Πανής Χρυσόστομος , ΑΕΜ: 3219

Binary Tree:

Η κλάση BinaryTree αποτελείται από ένα struct node το οποίο αναπαριστά τον κάθε κόμβο του δέντρου και έχει 4 μεταβλητές. Μια string data για την λέξη, μια int amount για τον αριθμό εμφάνισης αυτης της λέξης και 2 δείκτες left και right τύπου node που δείχνουν στο αριστέρο και δεξί υποδέντρο αντίστοιχα του κόμβου. Επίσης περιέχει τις εξής public μεθόδους: μέθοδο εισαγωγής void insert(string key) , μέθοδο διαγραφής void remove(string key), μέθοδο αναζήτησης node *search(string key), μέθοδο καταστροφής δέντρου void destroy_tree(), τις μεθόδους void inorder_print(), void postorder_print(), void preorder_print(). Περιέχει επίσης τις αντίστοιχες private μεθόδους : void destroy_tree(node *leaf), node* remove(node* root, string key), void insert(string key, node *leaf), node *search(string key, node *leaf), void inorder_print(node *leaf), void postorder_print(node *leaf), void preorder_print(node *leaf) καθώς και την μέθοδο node* minValueNode(node* node) και έναν δείκτη τύπου node ονόματι root. Ο κατασκευαστής αρχικοποιεί τον δείκτη root σε NULL. Ο καταστροφέας καλεί την public μέθοδο destroy_tree() η οποία καλει τη private μέθοδο με όρισμα τον δείκτη της ρίζας του δέντρου (root) η οποία ελέγχει εάν ο δείκτης είναι NULL,εάν δεν είναι τότε καλεί αναδρομικά τον εαυτό της με ορίσματα τους δείκτες για το αριστερό παιδί του κόμβου και το δεξί αντίστοιχα, τέλος διαγράφει τον δείκτη. Η public μέθοδος insert(string) ελέγχει εάν η ρίζα του δέντρου είναι NULL, αν δεν είναι δημιουργεί τον κόμβο της ρίζας αλλάζοντας τα δεδομένα του(data και amount) και τους δείκτες τον παιδιών του σε NULL. Εάν δεν είναι NULL τότε καλεί την private μέθοδο insert() me ορίσματα την λέξη και τον δείκτη της ρίζας. Εκείνη με την σειρά της ελέγχει εάν η λέξη π εισάγεται ειναι ίδια με την λέξη του κόμβου,εάν ναι τότε αυξάνει το amount της κατα 1, εάν οχι ελέγχει εαν είναι λεξικογραφικά μικρότερη και αν ναι συνεχίζει να ψάχνει αναδρομικά καλώντας τον εαυτό της μέχρι να βρεί ίδια λέξη η NULL δείκτη που να μπορει να εισαγχθεί. Αντίστοιχα ελέγχει και αν η λέξη είναι λεξικογραφικά μεγαλύτερη απο τη λέξη του κόμβου. Η public μεθοδος remove() αποθηκεύει στον δείκτη root το αποτελεσμα της private συνάρτησης η οποία αρχικά ελέγχει αν το δέντρο είναι άδειο αν δεν είναι ψάχνει την λέξη αναδρομικά κάνοντας τις καταλληλες συγκρίσεις. Μόλις την βρεί αν υπαρχει πανω απο φορές απλα αφαιρεί από το amount της και αν ειναι φύλλο απλα τον διαγραφει. Αν ειναι μόνο μια και έχει ενα παίδι τοτε ενώνει τον πρόγονο του κόμβου με το παιδί του και αν έχει δύο παιδία τότε ψάχνει μέσω της minValueNode() τον inorder διαδοχο του αντιγραφει τα δεδομενα στον κομβο και υστερα διαγραφει τον διαδοχο. Η public κλάση search() καλει την αντιστοιχη private η οποια αναδρομικα και με λεξικογραφικες συγκρισεις ψαχνει τον κομβο π περιεχει τη λεξη και τον επιστρεφει. Με τις inorder μεθοδους επισκεπτομαστε και εμφανιζουμε πρωτα το αριστερο υποδεντρο καθε κομβου μετα τον ιδιο τον κομβο και υστερα το δεξι του υποδεντρο μεχρι η καθε κινηση να μην γινεται. Με τις preorder μεθοδους επισκεπτομαστε και εμφανιζουμε πρωτα τον ιδιο τον κομβο μετα το αριστερο υποδεντρο και υστερα το δεξι του υποδεντρο μεχρι η καθε κινηση να μην γινεται. Με τις postorder μεθοδους επισκεπτομαστε και εμφανιζουμε πρωτα το αριστερο υποδεντρο καθε κομβου μετα το δεξι του υποδεντρο και τελος τον ιδιο τον κομβο μεχρι η καθε κινηση να μην γινεται.

