Τεχνική Έκθεση Εργασίας Δομών Δεδομένων 2020

Μπαρακλιλής Ιωάννης

ΑΕΜ: 3685

Email: imparakl@csd.auth.gr

Κατσάκη Σοφία

ΑΕΜ: 3656

Email: skatsaks@csd.auth.gr

Εισαγωγή: Η συνάρτηση main

Η συνάρτηση main ανοίγει το αρχείο με όνομα “input.txt”, διαβάζει μία προς μία τις γραμμές του αρχείου αυτού και τις αναλύει σε μεμονωμένες λέξεις τις οποίες εισάγει στις δομές BinarySearchTree και AVL_Tree. Παράλληλα, διαλέγει Q_SIZE (ορίζεται στη αρχή με #define) λέξεις που αποτελούν ένα σύνολο Q και τις αναζητά στις παραπάνω δομές. Για κάθε μία δομή δημιουργεί αντίστοιχο αρχείο κειμένου με όνομα “BinarySearchTree_output.txt”, “AVL_Search_Tree_output.txt” για τη δομή BinarySearchTree, AVL_Tree αντίστοιχα όπου τυπώνει τα αποτελέσματα της αναζήτησης κάθε στοιχείου του Q για κάθε δομή με μορφή:

“Word : [λέξη του Q], appeared: [αριθμός εμφανίσεων] times”

Τέλος, δημιουργεί ένα αρχείο με όνομα “Time_Count_output.txt” όπου τυπώνονται οι χρόνοι αναζήτησης (συμπεριλαμβανομένου το χρόνου της εκτύπωσης των αποτελεσμάτων στα αρχεία) σε millisecond όλων των λέξεων του Q για κάθε δομή.

Επίσης, πρέπει να σημειωθεί ότι σε αυτήν την υλοποίηση, λέξη θεωρείται ακολουθία αλφαβητικών χαρακτήρων (λατινικών).

Επιπλέον, πρέπει να σημειωθεί ότι αν υπάρξει η αδυναμία να ανοίξει οποιοδήποτε από τα παραπάνω αρχεία, το πρόγραμμα τερματίζει με κωδικό εξόδου 1.

Αναλυτικά η λειτουργία της main επιτελεί τα εξής:

Αρχικά, ανοίγει το αρχείο με όνομα input.txt (αν δεν ανοίξει, το πρόγραμμα τερματίζει με κωδικό εξόδου 1), γίνεται δέσμευση χώρου για το σύνολο Q (θέσεων Q_SIZE που δηλώνεται στην αρχή με #define) και δηλώνονται οι δύο δομές τύπου BinarySearchTree, AVL_Tree.

Στη συνέχεια, διαβάζονται μία προς μία οι γραμμές του input.txt και εισάγονται στις δομές με τρόπο: Περνώνται ένας-ένας οι χαρακτήρες της γραμμής και αν είναι αλφαβητικοί (λατινικοί) μετατρέπονται (ο έλεγχος για το αν είνας χαρακτήρας είναι αλφαβητικός καθώς η μετατροπή σε πεζό γίνονται με τη συνάρτηση processChar,η περιγραφή της οποίας βρίσκεται παρακάτω) σε πεζούς (αν δεν είναι ήδη) και επικολλούνται στο τέλος μίας προσωρινής συμβολοσειράς temp που αρχικά είναι κενή. Αν βρεθεί χαρακτήρας που δεν είναι αλφαβητικός, τότε αν η temp δεν είναι κενή δηλαδή αν περιέχει λέξη προς εισαγωγή (δηλαδή αν υπάρχουν πολλοί μη-αλφαβητικοί χαρακτήρες στην σειρά ώστε δεν εισάγεται το κενό) εισάγεται στις δομές και στη συνέχεια αρχικοποιείται σε κενή συμβολοσειρά για να χρησιμοποιηθεί στην συνέχεια για να εισάγει νέα λέξη. Όταν τελειώσουν οι χαρακτήρες μίας γραμμής γίνεται μία (τελευταία για την γραμμή) προσπάθεια εισαγωγής της temp (εφόσον είναι μη κενή) στις δομές εφόσον μπορεί να μην υπάρχει μη-αλφαβητικός χαρακτήρας στο τέλος της γραμμής που μπορεί να προκαλέσει εισαγωγή της temp στις δομές. Παράλληλα, σε κάθε εισαγωγή λέξης στις δομές καλείται η randomInsert (η περιγραφή της βρίσκεται παρακάτω) που δίνεται η temp ως όρισμα για να προστεθεί πιθανώς η temp στο σύνολο Q.

