ΕΡΓΑΣΙΑ 2

ΚΑΛΟΓΕΡΟΠΟΥΛΟΣ ΔΗΜΗΤΡΙΟΣ
A.M. 1115201000058
O.S. Linux Ubuntu 17.10 64-bit
gcc 7.2.0
Η εργασία υλοποιήθηκε ατομικά

Σχετικά με το B+ δέντρο:
-Σύμβαση (σύμφωνα με όλα τα παραδείγματα που έγιναν στο μάθημα), ο μέγιστος αριθμός εγγραφών στα μπλοκ δεδομένων ισούται με το μέγιστο αριθμό δεικτών στα μπλοκ ευρετηρίου. (δηλαδή για τον παράγοντα διακλάδωσης των μπλοκ ευρετηρίου ισχύει , b = max records per block)
-Όταν ανιχνεύω εγγραφή με διπλότυπο κλειδί την αποθηκεύω σε ειδικό μπλοκ υπερχείλισης για το συγκεκριμένο κλειδί.
-Σχετικά με τη ρίζα ,το δείχτη μπλοκ υπερχείλισης και το δείχτη σε γειτονικό μπλοκ αριθμός μπλοκ -1 σημαίνει ότι το μπλοκ δεν υπάρχει .

Σχετικά με τη μορφοποίηση των μπλοκ:
-Για τα μπλοκ ευρετηρίου:
Το μπλοκ έχει στην αρχή μια επικεφαλίδα όπου, στα πρώτα sizeof(char) byte αποθηκεύται ο τύπος του μπλοκ(index ‘i’),στα επόμενα sizeof(int) byte αποθηκεύται ο αριθμός των εγγραφών που έχουν γίνει.Μετά την επικεφαλίδα το μπλοκ έχει τη μορφή <δείχτης 0> κλειδί 1 <δείχτης 1 > …. <δείχτης b>.Ισχύουν οι τύποι:

ceil(b/2) <= μέγιστος αριθμός δείκτών ανά μπλοκ <= b

και

ceil(b/2) – 1 <= μέγιστος αριθμός κλειδιών ανά μπλοκ <= b – 1

για τη μπλοκ ευρετηρίου ρίζα ισχύει:

2<= μέγιστος αριθμός δεικτών <= b

το ‘μεσσαίο’ κλειδί που επιλέγεται για να ανέβει σε παραπάνω επίπεδο σε περίπτωση σπασίματος μπλοκ είναι το ceil(b/2) .
(-1 σε όλους τους τύπους για αρίθμηση πινάκων)
-Για τα μπλοκ δεδομένων:
Το μπλοκ έχει στην αρχή μια επικεφαλίδα όπου, στα πρώτα sizeof(char) byte αποθηκεύται ο τύπος του μπλοκ(leaf ‘l’),στα επόμενα sizeof(int) byte αποθηκεύται ο αριθμός των εγγραφών που έχουν γίνει και στα επόμενα sizeof(int) byte αποθηκεύεται ο αριθμός του δεξιού γειτονικού μπλοκ .
Στη συνέχεια το μπλοκ αποτελείται από εγγραφές που έχουν προσκολήμένο στην αρχή τον αριθμό ενός ,ξεχωριστού για το κλειδί τους ,μπλοκ υπερχείλισης.Συγκεκριμένα στα πρώτα sizeof(int) bytes αποθηκεύται ο αριθμός μπλοκ υπερχείλισης ,στα επόμενα attrLength1 bytes αποθηκεύται το κλειδί της εγγραφής και στα επόμενα attrLength2 bytes αποθηκεύται το δεύτερο πεδίο της εγγραφής.Ισχύουν οι τύποι:

ceil(b/2) <= μέγιστος αριθμός εγγραφών ανά μπλοκ <= b

για το μπλοκ δεδομένων ρίζα(το πρώτο μπλοκ του δέντρου είναι η πρώτη ρίζα) ισχύει:

