Πληροφορική & Τηλεπικοινωνίες

Κ18 - Υλοποίηση Συστημάτων Βάσεων Δεδομένων
Χειμερινό Εξάμηνο 2015 – 2016

Ι. Ιωαννίδης

Άσκηση 2: Υλοποίηση Ευρετηρίου B+ Δένδρου
Προθεσμία: 22 Δεκεμβρίου 2015, 11:59μμ

Για την άσκηση αυτή, θα υλοποιήσετε μία μέθοδο προσπέλασης βασισμένη στα B+ δένδρα που υποστηρίζει τη διεπαφή που περιγράφεται παρακάτω. Θα οργανώσετε ένα αρχείο δύο πεδίων σε B+ δένδρα, χρησιμοποιώντας το πρώτο πεδίο του αρχείου ως το πεδίο-κλειδί. Κάθε αρχείο που αντιστοιχεί σε ένα B+ δένδρο θα αποτελείται από block, τα οποία θα οργανώσετε εσείς κατάλληλα ώστε να έχουν την απαιτούμενη δομή για να λειτουργήσουν ως τέτοια. Κάθε block του αρχείου έχει μέγεθος BLOCKSIZE = 1024 bytes.

Η εργασία είναι ομαδική 2 ή 3 ατόμων. Η γλώσσα υλοποίησης είναι η C ή C++ (χωρίς χρήση STL και strings).

Συναρτήσεις Επιπέδου Ευρετηρίου AM (Access Method)

AM_Init()

void AM_Init(void)

Η ρουτίνα αυτή χρησιμοποιείται για να αρχικοποιήσετε τις όποιες καθολικές (global) εσωτερικές δομές δεδομένων αποφασίσετε ότι χρειάζεστε να έχετε για την χρήση του B+ δένδρου. Δεν έχει καμία παράμετρο εισόδου και δεν παράγει καμία έξοδο.

AM_CreateIndex()

```
int AM_CreateIndex(
   char  *fileName,     /* όνομα αρχείου */
   char   attrType1,    /* τύπος πρώτου πεδίου: 'c' (συμβολοσειρά), 'i' (ακέραιος), 'f' (πραγματικός) */
   int    attrLength1,  /* μήκος πρώτου πεδίου: 4 για 'i' ή  'f', 1-255 για 'c' */
   char   attrType2,    /* τύπος δεύτερου πεδίου: 'c' (συμβολοσειρά), 'i' (ακέραιος), 'f' (πραγματικός) */
   int    attrLength2   /* μήκος δεύτερου πεδίου: 4 για 'i' ή  'f', 1-255 για 'c' */
)
```

Η ρουτίνα αυτή δημιουργεί ένα αρχείο με όνομα fileName, βασισμένο σε ευρετήριο B+ δένδρου. Το αρχείο δεν πρέπει να υπάρχει ήδη. Ο τύπος και το μήκος του πρώτου πεδίου (αυτού που

χρησιμοποιείται για την εισαγωγή στο B+ δένδρο ως κλειδί) περιγράφονται από την δεύτερη και την τρίτη παράμετρο, αντίστοιχα. Παρόμοια, ο τύπος και το μήκος του δεύτερου πεδίου περιγράφονται από την τέταρτη και την πέμπτη παράμετρο, αντίστοιχα. Η ρουτίνα επιστρέφει AME_OK εάν επιτύχει, αλλιώς κάποιον κωδικό λάθους.

AM_DestroyIndex()

```
int AM_DestroyIndex(
   char   *fileName      /* όνομα αρχείου */
)
```

Η ρουτίνα αυτή καταστρέφει το αρχείο με όνομα fileName, διαγράφοντας το φυσικό αρχείο από το δίσκο. Ένα αρχείο δε μπορεί να διαγραφεί αν υπάρχουν ενεργά ανοίγματα σε αυτό (δείτε στη συνέχεια). Επιστρέφει AME_OK εάν επιτύχει, αλλιώς κάποιον κωδικό λάθους.

