**Εργασία Εξαμήνου Δομές δεδομένων 2020**

Φοιτητές της ομάδας **:**

Φίλιππος Πάνος - ΑΕΜ :3603 Παναγιώτης Παπασωτηρίου – ΑΕΜ : 3602

Η εργασία αποτελείται από τα αρχεία : main.cpp, BstNode.h, BstNode.cpp, AvlTree.h, AvlTree.cpp, HashMap.h, HashMap.cpp, HashTable.h, HashTable,cpp, menuFunctions.h, menuFunctions.cpp

Συγκεκριμένα :

**main.cpp**:

Αρχικά θέτει με define στο RANDOMWORDS πόσες λέξεις θα μπουν στον πίνακα των τυχαίων λέξεων. Στην συνέχεια δεσμεύει δυναμικά χώρο για τον πίνακα randomWords[], έπειτα διαβάζει από τον χρήστη το αρχείο κειμένου που πρόκειται να ανοίξει και θέτει την ρίζα του BST και του AVL (BSTroot και AVLroot αντίστοιχα) σε NULL και δημιουργεί ένα καινούριο αντικείμενο του HashMap. Μετά καταχωρεί στο string linestr κάθε φορά μια μια λέξει και ελέγχει με την ispunct αν στο linestr υπάρχουν σημεία στίξης, αν βρεθούν τα διαγράφει με την erase και προσθέτει σε κάθε δομή την λέξη (το string linestr), επίσης σε αυτό το σημείο αποφασίζει τυχαία για το αν η εκάστοτε λέξη θα μπει στον πίνακα randomWords ή όχι μέχρι να φτάσει ο μετρητής randomWordsIndex το define του RANDOMWORDS. Τέλος καλεί την searchRandom, συνάρτηση της menuFunctions περνώντας σαν ορίσματα το αντικείμενο του HashMap , την ρίζα του BST,την ρίζα του AVL, το πλήθος των τυχαίων λέξεων του πίνακα randomWords[] και την διεύθυνση του πίνακα randomWords[].

**BstNode.h & BstNode.cpp:**

Στο BstNode.h υπάρχει δηλωμένο το struct BstNode και οι συναρτήσεις με τις λειτουργείες του BST. ΤοBstNode αντιπρoσωπεύει το κάθε node του δυαδικού δέντρου αναζήτησης, ποιο συγκεκριμένα το data είναι η λέξη του node, το right,left είναι οι δείκτες του δεξιού και αριστερού παιδιού αντίστοιχα του node και η μεταβλητή counter δείχνει πόσες φορές εμφανίζεται η κάθε λέξη.

● Η συνάρτηση **GetNewNode** δέχεται ως όρισμα ένα string και δημιουργεί ένα καινούριο node με data το string του ορίσματος, NULL δείκτες για το αριστερό και δεξί υποδέντρο-παιδί και counter=1.

● Η συνάρτηση **Insert** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης και ένα string.Αρχικά ,αν το BST είναι άδειο προσθέτει την λέξη-string data στο δένδρο ,διαφορετικά “κατεβαίνει” αναδρομικά το δέντρο με τον σωστό δρόμο (αν το data < του data της εκάστοτε ρίζας πηγαίνει στο αριστερό υποδέντρο, αντίθετα στο δεξί) μέχρι να βρει φύλλο (right ή left δείκτης NULL) όπου και καλεί την GetNewNode για την δημιουργία καινούριου κόμβου. Αν καθώς “κατεβαίνει” το δέντρο βρει την ίδια λέξη, αυξάνει τον counter του εκάστοτε node.Τέλος επιστρέφει την ρίζα root του εκάστοτε node.

●H συνάρτηση **BstSearch** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης, ένα string και έναν ακέραιο αριθμό με αναφορά. Ψάχνει – κατεβαίνει το BST αναδρομικά όπως η Insert μέχρι να βρει το string του ορίσματος, αν το βρει δίνει στον ακέραιο του ορίσματος το counter του node και επιστρέφει την τιμή TRUE. Διαφορετικά αν φτάσει σε φύλλο χωρίς να έχει βρει την λέξη του string επιστρέφει FALSE.

