Χειμερινό Εξάμηνο 2023-2024
Υλοποίηση Συστημάτων Βάσεων Δεδομένων
Διδάσκων Γ. Ιωαννίδης
2η Εργασία

Στυλιανός Δημητριάδης 1115201900050
Παναγιώτης Κοντοειδής 1115201900266

Για την υλοποίηση της εργασίας συμπληρώθηκαν οι δομές HT_info και HT_block_info στο αρχείο include/ht_file.h.

Κάθε αρχείο επεκτατού κατακερματισμού έχει στο πρώτο μπλοκ του μια δομή HT_info η οποία περιέχει πληροφορίες για το ίδιο το αρχείο και κάθε μπλοκ - bucket έχει στην αρχή του μια δομή HT_block που περιέχει πληροφορίες για το ιδιο το μπλοκ.

Σχεδιαστικές επιλογές των ζητούμενων συναρτήσεων.

HT_Init

Η συνάρτηση αρχικοποιεί και δημιουργεί έναν πίνακα από HT_info (hash_file_array), όπου θα αποθηκεύονται οι πληροφορίες των ανοιχτών αρχείων. Ο μέγιστος αριθμός αρχείων ορίζεται από την μεταβλητή MAX_OPEN_FILES.

HT_Close

Η συνάρτηση αποδεσμεύει τον πίνακα των ανοιχτών αρχείων.

HT_CreateIndex

Η συνάρτηση ανοίγει και δημιουργεί ένα αρχείο κατακερματισμού. Συγκεκριμένα, αρχικοποιούνται οι πληροφορίες που αφορούν το αρχείο, όπως ένα χαρακτηριστικό string, ο αριθμός των μπλοκ(buckets) που περιέχει και το ολικό βάθος του ευρετηρίου.

Κάθε μπλοκ που δημιουργείται αντιπροσωπεύει ένα bucket και αρχικοποιείται αντίστοιχα, από την βοηθητική συνάρτηση **createBucket,** στο τέλος του αρχείου.

Το ευρετήριο βρίσκεται στο πρώτο μπλοκ κάθε αρχείου, μετά την δομή HT_info. Το ευρετήριο δημιουργείται δυναμικά ως ένας δείκτης σε έναν πίνακα από ints, όπου κάθε int είναι το νούμερο του μπλοκ στο οποίο δείχνει κάθε τιμή του πίνακα.

HT_OpenIndex

Η συνάρτηση ανοίγει το αρχείο κατακερματισμού και αντιγράφει τις πληροφορίες του στην πρώτη διαθέσιμη θέση στον πίνακα των ανοιχτών αρχείων, αλλιώς επιστρέφει HT_Error.

HT_CloseFile

Η συνάρτηση κλείνει το αρχείο κατακερματισμού και αφαιρεί τις πληροφορίες του από τον πίνακα των ανοιχτών αρχείων.

HT_InsertEntry

Η συνάρτηση τοποθετεί μια εγγραφή στο αρχείο κατακερματισμού. Η συνάρτηση κατακερματισμού των id των εγγραφών που χρησιμοποιήθηκε είναι παρόμοια με αυτή που παρουσιάστηκε στο μάθημα. Το id κάθε εγγραφής αντιστρέφεται μετατρέπεται σε bits και με βάση το Global Depth n , μέσω μιας μάσκας, κρατάμε τα πρώτα n - bit για την ταξινόμηση. Οι βοηθητικές συναρτήσεις για αυτή τη διαδικασία είναι οι **bitExtracted, reverseBits, extract_bits**.

Με βάση το hash key που προκύπτει από το ID της εγγραφή, ελέγχεται αν χωράει στο bucket που προορίζεται να τοποθετηθεί. Αν δεν χωράει στο bucket ακολουθείται ο αλγόριθμος του μαθήματος.

Ειδικότερα, αν το τοπικό βάθος του bucket είναι ίσο με το ολικό βάθος του ευρετηρίου, διπλασιάζεται το βάθος του ευρετηρίου και αυξάνεται το ολικό βάθος κατά ένα. Ο διπλασιασμός γίνεται με την βοηθητική συνάρτηση **expandingHashingTable** όπου δημιουργείται ένα νέο ευρετήριο προσαρμόζοντας τις παλιές τιμές έτσι ώστε κάθε θέση που δείχνει σε d+1 bits να δείχνει εκεί που έδειχνε η θέση του παλιού πίνακα με d bits, καλώντας την **extract_bits**. Τέλος, ξαναεισάγεται αναδρομικά η εγγραφή.