Μετά, ανοίγουν τα αρχεία “AVL_Search_Tree_output.txt” για τις δομές BinarySearchTree, AVL_Tree αντίστοιχα και το “Time_Count_output.txt” (αν δεν ανοίξουν, το πρόγραμμα τερματίζει με κωδικό εξόδου 1).

Μετά, γίνεται αναζήτηση των λέξεων του συνόλου Q σε κάθε δομή, τυπώνονται τα αποτελέσματα των αιτημάτων αναζήτησης στα αντίστοιχα αρχεία και αποθηκεύονται οι χρόνοι των ερωτημάτων και εκτύπωσης τους στο αρχείο (με την χρήση της βιβλιοθήκης chrono σε nanosecond που μετατρέπονται στην συνέχεια σε millisecond) οι οποίοι τυπώνονται στο τέλος στο αρχείο “Time_Count_output.txt”.

Τέλος κλείνουν όλα τα αρχεία που χρησιμοποίησε το πρόγραμμα.

Στο αρχείο main.cpp περιλαμβάνονται επίσης (όπως αναφέρθηκε και παραπάνω) και οι συναρτήσεις processChar και randomInsert:

- Η συνάρτηση bool processChar(char &c) δέχεται τον χαρακτήρα c ως όρισμα με αναφορά και ελέγχει αν είναι αλφαβητικός. Αν είναι, μετατρέπει τον c σε πεζό (αν δεν είναι ήδη) και επιστρέφει true. Διαφορετικά, αν δηλαδή δεν είναι πεζός, επιστρέφει false και δεν αλλάζει τον c.
- Η συνάρτηση void randomInsert(const std::string &word, std::string *Q, long &currQ) η οποία αποφασίζει αν θα εισαχθεί η word στο σύνολο Q και σε ποια θέση με τον τρόπο: Αν το currQ είναι μικρότερο από Q_SIZE (το Q έχει λιγότερα από Q_SIZE στοιχεία) η word εισάγεται στο Q στην θέση currQ (επόμενη της προηγούμενης που εισήχθηκε προηγουμένως ή στην πρώτη θέση αν είναι η πρώτη λέξη του Q) και αυξάνεται η τιμή του currQ. Ο σκοπός αυτού είναι να «γεμίσει» πρώτα ο πίνακας «σίγουρα» πριν εισάγουμε τυχαία τις λέξεις. Διαφορετικά, με πιθανότητα 15% εισάγεται η word σε μία τυχαία θέση του Q.

Μέρος πρώτο: Απλό δυαδικό δένδρο αναζήτησης

Μία σύντομη παρουσίαση του απλού δυαδικού δένδρου αναζήτησης

Η κλάση BinarySearchTree έχει ως μέλη δεδομένων τα:

- root τύπου BST_Node*, που είναι ένας δείκτης που δείχνει στην ρίζα (που είναι κόμβος) του δένδρου (οι λεπτομέρειες υλοποίησης του BST_Node βρίσκονται παρακάτω)
- numOfNodes τύπου long, που περιέχει το πλήθος των κόμβων του δένδρου

Ως μεθόδους public που μπορεί να καλέσει ο χρήστης έχει τις:

