Υλοποίηση Συστημάτων Βάσεων Δεδομένων
Εργασία 1

Μέλη ομάδας:

- Ιωάννης Καπετανγεώργης
  Αριθμός Μητρώου: 1115201800061

- Δημήτριος Σιταράς
  Αριθμός Μητρώου: 1115201800178

Εξάμηνο: 5ο

Δομή Αρχείων

Στο παραδοτέο directory περιέχονται:

- Το directory src το οποίο περιέχει τα αρχεία HP.c και HT.c με τις υλοποιήσεις του αρχείου σωρού και αρχείου κατακερματισμού αντίστοιχα.
- Το directory include το οποίο περιέχει τα αρχεία BF.h, HP.h και HT.h τα οποία γίνονται include από τα αντίστοιχα .c αρχεία του φακέλου src
- Το directory examples το οποίο περιέχει τις main συναρτήσεις οι οποίες επιδεικνύουν την λειτουργία του αρχείου σωρού και του αρχείου κατακερματισμού. Πιο συγκεκριμένα, το αρχείο main_HP.c επιδικνύει την λειτουργία του αρχείου σωρού και το αρχείου main_HT.c επιδεικνύει την λειτουργία του αρχείου κατακερματισμού. Οι ακριβείς λειτουργίες των αρχείων αυτών εξηγούνται αναλυτικότερα παρακάτω.
- Το directory build το οποίο περιέχει τα δύο εκτελέσιμα που δημιουργούνται έπειτα από την μεταγλώττιση. Πιο συγκεκριμένα, δημιουργείται το εκτελέσιμο demoHP το οποίο αφορά το αρχείο σωρού και το εκτελέσιμο demoHT το οποίο αφορά το αρχείο κατακερματισμού.
- Το directory lib το οποίο περιέχει τα αρχεία BF_64.a και BF_32.a που μας δόθηκαν και περιέχουν την υλοποίηση της βιβλιοθήκης BF.
- Το αρχείο Makefile μέσω του οποίου γίνεται το compilation των αρχείων. Το περιεχόμενο του Makefile εξηγείται στην επόμενη παράγραφο.

Μεταγλώττιση του προγράμματος

Μέσα στο αρχικό directory πληκτρολογούμε στο τερματικό την εντολή make (αν θελουμε να διαγραψουμε τα αρχεία που παραγονται απλά πληκτρολογούμε make clean). Έτσι, δημιουργούνται τα εκτελέσιμα demoHP και demoHT στο directory build τα οποία αφορούν το αρχείο σωρού και αρχείο κατακερματισμού αντίστοιχα.

Εκτέλεση του προγράμματος

Πηγαίνουμε στο directory /build και “τρέχουμε” το εκτελέσιμο που θέλουμε, για την προσομοίωση της λειτουργίας του σωρου “τρέχουμε” το demoHP και αντίστοιχα για την προσομοίωση της λειτουργίας του αρχείου στατικού κατακερματισμού “τρέχουμε” το demoHT.

Υλοποίηση Αρχείου Σωρού (./src/HP.c)

HP_CreateFile(...)

Η παραπάνω συνάρτηση δημιουργει και αρχικοποιεί ένα αρχείο σωρού με το αντίστοιχο όνομα που δίνεται ως όρισμα.
Αρχικά, δημιουργείται το αρχείο μέσω της BF_OpenFile. Στη συνέχεια, δεσμεύεται ένα block στο οποιο αποθηκεύουμε την επικεφαλίδα του αρχείου σωρού (HP_info). Επίσης εκτός από την επικεφαλίδα, στο πρώτο block αποθηκεύεται και ένας “pointer” στο επόμενο block του σωρού (δηλαδή το πρώτο block το οποίο περιέχει εγγραφές). Ο pointer αυτός αρχικοποιείται με -1 καθώς ακόμα δεν υπάρχουν blocks που περιέχουν εγγραφές στο αρχείο σωρού. Αποθηκεύονται οι αλλαγές στο block μέσω της BF_WriteBlock και τέλος “κλείνουμε” το αρχείο. Σε περίπτωση που τα παραπάνω εκτελεστούν επιτυχώς, η συνάρτηση επιστρέφει 0. Αντίθετα, σε περίπτωση λάθους η συνάρτηση επιστρέφει -1.

