ΤΕΧΝΙΚΗ ΕΚΘΕΣΗ

**Πίνακας Κατακερματισμού ( HashTable.h – HashTable.cpp )**

Αρχικά, έφτιαξα το struct Node στο οποίο αποθηκεύεται η λέξη και η μεταβλητή η οποία μετράει το πόσες φορές εμφανίζεται η λέξη στο αρχείο και έπειτα κατασκεύασα τον πίνακα κατακερματισμού ως ένα δυναμικό array από Nodes στο heap , καθώς το μέγεθος του είναι πάρα πολύ μεγάλο για να μπει στο stack.

Η κλάση HashTable περιλαμβάνει τον πίνακα table, τον κατασκευαστή HashTable, καθώς και τις δύο μεθόδους insert και search που ζητούσε η άσκηση.

Στον κατασκευαστή HashTable δίνεται μνήμη στον πίνακα table και αρχικοποιούνται όλες οι μεταβλητές emfaniseis στο 0.

Στη μέθοδο insert γίνεται η εισαγωγή των λέξεων στο HashTable. Έχει μια παράμετρο, την λέξη την οποία εισάγουμε , η οποία αρχικά εισάγεται στην συνάρτηση κατακερματισμού hashfunc.

Στην συνάρτηση κατακερματισμού αθροίζω όλους τους κωδικούς ASCII των χαρακτήρων της λέξης για να προσδιορίσω την θέση στην οποία θα τοποθετηθεί στον πίνακα και επιστρέφω την απόλυτη τιμή του αθροίσματος για να μην έχω κάποιο θέμα με παράξενους και μη αγγλικούς χαρακτήρες. Με αυτή την συνάρτηση δεν δημιουργούνται πολλά collisions καθώς το να προκύψει το ίδιο αποτέλεσμα για δύο διαφορετικές λέξεις είναι σχετικά σπάνιο.

Έπειτα γίνεται η εισαγωγή των λέξεων μέσα σε μια for με αριθμό επαναλήψεων όσο το μέγεθος του πίνακα για να ελεγχθούν όλα τα στοιχεία του. Αρχικά γίνεται προσπάθεια εισαγωγής στη θέση που προκύπτει από την συνάρτηση κατακερματισμού. Αν η θέση είναι άδεια τότε εισάγεται κανονικά η λέξη εκεί και αυξάνεται ο αριθμός των εμφανίσεων. Αν προκύψει collision , τότε ψάχνω την επόμενη διαθέσιμη θέση εισαγωγής της λέξης με σταθερό βήμα step=101. Βάζω ως βήμα το 101 απλά και μόνο επειδή είναι πρώτος αριθμός και έτσι θα ελεγχθούν όλες οι θέσεις του πίνακα αν χρειαστεί. Τέλος , αν η λέξη υπάρχει ήδη, απλά αυξάνω τις εμφανίσεις της. Αυτή είναι η διαδικασία της εισαγωγής.

Στη μέθοδο search γίνεται η αναζήτηση των λέξεων. Έχει ως μοναδική παράμετρο την λέξη την οποία αναζητούμε, η οποία εισάγεται πάλι αρχικά στη συνάρτηση κατακερματισμού , της οποίας η λειτουργία εξηγήθηκε πιο πάνω.

Η αναζήτηση γίνεται πάλι σε ένα for με τον ίδιο αριθμό επαναλήψεων. Αρχικά ελέγχω την θέση που προκύπτει από την συνάρτηση κατακερματισμού και αν δε βρίσκεται εκεί ελέγχω όλο τον πίνακα με σταθερό βήμα 101. Αν η λέξη υπάρχει επιστρέφω την τοποθεσία της λέξης στον πίνακα και εμφανίζεται ένα μήνυμα εξόδου :

The word “x” has appeared “y” times.

Αν η λέξη δεν υπάρχει θα εμφανιστεί ένα μήνυμα:

The word “x” was not found.

και επιστρέφεται nullptr.

**Δυαδικό δένδρο αναζήτησης ( BinaryTree.h - BinaryTree.cpp )**

Η κλάση που είναι υλοποιημένο το δυαδικό δένδρο αναζήτησης είναι η BinaryTree.h στην οποία υπάρχουν δηλωμένες όλες οι συναρτήσεις που αναφέρει η εκφώνηση και είναι υλοποιημένες στο αρχείο BinaryTree.cpp.

