Δομές Δεδομένων

Project 2018

Υλοποιημένο σε C

Λασκαρέλιας Βάιος, 1054432

laskarelias@ceid.upatras.gr

Γενικές Παρατηρήσεις

Οδηγίες μεταγλώττισης και εκτέλεσης:

Για την μεταγλώττιση χρησιμοποιείται Makefile. Συνεπώς τρέχουμε την εντολή:

```
> make
```

Έτσι, δημιουργείται το εκτελέσιμο αρχείο main. Για να το εκτελέσουμε τρέχουμε την εντολή:

```
> ./main
```

Τα αρχεία πηγαίου κώδικα (*.c) περιέχουν εκτενή σχόλια για το προγραμματιστικό και το αλγοριθμικό κομμάτι. Μόνο το αλγοριθμικό κομμάτι περιγράφεται στην αναφορά. Τα header files (inc/*.h) περιέχουν ορισμούς συναρτήσεων και δομών δεδομένων, και όχι την υλοποίηση τους, για γρήγορη αναφορά στην χρήση τους στο βασικό πρόγραμμα. Ο πηγαίος κώδικας είναι χωρισμένος σε αρχεία, για τον ευκολότερο διαχωρισμό των ερωτημάτων και την διόρθωσή τους. Σε κάθε μέρος αναφέρονται τα αρχεία που χρησιμοποιούνται.

Το αρχείο πηγαίου κώδικα fileio.c και το ανάλογο header του, inc/fileio.h, αποτελούν καθαρά βοηθητικά αρχεία, που χρησιμεύουν στην μετατροπή των δοσμένων αρχείων εισόδου (integers.txt και words.txt) σε δομές προς επεξεργασία από το πρόγραμμα. Ο κώδικας των αρχείων δεν περιέχει αλγοριθμικό κομμάτι.

Το αρχείο πηγαίου κώδικα bench.c και το ανάλογο header του, inc/bench.h χρησιμεύουν μόνο για την μέτρηση χρόνου των αναζητήσεων και δεν περιέχουν αλγοριθμικό κομμάτι.

Μέρος Α (Merge Sort)

Η συνάρτηση mergesort (</>mergesort.c) διαιρεί τον πίνακα σε 2 ίσα μέρη. Αναδρομικά, συνεχίζει την διαίρεση στους δύο υποπίνακες και τους ταξινομεί με την βοήθεια της συνάρτησης merge. Τέλος, συγχωνεύει (με κλήση της συνάρτησης merge) τους δύο πίνακες, επιλέγοντας το μικρότερο στοιχείο από τον καθένα κάθε φορά. Το αποτέλεσμα είναι ο αρχικός πίνακας να είναι ταξινομημένος.

Αρχεία:
</> fileio.c
</> mergesort.c
</> inc/fileio.h
</> inc/sort.h

Μέρος Β (Αναζήτηση)

Οι αναζητήσεις επιστρέφουν την θέση του στοιχείου στον πίνακα εφόσον υπάρχει, αλλιώς επιστρέφουν την τιμή -1

Γραμμική αναζήτηση:

Αρχεία:
</> linearsearch.c
</> inc/search.h

Η γραμμική αναζήτηση προσπελαύνει με την σειρά τον πίνακα, ελέγχοντας κάθε στοιχείο με το δοσμένο στοιχείο προς αναζήτηση. Σημαντική παρατήρηση είναι πως η γραμμική αναζήτηση δουλεύει και για μη ταξινομημένα στοιχεία

Binary searh:

Αρχεία:
</> binarysearch.c
</> inc/search.h

Η δυαδική αναζήτηση (binary search) συγκρίνει το στοιχείο προς αναζήτηση με το μέσο στοιχείο του πίνακα. Στην συνέχεια, κάνει αναδρομικά την ίδια διαδικασία χρησιμοποιώντας το αριστερό ή το δεξί μέρος του πίνακα, αν το στοιχείο στην μέση είναι μεγαλύτερο ή μικρότερο από το στοιχείο προς αναζήτηση αντίστοιχα. Η διαδικασία σταματάει όταν το μεσαίο στοιχείο είναι αυτό που αναζητάμε, ή το στοιχείο δεν υπάρχει. Η μέθοδος αυτή δεν δουλεύει σε μη ταξινομημένους πίνακες.