1 <= μέγιστος αριθμός εγγραφών <= b

Η ‘μεσσαία’ εγγραφή της οποίας το κλειδί επιλέγεται για να ανέβει σε παραπάνω επίπεδο σε περίπτωση σπασίματος μπλοκ είναι η ceil(b/2) + 1 .
(-1 σε όλους τους τύπους για αρίθμηση πινάκων)
-Για το μπλοκ υπερχείλισης:
Το μπλοκ έχει στην αρχή μια επικεφαλίδα όπου, στα πρώτα sizeof(char) byte αποθηκεύται ο τύπος του μπλοκ(overflow ‘o’),στα επόμενα sizeof(int) byte αποθηκεύται ο αριθμός των εγγραφών που έχουν γίνει.Στη συνέχεια το μπλοκ αποτελείται από εγγραφές .Συγκεκριμένα στα πρώτα

attrLength1 bytes αποθηκεύται το κλειδί της εγγραφής και στα επόμενα attrLength2 bytes αποθηκεύται το δεύτερο πεδίο της εγγραφής.Δε γίνεται έλεγχος για full μπλοκ γιατί θεωρώ (από εκφώνηση) ότι οι διπλότυπες εγγραφές χωρούν άνετα σε ένα μπλοκ.Επιπλέον οι εγγραφές σε μπλοκ υπερχείλισης είναι αταξινόμητες.(δηλαδή μπαίνουν στην πρώτη άδεια θέση)
-Για το block 0:
Το πρώτο μπλοκ κάθε αρχείου δεσμεύται για να αποθηκεύει metadata σχετικά με το B+ δέντρο που αντιστοιχεί στο αρχείο.Συγκεκριμένα αποθηκεύονται οι παρακάτω πληροφορίες σε σειρά

|int root block number | char attrType1|int attrLength1|char attrType1|int attrLength2|int max number of records per block| υπόλοιπος / κενός χώρος του μπλοκ |

Αν δημιουργηθεί νέα ρίζα τότε πρέπει να ενημερωθεί η αντίστοιχη καταχώρηση στο μπλοκ 0.

---Αρχείο mystructs.h:

Ορίζονται μέσω define τα μεγέθη των πινάκων ,με καταχωρήσεις για ανοίγματα αρχείων και ενεργά scans.
Δηλώνονται οι πίνακες ως extern προκειμένου να γίνεται να χρησιμοποιηθούν από συναρτήσεις σε άλλα αρχεία.
Ορίζονται οι τύποι :

-BLOCK_NUM ,χρησιμοποιείται ως δομή/κόμβος μιας στοίβας.Περιέχει τον αριθμό ενός μπλοκ και δείχτη στον επόμενο κόμβο της στοίβας(κατώτερο).

-PATH, είναι δομή που εξυπηρετεί ως κεφαλή της στοίβας.Περιέχει τον αριθμό των κόμβων στη στοίβα και ένα δείκτη στο κορυφαίο στοιχείο της.

-INDEX_DATA , δομή της καταχώρησης στον πίνακα ανοιχτών αρχείων.Κατά τη δημιουργία μιας καταχώρησης αντιγράφονται όλα τα δεδομένα από το block 0 και αρχικοποιούνται τα υπόλοιπα μέλη.Το πεδίο fileDesc αποθηκεύει το πραγματικό αναγνωριστικό του αρχείου που ανοίχτηκε ,έτσι όπως το επέστρεψε η BF_OpenFile.Το πεδίο open_scans καταγράφει τον αριθμό τον ανοιχτών scan πάνω στη συγκεκριμένη καταχώρηση του αρχείου.

-SCAN_DATA, δομή της καταχώρησης στον πίνακα ανοιχτών σαρώσεων.Κατά τη δημιουργία μιας καταχώρησης αποθηκεύεται ο αριθμός του πρώτου μπλοκ από το οποίο θα ξεκινήσει μια σάρωση,η τιμή και ο τελεστής που δίνονται ,αλλά και το αναγνωριστικό της καταχώρησης ανοίγματος αρχείου πάνω στην οποία θα τρέξει η σάρωση και αρχικοποιούνται τα υπόλοιπα πεδία.Το πεδίο overflow_block περιέχει τον αριθμό ενός μπλοκ υπερχείλισης και υποδεικνύει αν έχει βρεθεί εγγραφή που ικανοποιεί τους όρους της αναζήτησης και έχει δείκτη σε μπλοκ υπερχείλισης,ουσιαστικά προτρέπει την αναζήτηση να σαρώσει όλο το μπλοκ υπερχείλισης πρωτού συνεχίσει να σαρώνει το μπλοκ δεδομένων.Περιέχει -1 αν δεν υπάρχει τέτοιο μπλοκ υπερχείλισης.Τα πεδία next_record και next_overflow_record παραπέμπουν στη σχετική θέση ,μέσα στο μπλοκ, της επόμενης εγγραφής που πρέπει να σαρωθεί.(π.χ. 0 για τη πρώτη εγγραφή ενός μπλοκ)