HP_OpenFile(...)

Η παραπάνω συνάρτηση απλά “ανοίγει” το ήδη δημιουργηθέν αρχείο σωρού με το όνομα που δόθηκε ως όρισμα και διαβάζει το πρώτο block το οποίο περιέχει την επικεφαλίδα του αρχείου σωρού. Στην συνέχεια δημιουργεί ένα αντίγραφο της επικεφαλίδας αυτής το οποίο και επιστρέφει. Σε περίπτωση λάθους η συνάρτηση επιστρέφει NULL.

HP_CloseFile(...)

Η παραπάνω συνάρτηση απλά “κλείνει” το ήδη δημιουργηθέν αρχείο σωρού με το όνομα που δόθηκε ως όρισμα. Επίσης, αποδεσμεύει το αντίγραφο της επικεφαλίδας το οποίο είχε δημιουργηθεί (και δεσμευθεί δυναμικά) από την HP_OpenFile. Σε περίπτωση που τα παραπάνω εκτελεστούν επιτυχώς, η συνάρτηση επιστρέφει 0. Αντίθετα, σε περίπτωση λάθους η συνάρτηση επιστρέφει -1.

HP_InsertEntry(...)

Η παραπάνω συνάρτηση εισάγει στον σωρό μια νέα εγγραφή που δίνεται ως όρισμα.
Κάθε block εγγραφών περιέχει στην αρχή έναν ακέραιο ο οποίο “κρατάει” τον αριθμό τον εγγραφών που είναι αποθηκευμένες την τρέχουσα στιγμή στο συγκεκριμένο block. Επίσης, μετά από τον ακέραιο αυτό, αποθηκεύεται ακόμα ένας ακέραιος ο οποίος λειτουργεί σαν pointer στο επόμενο block του αρχείου σωρού έτσι ώστε να δημιουργηθεί μια “αλυσίδα”.
Αρχικά ελέγχουμε αν το αρχείο σωρού είναι άδειο (δηλαδή δεν έχει κανένα block με εγγραφές). Στην περίπτωση αυτή δημιουργούμε ένα νέο block στο οποίο θα αποθηκεύσουμε την εγγραφή αυτή. Αρχικοποιούμε τον μετρητή εγγραφών σε 1, τον pointer του επόμενου block σε -1 (καθώς δεν υπάρχει επόμενο block) και τέλος αποθηκεύουμε το record έπειτα από τους δύο ακέραιους. Τέλος, ανανεώνουμε τον pointer του πρώτου block του αρχείου έτσι ώστε να “δείχνει” στο πρώτο block με εγραφών που μόλις δημιουργήσαμε. Η εισαγωγή της εγγραφής έχει ολοκληρωθεί, οπότε επιστρέφουμε τον αριθμό του block στο οποίο αποθηκεύσαμε την εγγραφή.
Σε περίπτωση που υπάρχει ήδη κάποια εγγραφή στο αρχείο σωρού, αρχικά ελέγχουμε αν η εγγραφή που θέλουμε να εισάγουμε υπάρχει ήδη μέσα στον σωρό. Ο έλεγχος αυτός γίνεται μέσω της συνάρτησης HP_GetAllEntries της οποίας η λειτουργία εξηγείται παρακάτω. Σε περίπτωση που η εγγραφή ήδη υπάρχει εκτυπώνεται το αντίστοιχο μήνυμα και επιστρέφουμε -1.
Σε περίπτωση που η εγγραφή δεν υπάρχει στον σωρό, ξεκινάμε να ανατρέχουμε ένα προς ένα τα blocks της “αλυσίδας”. Σε κάθε ένα block ελέγχουμε αν υπάρχει διαθέσιμος χώρος για να αποθηκεύσουμε την νέα εγγραφή. Σε περίπτωση που υπάρχει χώρος σε ένα ήδη υπάρχον block, αποθηκεύουμε την νέα εγγραφή μετά από τις ήδη υπάρχουσες, αυξάνουμε τον μετρητή εγγραφών που υπάρχει στην αρχή του block και τέλος επιστρέφουμε τον αριθμό του block στο οποίο μόλις αποθηκεύσαμε το νέο record. Σε περίπτωση που φτάσουμε στο τέλος της αλυσίδας και δεν έχουμε βρει διαθέσιμο χώρο για να εισάγουμε την εγγραφή, χρειάζεται να δημιουργήσουμε ένα νέο block. Έτσι, δεσμεύουμε το νέο block, αρχικοποιούμε τον μετρητή εγγραφών σε 1, αρχικοποιούμε τον pointer του επόμενου block σε -1 (καθώς είναι το τελευταίο block της αλυσίδας) και τέλος αποθηκεύουμε το record έπειτα από τους δύο ακέραιους. Επίσης στον pointer του πρώην τελευταίου block αποθηκεύουμε τον αριθμό του νέου block που δημιουργήσαμε έτσι ώστε να “συνδεθεί” στην αλυσίδα το νέο τελευταίο block. Έπειτα από όλα αυτά επιστρέφουμε τον αριθμό του νέου block που δημιουργήσαμε και στο οποίο αποθηκεύτηκε η νέα εγγραφή.

