Εργασια C++

Τσαπούρης Γεώργιος ΑΕΜ: 1356

Δομή Εργασίας

Περιγράφουμε αναλυτικά την δομή που ακολουθήθηκε για την υλοποίηση των δομών δεδομένων:

- Node.h → Χρησιμοποιείται για τα δένδρα ως δομή αποθήκευσης μιας λέξης (key) και του αντίστοιχου μετρητή (num) της. Επίσης, περιέχει δείκτες για γειτονικά Nodes (l,r) καθώς και το ύψος (h) που χρειάζεται για τα avl δένδρα.
- DataStructure.h → Είναι μία αφηρημένη κλάση που περιέχει τις βασικές λειτουργίες μιας δομής δεδομένων (εισαγωγή, διαγραφή, αναζήτηση). Επίσης, περιέχει και τον βασικό δείκτη για την αντίστοιχη βοηθητική δομή που χρησιμοποιείται (π.χ Node για τα δένδρα). Το σκεπτικό ήταν να κληρονομείται η κλάση από όλες τις δομές, αλλά εν τέλει περιοριστήκαμε στα δένδρα. Η κλάση DataStructure είναι ένα πρότυπο και ανάλογα με τον τύπο της βοηθητικής δομής δημιουργείται μία εξιδεικευμένη κλάση.
- Tree.h, Tree.cpp → Η κλάση αυτή κληρονομεί από την DataStructure τις βασικές λειτουργίες και την βοηθητική δομή δεδομένων, ενώ προσθέτει και επιπρόσθετες λειτουργίες που έχουν μόνο τα δένδρα (π.χ. διασχίσεις). Εδώ, έχουμε υλοποιήσει το σώμα της αναζήτησης καθώς είναι κοινό και για τα δυαδικά δένδρα αναζήτησης αλλά και για τα avl. Ενώ η εισαγωγή και διαγραφή υλοποιούνται διαφορετικά για τις παράγωγες κλάσεις. Το κύριο σκεπτικό για την υλοποίηση των συναρτήσεων που βλέπει ο χρήστης ήταν να δημιουργηθούν αντίστοιχες ιδιωτικές συναρτήσεις που θα υλοποιούν την κάθε συνάρτηση. Για παράδειγμα, για την Inorder συνάρτηση που είναι δημόσιο μέλος, έχει χρησιμοποιηθεί η inorder(Node*) που υλοποιεί την λειτουργία. Για τις λειτουργίες των μεθόδων χρησιμοποιήθηκε αναδρομή.
- BinarySearchTree.h, BinarySearchTree.cpp → Κληρονομεί από την Tree και υλοποιεί της δύο βασικές μεθόδους (εισαγωγή, διαγραφή) οι οποίες με το ίδιο σκεπτικό που εφαρμόστηκε παραπάνω χρησιμοποιούν τις ιδιωτικές συναρτήσεις add_a_node(Node*, string) και delete_a_node(Node*, string) αντίστοιχα. Η add_a_node αναδρομικά προσπαθεί να προσθέσει την λέξη μέσα στο δένδρο ακολουθώντας είτε το δεξί ή το αριστερό υποδένδρο. Αν η λέξη υπάρχει αυξάνει τον μετρητή της αλλιώς δημιουργεί έναν νέο κόμβο για το δένδρο. Κατά την διαγραφή με την delete_a_node αν βρεθεί η λέξη σε έναν κόμβο, τότε από το αριστερό παιδί του κόμβου ψάχνουμε την μεγαλύτερη λέξη που βρίσκεται στο τέρμα δεξιά φύλλο του υποδένδρου του κόμβο και το αντικαθιστούμε με τη λέξη του κόμβου.
- AVLTree.h, AVLTree.cpp → Κληρονομεί και αυτή η κλάση από την Tree και υλοποιεί τις αντίστοιχες βασικές μεθόδους. Για την εισαγωγή και διαγραφή έχουμε τις ίδιες περιγραφές συναρτήσεων ενώ έχουμε δημιουργήσει και ιδιωτικές συναρτήσεις για να υλοποιηθεί κάθε περιστροφή ξεχωριστά. Τέλος έχουμε προσθέσει μία συνάρτηση που επιστρέφει το ύψος ενός δένδρου καθώς και ένα σκορ για την ισορροπία του δένδρου (η τελευταία χρησιμοποιείται από τις περιστροφές). Οι περιστροφές έχουν διαφορετικές συνθήκες για το πότε πρέπει να πραγματοποιηθούν ανάλογα με την εισαγωγή ή την διαγραφή. Έτσι, σε κάθε περιστροφή έχουμε προσθέσει και την μεταβλητή condition όπου ανάλογα με την συνάρτηση που έχουμε καλέσει, χρησιμοποιούμε διαφορετική συνθήκη.