Επίσης ορίζεται ο τύπος AME_ErrorCode που καθορίζει τους κωδικούς σφαλμάτων.

---Αρχείο mystructs.c:

Περιέχει τις υλοποιήσεις των συναρτήσεων μελών των δομών που ορίστικαν στο αρχείο επικεφαλίδα.

Ορίζονται οι πίνακες Open_files[OPEN_FILES_SIZE] , Open_scans[OPEN_SCANS_SIZE]

είναι πίνακες δεικτών στους αντίστοιχους τύπους καταχωρήσεων(που αναφέρθηκαν πιο πάνω)

και τα μεγέθη τους ορίζονται μέσω define στο αρχείο mystructs.h

---Αρχείο myfunctions.h:

Κάνει include το mystructs.h .

Δηλώνει ως εξωτερικές τις global μεταβλητές που περιέχουν το μέγεθος της επικεφαλίδας του κάθε τύπου μπλοκ,προκειμένου να γίνεται να χρησιμοποιηθούν από συναρτήσεις σε άλλα αρχεία.

Περιέχει τις δηλώσεις όλων των συναρτήσεων που υλοποίησα.

---Αρχείο myfunctions.c:

Ορίζονται οι global μεταβλητές overflow_header_size , index_header_size και leaf_header_size και

περιέχουν τα αντίστοιχα μεγέθη επικεφαλίδων.

Ορίζονται όλες οι συναρτήσεις που υλοποίησα:

--int size_error(char type,int len) , ελέγχει αν το μέγεθος αντιστοιχεί στον τύπο που δίνεται ,αν το μέγεθος είναι > 0 και αν ο τύπος είναι έγκυρος.Επιστρέφει -1 σε περίπτωση που εντοπίσει μη έγκυρο τύπο ή μέγεθος ,αλλιώς 0.

--int descend_tree(int fileDesc,void *value1,PATH *path),κατάβαση του δέντρου αρχίζοντας από τη ρίζα.Ψάχνει κάθε μπλοκ για κλειδί μεγαλύτερο από το value1 .Αν βρεθεί διαλέγει τον αριστερό δείχτη , εκείνου του κλειδιού ,για κατάβαση,αλλιώς επιλέγει τον δεξιότερο δείκτη όλου του μπλοκ.Σταματάει όταν βρει μπλοκ δεδομένων.Ωθεί στη στοίβα του path όλους τους κόμβους από τους οποίους περνάει ο έλεγχος.Αν υπάρχει μόνο ένα μπλοκ δεδομένων το path θα περιέχει μόνο ένα στοιχείο .Αν το δέντρο είναι κενό(root_block = -1) τότε θα είναι κενό το path.Επιστρέφει -1 σε περίπτωση λάθους διαφορετικά 0.

--int BlockCreate_Leaf(int fileDesc,BF_Block *block,int *block_no),

int BlockCreate_Index(int fileDesc,BF_Block *block,int *block_no),

int BlockCreate_Overflow(int fileDesc,BF_Block *block,int *block_no),

κάνουν allocate χώρο για ένα block και το μορφοποιούν ανάλογα τον τύπο,αρχικοποιώντας την επικεφαλίδα .Αποθηκεύουν στη μεταβλητή block_no τον αριθμό του block που δημιουργήθηκε.Επιστρέφουν -1 σε περίπτωση λάθους ,αλλιώς 0.

--float compare_key(void* value1,void* key,char type), συγκρίνει τα value1 και key με βάση τον τύπο τους.Επιστρέφει αποτέλεσμα που είναι μεγαλύτερο του μηδενός αν το value1 είναι μεγαλύτερο από το key , μικρότερο του μηδενός αν το value1 είναι μικρότερο από το key , και ίσο με το μηδέν αν οι τιμές τους είναι ίσες.