Στην περίπτωση που το τοπικό βάθος του bucket είναι μικρότερο από το ολικό βάθος του ευρετηρίου, θα χρειαστεί να γίνει split του bucket. Ετσι, δημιουργείται ένα νέο bucket στο τέλος του αρχείου, αυξάνεται και στα δύο buckets, νέο και παλιό, το τοπικό βάθος και δημιουργείται ένα νέο ευρετήριο με την διαδικασία που περιγράφηκε παραπάνω. Η διαφορά με την παραπάνω περίπτωση είναι ότι γίνεται και διαμοιρασμός των κελιών του ευρετηρίου που δείχνουν στο παλιό bucket έτσι ώστε να δείχνουν και στο καινούργιο. Δηλαδή, βρίσκονται όλες οι θέσεις των δεικτών στον πίνακα κατακερματισμού (αριθμοί στο bucket) που δείχνουν στο bucket που έγινε split. Αυτοί χωρίζονται σε δύο όμοια σύνολο και οι μισοί από αυτούς παραμένουν ίδιοι (δείχνουν στο ίδιο bucket),ενώ οι άλλοι μισοί αλλάζουν και τώρα δείχνουν στο νέο bucket που δημιουργήθηκε.

Οι ήδη υπάρχουσες εγγραφές του bucket θα αποθηκευτούν σε ένα array (array_of_record) και μαζί με την νέα θα ξανα εισαχθούν αναδρομικά. Στο παλιό bucket δεν διαγράφονται οι παλιές εγγραφές αλλά, μηδενίζεται ο αριθμός των εγγραφών που περιέχει έτσι ωστε, οι καινούργιες που θα εισαχθούν σε αυτόν να κάνουν overwrite τις παλιές.

HT_PrintAllEntries

Η συνάρτηση εκτυπώνει όλες τις εγγραφές που υπάρχουν σε ένα αρχείο κατακερματισμού αν το id που δωθεί είναι null ή εκτυπώνει όλες τις εγγραφές που έχουν την τιμή id, αν δεν υπάρχει καμία εγγραφή με αυτό το id εκτυπώνεται κατάλληλο μήνυμα.

HT_HashStatistics

Η συνάρτηση τυπώνει τον αριθμό των μπλοκ που έχει το αρχείο κατακερματισμού.Τον ελάχιστο αριθμό εγγραφών που θα βρει σε ενα μπλοκ, τον μέσο αριθμό εγγραφών όλων των μπλοκ και τον μέγιστο αριθμό εγγραφών που θα βρει σε ένα μπλοκ.

Υπολογίστηκε ότι σε κάθε μπλοκ - bucket χωράνε 8 εγγραφές και για αυτό πολλές φορές ο μέγιστος αριθμός εγγραφών που θα βρεθεί είναι 8 και ο μέσος είναι γύρω στις 5.

Εκτέλεση του κώδικα:

Για τον έλεγχο της πλήρους λειτουργικότητας του κώδικα δημιουργήθηκαν δύο ξεχωριστές main εκτός της δοσμένης.
Στην ht_main, βρίσκεται η έτοιμη main η οποία μας δόθηκε.

```
make
./build/runner
```

Στην **test_main1,** δημιουργούνται 1000 τυχαίες εγγραφές τοποθετούνται σε ένα αρχείο κατακερματισμού, εκτυπώνονται από το αρχείο και εκτυπώνονται και τα στατιστικά εισαγωγής τους.

```
make main1
./build/runner
```

Στην **test_main2,** γίνεται η ίδια διαδικασία με την test_main1 αλλά για τρία διαφορετικά αρχεία κατακερματισμού τα οποία ανοίγουν ταυτόχρονα. Σε κάθε ένα από αυτά τα αρχεία εισάγονται διαφορετικές τυχαίες εγγραφές.

```
make main2
./build/runner
```

Επίσης μετά από κάθε make μπορεί να γίνει και make clean.

Τέλος το πρόγραμμα εκτελώντας το σε περιβάλλον valgrind με την εντολή :

```
valgrind --leak-check=full --show-leak-kinds=all ./build/runner
```

δεν παρουσιάζει κανένα memory leak και τρέχει κανονικά.

Παρόλα αυτά, τρέχοντας το στο terminal παρουσιάζει τον κωδικό λάθους :

```
BF Error: The block number doesn't exists into the file
```

Εντοπίσαμε το πρόβλημα να είναι στην γραμμή 347 στο αρχείο src/hash_file.c, όπου γίνεται αποδέσμευση του δείκτη p. Για αυτόν τον λόγο, αφαιρέσαμε σε σχόλια την γραμμή αυτή και μπορεί ενδεχομένως να φανούν κάποια leaks.