HP_GetAllEntries(...)

Η παραπάνω συνάρτηση αναζητά στο αρχείο σωρού την εγγραφή που δίνεται ως όρισμα με βάση το κλειδί της.
Ξεκινάμε, την αναζήτηση από το δεύτερο μπλοκ του σωρού καθώς στο πρώτο δεν εχουμε εγγραφές (έχουμε αποθηκεύσει εκεί το αντίστοιχο αρχείο κεφαλίδας του σωρου). Διατρέχουμε ένα προς ένα τα blocks και σε κάθε μπλοκ ελέγχουμε μια-μια τις εγγραφές που ειναι αποθηκευμένες σε αυτό προκειμένου να βρούμε την εγγραφη που δίνεται ως όρισμα. Κάθε φορά συγκρίνουμε το κλειδί της εγγραφής που ψάχνουμε με το κλειδί της εγγραφής που ελέγχουμε. Στην περίπτωση που βρούμε την εγγραφή επιστρέφουμε τον αριθμό του block στον οποίο είναι αποθηκευμένη. Σε περίπτωση που φτασουμε στο τελευταίο block, δηλαδή ο δείκτης για το επόμενο block ειναι -1, και δεν έχουμε βρει την εγγραφή που ψάχνουμε η διαδικασία της αναζήτησης σταματάει και η συνάρτηση επιστρέφει -1.

HP_DeleteEntry(...)

Η συνάρτηση αυτή ακολουθεί παρόμοια διαδικασία με την συνάρτηση HP_GetAllEntries. Πιο συγκεκριμένα διατρέχει ένα προς ένα τα blocks και σε κάθε block ελέγχει μια προς μία τις εγγραφές συγκρίνοντας το κλειδί της εγγραφής που θέλουμε να διαγράψουμε με το κλειδί της εγγραφής την οποία ελέγχουμε κάθε φορά. Μόλις βρούμε την εγγραφή την οποία θέλουμε να διαγράψουμε, μειώνουμε το recordCount κατα 1 και διαγράφουμε την εγγραφή και μετατοπίζουμε την τελευταία εγγραφή του block στην θέση της εγγραφής την οποία μόλις διαγράψαμε έτσι ώστε να καλύψουμε την “κενή θέση” που δημιουργήθηκε από την διαγραφή. Σε περίπτωση που η διαγραφή γίνει με επιτυχία επιστρέφουμε 0. Σε περίπτωση που η εγγραφή που θέλουμε να διαγράψουμε δεν υπάρχει στον σωρό ή γίνει κάποιο λάθος κατα την διαδικασία της διαγραφής επιστρέφουμε -1.

Παρατηρήσεις:

- Η δομή των records τα οποία εισάγονται στο αρχείο σωρού, δηλώνεται στο αρχείο HP.h και είναι ίδια με αυτή που περιγράφεται στην εκφώνηση της άσκησης.

