Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Πανεπιστήμιο Αθηνών

**Υλοποίηση Συστημάτων Βάσεων Δεδομένων (K18)**

**Περίοδος 2021-2022**

**Καθηγητής Ι. Ιωαννίδης**

# 1η Εργασία


Ομάδα:

Ονομ/νυμο: Μιχαήλ Δεικτάκης AM: 1115200800018

Ονομ/νυμο: Ιωάννης Ψαρρός AM: 1115201800216

Ονομ/νυμο: Φίλιππος-Γεώργιος Μπράτης AM: 1115201400121


**Γενικές Παραδοχές:**

Ένα hash file έχει το πρώτο του block με indexing 0 το οποίο κρατάει info. Αυτό στην περίπτωση μας είναι το global depth του hash table.

Από το 2ο έως το kο block με indexing μέσα στο αρχείο 1-MAX_HASH_BLOCKS θα κρατάμε τα directories μας που δείχνουν τα blocks of records (buckets) με τη μορφή int ο οποίος είναι ο αριθμός του εκάστωτε block.

Από το k+1 έως το Νο block κρατάμε τα buckets όπου 1 block = 1 bucket.

Η μορφή ενός bucket είναι στα πρώτα BF_BLOCK_SIZE – 8 bytes αποθηκεύονται όσα blocks χωράνε ενώ στα επόμενα 4 bytes κρατάμε το local_depth του bucket και στα τελευταία 4 κρατάμε το πλήθος των records που περιέχει.

Δεν έχουμε υλοποιήσει dynamic allocating του hash table , όμως δίνουμε τη δυνατότητα στον χρήστη μέσω της σταθερής μεταβλητής MAX_HASH_BLOCKS να ορίσει από την αρχή πόσα blocks θα χρησιμοποιηθούν για το hashing.

Σχετικά με τα records μπορούν να δωθούν διπλότυπα σε Record.id μέχρι το max πλήθος των records που χωράνε σε ένα bucket όμως (δεν έχουμε λίστες υπερχείλησης) οπότε αν πάει να μπει μεγαλύτερο πλήθος θα γίνεται συνεχόμενα expand στο directory μέχρι να γεμίσει το hash table και να επιστρέψει error.

Τέλος το hashing ενός αριθμού γίνεται παίρνοντας ανάποδα τα bits του αριθμού έτσι ώστε να αποφευχθούν πολλά overflow από την αρχή (πχ id = 10 , binary = 000000....1010 , το hash με depth 4 θα επιστρέψει 0101 = 5) το οποίο επιβεβαιώθηκε και από ερώτημα στο e-class πως είναι αποδεκτό.

**HT_INIT:**

Αρχικά έχουμε τις global variables num_files που κρατάει τον αριθμό των ανοιχτών αρχείων και την openfiles η οποία είναι pointer σε array από struct files. Ένα struct file έχει ένα filedescriptor και το global_depth του συγκεκριμένου file.

Στην init αρχικοποιούμε το array με descriptors = -1 (σημαίνει πως σε αυτή τη θέση δεν υπάρχει ανοιχτό αρχείο) και global_depth = 0 , αυτό θα το πάρουμε από την create.

**HT_CREATEINDEX:**

Στην createindex φτιάχνουμε ένα αρχείο. Αρχικά θέτουμε στο block 0 το global depth που μας δώθηκε. Μετά δημιουργούμε τα blocks του hash table σύμφωνα με το MAX_HASH_BLOCKS.

Κάνουμε μετά allocate τα πρώτα Record blocks (buckets) και στη συνέχεια θέτουμε τα directories να δείχνουν με τη σειρά σε κάθε ένα από αυτά τα blocks θέτοντας το local_depth=global_depth και το πλήθος των records = 0.

Θεωρούμε πως αν κάποιος φτιάξει ένα αρχείο με σχετικά μεγάλο global depth θα θέλει να αποθηκεύσει πολλά records οπότε τα buckets αυτά θα γεμίσουν.

**HT_OPENINDEX:**

Όταν ανοίγουμε ένα index αρχικά αποθηκεύουμε το αναγνωριστικό του αρχείου στον πίνακα με τα ανοιχτά αρχεία.

Μετά θέτουμε το global depth του αρχείου να είναι ίσο με αυτό που έχει αποθηκευτεί στο block 0 έτσι αν ένα αρχείο ανοίχτηκε από άλλο χρήστη και έχει αλλάξει το global depth δεν χάνεται.