AM_OpenIndex()

```
int AM_OpenIndex (
   char   *fileName      /* όνομα αρχείου */
)
```

Η ρουτίνα αυτή ανοίγει το αρχείο με όνομα fileName. Εάν το αρχείο ανοιχτεί κανονικά, η ρουτίνα επιστρέφει έναν μικρό, μη αρνητικό ακέραιο, ο οποίος χρησιμοποιείται για να αναγνωρίζεται το αρχείο (όπως περιγράφουμε παρακάτω). Σε διαφορετική περίπτωση, επιστρέφει κάποιον κωδικό λάθους.

Θα πρέπει να κρατάτε στην μνήμη έναν πίνακα για όλα τα ανοιχτά αρχεία. Ο ακέραιος που επιστρέφει η *AM_OpenIndex* είναι η θέση του πίνακα που αντιστοιχεί στο αρχείο που μόλις ανοίχτηκε. Σ' αυτόν τον πίνακα θα κρατάτε οτιδήποτε σχετικό κρίνετε ότι πρέπει να είναι άμεσα διαθέσιμο για κάθε ανοιχτό αρχείο (π.χ., γενικές πληροφορίες για το αρχείο, το αναγνωριστικό του αρχείου έτσι όπως έχει ανοιχθεί από το λειτουργικό σύστημα, κτλ. - χωρίς τα παραπάνω να θεωρούνται απαραίτητα). Το ίδιο αρχείο μπορεί να ανοιχτεί πολλές φορές και για κάθε άνοιγμα καταλαμβάνει διαφορετική θέση στον πίνακα που κρατάτε στη μνήμη. Μπορείτε να υποθέσετε ότι οποιαδήποτε στιγμή δεν θα υπάρχουν περισσότερα από MAXOPENFILES = 20 ανοιχτά αρχεία.

AM_CloseIndex()

```
int AM_CloseIndex (
   int    fileDesc       /* αριθμός που αντιστοιχεί στο ανοιχτό αρχείο */
)
```

Η ρουτίνα αυτή κλείνει το αρχείο που υποδεικνύεται από την παράμετρό της. Επίσης σβήνει την καταχώρηση που αντιστοιχεί στο αρχείο αυτό στον πίνακα ανοιχτών αρχείων. Για να κλείσει επιτυχώς το αρχείο που υποδεικνύεται από το fileDesc, θα πρέπει να μην υπάρχουν ανοιχτές σαρώσεις σε αυτό.
Η συνάρτηση επιστρέφει AME_OK εάν το αρχείο κλείσει επιτυχώς, αλλιώς κάποιον κωδικό λάθους.

AM_InsertEntry()

```
int AM_InsertEntry(
   int     fileDesc,      /* αριθμός που αντιστοιχεί στο ανοιχτό αρχείο */
   void    *value1,       /* τιμή του πεδίου-κλειδιού προς εισαγωγή */
   void    *value2        /* τιμή του δεύτερου πεδίου της εγγραφής προς εισαγωγή */
)
```

Η ρουτίνα αυτή εισάγει το ζευγάρι (value1, value2) στο αρχείο που υποδεικνύεται από την παράμετρο fileDesc. Η παράμετρος value1 δείχνει στην τιμή του πεδίου-κλειδιού που εισάγεται στο αρχείο και η παράμετρος value2 αντιπροσωπεύει το άλλο πεδίο της εγγραφής. Η ρουτίνα επιστρέφει AME_OK εάν επιτύχει, αλλιώς κάποιον κωδικό λάθους.

