Μαθητές που πήραν μέρος στην εργασία:

* Μαζέρας Αντώνιος - 1115201400092
* Μάλλιος Χρήστος-Ιωάννης - 1115201900106

Σχόλια εργασίας:

Για την υλοποίηση της εργασίας χρειαστήκαμε κάποιες βοηθητικές δομές και συναρτήσεις, αυτές είναι:

Structs:

* Buckets
* Hashtable

Συναρτήσεις:

* HashFunction
* Printrecord
* Power
* updateDirectory
* getDirectory

Αρχικά, το struct buckets είναι ένα struct που αποτελείται από ένα hashcode που είναι ο κωδικός του bucket, number of block που είναι ο αριθμός που αποθηκεύονται οι εγγραφές αυτού του bucket, number of registries που είναι το πόσες εγγραφές υπάρχουν ήδη μέσα στο bucket αυτό, local depth που είναι το τοπικό βάθος του bucket, και maxsize που είναι το πόσες εγγραφές χωράει συνολικά το bucket

Στη συνέχεια, το struct Hashtable που αποτελείται από το global depth του hashtable, ένα πίνακα από buckets και το maxbuckets που είναι το πόσα buckets χωράνε σε κάθε μπλοκ για να ξέρουμε πόσα μπορούμε να αποθηκεύσουμε στο block όταν περνάμε το hashtable στη μνήμη.

Για τις συναρτήσεις, έχουμε την hashFunction που παίρνει έναν δεκαδικό αριθμό και τον μετατρέπει σε δυαδικό αλλά δεν γυρνάει όλα του τα ψηφία (στο δυαδικό) αλλά μόνο τα depth πρώτα γιατί δεν χρειάζεται να ελεγχθούν τα υπόλοιπα. Έπειτα έχουμε την printrecord που παίρνει σαν όρισμα ένα record και απλά το εκτυπώνει. Η power απλά υπολογίζει την μαθηματική έκφραση .

Οι update και get Directory είναι βοηθητικές συναρτήσεις για κάθε φορά που έχει αλλαχθεί το ευρετήριο είτε λόγο εισαγωγής νέου στοιχείου είτε λόγο κάποιας ενέργειας expand ή split και κάθε φορά που θέλουμε να πάρουμε το ευρετήριο από την μνήμη για να ψάξουμε κάποια εγγραφή που θέλουμε να τυπώσουμε για να εισάγουμε μία νέα και να δούμε σε ποιο bucket πρέπει να εισαχθεί και να μπορέσουμε μετά να το ενημερώσουμε.

Για αρχή η δομή που δώσαμε στα μπλοκ του ευρετηρίου είναι η εξής:

* 1 integer για το global\_depth
* 1 integer για το max\_buckets
* 1 integer για το number\_of\_buckets(πόσα bucket είναι αποθηκευμένα σε αυτό το μπλοκ)
* 1 integer για το number\_of\_next\_block(ο αριθμός του επόμενου μπλοκ εάν αυτό έχει δεσμευτεί, εάν δεν έχει θα είναι -1)
* Όλος ο εναπομένοντας χώρος αφιερώνεται για την αποθήκευση των bucket του ευρετηρίου

updateDirectory: αρχικά ελέγχουμε αν χωράνε όλες οι εγγραφές σε ένα μπλοκ, εάν ναι τότε απλά τις βάζουμε στο ήδη υπάρχον μπλοκ και τερματίζουμε το πρόγραμμα. Εάν όχι, τότε βάζουμε τις πρώτες εγγραφές στο πρώτο μπλοκ που είναι δεσμευμένο για το ευρετήριο και παίρνουμε από αυτό, τον αριθμό που μας δείχνει στο επόμενο μπλοκ ευρετηρίου. Αν ο αριθμός αυτός είναι αρνητικός τότε σημαίνει ότι δεν υπάρχει επόμενο δεσμευμένο μπλοκ για το ευρετήριο άρα πρέπει να δεσμεύσουμε και να εισάγουμε εκεί τα επόμενα μπλοκ του ευρετηρίου. Η διαδικασία αυτή επαναλαμβάνεται όσες φορές χρειαστεί για να αποθηκευτούν όλα τα bucket του hashtable στη μνήμη. Αν ο αριθμός δεν είναι αρνητικός σημαίνει ότι υπάρχει επόμενο δεσμευμένο μπλοκ για το ευρετήριο. Το καλούμε από τον δίσκο και αρχίζουμε να του εισάγουμε τα bucket. Εάν και όταν γεμίσει παίρνουμε τον αριθμό του επόμενου μπλοκ και πάλι απ’ την αρχή ελέγχουμε αν είναι αρνητικός ή θετικός και επαναλαμβάνουμε μέχρι να αποθηκευτούν όλα τα bucket του ευρετηρίου.