**HT_CLOSEFILE:**

Όταν κλείνουμε ένα αρχείο διαγράφουμε το αναγνωριστικό του από τον πίνακα με τα ανοιχτά αρχεία και θέτουμε το global depth στο block 0 ίσο με αυτό που έχουμε εκείνη τη στιγμή στον πίνακά μας.(Αυτό «πειράζουμε» όταν κάνουμε τα insert οπότε πρέπει να αποθηκευτεί πάλι στο αρχείο)

**HT_INSERTENTRY:**

Αρχικά κοιτάμε αν το global depth που έχουμε είναι μεγαλύτερο από αυτό που μπορεί να φτάσει ο πίνακας με τα directories οπότε αν είναι επιστρέφουμε error.

Μετά κοιτάμε αν το αρχείο στο οποίο προσπαθεί ο χρήστης να εισάγει το record είναι ανοιχτό.

Στη συνέχεια κάνουμε hash το id και βρίσκουμε μέσω των directories σε ποιο bucket πρέπει να μπει το record. Η υλοποίηση των id των directories είναι ουσιαστικά η θέση στο block που βρίσκεται ενώ το bucket στο οποίο δείχνει βρίσκεται μέσω του αριθμού του block που έχουμε αποθηκεύσει στη συγκεκριμένη θέση. Επειδή τα blocks του hash table είναι περισσότερα από ένα βρίσκουμε πρώτα σε ποιο  block είναι το directory (hash(id)/(BF_BLOCK_SIZE/sizeof(int)) και τη θέση μέσω (hash(id) % (BF_BLOCK_SIZE/sizeof(int)).

Αφου βρούμε το bucket κοιτάμε πρώτα αν χωράει και αν ναι το τοποθετούμε.

Αν όχι κοιτάμε και το local depth του bucket:

Αν είναι ίσο με global depth κάνουμε expand στα directories και μετά split. Το expand γινεται με βοηθητικά arrays όπου αποθηκεύουμε πρώτα τα buckets που δείχνουν, μετά φτιάχνουμε νέο array διπλάσιο και θέτουμε να δείχνει και αυτό σε σωστά buckets και τέλος ξαναγράφουμε το array στο table. Μετά κάνουμε allocate το νέο block και υλοποιούμε το split με βοηθητικό array of records κρατάμε τα records που χρειάζεται να ξαναγίνουν hash διαγράφουμε τα δεδομένα του bucket και τα ξαναγράφουμε σωστά καλώντας την insert για κάθε ένα από αυτά με το νέο global kai local depth (+1).

Αν είναι μικρότερο από global depth απλά κάνουμε τη διαδικασία του split που περιγράφεται παραπάνω με νέο local depth(+1).

**HT_PRINTALLENTRIES:**

Ανάλογα με το αν μας δόθηκε συγκεκριμένο id ή NULL:

Στην περίπτωση NULL εκτυπώνονται όλα τα Records με τη σειρά που είναι στα buckets και όχι με τη σειρά του hashing τους όπου φαίνεται οτι όλα τα buckets περιέχουν ids τα οποία έχουν ίδιο hashing ανάλογα με το global depth (εκτυπώνεται κάτω κάτω για ευκολία στον έλεγχο).

Στην περίπτωση συγκεκριμένου id βρίσκουμε μέσω του hashing το bucket στο οποίο βρίσκεται και το εκτυπώνουμε με πολυπλοκότητα O(1) αφού ξέρουμε που βρίσκεται ακριβώς (Τυπικά θα είναι O(BLOCK_SIZE/sizeof(Record)) αφού δεν είναι 1block = 1 Record).

**HT_HASHSTATISTICS:**

Βρίσκουμε το σύνολο των records σε κάθε bucket και επιστρέφουμε το bucket με τον ελάχιστο αριθμό και αυτό με τον μέγιστο , και τέλος εκτυπώνεται και το average πλήθος records που έχουν τα buckets.

**MAIN:**

Η ht_main.c μας δείχνει οτι μποτούμε να κλείσουμε ένα αρχείο να το ξανανοίξουμε χωρίς να χάσουμε τα δεδομένα μας.

Στην new_main.c μπορούμε να δούμε ότι μπορούμε να έχουμε πολλαπλά αρχεία με διαφορετικό global depth ανοιχτά ταυτόχρονα και να τα «πειράζουμε» ξεχωριστά.