Στην κλάση λοιπόν έχουμε τον δείκτη σε BinaryTree root ο οποίος δείχνει στην ριζά το δένδρου . Ακόμα υπάρχει η μεταβλητή numberOfInserts που κρατάει τον αριθμό τον duplicates λέξεων ,την τύπου string μεταβλητή data που αποθηκεύει την λέξη μέσα στα nodes του δένδρου και 2 δείκτες σε BinaryTree οπού αναπαριστούν τα left και right παιδιά του κάθε κόμβου.

Ακόμα η κλάση περιλαμβάνει έναν κενό κατασκευαστή την μέθοδο init,τις συναρτήσεις για την εισαγωγή :insert\_root, insert\_node, insert\_right, insert\_left ,την Delete για την διαγραφή ενός κόμβου , την search και τις preorder,inorder,postorder.

Στον κενό κατασκευαστή αρχικοποιούνται οι μεταβλητές Lchild Και Rchild στο null και η numberOfInserts στο 0,ενώ στην init αρχικοποιείται ο δείκτης root στο null.

Για την εισαγωγή των λέξεων στο δένδρο, καλείτε η insert\_node η οποία παίρνει σαν όρισμα την λέξη που θέλουμε να εισάγουμε . Αρχικά ελέγχει αν το δένδρο είναι κενό . Αν είναι καλεί την συνάρτηση insert\_root που δέχεται σαν όρισμα την λέξη και δημιουργεί έναν νέο κόμβο στο δένδρο με data την λέξη που δέχτηκε , Lchild,Rchild=null και βάζει τον δείκτη root να δείχνει στον νέο κόμβο και επιστρέφει true . Εφόσον δεν είναι κενό το δένδρο μέσο μιας επαναληπτικής διαδικασίας γίνεται αναζήτηση τις λέξεις . Εάν υπάρχει ήδη στο δένδρο αυξάνεται κατά 1 η numberOfInserts μεταβλητή του τρέχων κόμβου . Εάν δεν υπάρχει τότε συγκρίνει την λέξη με τις υπόλοιπες του δένδρου και ανάλογα καλεί η την insert\_left η την insert\_right οι οποίες δέχονται σαν όρισμα ένα δείκτη current σε binatytree που κρατάει τον κόμβο στο σημείο που θέλουμε να προσθέσουμε την λέξη και την τρέχων λέξη .Οι συναρτήσεις κάνουν την ιδιά διαδικασία δηλαδή δημιουργούν έναν νέο κόμβο newnode ,προσθέτουν σε αυτόν την λέξη και στο τέλος βάζουν στο δεξί - αριστερό παιδί του δείκτη current αντίστοιχα να δείχνει στον newnode.Σε κάθε περίπτωση που γίνει επιτυχής εισαγωγή η συνάρτηση επιστρέφει true . Αντίθετα false.

Η συνάρτηση search πραγματοποιεί αναζήτηση στο δένδρο . Δέχεται σαν όρισμα μια λέξη και μέσο μιας while γίνετε η αναζήτηση . Εάν βρεθεί η λέξη εμφανίζει το μήνυμα:the word «λέξη» has appeared «τον αριθμό των εισαγωγών» και επιστρέφει true.Αν όχι τότε επιστρέφει false.

H delete διεξάγει την διαγραφή του κόμβου που περιέχει την λέξη που δέχεται σαν όρισμα η συνάρτηση. Αρχικά μέσο μιας επανάληψης while αναζητούμε την λέξη μέσα στο δένδρο . Αν δεν βρεθεί επιστρέφει false και τελειώνει . Εάν βρεθεί ελέγχει εάν ο κόμβος ο οποίος κρατάει την λέξη έχει 2,1 η κανένα παιδιά . Αρχικά δημιουργούμε 2 δείκτες σε binary tree τον p που εκφράζει τον κόμβο που θέλουμε να διαγράψουμε και τον pp που εκφράζει τον γονέα αυτού . Εφόσον η λέξη βρεθεί και ο κόμβος της έχει 2 παιδιά δημιουργούμε άλλους 2 δείκτες ,τον s τον ps γονέα του s . Έτσι μέσω μιας while βρίσκουμε το μικρότερο στοιχείο του δεξιού υποδένδρου αποθηκεύοντας το στο s και έπειτα το μεταφέρουμε στο p καθώς έτσι το δένδρο θα διατηρήσει την κανονικότητα του . Στην περίπτωση τώρα που δεν έχει 2 παιδιά ελέγχουμε αν έχει το πολύ 1 παιδί και δημιουργούμε ένα δείκτη current που θα δείχνει στο αριστερό – δεξί παιδί του κόμβου .Εάν έχει μόνο αριστερό παιδί βάζουμε τον current να δείχνει σε αυτό ,αντίστοιχα αν έχει μόνο δεξί παιδί . Τέλος αν δεν έχει παιδιά προβαίνουμε στην διαγραφή του κόμβου και επιστρέφουμε true.Ενδιάμεσα γίνεται και ένας έλεγχος εάν ο κόμβος που διαγράφουμε είναι ο μοναδικός στο δένδρο και εάν ναι τότε βάζουμε το root να δείχνει στο current και συνεχίζουμε στην διαγραφή.