AM_OpenIndexScan()

```
int AM_OpenIndexScan(
   int     fileDesc,      /* αριθμός που αντιστοιχεί στο ανοιχτό αρχείο */
   int     op,            /* τελεστής σύγκρισης */
   void    *value         /* τιμή του πεδίου-κλειδιού προς σύγκριση */
)
```

Η ρουτίνα αυτή ανοίγει μία σάρωση (αναζήτηση) του αρχείου που υποδεικνύεται από την παράμετρο fileDesc. Η σάρωση έχει σκοπό να βρει τις εγγραφές των οποίων οι τιμές στο πεδίο-κλειδί του αρχείου ικανοποιούν τον τελεστή σύγκρισης *op* σε σχέση με την τιμή που δείχνει η παράμετρος *value*. Οι διάφοροι τελεστές σύγκρισης κωδικοποιούνται ως εξής:

- **1** EQUAL (πεδίο-κλειδί == τιμή της value)
- **2** NOT EQUAL (πεδίο-κλειδί != τιμή της value)
- **3** LESS THAN (πεδίο-κλειδί < τιμή της value)
- **4** GREATER THAN (πεδίο-κλειδί > τιμή της value)
- **5** LESS THAN or EQUAL (πεδίο-κλειδί <= τιμή της value)
- **6** GREATER THAN or EQUAL (πεδίο-κλειδί >= τιμή της value)

Η ρουτίνα επιστρέφει έναν μη αρνητικό ακέραιο που αντιστοιχεί σε μία θέση κάποιου πίνακα που πρέπει να υλοποιήσετε και να κρατάτε στη μνήμη ενήμερο σχετικά με όλες τις σαρώσεις αρχείων που είναι ανοιχτές κάθε στιγμή. Πιθανές πληροφορίες που χρειάζεται κανείς να κρατάει για κάθε ανοιχτή σάρωση σ' αυτόν τον πίνακα είναι το αναγνωριστικό της τελευταίας εγγραφής που διαβάστηκε, ο αριθμός που αντιστοιχεί στο ανοιχτό αρχείο που σαρώνεται, κτλ. (Θα πρέπει να σχεδιάσετε το περιεχόμενο κάθε καταχώρησης στον πίνακα αυτό με βάση αυτά που θα δείτε ότι πρέπει να ξέρετε.) Μπορείτε να υποθέσετε ότι δεν θα υπάρχουν ποτέ πάνω από MAXSCANS = 20 ταυτόχρονες ανοιχτές σαρώσεις αρχείων. Εάν ο πίνακας σαρώσεων είναι γεμάτος, τότε η ρουτίνα επιστρέφει κάποιον κωδικό λάθους. Αντίστοιχους κωδικούς λαθών επιστρέφετε και σε άλλες περιπτώσεις κακής λειτουργίας.

AM_FindNextEntry()

```
void *AM_FindNextEntry(
   int    scanDesc       /* αριθμός που αντιστοιχεί στην ανοιχτή σάρωση */
)
```

Η ρουτίνα αυτή επιστρέφει την τιμή του δεύτερου πεδίου της επόμενης εγγραφής που ικανοποιεί την συνθήκη που καθορίζεται για την σάρωση που αντιστοιχεί στο scanDesc. Αν δεν υπάρχουν άλλες εγγραφές επιστρέφει NULL και θέτει τη global μεταβλητή *AM_errno* σε AME_EOF. Αν συμβεί κάποιο σφάλμα, επιστρέφει NULL και θέτει τη global μεταβλητή *AM_errno* στον κατάλληλο κωδικό λάθους.

AM_CloseIndexScan()

```
int AM_CloseIndexScan(
   int    scanDesc       /* αριθμός που αντιστοιχεί στην ανοιχτή σάρωση */
)
```

Η ρουτίνα αυτή τερματίζει μία σάρωση ενός αρχείου και σβήνει την αντίστοιχη καταχώρηση από τον πίνακα ανοιχτών σαρώσεων. Επιστρέφει AME_OK εάν επιτύχει, αλλιώς κάποιον κωδικό λάθους.