getDirectory: παίρνουμε από τον δίσκο το πρώτο μπλοκ του αρχείου που είναι πάντα το πρώτο μπλοκ ευρετηρίου και από αυτό παίρνουμε τους πρώτους 3 ακέραιους. Ορίζουμε έναν μετρητή και μέχρι αυτός να γίνει ίσος με το σημαίνει ότι υπάρχουν buckets στη μνήμη που πρέπει να φέρουμε στην μεταβλητή HT. Τώρα σε εμφωλευμένη επανάληψη για όσα είναι τα number\_of\_buckets παίρνουμε τα buckets ένα ένα από το μπλοκ και τα βάζουμε στην HT. Στη συνέχεια παίρνουμε τον αριθμό του επόμενου block εάν είναι αρνητικός σημαίνει ότι δεν υπάρχει επόμενο block άρα και ο μετρητής θα τερματίσει την while αλλιώς φέρνουμε το επόμενο μπλοκ στην μνήμη από τον δίσκο αποδεσμεύοντας πρώτα το προηγούμενο και συνεχίζουμε μέχρι να τελειώσει η επανάληψη.

Κύριες συναρτήσεις του προγράμματος:

* HT\_Init: απλά αρχικοποιούμε τον πίνακα Open\_files
* HT\_CreateIndex: μία συνάρτηση που δημιουργεί ένα αρχείο με όνομα filename, δημιουργεί ένα αρχικά κενό ευρετήριο για αυτό με global depth ίσο με depth και κάδους κενούς. Κάνει allocate τα απαραίτητα block για να αποθηκευτούν όλα αυτά στο δίσκο και τερματίζει
* HT\_OpenIndex: ανοίγει ένα αρχείο και επιστρέφει στη μεταβλητή IndexDesc το αναγνωριστικό αυτού του αρχείου με τη βοήθεια της BF\_OpenFile. Επίσης προσθέτει αυτό το αναγνωριστικό στο Open\_files.
* HT\_CloseFile: κλείνει ένα αρχείο και αφαιρεί το indexDesc του από το Open\_files.
* ΗΤ\_InsertEntry: αρχικά παίρνει το ευρετήριο από το δίσκο ψάχνει να βρει η εγγραφή που θέλει να προσθέσει στο αρχείο σε ποιο bucket πηγαίνει και εάν έχει χώρο την αποθηκεύει εκεί. Αν δεν έχει χώρο διαλέγει αν πρέπει να κάνει expand και μετά κάνει split. Το expand έτσι όπως το έχουμε υλοποιήσει απλά διπλασιάζει τον πίνακα των buckets και βάζει στους καινούργιους buckets τις αντίστοιχες τιμές με το φιλαράκι τους, εκτός από το hascode που είναι μοναδικό για κάθε bucket. Άρα και μετά το expand η διαδικασία του split χρειάζεται να πραγματοποιηθεί. Μέσα στο split χρησιμοποιούμε ένα τύπο για να ορίσει ποιοι κουβάδες πρέπει να χωριστούν (z = j\*power(2, HT->bucket[i].local\_depth-1)) αυτός ο τύπος είναι εκεί για την περίπτωση που τα id των εγγραφών δεν είναι συνεχόμενοι ακέραιοι αλλά τυχαίοι αριθμοί οπότε μπορεί η διαφορά global\_depth-local\_depth να είναι μεγαλύτερη του 1 άρα να έχεις 4 ή και περισσότερους κουβάδες να δείχνουν στον ίδιο μπλοκ άρα πρέπει να χωρίσεις αυτούς τους 4 σε 2 ζευγάρια. Τέλος, μετά το split καλούμε αναδρομικά την insert για να ανακατανείμουμε τις ήδη υπάρχοντες, στο μπλοκ που έγινε το split, εγγραφές σύμφωνα με το hashcode τους.
* ΗΤ\_PrintAllEntries: παίρνει σαν όρισμα ένα indexDesc και ένα id( μπορεί να είναι και NULL). Τυπώνει από το αρχείο με αναγνωριστικό indexDesc είτε την εγγραφή με id εάν αυτό δεν είναι NULL. Αν είναι τότε τυπώνει όλο το ευρετήριο. Οι αναζήτηση των εγγραφών και στις δύο περιπτώσεις γίνονται με τη βοήθεια του ευρετηρίου.