●H συνάρτηση **BstSearchNode** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης και ένα string.Δουλεύει-ψάχνει το string στο δέντρο όπως και η BstSearch αλλά στην περίπτωση που βρεθεί επιστρέφει την διεύθυνση του εκάστοτε κόμβου.

● H συνάρτηση **DelleteEllement** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης, ένα string και έναν ακέραιο αριθμό (ο οποίος στην αρχή είναι πάντα 0). Αρχικά με την BSTSearchNode βρίσκει το node στο οποίο υπάρχει η λέξη και έπειτα ελέγχει αν η λέξη εμφανίζεται μια ί παραπάνω από μια φορές. Στην περίπτωση που εμφανίζεται παραπάνω από μια φορές απλώς μειώνει τον counter κατά ένα. Διαφορετικά ψάχνει στο δέντρο μέχρι να βρει την λέξη και ανάλογα με το πλήθος των παιδιών της ακολουθεί την ανάλογη πορεία. Αν δεν έχει κανένα παιδί απλά διαγράφει το node ,αν έχει ένα παιδί το εκάστοτε παιδί μπαίνει στην θέση του κόμβου προς διαγραφή και ο κόμβος διαγράφεται. Αν έχει δύο παιδιά, ψάχνει με την FindMin το ελάχιστο στοιχείο του δεξιού υποδέντρου του κόμβου με την λέξη προς διαγραφή και αυτό το στοιχείο μπαίνει στην θέση του κόμβου προς διαγραφή.

● H συνάρτηση **FindMin** δέχεται ως ορίσματα έναν κόμβο ή ρίζα του δυαδικού δέντρου αναζήτησης και βρίσκει το ελάχιστο στοιχείο του υποδέντρου του και επιστρέφει την διεύθυνση του.

●Η **PreOrder** δέχεται ως όρισμα την ρίζα του BST και εμφανίζει όλα τα στοιχεία του με προδιατεταγμένη σειρά.

●Η **PostOrder** δέχεται ως όρισμα την ρίζα του BST και εμφανίζει όλα τα στοιχεία του με μεταδιατεταγμένη σειρά.

●Η **InOrder** δέχεται ως όρισμα την ρίζα του BST και εμφανίζει όλα τα στοιχεία του με ενδοδιατεταγμένη σειρά.

**AvlTree.h & AvlTree.cpp:**

Στο AvlTree.h υπάρχει δηλωμένο το struct AvlTreeNode και οι συναρτήσεις με τις λειτουργείες του AVL. Το AvlTreeNode αντιπροσωπεύει το κάθε node του AVL δυαδικού δέντρου αναζήτησης, ποιο συγκεκριμένα το data είναι η λέξη του node, το right,left είναι οι δείκτες του δεξιού και αριστερού παιδιού του node αντίστοιχα, η μεταβλητή n δείχνει πόσες φορές εμφανίζεται η κάθε λέξη και η μεταβλητή height δείχνει το ύψος του εκάστοτε κόμβου.

● Η συνάρτηση **AVLnewNODE** δέχεται ως όρισμα ένα string και δημιουργεί ένα καινούριο node με data το string του ορίσματος, NULL δείκτες για το αριστερό και δεξί υποδέντρο-παιδί, counter=1 και height=1 και επιστρέφει τον καινούριο κόμβο.

● Η συνάρτηση **getHeight** δέχεται ως όρισμα έναν κόμβο – ρίζα υποδέντρου και επιστρέφει το ύψος του. (Προσοχή: δεν υπολογίζει το ύψος ,απλά επιστρέφει το height του κόμβου, το ύψος υπολογίζεται-ενημερώνεται στην AvlInsertion)

● Η συνάρτηση **MaxOfTwoHeights** δέχεται ως ορίσματα δύο ακαίρους αριθμούς και επιστρέφει τον μεγαλύτερο. Χρησιμοποιείται για τον υπολογισμό-ενημέρωση του ύψους ενός κόμβου στην AvlInsertion και στις συναρτήσεις περιστροφής.

