Προγραμματιστική Εργασία – Παραδοτέο 3

Ημερομηνία Παράδοσης: 18 Ιουνίου 2008

Για το τρίτο παραδοτέο της προγραμματιστικής σας εργασίας θα πρέπει να εμπλουτίσετε το πρόγραμμά σας, δίνοντας τη δυνατότητα να χρησιμοποιήσετε πίνακες κατακερματισμού αντί για δυαδικά δέντρα αναζήτησης ή για διπλές λίστες, για την αποθήκευση των καταχωρήσεων τύπου `BibtexEntry`. Για την ακρίβεια πρέπει να υλοποιήσετε πίνακα κατακερματισμού με chaining. Ο πίνακας κατακερματισμού σας θα πρέπει να έχει μέγεθος 1000, δηλαδή θα μπορεί να αποθεκεύσει 1000 διαφορετικές αλυσίδες καταχωρήσεων τύπου `BibtexEntry`.

Η συνάρτηση κατακερματισμού h που θα χρησιμοποιήσετε είναι μια τροποποιημένη μορφή αυτής που αναφέραμε στις διαλέξεις του μαθήματος για αλφαριθμητικά. Πιο συγκεκριμένα αν σας δίνεται μια λέξη $w = a_0a_1 \ldots a_{n-1}$ (το a_i είναι το $(i+1)$-στό γράμμα της w διαβάζοντας την από αριστερά), αποτιμούμε το πολυώνυμο $p(z) = a_0 + a_1z + a_2z^2 + \ldots + a_{n-1}z^{n-1}$, σε μια σταθερή τιμή z, και στη συνέχεια υπολογίζουμε το υπόλοιπο της διαίρεσης με το 1000. Στη δική μας περίπτωση, δεδομένου ενός κλειδιού (π.χ., το `c-uvcld-93`), αφαιρούμε πρώτα τις όποιες παύλες έχει (οπότε στο παράδειγμά μας παίρνουμε τη λέξη `cuvcld93`), την οποία και υπολογίζουμε με την ως άνω συνάρτηση κατακερματισμού για $z = 33$ (στο παράδειγμά μας $h($`cuvcld93`$) = 949$). Προς δική σας διευκόλυνηση, στην ιστοσελίδα του μαθήματος για την προγραμματιστική εργασία σας δίνεται το αρχείο `hash_function.h` στο οποίο υπάρχει η συνάρτηση με prototype:

```
unsigned int h(const char* s, unsigned int n, unsigned int z);
```

όπου `s` είναι το αλφαριθμητικό για το οποίο θέλουμε να υπολογίσουμε τη συνάρτηση κατακερματισμού, `n` το πλήθος των γραμμάτων του `s` και `z` η τιμή για την οποία πρέπει να υπολογιστεί το πολυώνυμο (στην περίπτωσή μας το `z` είναι πάντα 33. Η συνάρτηση `h` **δεν** αφαιρεί τις παύλες, κατά συνέπεια πρέπει εσείς να τις αφαιρέσετε από το κλειδί πριν υπολογίσετε την τιμή της συνάρτησης κατακερματισμού.

Κατά τα άλλα το πρόγραμμά σας θα πρέπει να συμπεριφέρεται προς το χρήστη όπως και για παραδοτέο δύο, λαμβάνοντας όμως υπόψη τις παρακάτω αλλαγές:

1. Το πρόγραμμά σας θα πρέπει να παίρνει δύο command-line arguments, τα οποία όμως είναι διαφορετικά από αυτά του δεύτερου παραδοτέου. Το πρώτο είναι το όνομα του αρχείου με τα δεδομένα όπως και στο πρώτο παραδοτέο, ενώ το δεύτερο θα είναι ένας χαρακτήρας, ο οποίος πρέπει να είναι l, t, r, h ή a. Όταν ο χρήστης εισάγει στο command line ένα από τα τέσσετα πρώτα παραπάνω γράμματα (l ή t ή r ή h), τότε το πρόγραμμα θα επιλέγει εσωτερικά να αποθηκεύει τα δεδομένα που διαβάζει από το αρχείο σε διπλή λίστα, δυαδικό δέντρο αναζήτησης, μελανέρυθρο δέντρο και πίνακα κατακερματισμού αντίστοιχα. Σε ότι αφορά το τελευταίο γράμμα, το a, το πρόγραμμά σας θα πρέπει να σώσει τα δεδομένα σε τέσσερεις διαφορετικές δομές ταυτόχρονα και να υπολογίζει χρόνους αναζήτησης όπως θα εξηγηθεί παρακάτω.

Τί πρέπει να κάνει το πρόγραμμά σας όταν το command-line argument είναι a; Στην περίπτωση αυτή θα πρέπει τα δεδομένα τύπου `BibtexEntry` να τα αποθηκεύετε ταυτόχρονα και στις 4 δομές δεδομένων που προγραμματίσατε (διπλή λίστα, δυαδικό δέντρο αναζήτησης, μελανέρυθρο δέντρο και

πίνακα κατακερματισμού). Στην συνέχεια όταν σας ζητήται να αναζητήσετε κάποιο κλειδί, το αναζητάτε και στις 4 δομές, ενώ ταυτόχρονα θα πρέπει να υπολογίζετε το χρόνο εκτέλεσης της αναζήτησης για κάθε δομή. Επειδή οι χρόνοι αυτοί ενδέχεται να είναι πολύ μικροί, ουσιαστικά θα πρέπει να υποογίζετε το χρόνο για 1000 αναζητήσεις. Για να το κάνετε αυτό μπορείτε να χρησιμοποίησετε τη συνάρτηση `time` της `C` η οποία ορίζεται στο αρχείο `time.h`. Ο κώδικάς σας θα πρέπει γενικά να μοιάζει με τον παρακάτω, όπου θεωρούμε ότι υπολογίζουμε το χρόνο αναζήτησης του κλειδιού `key` για τη δομή `my_ds` (που είναι μία από τις τέσσερεις δομές), μέσω της συνάρτησης αναζήτησης `search`:

```
time_t secs_start;
time_t secs_end;
time_t secs_elapsed_for_my_ds;
int i;

secs_start = time(NULL);
for (i = 0; i < 1000; ++i) {
  search(my_ds, key);
}
secs_end = time(NULL);
secs_ellapsed_for_my_ds = secs_end - secs_start;
```

Το υπολογισμό του εν λόγω χρονικού κόστους θα πρέπει να το κάνετε και για τις 4 δομές και στη συνέχεια να εκτυπώνετε στην οθόνη τα αποτελέσματα. Το `output` του προγράμματός σας θα πρέπει να έχει τη μορφή:

```
Key: c-uvcld-93
Search time for List: 1.5 sec
Search time for Binary-search tree: 1.5 sec
Search time for Red-black tree: 1.5 sec
Search time for Hash table: 1.5 sec
```

όπου οι τιμές που υπάρχουν παρακάτω είναι ενδεικτικές.

Για να αποφύγουμε τα όποια προβλήματα κατά την εξέταση των παραδοτέων σας, βεβαιώστε ότι το πρόγραμμά σας μεταγλωττίζεται με τον μεταγλωττιστή `gcc` και τρέχει σε μηχανήματα Unix/Linux. Για δική σας διευκόλυνση καλό θα ήταν κατά τη μεταγλώττιση να χρησιμοποιήσετε τα options `-Wall` και `-ansi` του `gcc` (ή αντίστοιχα για άλλους compilers), ώστε με τον τρόπο αυτό να βεβαιώσετε ότι ο κώδικας σας ακολουθεί το ANSI `C` standard, γεγονός που βοηθάει στην ορθότητα και το portability του.