Interpolation search:

Αρχεία:
</> interpolationsearch.c
</> inc/search.h

Η αναζήτηση παρεμβολής (interpolation search) λειτουργεί ξεκινώντας από μια συγκεκριμένη θέση στον πίνακα, όπου βρίσκεται κοντά στο στοιχείο προς αναζήτηση. Η θέση αυτή επιλέγεται εκμεταλλεύοντας τις ιδιότητες των κανονικών κατανομών και των ακραίων στοιχείων του πίνακα. Στην συνέχεια, αλλάζει τα όρια αναζήτησης στον πίνακα, αποκλείοντας το κομμάτι που δεν ανήκει το στοιχείο προς αναζήτηση (με παρόμοιο τρόπο της binary search), επαναλαμβάνοντας την διαδικασία μέχρις ότου να βρεθεί η θέση του στοιχείου, ή η αναζήτηση να αποτύχει. Η μέθοδος αυτή δεν δουλεύει σε μη ταξινομημένα στοιχεία και είναι πολύ αποδοτικότερη σε στοιχεία όπου οι τιμές τους ακολουθούν κανονική κατανομή

```
Επιλογή θέσης i,με left και right τις ακραίες θέσεις του πίνακα, και s την τιμή προς αναζήτηση:
int i = left + (((double)(right - left) / (array[right] - array[left]))*(s – array[left]));
```

Μέρος Γ (Red-Black Tree)

Αρχεία:
</> rb.c
</> inc/rb.h

Το red-black tree πρόκειται για ένα ισορροπημένο δυαδικό δέντρο, στο οποίο μπορούμε να αποθηκεύσουμε αριθμητικά στοιχεία. Ένας κόμβος του red-black tree είναι δομημένος ως εξής:

```
struct rb_node;
typedef struct rb_node rbnode;
```

```
typedef rbnode* rbtree;
struct rb_node
{
	int d;		/* δεδομένα */
	rbtree l;	/* δείκτης στο αριστερό παιδί του κόμβου */
	rbtree r;	/* δείκτης στο δεξί παιδί του κόμβου */
	colortype c;	/* χρώμα (“red” ή “black”) */
};
```

Το δέντρο δημιουργείται βρίσκοντας πρώτα το σωστό φύλλο όπου πρέπει να εισαχθούν τα δεδομένα (συνάρτηση rbinsert), και διορθώνει το δέντρο χρησιμοποιώντας την βοηθητική συνάρτηση rbfix (έλεγχος και διόρθωση δέντρου) και rbrotate (Περιστροφή δύο κόμβων).

Συνάρτηση rbinsert:

Αρχικά, βρίσκει την θέση που πρέπει να εισαχθεί ο καινούριος κόμβος, κάνοντας χρήση των βοηθητικών κόμβων x, p, gp, ggp για να προχωρήσει προς τα φύλλα του δέντρου.

Εφόσον ο κόμβος δεν υπάρχει, τον δημιουργεί και τον συνδέει με τον πατέρα του, συνεπώς με το δέντρο.

Χρωματίζει τον εισαχθέντα κόμβο κόκκινο, και ελέγχει και διορθώνει το δέντρο (με την συνάρτηση rbfix)

Συνάρτηση rbfix:

Χρωματίζει τον δοσμένο στις παραμέτρους κόμβο με κόκκινο, και τα παιδιά του με μαύρο.

Ελέγχει αν ο πατέρας του κόμβου είναι κόκκινος, για να καθορίσει αν χρειάζεται περιστροφή. Εφόσον το δέντρο χρειάζεται περιστροφή, ελέγχει αν χρειάζεται διπλή περιστροφή (Δεξιά – αριστερή ή αριστερή – δεξιά) και εφαρμόζει την επιπλέον περιστροφή (κόμβου με πατέρα). Συνεχίζει κάνοντας και την απαραίτητη περιστροφή που δεν έχει γίνει ακόμα (κόμβου με παππού ή αλλαγμένου κόμβου με παππού) και χρωματίζει μαύρο τον κόμβο.

Τέλος, χρωματίζει μαύρη την ρίζα.

Αυτός ο τρόπος χρωματισμών και περιστροφών εξασφαλίζει την ορθότητα του red-black tree χωρίς τον έλεγχο των άλλων “συγγενών” του κόμβου.

Συνάρτηση rbsearch:

Αφού το red-black tree πρόκειται για δυαδικό δέντρο, μπορούμε να αναζητήσουμε τιμές μέσα σε αυτό με αναδρομικό τρόπο και συγκρίσεις. Συγκρίνοντας το δεδομένο προς αναζήτηση με το δεδομένο του κόμβου, μπορούμε να συνεχίσουμε την αναζήτηση στο αριστερό ή δεξί υποδέντρο, αν το δεδομένο προς αναζήτηση είναι μικρότερο ή μεγαλύτερο αντίστοιχα.