- Η δομή η οποία περιέχει την επικεφαλίδα του αρχείου σωρού (HP_info) επίσης δηλώνεται στο αρχείο HP.h

- Η αρχικοποίηση της βιβλιοθήκης BF γίνεται στην αντίστοιχη main που χρησιμοποιεί τις παραπάνω συναρτήσεις (με την κλήση της συνάρτησης BF_Init()) για την δοκιμή του αρχείο σωρού.

Υλοποίηση Αρχείου Κατακερματισμού(./src/HT.c)

HT_CreateFile(...)

Η παραπάνω συνάρτηση δημιουργει και αρχικοποιεί ένα αρχείο στατικού κατακερματισμού με το αντίστοιχο όνομα που δίνεται ως όρισμα.
Αρχικά, δημιουργείται το αρχείο μέσω της BF_OpenFile. Στη συνέχεια, δεσμεύεται ένα block στο οποιο αποθηκεύουμε την επικεφαλίδα του αρχείου κατακερματισμού (HT_info) καθώς και ένας pointer στο επόμενο block το οποίο περιέχει τον πίνακα κατακερματισμού (ή μέρος αυτού) και αποθηκεύονται οι αλλαγές στο block αυτό μέσω της BF_WriteBlock.
Στην συνέχεια δημιουργούμε και αρχικοποιούμε τον πίνακα κατακερματισμού. Ο πίνακας ενδεχομένως να χρειάζεται παραπάνω από ένα block για να αποθηκευτεί. Έτσι, μέσω μιας επαναληπτικής διαδικασίας κάθε φορά δημιουργούμε ένα block και αποθηκεύουμε τον μέγιστο αριθμό blocks που χωράνε στο block αυτό. Η διαδικασία αυτή ολοκληρώνεται όταν έχουμε δημιουργήσει τον αριθμό των buckets ο οποίος δόθηκε σαν όρισμα. Προφανώς αν ο πίνακας κατακερματισμού χωράει ολόκληρος σε ένα block, απλά δημιουργούμε ένα block στο οποίο και αποθηκεύουμε τα buckets που θέλουμε.
Κάθε ένα από αυτά τα blocks αρχικά περιέχει έναν ακέραιο ο οποίος μας “δείχνει” το πόσα buckets περιέχει το block αυτό. Επίσης περιέχει έναν pointer στο επόμενο block το οποίο περιέχει τον υπόλοιπο πίνακα κατακερματισμού. Σε περίπτωση που ο πίνακας κατακερματισμού “χωράει” σε μόνο ένα block, αλλά και στην περίπτωση του block που περιέχει το τελευταίο κομμάτι του πίνακα, ο pointer αυτός περιέχει την τιμή -1.
Έπειτα από τους αριθμούς αυτούς αποθηκεύονται τα buckets. Το κάθε bucket ουσιαστικά είναι ένας ακέραιος ο οποίος μας “δείχνει” στο block το οποίο περιέχει τις εγγραφές που αντιστοιχούν στο bucket αυτό. Το κάθε bucket αρχικοποιείται με την τιμή -1 καθώς στην τρέχουσα κατάσταση δεν υπάρχει κανένα block με εγγραφές.
Σε περίπτωση που τα παραπάνω εκτελεστούν επιτυχώς, η συνάρτηση επιστρέφει 0. Αντίθετα, σε περίπτωση λάθους η συνάρτηση επιστρέφει -1.

HT_OpenIdex(...)

Η παραπάνω συνάρτηση απλά “ανοίγει” το ήδη δημιουργηθέν αρχείο κατακερματισμού με το όνομα που δόθηκε ως όρισμα και διαβάζει το πρώτο block το οποίο περιέχει την επικεφαλίδα του αρχείου κατακερματισμού. Στην συνέχεια δημιουργεί ένα αντίγραφο της επικεφαλίδας αυτής το οποίο και επιστρέφει.
Σε περίπτωση λάθους η συνάρτηση επιστρέφει NULL.

HT_CloseIndex(...)