Οι συναρτήσεις τώρα preorder , inorder , postorder είναι παρόμοιες και δέχονται σαν όρισμα έναν δείκτη σε BinaryTree ο οποίος είναι το root.Ελέγχουν αν είναι ίσος με null.Εάν όχι τότε στην preorder έχουμε πρώτα εμφάνιση τις λέξης του κόμβου και μετρά δυο διαδοχικές αναδρομικές κλήσεις πρώτα του αριστερού υποδένδρου της ρίζας και μετρά του δεξιού . Στην inorder καλούμε πρώτα την αναδρομική συνάρτηση για το αριστερό υποδένδρου μετά εμφανίζουμε την λέξη και τέλος καλούμε την αναδρομική συνάρτηση για το δεξί υποδένδρου και αντίστοιχα και για την postorder κάνουμε πρώτα αναδρομή για τα υπόδενα και μετά εμφανίζουμε την λέξη.

**Δυαδικό δένδρο τύπου AVL ( avl.h - avl.cpp )**

Το δυαδικό δένδρο τύπου AVL αναπαρίσταται μέσω της κλάσης avl.h οπού υπάρχουν δηλωμένες οι συναρτήσει που ζητάει η εκφώνηση καθώς και υλοποιημένες στο αρχείο της κλάσης avl.cpp.

H κλάση avl.h περιλαμβάνει την μεταβλητή τύπου string dataAVL που αναπαριστά την λέξη που θα αποθηκεύετε μέσα στα nodes του δένδρου ,την int μεταβλητή h οπού θα κρατάει το ύψος του κάθε κόμβου , επίσης την int numberOfInsertsAVL που θα κρατάει τα duplicates των λέξεων και 2 δείκτες σε avl LchildAVL,RchildAVL για το αριστερό και δεξί παιδί του κάθε κόμβου.

Από συναρτήσεις , η κλάση περιέχει έναν κατασκευαστή χωρίς ορίσματα την newnodeAVL,την insert\_node,την height,την deleteAVL,την balance,τις R,L,LR,RL ,την search και τις preorder ,postorder,inorder.

Στον κατασκευαστή χωρίς ορίσματα γίνεται η αρχικοποίηση των LchildAVL,RchildAVL στο null και τα numberOfInsertsAVL και h στο 0.

Για να γίνει η εισαγωγή μιας λέξης μέσα στο AVL καλούμε την συνάρτηση insert\_nodeAVL που δέχεται σαν ορίσματα ένα δείκτη σε avl και την λέξη που θέλουμε να εισάγουμε .Για να δημιουργηθεί ένας νέος κόμβος χρησιμοποιούμε μια αναδρομική λειτουργία που κάνει αναζήτηση της λέξης μέσα στο δένδρο .Εάν βρεθεί η λέξη που θέλουμε να εισάγουμε απλά αυξάνουμε την μεταβλητή numberOfInsertsAVL κατά 1.Αλλιως γίνονται συνεχής συγκρίσεις με τις υπόλοιπες λέξεις του δένδρο μέχρι να φτάσουμε σε κόμβο που είναι ίσος με null.Τότε καλούμε την συνάρτηση newnodeAVL που δέχεται σαν όρισμα την λέξη και δημιουργεί έναν νέο δείκτη σε AVL newnode βάζοντας στο dataAVL αυτού την λέξη που δεχτικέ σαν όρισμα και αρχικοποιόντας τα LchildAVL, RchildAVL , numberOfInsertsAVL, h.Τέλος επιστρέφει τον καινούριο κόμβο .Καθώς έχει πραγματοποιηθεί η εισαγωγή του κόμβου και έχει ΄΄γυρίσει΄΄η αναδρομή καλούμε την συνάρτηση balance που δέχεται τον καινούργιο δείκτη του κόμβου και την λέξη σαν ορίσματα και εξισορροπεί το δένδρο. Αρχικά ενημερώνει τα ύψη του δένδρου καλώντας την height που επιστρέφει τα ύψη του δεξιού και του αριστερού υποδένδρου και οποίο από αυτά είναι μεγαλύτερο το εισάγει στο h του κόμβου προσθέτοντας +1 για το ίδιο τον κόμβο .Έπειτα σε μια μεταβλητή k αποθηκεύει την διαφορά των υψών των δυο υποδένδρων του, εάν αυτή είναι ίση με 2 η -2 καλεί μια από τις συναρτήσει R (δεξιά περιστροφή) ,L(αριστερή περιστροφή) ,LR(πρώτα μια αριστερή και μετά μια δεξιά περιστροφή) και την RL(πρώτα μια δεξιά και μετά μια αριστερή περιστροφή) ώστε το δένδρο να πάρει μορφή AVL.