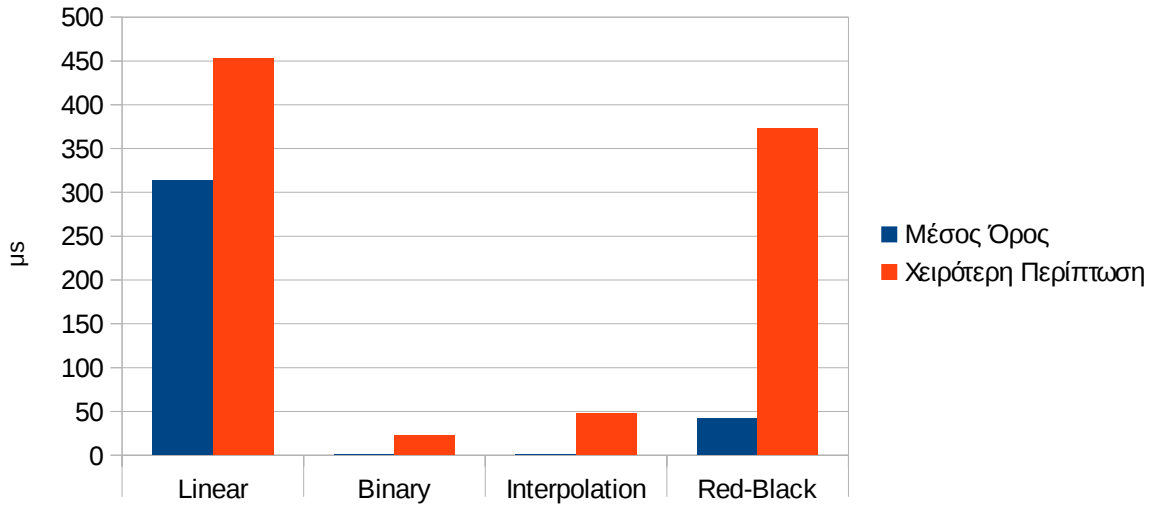
Μέρος Δ (Χρόνοι αναζήτησης)

Αρχεία:
</> linearsearch.c
</> binarysearch.c
</> interpolationsearch.c
</> inc/search.h
</> rb.c
</> inc/rb.h
</> bench.c
</> inc/bench.h

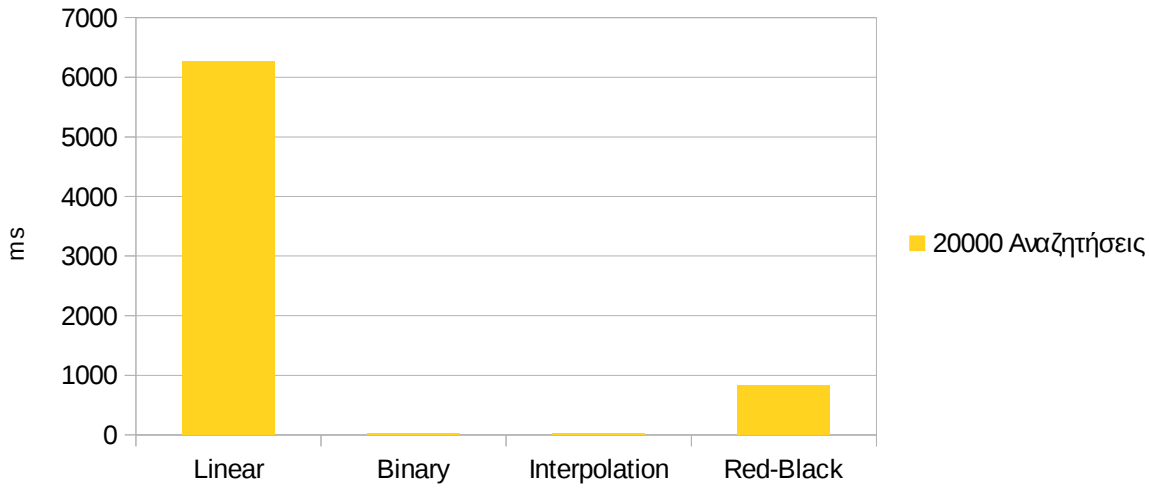
Τα αποτελέσματα των χρόνων αναζήτησης δίνονται από την συνάρτηση bench, για τις αναζητήσεις Linear search, Binary search και Interpolation search, και από την συνάρτηση benchrb για την αναζήτηση στο red-black tree. Τα αποτελέσματα για 20000 τυχαίες αναζητήσεις είναι:

```
Average time for 1 Linear search = 0.000314 sec.
Total time for 20000 Linear searches = 6.275475 sec.
Worst time for 1 Linear search = 0.000453 sec.

Average time for 1 Binary search = 0.000001 sec.
Total time for 20000 Binary searches = 0.027339 sec.
Worst time for 1 Binary search = 0.000023 sec.

Average time for 1 Interpolation search = 0.000001 sec.
Total time for 20000 Interpolation searches = 0.019462 sec.
Worst time for 1 Interpolation search = 0.000048 sec.

Average time for 1 Red-Black search = 0.000042 sec.
Total time for 20000 Red-Black searches = 0.832598 sec.
Worst time for 1 Red-Black search = 0.000373 sec.
```