Η παραπάνω συνάρτηση απλά “κλείνει” το ήδη δημιουργηθέν αρχείο κατακερματισμού με το όνομα που δόθηκε ως όρισμα. Επίσης, αποδεσμεύει το αντίγραφο της επικεφαλίδας το οποίο είχε δημιουργηθεί (και δεσμευθεί δυναμικά) από την HT_OpenIndex. Σε περίπτωση που τα παραπάνω εκτελεστούν επιτυχώς, η συνάρτηση επιστρέφει 0. Αντίθετα, σε περίπτωση λάθους η συνάρτηση επιστρέφει -1.

HT_InsertEntry(...)

Η παραπάνω συνάρτηση εισάγει στον πίνακα κατακερματισμού μια νέα εγγραφή που δίνεται ως όρισμα.
Κάθε block εγγραφών περιέχει στην αρχή έναν ακέραιο ο οποίος “κρατάει” τον αριθμό τον εγγραφών που είναι αποθηκευμένες την τρέχουσα στιγμή στο συγκεκριμένο block. Επίσης, μετά από τον ακέραιο αυτό, αποθηκεύεται ακόμα ένας ακέραιος ο οποίος λειτουργεί σαν pointer στο επόμενο block έτσι ώστε να δημιουργηθεί μια “αλυσίδα” για τα overflow blocks.

Αρχικά, ελέγχουμε αν η εγγραφή που θέλουμε να εισάγουμε υπάρχει ήδη μέσα στον πίνακα κατακερματισμού. Ο έλεγχος αυτός γίνεται μέσω της συνάρτησης HT_GetAllEntries της οποίας η λειτουργία εξηγείται παρακάτω. Σε περίπτωση που η εγγραφή ήδη υπάρχει εκτυπώνεται το αντίστοιχο μήνυμα και επιστρέφουμε -1. Στη συνέχεια, μέσω της συνάρτησης κατακερματισμου hash(...) (περισσότερα στις παρατηρήσεις), στην οποία δίνουμε ως όρισμα το κλειδί της εγγραφής που θέλουμε να εισάγουμε, βρίσκουμε το αντίστοιχο bucket του πινακα που πρέπει να εισαχθεί η εγγραφή. Έπειτα, αναζητουμε το block στο οποίο βρίσκεται το bucket αυτό (καθώς ο πίνακας κατακερματισμού μπορεί να είναι αποθηκευμένος σε πολλαπλά blocks). Σε περίπτωση που το bucket αυτό δεν δείχνει σε κανένα block, δηλαδή δεν περιέχει καμία εγγραφή, αρχικά χρειάζεται να δημιουργήσουμε ένα νέο block. Στο block αυτό αρχικοποιούμε τον μετρητή εγγραφών σε 1, τον pointer για το επόμενο block σε -1 (καθώς δεν υπάρχει overflow block) και τέλος αποθηκεύουμε το record έπειτα από τους δύο αυτούς ακέραιους. Τέλος, ανανεώνουμε την τιμή του bucket με τον αριθμό του block που μόλις δημιουργήσαμε, έτσι ώστε πλέον το bucket να “δείχνει” στο block αυτό. Ειδάλλως, το bucket δείχνει σε ένα block εγγραφών, το οποίο ίσως και να “δείχνει” σε μια αλυσίδα από overflow blocks. Έτσι, ξεκινάμε να ανατρέχουμε ένα προς ένα τα blocks της “αλυσίδας”. Σε κάθε ένα block ελέγχουμε αν υπάρχει διαθέσιμος χώρος για να αποθηκεύσουμε την νέα εγγραφή. Σε περίπτωση που υπάρχει χώρος σε ένα ήδη υπάρχον block, αποθηκεύουμε την νέα εγγραφή μετά από τις ήδη υπάρχουσες, αυξάνουμε τον μετρητή εγγραφών που υπάρχει στην αρχή του block και τέλος επιστρέφουμε τον αριθμό του block στο οποίο μόλις αποθηκεύσαμε το νέο record. Σε περίπτωση που φτάσουμε στο τέλος της αλυσίδας και δεν έχουμε βρει διαθέσιμο χώρο για να εισάγουμε την εγγραφή, χρειάζεται να δημιουργήσουμε ένα νέο block. Συνεπώς, δεσμεύουμε το νέο block, αρχικοποιούμε τον μετρητή εγγραφών σε 1, αρχικοποιούμε τον pointer του επόμενου block σε -1 (καθώς είναι το τελευταίο block της αλυσίδας) και τέλος αποθηκεύουμε το record έπειτα από τους δύο ακέραιους. Επίσης στον pointer του πρώην τελευταίου block αποθηκεύουμε τον αριθμό του νέου block που δημιουργήσαμε έτσι ώστε να “συνδεθεί” στην αλυσίδα το νέο τελευταίο block. Έπειτα από όλα αυτά επιστρέφουμε τον αριθμό του νέου block που δημιουργήσαμε και στο οποίο αποθηκεύτηκε η νέα εγγραφή.