AM_PrintError()

```
void AM_PrintError(
   char    *errString            /* κείμενο για εκτύπωση */
)
```

Η ρουτίνα τυπώνει το κείμενο που δείχνει η παράμετρος errString και μετά τυπώνει το μήνυμα που αντιστοιχεί στο τελευταίο σφάλμα που προέκυψε από οποιαδήποτε από τις ρουτίνες του AM επιπέδου. Για τον σκοπό αυτό, η ρουτίνα αυτή χρησιμοποιεί μία καθολική (global) μεταβλητή *AM_errno* η οποία αποθηκεύει πάντα τον κωδικό του πλέον πρόσφατου σφάλματος. Ο κωδικός αυτός σφάλματος πρέπει πάντα να ενημερώνεται σωστά σε όλες τις άλλες ρουτίνες. Η ρουτίνα αυτή δεν έχει δική της τιμή επιστροφής.

Σχόλια για την Υλοποίηση

Εσείς θα αποφασίσετε πώς να δομήσετε εσωτερικά κάθε μπλοκ (κεφαλίδα, ζευγάρια τιμής-δείκτη, κτλ.), τόσο για τα εσωτερικά μπλοκ όσο και για τα μπλοκ-φύλλα του δένδρου. Τα B+ δένδρα που θα υλοποιήσετε θα πρέπει να καλύπτουν την περίπτωση που πολλά ζευγάρια (*κλειδί, δείκτης*) να έχουν την ίδια τιμή στο κλειδί τους. Μπορείτε όμως να υποθέσετε ότι, για οποιαδήποτε τιμή, τα ζευγάρια με την τιμή αυτή θα είναι λίγα και θα χωρούν σε ένα μπλοκ.

Επίσης, θα πρέπει να υλοποιήσετε πλήρως τους αλγορίθμους αναζήτησης και εισαγωγής που υπάρχουν για τα B+ δέντρα, συμπεριλαμβανομένης της διάσπασης κόμβων στη διάρκεια εισαγωγής δεδομένων.

Σχολιασμός, Έλεγχος Σφαλμάτων, και Γενική Μορφοποίηση

Όπως πάντοτε, αναμένεται καλός σχολιασμός του προγράμματος, και εσωτερικός (ανάμεσα στις γραμμές κώδικα) και εξωτερικός (στην αρχή κάθε ρουτίνας). Ένας γενικός κανόνας είναι να σχολιάζετε τα προγράμματά σας σαν να πρόκειται να τα δώσετε σε κάποιον άλλον ο οποίος θα τα επεκτείνει και ο οποίος δεν έχει ιδέα για το τι κάνατε όταν τα γράφατε (και δεν μπορεί ούτε να σας βρει να σας ρωτήσει).
Επίσης, θα πρέπει να ελέγχετε για διάφορα σφάλματα που μπορούν να προκύψουν και να βεβαιωθείτε ότι ο κώδικάς σας τερματίζει ομαλά, με μηνύματα που έχουν νόημα, σε όλες τις εισόδους που ικανοποιούν την παραπάνω περιγραφή.

Λεπτομέρειες εργασίας και παράδοσής της

Η εργασία είναι ομαδική, **2 ή 3 ατόμων**.

Γλώσσα υλοποίησης: C / C++ (Χωρίς χρήση STL και string)

Περιβάλλον υλοποίησης: Θα χρησιμοποιήσετε το BF επίπεδο που σας δίνετε όπως στη προηγούμενη εργασία. Μπορείτε να αναπτύξετε την εργασίας σας σε Linux (gcc 4.3+) ή Windows (Visual Studio 2012) αλλά η εξέταση θα γίνει στα Linux της σχολής συνεπώς τα προγράμματά σας πρέπει να μεταγλωττίζονται και να τρέχουν στα Linux της σχολής.

Παραδοτέα: Τα αρχεία πηγαίου κώδικα (sources) και τα αντίστοιχα αρχεία κεφαλίδας (headers) καθώς και αρχείο readme με περιγραφή / σχόλια πάνω στην υλοποίησή σας. Η παράδοση θα γίνει μέσω της πλατφόρμας e-class.

Προθεσμία Παράδοσης: 22 Δεκεμβρίου 2015, 11:59 μμ.