Για την δεξιά περιστροφή η συνάρτηση R δέχεται τον κόμβο parent σαν όρισμα και δημιουργεί έναν νέο δείκτη σε avl l. Έπειτα τον αποθηκεύει στο αριστερό παιδί του parent.Στην συνέχεια βάζει το αριστερό παιδί του parent να δείχνει στο δεξί παιδί του l και στο δεξί παιδί του l να δείχνει στον κόμβο parent.Τέλος ανανεώνει τα ύψη τον δυο κόμβων και επιστρέφει τον l που έχει πάρει την θέση το parent.Αντίστοιχα και για την περιστροφή L τα βήματα είναι εντελώς ανάλογα μόνο που κάνει την περιστροφή προς τα αριστερά.

Η RL συνάρτηση που δέχεται ένα κόμβο σαν όρισμα και κάνει πρώτα μια δεξιά περιστροφή στο δεξί παιδί αυτού και μετά μια αριστερή στον ίδιο τον κόμβο τον οποίο και επιστρέφει .Αντίστοιχα και η LR κάνει τις ιδίες περιστροφές αλλά ανάποδα.

Η Αναζήτηση μιας λέξης μέσα στο δένδρο γίνεται με τον ίδιο τρόπο όπως και στο διάδικο δένδρο αναζήτησης με την διαφορά ότι δέχεται ένα κόμβο σαν όρισμα με τον οποίο θα πραγματοποιούνται οι συγκρίσεις με τις υπόλοιπες λέξεις .Τι ίδιο ισχύει και για τις preorder, inorder,postorder .

Για την διαγραφή ενός κόμβου τώρα χρησιμοποιούμε τη DeleteAVL που δέχεται σαν ορίσματα έναν δείκτη σε avl node και την λέξη που θέλουμε να διαγράψουμε .Αρχικά κάνει μια αναδρομική αναζήτηση μέσα στον δένδρο .Εάν η λέξη που ψάχνουμε δεν βρεθεί εμφανίζει σχετικό μήνυμα : no word like "λέξη" in the tree και επιστρέφει null.Εάν βρεθεί ελέγχει αν έχει 2,1η κανένα παιδιά .Στη περίπτωση που έχει 2 παιδιά δημιουργεί έναν δείκτη current τον οποίο βάζει να δείχνει στο μικρότερο στοιχείο του δεξιού υποδένδρου του κόμβου node μέσο της συνάρτησης Min.Μεταφέρει το data από το current στο node και συνεχίζει αυτήν την διαδικασία καλώντας αναδρομικά την deleteAVL για το δεξί παιδί του node.Στην περίπτωση που έχει το πολύ ένα παιδί βάζει τον current να δείχνει στο node και μετά ελέγχει αν έχει δεξί η αριστερό παιδί .Αν έχει βάζει το node να δείχνει στο δεξί η το αριστερό του παιδί αντίστοιχα και προβαίνει στην διαγραφή του current που δείχνει στο κόμβο που θέλουμε να διαγράψουμε .Εντωμεταξύ γίνεται έλεγχος για το αν ο κόμβος με την λέξη που θέλουμε να διαγράψουμε είναι ο μοναδικός του δένδρου .Τέλος ενημερώνει τα ύψη των κόμβων και αν είναι αναγκαίο κάνει τις κατάλληλες περιστροφές με τον τρόπο που περιγραφηκέ ποιο πάνω.