HT_GetAllEntries(...)

Η παραπάνω συνάρτηση αναζητά στον πίνακα κατακερματισμού την εγγραφή που δίνεται ως όρισμα με βάση το κλειδί της.
Αρχικά, ομοίως με την συνάρτηση HT_InsertEntry καλούμε την συνάρτηση hash έτσι ώστε να βρούμε σε ποιο bucket του πίνακα κατακερματισμού βρίσκεται η εγγραφή που ψάχνουμε.
Στην συνέχεια ανατρέχουμε τα blocks στα οποία είναι αποθηκευμένο το hash table έτσι ώστε να βρούμε το bucket αυτό.
Μόλις βρούμε το bucket ελέγχουμε αν “δείχνει” σε κάποιο κάποιο block εγγραφών. Σε περίπτωση που δεν δείχνει σε κάποιο block, αυτό σημαίνει ότι δεν υπάρχει καμία εγγραφή που να αντιστοιχεί στο bucket αυτό, δηλαδή δεν υπάρχει και η εγγραφή που ψάχνουμε. Έτσι, επιστρέφουμε -1.
Σε περίπτωση που το bucket δείχνει σε ένα block, τότε ανατρέχουμε στο block αυτό. Ελέγχουμε μια προς μία τις εγγραφές αυτού του block συγκρίνοντας το κλειδί της εγγραφής που ψάχνουμε με το κλειδί της εγγραφής που ελέγχουμε κάθε φορά. Αν βρούμε την εγγραφή αυτή τότε τυπώνουμε το αντίστοιχο μήνυμα και επιστρέφουμε τον αριθμό των block τα οποία “διαβάσαμε” μέχρι να βρούμε την εγγραφή αυτή. Σε περίπτωση που δεν βρούμε την εγγραφή σε αυτό το block ακολουθούμε την ίδια ακριβώς διαδικασία επαναληπτικά για τα overflow blocks (εάν αυτά υπάρχουν). Τα blocks αυτά τα προσπελαύνουμε μέσω του "pointer" που περιέχεται σε κάθε block και “δείχνει” στο επόμενο block της αλυσίδας.
Αν ανατρέξουμε όλη την αλυσίδα χωρίς να βρούμε την εγγραφή που ψάχνουμε, σημαίνει ότι η εγγραφή δεν υπάρχει στο αρχείο κατακερματισμού. Έτσι, επιστρέφουμε -1.

HT_DeleteEntry(...)

Η συνάρτηση αυτή ακολουθεί παρόμοια διαδικασία με την συνάρτηση HT_GetAllEntries έτσι ώστε να βρούμε την εγγραφή την οποία θέλουμε να διαγράψουμε. Μόλις βρούμε την εγγραφή την οποία θέλουμε να διαγράψουμε, μειώνουμε το recordCount κατα 1, διαγράφουμε την εγγραφή και μετατοπίζουμε την τελευταία εγγραφή του block στην θέση της εγγραφής την οποία μόλις διαγράψαμε έτσι ώστε να καλύψουμε την “κενή θέση” που δημιουργήθηκε από την διαγραφή. Σε περίπτωση που η διαγραφή γίνει με επιτυχία επιστρέφουμε 0. Σε περίπτωση που η εγγραφή που θέλουμε να διαγράψουμε δεν υπάρχει στον πίνακα κατακερματισμού ή γίνει κάποιο λάθος κατα την διαδικασία της διαγραφής επιστρέφουμε -1.