Γραφικές παραστάσεις:



Παρατηρούμε αποτελέσματα αρκετά κοντά στην θεωρία



Η διαφορά φαίνεται αισθητά όσο ανεβαίνει ο αριθμός αναζητήσεων

Θεωρητικά:

Τρόπος αναζήτησης	Μέσος όρος	Χειρότερη περίπτωση
Linear search	O(n)	O(n)
Binary search	O(logn)	O(logn)
Interpolation search	O(loglogn) *κανονική κατανομή	O(n)
Red-Black tree search	O(logn)	O(logn)

Η απόκλιση που παρατηρείται στο red-black tree προκύπτει από την φύση της δομής δεδομένου του δέντρου. Οι υπόλοιπες υλοποιήσεις χρησιμοποιούν πίνακα ακεραίων, που κάθε στοιχείο του προσπελαύνεται πολύ πιο γρήγορα από ένα στοιχείο λίστας με περαιτέρω πληροφορία (δείκτες σε άλλους κόμβους και χρώμα). Ωστόσο, η αργότερη προσπέλαση μνήμης με γρηγορότερο αλγόριθμο αποδίδει περισσότερο από την περίπτωση της γραμμικής αναζήτησης, τόσο σε μία, όσο και σε πολλές αναζητήσεις.

Μέρος Ε (Trie)

Αρχεία:
</> trie.c
</> inc/trie.h
</> fileio.c
</> inc/fileio.h

Το trie (Digital tree) είναι μια δομή που μας επιτρέπει να αποθηκεύσουμε διαφορετικές σειρές χαρακτήρων (πρακτικά λέξεις), με μικρό κόστος μνήμης. Η δομή ενός κόμβου του trie υλοποιείται ως εξής:

```
struct trie_node /* Ορισμός κόμβου trie */
{
	struct trie_node* character[26]; /* Πίνακας επόμενου χαρακτήρα */
	bool leaf; /* Αν τελειώνει εδώ η λέξη */
};
typedef struct trie_node trienode;
```

Κάθε κόμβος αποτελείται από έναν πίνακα με δείκτες σε επόμενους κόμβους, όπου η θέση τους στον πίνακα καθορίζει τον χαρακτήρα, και μια μεταβλητή που δηλώνει το τέλος μιας λέξης.

Η εισαγωγή λέξεων στο trie γίνεται με την συνάρτηση trieinsert, η αναζήτηση με την συνάρτηση triesearch, και η διαγραφή με την συνάρτηση triedelete.

Συνάρτηση trieinsert:

Προσπελαύνει την λέξη σειριακά, ανά χαρακτήρα. Δημιουργεί έναν νέο κόμβο στην κατάλληλη θέση του πίνακα, και πηγαίνει στον κόμβο αυτό. Η διαδικασία συνεχίζεται εώς ότου τελειώσουν οι χαρακτήρες μιας λέξης, όπου τότε σημειώνεται και το τέλος της στην μεταβλητή leaf του κόμβου.

Συνάρτηση triesearch:

Η αναζήτηση λειτουργεί με παρόμοιο τρόπο της trieinsert. Ελέγχει σειριακά την ύπαρξη κόμβων στην κατάλληλη θέση του πίνακα, και αν ο κόμβος δεν υπάρχει, ή ο κόμβος δεν είναι φύλλο, η λέξη δεν υπάρχει στο δέντρο.

Συνάρτηση triedelete:

Η συνάρτηση διαγραφής δουλεύει αναδρομικά. Μόλις φτάσει στον τελευταίο χαρακτήρα της λέξης προς διαγραφή, ελέγχει την ύπαρξη παιδιών στον πίνακα επόμενων κόμβων. Αν ο πίνακας είναι κενός, ο κόμβος διαγράφεται και ελευθερώνεται η αντίστοιχη μνήμη, και συνεχίζει στο ανώτερο επίπεδο του trie. Αλλιώς, η λέξη προς διαγραφή είναι υποσύνολο άλλης λέξης (για παράδειγμα: hell και hello), οπότε η μεταβλητή leaf παίρνει την τιμή false, που σηματοδοτεί ότι πλέον η λέξη δεν τελειώνει εκέι, άρα η λέξη δεν υπάρχει.