--int get_update_entries(char* data,int inc),προσθέτει στον αριθμό εγγραφών το inc και επιστρέφει τον ενημερωμένο αριθμό εγγραφών.(για inc = 0 απλώς επιστρέφει τον αριθμό εγγραφών)

--int leaf_insert(int fileDesc,void* value1,void* value2,PATH* path,int *new_block_no,void *index_value),ελέγχει αν το path περιέχει τουλάχιστον ένα στοιχείο,αν όχι τότε δημιουργεί ρίζα – μπλοκ δεδομένων.Αλλιώς παίρνει το block από το path και ελέγχει αν υπάρχει διπλότυπη εγγραφή ,οπότε και εισάγει την τρέχουσα εγγραφή σε μπλοκ υπερχείλισης.Αν δεν υπάρχει διπλότυπη εγγραφη τότε ελέγχει αν το μπλοκ είναι γεμάτο.Αν ναι ,τότε δημιουργεί νέο μπλοκ, θέτει τη μεταβλητή split = 1 και καλεί τη συνάρτηση Leaf_split_push προκειμένου να εισάγει την τρέχουσα εγγραφή ,μοιράζοντας τις υπόλοιπες.Αν δεν είναι γεμάτο τότε καλεί την Leaf_push και εισάγει την τρέχουσα εγγραφή κάνοντας τις αντίστοιχες μετακινήσεις.Αν έγινε split αποθηκεύει στη μεταβλητή index_value ,τη τιμή του ‘μεσσαίου’ κλειδιού που θα πρέπει να εισαχθεί στο parent block του παραπάνω επιπέδου , και στη μεταβλητή new_block_no τον αριθμό του νέου μπλοκ που δήμιουργήθηκε και πρέπει να συνδεθεί με το parent block.Επιστρέφει -1 σε περίπτωση λάθος ,αλλιώς 1 αν έγινε split και 0 αν όχι.

--int Leaf_push(int fileDesc,char* data,void* carried_value,void *carried_value2), ψάχνει την πρώτη εγγραφή που έχει κλειδί μεγαλύτερο από το carried_value και τοποθετεί στη θέση της την τρέχουσα εγγραφή .Στην συνέχεια,κάνει το ίδιο για κάθε επόμενο carried_value ,μετακινώντας ,ουσιαστικά ,τις υπόλοιπες εγγραφές μία θέση δεξιά.

--int Leaf_split_push(int fileDesc,char* old_data,char* new_data,void* carried_value,void *carried_value2,void *index_value),

Παρόμοια με Leaf_push με τη διαφορά ότι όταν συναντήσει το ‘μεσσαίο στοιχείο’ το αποθηκεύει στο index_value ,και από εκεί και μετά αποθηκεύει τις υπόλοιπες εγγραφές στο νέο μπλοκ.

--int check_duplicate(int fileDesc,char *data,void *value1 ,void *value2),ελέγχει αν υπάρχει διπλότυπη εγγραφή στο buffer.Αν υπάρχει, τότε καλεί την overflow_push προκειμένου να αποθηκεύσει την εγγραφή στο μπλοκ υπερχείλισης που της αντιστοιχεί.Επιστρέφει 1 αν βρει διπλότυπη εγγραφή και 0 σε κάθε άλλη περίπτωση.

--int overflow_push(int fileDesc,char* leaf_data,int position,void* value1,void* value2),

τσεκάρει αν υπάρχει μπλοκ υπερχείλισης που αντιστοιχεί στο κλειδί της εγγραφής ,αν δεν υπάρχει το δημιουργεί.Ψάχνει την πρώτη κενή θέση και αποθηκεύει την εγγραφή.Σε περίπτωση λάθους επιστρέφει -1.

--int index_insert(int fileDesc,void *index_value,PATH *path,int new_block_no),

παρόμοια με την leaf_insert.Εισάγει το index_value στο parent block του επόμενου επιπέδου

εως ότου να συμβεί split (δηλαδή να μη υπάρχει άλλο index value για να εισαχθεί πιο πάνω) ή το path να αδειάσει(δηλαδή έγινε split και η ρίζα).