- TableEntry.h → Βασική δομή για τον πίνακα κατακερματισμού όπου έχει την λέξη και έναν μετρητή.
- HashTable.h, HashTable.cpp → Ως μεταβλητές ορίζονται το μέγεθος του πίνακα την δεδομένη στιγμή, το συνολικό μέγεθος του πίνακα και έναν πίνακα δεικτών για TableEntries. Οι βοηθητικές συναρτήσεις που χρησιμοποιήθηκαν είναι η hash_key που επιστρέφει ένα κλειδί για μία λέξη και η new_entry() που δημιουργεί ένα νέο TableEntry για τον πίνακα. Κατά την δημιουργία του αντικειμένου δηλώνεται ένας πίνακας 80000 θέσεων ο οποίος αρχικοποιείται με 0. Η εισαγωγή προσθέτει ένα νέο στοιχείο στον πίνακα μόνο αν η hkey θέση του πίνακα είναι κενή. Αλλιώς, υπάρχει σύγκρουση και πρέπει να ψάξει έπειτα στον πίνακα για να βάλει το στοιχείο. Αν το μέγεθος των στοιχείων μέσα στον πίνακα υπερβεί το δηλωμένο μέγεθος, τότε ενημερώνουμε το πίνακα για νέο μέγεθος και εφαρμόζουμε rehashing για την τοποθέτηση των στοιχείων στον νέο πίνακα. Για την αναζήτηση, τροποποιήσαμε κατάλληλα την συνάρτηση εισαγωγής.
- Main.cpp → Δημιουργούμε τις τρεις δομές δεδομένων και καλούμε την readfile(). Η readfile, παίρνει το όρισμα που δόθηκε κατά την εκτέλεση του προγράμματος και είναι το μονοπάτι στο αρχείο, το σύνολο Q που θα περιέχει τις λέξεις προς αναζήτηση και τις δομές δεδομένων. Η readfile διαβάζει το αρχείο λέξη προς λέξη και τοποθετεί τις λέξεις στις δομές. Χρησιμοποιείται η include για να ελεγχθεί αν η λέξη μπορεί να μπει ή όχι και στο σύνολο Q (δημιουργία τυχαιότητας). Αφού γίνει η ανάγνωση, υπολογίζεται μετά ο χρόνος εκτέλεσης της αναζήτησης για κάθε μία από τις δομές (calculate_time). Τέλος, τυπώνονται οι λέξεις του συνόλου και οι εμφανίσεις των λέξεων σε κάθε δομή δεδομένων.

Μεταγλώττιση και Εκτέλεση

Για την μεταγλώττιση του προγράμματος θα πρέπει να δοθούν ως ορίσματα τα -fpermissive και το -std=c++11_και να περαστούν όλα τα αρχεία .cpp για μεταγλώττιση. Παράδειγμα μεταγλώττισης:

g++ -fpermissive -std=c++11 avl_tree.cpp binary_search_tree.cpp data_structure.cpp hash_table.cpp node.cpp table_entry.cpp tree.cpp main.cpp

Για την εκτέλεση του προγράμματος θα πρέπει να δοθεί ως όρισμα το μονοπάτι για το αρχείο εισόδου. Για παράδειγμα:

a.exe small-file.txt