HashStatistics(...)

Η συνάρτηση αυτή εκτυπώνει διάφορα στατιστικά του αρχείου κατακερματισμού με το αντίστοιχο όνομα που δίνεται ως όρισμα στην συνάρτηση.
Για τον υπολογισμό των στατιστικών αυτών χρειάζεται να ανατρέξουμε ολόκληρο τον πίνακα κατακερματισμού. Πιο συγκεκριμένα, ανατρέχουμε ένα προς ένα τα buckets του hash table, και για κάθε ένα bucket, ανατρέχουμε ένα προς ένα τα blocks τα οποία περιέχουν τις εγγραφές που αντιστοιχούν στο bucket αυτό, κρατώντας τις καταλληλες πληροφορίες που χρειαζόμαστε για τον υπολογισμό των στατιστικών που ζητούνται.
Τα στατιστικά που εκτυπώνονται είναι τα παρακάτω:

1) Τον συνολικό αριθμό των blocks που έχουν δημιουργηθεί στο αρχείο καθώς και τον αριθμό των blocks που περιέχουν εγγραφές. Για τον συνολικό αριθμό των blocks εκτυπώνουμε την τιμή που επιστρέφει η συνάρτηση BF_GetBlockCounter( ), ενώ για τον αριθμό των blocks εγγραφών αφαιρούμε από την τιμή αυτή το πρώτο block (στο οποίο είναι αποθηκευμένη η επικεφαλίδα) και τον αριθμό των blocks στα οποία είναι αποθηκευμένο το hash table.
2) Το ελάχιστο, το μέγιστο και το μέσο πλήθος εγγραφών που έχει το κάθε bucket ενός αρχείου. Αυτα τα βρίσκουμε ανατρέχοντας των hash Table, υπολογίζοντας σε κάθε bucket το πλήθος των εγγραφων που περιέχει επιλέγοντας στο τέλος τον μικρότερο και τον μεγαλύτερο αριθμό που υπολογίστηκαν. Για τον μέσο αριθμο blocks σε κάθε bucket διαιρούμε τον συνολικό αριθμό εγγραφών του αρχείου κατακερματισμού με τον αριθμό των buckets.
3) Τον μέσο αριθμό blocks που έχει καθε bucket. Το οποίο υπολογιζουμε χρησιμοποιώντας τον αριθμο blocks εγγραφων (που εχουμε υπολογίσει ηδη για το (1)) διαιρώντας τον με τον αριθμό των buckets.
4) Το πλήθος των “overflow blocks” για κάθε bucket και το συνολικό αριθμό των buckets τα οποια εχουν overflow blocks. Αυτά τα υπολογιζουμε μέσω ενός μονοδιάστατου πίνακα όπου κάθε θέση του αντιστοιχεί σε bucket. Καθώς ανατρέχουμε το Hash Table, αποθηκεύουμε κάθε φόρα για κάθε bucket τον αριθμό των overflow blocks που έχει. Φυσικά, το πρώτο block εγγραφών σε ενα bucket (και στο οποίο δείχνει το bucket) δεν θεωρείται overflow block. Τελικά, τυπώνουμε (μόνο για όσα buckets έχουν τουλάχιστον ένα overflow block) των αριθμό των blocks που περιέχεται στην θέση του πίνακα η οποία αντιστοιχεί στο κάθε bucket. Τέλος, τυπώνεται ο πλήθος των θέσεων του πίνακα που έχουν τιμή τουλάχιστον ίση με 1 το οποίο ταυτίζεται με το συνολικό αριθμό των buckets τα οποια εχουν overflow blocks.

Παρατηρήσεις:

- Η συνάρτηση κατακερματισμού που χρησιμοποιούμε είναι μια παραλλαγή της γνωστής συνάρτησης κατακερματισμού *djb2.*

- Η δομή των records τα οποία εισάγονται στο αρχείο κατακερματισμού, δηλώνεται στο αρχείο HT.h και είναι ίδια με αυτή που περιγράφεται στην εκφώνηση της άσκησης.