Αν το path είναι εξαρχής κενό ,ή γίνει κενό κατά τη διαδικασία ,δημιουργείται νέα ρίζα-μπλοκ ευρετηρίου καλώντας την Index_make_root

Επιστρέφει -1 αν γίνει κάποιο λάθος ,διαφορετικά 0.

--int Index_push(int fileDesc,char* data,void* carried_value,int carried_right),ψάχνει το πρώτο κλειδί που είναι μεγαλύτερο από το carried_value και τοποθετεί στη θέση του το carried_value αντικαθιστώντας και τον αντίστοιχο δεξιό του δείκτη(που είναι ο αριθμός του μπλοκ που δημιουργήθηκε στο προηγούμενο επίπεδο εξαιτίας του split) .Στην συνέχεια,κάνει το ίδιο για κάθε

επόμενο carried_value ,μετακινώντας ,ουσιαστικά ,τα υπόλοιπα ζευγάρια <κλειδί,δείχτης> μία θέση δεξιά.

--Index_split_push(int fileDesc,char* old_data,char* new_data,void* carried_value,int carried_right,void *index_value),
Παρόμοια με Index_push με τη διαφορά ότι όταν συναντήσει το ‘μεσσαίο κλειδί’ το αποθηκεύει στο index_value με τον δεξιό του δείκτη να γίνεται ο πρώτος δείκτης του νέου block που δημιουργήθηκε από το split,(το μεσσαίο στοιχείο κατ’ουσίαν διαγράφεται από αυτο το επίπεδο)και μετά αποθηκεύει τα υπόλοιπα ζευγάρια στο νέο μπλοκ.

--int Index_make_root(int fileDesc,void* index_value,int new_block_no),φτιάχνει νέα ρίζα-μπλοκ ευρετηρίου με πρώτο δείκτη τον αριθμό της προηγούμενης ρίζας και εισάγοντας το ζευγάρι <index_value,new_block_no>(που προέρχονται από το πιο κάτω επίπεδο).Επιστρέφει -1 σε περίπτωση λάθους ,αλλιώς 0.

--void *find_next_leaf_entry(int scanDesc),σαρώνει τα μπλοκ δεδομένων μέχρι να βρει match ή μέχρι να μην υπάρχει άλλο μπλοκ.Οι τελεστές ομαδοποιούνται ως εξής

-EQUAL:1 , 5 , 6

-LESS OR NOT EQUAL : 2, 3, 5

-GREATER OR NOT EQUAL: 2,4,6

Αν ικανοποιηθεί μία από τις παραπάνω συνθήκες ,τότε η εγγραφή είναι match .Αν η εγγραφή έχει overflow block ενημερώνω το πεδίο της αντίστοιχης καταχώρησης της σάρωσης.Αντίστοιχα ενημερώνω και το επόμενο μπλοκ δεδομένων και την επόμενη εγγραφή για σάρωση. Επιστρέφει την τιμή του δεύτερου πεδίου της εγγραφής που βρέθηκε ,αλλιώς NULL σε περίπτωση λάθους η σε περίπτωση που δεν υπάρχει άλλο μπλοκ για σάρωση.

--void *find_next_overflow_entry(int scanDesc),επιστρέφει την τιμή του δεύτερου πεδίου της επόμενης εγγραφής που θα συναντήσει μέσα στο μπλοκ υπερχείλισης.Αν πρόκειται για την τελευταία εγγραφή που υπάρχει στο μπλοκ ,θέτει το πεδίο overflow_block σε -1 προκειμένου η σάρωση να συνεχίσει σε μπλοκ δεδομένων(από εκεί που είχε μείνει).Σε περίπτωση λάθους επιστρέφει NULL.

Αρχείο AM.c:

--void AM_Init(),

Αρχικοποιεί το επίπεδο BF.Αρχικοποιεί τους πίνακες καταχωρήσεων σε NULL τιμές.