● Η συνάρτηση **HeightDiff** δέχεται ως όρισμα έναν κόμβο – ρίζα υποδέντρου και επιστρέφει την διαφορά ύψους μεταξύ του δεξιού και αριστερού υποδέντρου του. Χρησιμοποιείται για να δούμε αν χρειάζεται περιστροφή ή όχι.

Για να γίνουν οι **περιστροφές** χρησιμοποιούνται δύο συναρτήσεις, η **RightRot** και η **LeftRot** ,και οι δύο δέχονται ως όρισμα την διεύθυνση του κόμβου στον οποίο βρέθηκε διαφορά ύψους υποδέντρων >1.Η RightRot δημιουργεί έναν καινούριο κόμβο temp ο οποίος δείχνει στο αριστερό παιδί του κόμβου ορίσματος(πλέον temp= αριστερό παιδί του κόμβου ορίσματος), μετά βάζει το αριστερό παιδί του κόμβου ορίσματος να δείχνει στο δεξί παιδί του temp και το δεξιά παιδί του temp στον κόμβο ορίσματος. Τέλος ενημερώνει το ύψος του κόμβου ορίσματος και υπολογίζει και το ύψος του temp. Έτσι καταφέραμε να κάνουμε μια δεξιά περιστροφή και πλέον ο temp είναι η ρίζα του υποδέντρου και τον επιστρέφουμε.

Με παρόμοιο τρόπο λειτουργεί και η LeftRot.

● H συνάρτηση **AvlInsertion** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης και ένα string.Η διαδικασία εισαγωγής είναι ίδια με την Insert του BST. Δουλεύει αναδρομικά άρα καθώς “επιστρέφει” ενημερώνει το ύψος κάθε κόμβου. Επίσης καθώς “επιστρέφει” από την αναδρομή ελέγχει αν μετά την εισαγωγή, σε κάποιον κόμβο πρέπει να γίνει περιστροφή. Αν χρειάζεται ελέγχει σε πια από τις 4 πιθανές κατηγορίες ανήκει (LL,RL,LR,RR) και καλεί τις ανάλογες συναρτήσεις περιστροφής.

● H συνάρτηση **AVLFindMin** δέχεται ως ορίσματα έναν κόμβο ή ρίζα του δυαδικού δέντρου αναζήτησης και βρίσκει το ελάχιστο στοιχείο του υποδέντρου του και επιστρέφει την διεύθυνση του.

● H συναρτήσεις **AvlSearch** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης, ένα string και έναν ακέραιο αριθμό με αναφορά. Ψάχνει αναδρομικά το AVL μέχρι να βρει την λέξη. Στην περίπτωση που την βρει , αναθέτει στον ακέραιο αριθμό του ορίσματος το n του εκάστοτε κόμβου και επιστρέφει TRUE. Αντίθετα επιστρέφει FALSE.

●H συνάρτηση **AvlSearchNode** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης και ένα string. Ψάχνει το string αναδρομικά στο δέντρο όπως και η AvlSearch αλλά στην περίπτωση που βρεθεί επιστρέφει την διεύθυνση του εκάστοτε κόμβου. Αλλιώς επιστρέφει NULL.

●H συνάρτηση **Avldelete** δέχεται ως ορίσματα την ρίζα του δυαδικού δέντρου αναζήτησης, ένα string και έναν ακέραιο αριθμό (ο οποίος στην αρχή είναι πάντα 0). Αφού εκτελέσει την διαδικασία της διαγραφής όπως η DeleteEllement ,αφού γίνει η διαγραφή καθώς επιστρέφει από την αναδρομή της διαγραφής ελέγχει κάθε κόμβο αν χρειάζεται περιστροφή και κάνει ανάλογα.

Οι **preorder , postorder , inorder** λειτουργούν ανάλογα με το BST.