AVL Binary Tree:

Η κλάση BinaryTreeAVL αποτελείται από ένα struct node2 το οποίο αναπαριστά τον κάθε κόμβο του δέντρου και έχει 5 μεταβλητές. Μια string data για την λέξη, μια int amount για τον αριθμό εμφάνισης αυτης της λέξης, 2 δείκτες left και right τύπου node2 π δείχνουν στο αριστέρο και δεξί υποδέντρο αντίστοιχα, του κόμβου και μια int height για το υψος του δεντρου. Επίσης περιέχει τις εξής public μεθόδους: μέθοδο εισαγωγής void insert(string key) , μέθοδο διαγραφής void remove(string key), μέθοδο αναζήτησης node2 *search(string key), τις μεθόδους void inorder_print(), void postorder_print(), void preorder_print(). Περιέχει επίσης τις αντίστοιχες private μεθόδους : node2* remove(string key, node2* leaf), node2* insert(string key, node2 *leaf), node2 *search(string key, node *leaf), void inorder_print(node2 *leaf), void postorder_print(node2 *leaf), void preorder_print(node2 *leaf), έναν δείκτη τύπου node2 ονόματι root και ονομαστικα(θα εξηγηθουν παρακατω) τις μεθοδους : void makeEmpty(node2* leaf), node2* singleRightRotate(node2* &leaf), node2* singleLeftRotate(node2* &leaf), node2* doubleLeftRotate(node2* &leaf), node2* doubleRightRotate(node2* &leaf), node2* findMin(node2* leaf), node2* findMax(node2* leaf), int height(node2* leaf), int getBalance(node2* leaf). Ο κατασκευαστης οριζει τη ριζα (root) του δεντρου σε NULL. Η μεθοδος makeEmpty() διαγραφει το δεντρο. Η singleRightRotate () πραγματοποιει μια δεξια περιστοφη. Η singleLeftRotate () πραγματοποιει μια αριστερη περιστοφη. Η doubleRightRotate () πραγματοποιει δυο δεξιες περιστοφες. Η doubleLeftRotate () πραγματοποιει δυο αριστερες περιστοφες. Η findMin() βρισκει το αριστεροτερο φυλλο ενός υποδεντρου και η findMax() βρισκει το δεξιοτερο φυλλο ενός υποδεντρου. Η height επιστρεφει το υψος του δεντρου με ριζα τον κομβο-ορισμα. Η getBalance() επιστρεφει τη διαφορα μεταξυ υψων των δυο παιδιων ενός κομβου. Οι μεθοδοι print λειτουργουν αντιστοιχα με αυτές του απλου binary tree. Κατά την μεθοδο insert αν η λεξη υπαρχει ηδη αυξανεται απλα η μεταβλητη amount της, αν είναι λεξοκογραφικα μικροτερη καλειται ξανα η μεθοδος με ορισμα το αριστερο παιδι του κομβου Μολις η λεξη εισαγθει ελεγχεται αν πληρει τους κανονες του AVLTree και αν όχι γινονται οι αντιστοιχες περιστροφες ώστε να διορθωθουν τα υψη. Η μεθοδος search αναζητα την λεξη στο δεντρο με λεξικογραφικες συγκρισεις και η remove βρικσει την λεξη προς διαγραφη αν η λεξη υπαρχει πανω από μια φορες τοτε απλα μειωνει την μεταβλητη amount της κατά ένα, αν δεν ισχυει αυτό τοτε διαγραφει τον κομβο διατηρωντας τα καταλληλα υψη κανοντας τις απαραιτητες αλλαγες.

