόριθμος κατακερματισμού εμπνεύστηκε εν μέρει από τον αλγόριθμο 16.2(a) του συγγράμματος [1, σ. 574], ο οποίος φαίνεται στο σχήμα 2.

Algorithm 16.2. Two simple hashing algorithms: (a) Applying the mod hash function to a character string K. (b) Collision resolution by open addressing.

(a) *temp* ← 1;
for *i* ← 1 to 20 do *temp* ← *temp* * code(*K*[*i*]) mod *M* ;
hash_address ← *temp* mod *M*;

(b) *i* ← *hash_address*(*K*); *a* ← *i*;
if location *i* is occupied
then **begin** *i* ← (*i* + 1) mod *M*;
while (*i* ≠ *a*) and location *i* is occupied
do *i* ← (*i* + 1) mod *M*;
if (*i* = *a*) then all positions are full
else *new_hash_address* ← *i*;
end;

Σχήμα 2: Αλγόριθμος κατακερματισμού συγγράμματος για strings.

Η υλοποίηση της συνάρτησης φαίνεται στο listing 1.

```c
int SHT_Hash(char *name, int buckets) {

  /* Error handling */
  if (name[0] == '\0')
    return -1;

  int temp = 0;

  for (int i = 0; i < 15; i++) {

    /* name length < 15 */
    if (name[i] == '\0')
      break;

    temp += name[i] % buckets;
  }

  return temp % buckets;
}
```

Listing 1: Υλοποίηση συνάρτησης `SHT_Hash`.

ΣΤΑΤΙΣΤΙΚΑ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ

Η συνάρτηση κατακερματισμού **HashStatistics** βρίσκεται στο αρχείο **src/stats.c**, μαζί με τις βοηθητικές της συναρτήσεις. Καλείται στο τέλος της κάθε main. Το filetype κάθε αρχείου περιγράφεται από string έτσι ώστε η πιθανότητα λάθους να είναι ελάχιστη, κάτι που με έναν απλό int δεν θα ίσχυε.

Οι βοηθητικές συναρτήσεις της **HashStatistics**—οι οποίες αναπαριστούν και καθένα από τα βήματά της—είναι οι εξής:

- **int STATS_GetFiletype**: Επιστρέφει το filetype του αρχείου, όπου τιμή 1 δηλώνει αρχείο σωρού, τιμή 2 αρχείο κατακερματισμού και τιμή 3 αρχείο δευτερεύοντος ευρετηρίου. Αρχικοποιεί την μεταβλητή **filetype** σε -1, και την αλλάζει ανάλογα το είδος του αρχείου στην αντίστοιχη τιμή. Αν δεν βρεθεί αντιστοιχία σε κάποιο είδος αρχείου, παραμένει -1. Τέλος, η μεταβλητή **filetype** επιστρέφεται.
- **int STATS_NumberOfBlocks**: Επιστρέφει το πλήθος των blocks του αρχείου. Εκμεταλλεύεται την έτοιμη συνάρτηση **BF_GetBlockCounter**.
- **int STATS_MinBlocksNum**: Επιστρέφει το ελάχιστο πλήθος εγγραφών που έχει κάθε bucket του αρχείου. Για κάθε bucket διατρέχουμε τα blocks, βρίσκουμε πόσες εγγραφές έχει το καθένα κι αθροίζουμε τις εγγραφές των blocks. Αν οι εγγραφές του bucket είναι λιγότερες από το **min_records**, τότε το τελευταίο παίρνει την τιμή του πρώτου.
- **int STATS_MaxBlocksNum**: Ίδια με την παραπάνω, αλλά επιστρέφει το μέγιστο πλήθος εγγραφών.
- **int STATS_BucketsNum**: Επιστρέφει το πλήθος των κάδων του αρχείου κατακερματισμού/δευτερεύοντος ευρετηρίου. Χρησιμοποιείται για τον υπολογισμό του μέσου αριθμού των blocks που έχει κάθε bucket, καθώς ο τελευταίος θα ισούται με το πλήθος των blocks (**STATS_NumberOfBlocks**) προς το πλήθος των κάδων.
- **int STATS_PrintOverflowStats**: Εκτυπώνει το πλήθος των buckets που έχουν block υπερχείλισης, και πόσα block είναι αυτά για κάθε bucket. Διατρέχει τους κάδους κι υπολογίζει για τον καθέναν τις αντίστοιχες τιμές και τις εκτυπώνει.
- **int HashStatistics**: Καλεί τις παραπάνω συναρτήσεις. Αυτή καλείται από την εκάστοτε main με τα κατάλληλα ορίσματα.

Παρατήρηση

Οι "αυθαιρεσίες" όσον αφορά τους ορισμούς των συναρτήσεων βασίστηκε στις απαντήσεις του διδάσκοντα στο eClass.

ΣΗΜΕΙΩΣΕΙΣ

- Τα αρχεία των εργασιών 1 και 2 βρίσκονται στους ίδιους καταλόγους καθώς έχουν κοινά αρχεία, βιβλιοθήκη κτλ.

Αναφορές

[1] Ramez Elmasri και Sham Navathe. *Fundamentals of Database Systems*. Seventh edition. Hoboken, NJ: Pearson, 2016. 1242 pagetotals. ISBN: 978-0-13-397077-7.