- Η δομή η οποία περιέχει την επικεφαλίδα του αρχείο κατακερματισμού (HP_info) επίσης δηλώνεται στο αρχείο HT.h

- Η αρχικοποίηση της βιβλιοθήκης BF γίνεται στην αντίστοιχη main που χρησιμοποιεί τις παραπάνω συναρτήσεις (με την κλήση της συνάρτησης BF_Init()) για την δοκιμή του αρχείο κατακερματισμού.

Main Συναρτήσεις

(./examples/mainHP.c και ./examples/mainHT.c)

mainHP.c

Αφότου αρχικοποιήσουμε την βιβλιοθήκη BF μέσω της συνάρτησης BF_Init(), δημιουργούμε το αρχείο σωρού μέσω της HP_CreateFile. Στην συνέχεια, ανοίγουμε το αρχείο ενδεικτικών εγγραφών που μας έχει δοθεί και εισάγουμε 500 εγγραφες στο αρχείο σωρού που δημιουργήσαμε. Έπειτα, επιλέγουμε τυχαία 15 συνεχόμενα records από αυτά που εισάγαμε και για κάθε ένα από αυτά κάνουμε:

- Ελέγχουμε αν η εγγραφή αυτή υπάρχει στο αρχείο σωρού μέσω της συνάρτησης HP_GetAllEntries() και εκτυπώνουμε το αντίστοιχο μήνυμα.
- Διαγράφουμε την εγγραφή αυτή από το αρχείο σωρού μέσω της HP_DeleteEntry() και εκτυπώνουμε το αντίστοιχο μήνυμα.
- Ελέγχουμε ξανά αν η εγγραφή υπάρχει στο αρχείο σωρού (πάλι μεσω της HP_GetAllEntries()) έτσι ώστε να διαπιστώσουμε αν η διαγραφή ήταν επιτυχής και εκτυπώνουμε το αντίστοιχο μήνυμα.

Έτσι έχουμε ελέγξει την λειτουργία όλων τον συναρτήσεων που υλοποιήσαμε, όποτε τελικά κλείνουμε το αρχείο σωρού μέσω της συνάρτησης HP_CloseFile.

mainHT.c

Αφότου αρχικοποιήσουμε την βιβλιοθήκη BF μέσω της συνάρτησης BF_Init(), δημιουργούμε το αρχείο κατακερματισμού μέσω της HT_CreateIndex. Το αρχείο κατακερματισμού έχει 100 buckets. Στην συνέχεια, ανοίγουμε το αρχείο ενδεικτικών εγγραφών που μας έχει δοθεί και εισάγουμε 1000 εγγραφες στο αρχείο κατακερματισμού. Έπειτα, επιλέγουμε τυχαία 15 συνεχόμενα records από αυτά που εισάγαμε και για κάθε ένα από αυτά κάνουμε:

- Ελέγχουμε αν η εγγραφή αυτή υπάρχει στο αρχείο σωρού μέσω της συνάρτησης HT_GetAllEntries() και εκτυπώνουμε το αντίστοιχο μήνυμα.
- Διαγράφουμε την εγγραφή αυτή από το αρχείο σωρού μέσω της HT_DeleteEntry() και εκτυπώνουμε το αντίστοιχο μήνυμα.
- Ελέγχουμε ξανά αν η εγγραφή υπάρχει στο αρχείο σωρού (πάλι μεσω της HT_GetAllEntries()) έτσι ώστε να διαπιστώσουμε αν η διαγραφή ήταν επιτυχής και εκτυπώνουμε το αντίστοιχο μήνυμα.

Στην συνέχεια κλείνουμε το αρχείο κατακερματισμού μέσω της συνάρτησης HT_CloseFile.
Τέλος καλούμε την συνάρτηση HashStatistics έτσι ώστε να εκτυπώσουμε τα στατιστικά που ζητούνται για το αρχείο κατακερματισμού.
Έτσι έχουμε ελέγξει την λειτουργικότητα όλων των συναρτήσεων που υλοποιήσαμε.