Hash Table Open Addressing:

Η κλάση HashTableOpenAddressing αποτελείται από ένα struct hashNode το οποίο αναπαριστά τον κάθε κάθε λεξη με τις πληροφοριες που εχει και έχει 2 μεταβλητές, μια string key για την λέξη, μια int amount για τον αριθμό εμφάνισης αυτης της λέξης, και ένα κατασκευαστη με ορισμα string key. Η κλαση περιεχει μεθοδο εισαγωγης, αναζητησης ,απεικονισης(insert,search,display) και επισης εχει private μεταβλητη hashNode** arr που παριστανει τον πινακα,int capacity(χωρητικοτητα του πινακα),hashNode* dummy(βοηθητικος ρολος) και μεθοδο hashCode που υπολογιζει τη θεση που θα μπει η λεξη. Καλωντας τον κατασκευαστη της κλασης με ορισμα τον πληθος των λεξεων που θα εισαγθουν δημιουργειται πινακας διπλασιου μεγεθους από αυτό των λεξεων και αρχικοποιειται σε NULL και η μεταβλητη dummy σε ‘’-’’. Στη μεθοδο search υπολογιζουμε με τη βοηθεια της hashCode() την θεση που θα επρεπε να βρισκεται η λεξη αν δεν βρισκεται εκει χρησιμοποιουμαι τη μεθοδο του linear probing και συνεχιζουμαι να τη αναζητουμε μεχρι να την βρουμε η να βρουμε μια θεση στην οποια θα επρεπε να βρισκεται αλλα η θεση είναι NULL(αυτό θα σημαινει ότι η λεξη δεν υπαρχει στο πινακα). Στη μεθοδο insert αρχικα ψαχνουμε αν η λεξη υπαρχει ηδη με τη μεθοδο search στο πινακα, αν υπαρχει απλα αυξανουμε το πεδιο amount της κατά ένα,αν δεν υπαρχει χρησιμοποιουμε τη μεθοδο linear probing όπως προηγουμενως μεχρι να βρουμε μια ελευθερη θεση για τη λεξη μολις βρουμε την τοποθετουμε εκει. Η μεθοδος display με απλη δομη for εμφανιζει τα περιεχομενα του πινακα.

Main:

Στη main αρχικα διαβαζουμε το αρχειο με ονομα ‘’small-file.txt’’ και διαβαζουμε το κειμενο αρχικα μονο για μετρησουμε το πληθος των λεξεων ώστε να το χρησιμοποιησουμε στον hash table. Επειτα δημιουργουμε τους 3 μας δεικτες έναν binary tree,έναν avl tree και έναν hash table. Διαβαζουμε το κειμενο ανα γραμμη με την getline και διαχωριζουμε τις λεξεις με το κενο και την αλλαγη γραμμης. Αφαιρουμε από τις λεξεις τα εξης συμβολα--> ;?_,./!-01234*567[]89()':#<>{}! .

Διατηρουμε τη διαφοροποιηση κεφαλαιων με πεζων γραμματων και προσθετουμε την λεξη σε κάθε μια από τις 3 μας δομες. Ταυτοχρονα δημιουργουμε ένα πινακα string 5000 θεσεων για τις αναγκες της ασκησης στον οποιο προσθετουμε μια λεξη κάθε 10 που προστιθενται στις δομες. Τελος αναζητουμε αυτές τις 5000 λεξεις σε κάθε δομη μετρωντας τον χρονο που χρειαζεται για να ανταποκριθει η κάθε μια ξεχωριστα και αντιστοιχα για κάθε δομη εμαφανιζεται η κάθε λεξη και ο αριθμος π εμφανιζεται αυτή η λεξη μεσα στη κάθε δομη και στο τελος εμφανιζεται ο χρονος που χρειαζεται η κάθε δομη για να ανταποκριθει.