--int AM_CreateIndex(char *fileName,char attrType1, int attrLength1, char attrType2, int attrLength2),

Ελέγχει την ορθότητα των ορισμάτων,δημιουργεί το αρχείο ,αρχικοποιεί το πρώτο μπλοκ με τις απαραίτητες πληροφορίες για το δέντρο(αναφέρθηκαν πιο πάνω).Επιστρέφει -1 σε περίπτωση λάθους , αλλιώς 0.

--int AM_DestroyIndex(char *fileName),ελέγχει την ύπαρξη του αρχείου ή αν είναι ανοιχτό

και στη συνέχεια βρίσκει το μονοπάτι του και το διαγράφει.Επιστρέφει -1 σε περίπτωση λάθους ,διαφορετικά 0.

--int AM_OpenIndex (char *fileName) ,ανοίγει ένα υπάρχον αρχείο και δημιουργεί μιά αντίστοιχη καταχώρηση στον πίνακα ανοικτών αρχείων(αν υπάρχει θέση).Επιστρέφει -1 σε περίπτωση λάθους διαφορετικά επιστρέφει τη θέση της καταχώρησης στον πίνακα.

--int AM_CloseIndex (int fileDesc),αφαιρεί και διαγράφει μία υπάρχουσα καταχώρηση στη θέση fileDesc του πίνακα καταχωρήσεων,εφόσον δεν τρέχει καμία σάρωση πάνω της.

--int AM_InsertEntry(int fileDesc, void *value1, void *value2) ,καλεί την descend_tree

για να φτιάξει ένα path από τη ρίζα ως το ζητούμενο μπλοκ δεδομένων ,καλεί την leaf_insert με το path, προκειμένου να εισάγει την εγγραφή στο μπλοκ δεδομένων(ο αριθμός του οποίου βρίσκεται στην κορυφή της στοίβας του path) , η οποία της επιστρέφει ένα index value , και ένα αριθμό νέου μπλοκ.Αν έγινε διαχωρισμός στο επίπεδο μπλοκ δεδομένων καλείται η index insert για να εισάγει το index value και τον αριθμό στο γονικό μπλοκ ευρετηρίου(νέα κορυφή του path) συνεχίζοντας τη διαδικασία εισαγωγής όπως περιγράφεται πιο πάνω.Αν δεν έγινε διαχωρισμός στο επίπεδο μπλοκ δεδομένων η index insert δε καλείται.Επιστρέφει -1 σε περίπτωση λάθους και 0 διαφορετικά.

--int AM_OpenIndexScan(int fileDesc, int op, void *value),δημιουργεί μία καταχώρηση στον πίνακα σαρώσεων , που αντιστοιχεί στη θέση fileDesc του πίνακα ανοιχτών αρχείων.Για να βρεί το πρώτο μπλοκ δεδομένων από το οποίο θα ξεκινήσει η σάρωση , καλεί την descend_tree ,εφόσον ο τελεστής αναζήτησης είναι 1 ,4 ,6.Σε διαφορετική περίπτωση η αναζήτηση ξεκινά από το μπλοκ με αριθμό 1 (το αριστερότερο μπλοκ δεδομένων, το πρώτο μπλοκ του δέντρου και το δεύτερο μπλοκ του αρχείου).Επιστρέφει -1 σε περίπτωση λάθους ,διαφορετικά επιστρέφει τη θέση της εγγραφής στον πίνακα σαρώσεων.

--void *AM_FindNextEntry(int scanDesc),

καλεί την find_next_overflow_entry αν έχει βρεθεί overflow block.Αλλιώς, καλεί την find_next_leaf_entry για να σαρώσει το μπλοκ και την εγγραφή που υποδεικνύονται από τα πεδία της καταχώρησης.Επιστρέφει NULL σε περίπτωση που δεν υπάρχει άλλο μπλοκ για να διαβαστεί ή σε περίπτωση λάθους.Διαφορετικά επιστρέφει την διεύθυνση στην οποία είναι αποθηκευμένη η τιμή του δεύτερου πεδίου της εγγραφής που βρέθηκε.

--int AM_CloseIndexScan(int scanDesc) ,αφαιρεί μία υπάρχουσα καταχώρηση στη θέση scanDesc του πίνακα σαρώσεων

--void AM_PrintError(char *errString),εκτυπώνει μήνυμα σύμφωνα με τον κωδικό λάθους που περιέχει το AM_errno