- bool insertNode (const std::string&). Ως λειτουργία του έχει να εισάγει την συμβολοσειρά εισόδου στο δένδρο αν δεν υπάρχει ή να αυξάνει τον μετρητή εμφανίσεων της συγκεκριμένης συμβολοσειράς σε περίπτωση που υπάρχει. Αν επιτύχει η εισαγωγή επιστέφεται true, διαφορετικά false.
- const BST_Node* searchNode (const std::string&). Ως λειτουργία του έχει να εκτελεί αναζήτηση στο δένδρο και να επιστρέφει const BST_Node δείκτη στον κόμβο που βρέθηκε ή nullptr σε περίπτωση που δεν το βρει.
- bool deleteNode (const std::string&). Ως λειτουργία του έχει να διαγράφει από το δένδρο τον κόμβο με αποθηκευμένη τιμή την τιμή της δοθείσας συμβολοσειράς. Αν επιτύχει η διαγραφή επιστέφεται true, διαφορετικά false.
- void inorder(std::ofstream &out). Ως λειτουργία του έχει να διαπερνά το δένδρο με ενδοδιατεταγμένη διαπέραση και να εμφανίζει στο ρεύμα εισόδου out τα δεδομένα κάθε κόμβου με μορφή: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”, όπου τα στοιχεία για κάθε κόμβο εμφανίζονται σε ξεχωριστή γραμμή.
- void preorder(std::ofstream &out). Ως λειτουργία του έχει να διαπερνά το δένδρο με προδιατεταγμένη διαπέραση και να εμφανίζει στο ρεύμα εισόδου out τα δεδομένα κάθε κόμβου με μορφή: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”, όπου τα στοιχεία για κάθε κόμβο εμφανίζονται σε ξεχωριστή γραμμή.
- void postorder(std::ofstream &out). Ως λειτουργία του έχει να διαπερνά το δένδρο με μεταδιατεταγμένη διαπέραση και να εμφανίζει στο ρεύμα εισόδου τα δεδομένα κάθε κόμβου με μορφή: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”, όπου τα στοιχεία για κάθε κόμβο εμφανίζονται σε ξεχωριστή γραμμή.
- int getNumberOfNodes(). Ως λειτουργία του έχει να επιστρέφει τον αριθμό των στοιχείων (κόμβων) του δένδρου.

Ως private μεθόδους που δεν έχει πρόσβαση ο χρήστης έχει τις:

- void deleteTree (BST_Node *pt). Ως λειτουργία του έχει να διαγράφει από την μνήμη κάθε κόμβο του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt.
- void internalPrintInorder(BST_Node *pt, std::ofstream &out). Τυπώνει στην έξοδο out τα δεδομένα κάθε κόμβου του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt με ενδοδιατεταγμένη (inorder) διαπέραση.
- void internalPrintPreorder(BST_Node *pt, std::ofstream &out). Τυπώνει στην έξοδο out τα δεδομένα κάθε κόμβου του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt με προδιατεταγμένη (preorder) διαπέραση.
- void internalPrintPostorder(BST_Node *pt, std::ofstream &out) . Τυπώνει στην έξοδο out τα δεδομένα κάθε κόμβου του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt με μεταδιατεταγμένη (postorder) διαπέραση.

- BST_Node** internalSearchNode(const std::string &a_word). Αναζητά στην δομή τον κόμβο με τιμή Data==a_word και επιστρέφει δείκτη σε μέλος leftchild ή rightchild που δείχνει στον κόμβο που αναζητείται.
- void deleteNode_NoChildren(BST_Node **found). Διαγράφει τον κόμβο που μπορεί να βρεθεί από την διεύθυνση **found με μεθοδολογία διαγραφής κόμβου απλού δυαδικού δένδρου αναζήτησης που δεν έχει κανένα παιδί.
- void deleteNode_OneChild(BST_Node **found). Διαγράφει τον κόμβο που μπορεί να βρεθεί από την διεύθυνση **found με μεθοδολογία διαγραφής κόμβου απλού δυαδικού δένδρου αναζήτησης που έχει ένα μόνο παιδί.
- void deleteNode_TwoChildren(BST_Node **found). Διαγράφει τον κόμβο μπορεί να βρεθεί από την διεύθυνση **found με μεθοδολογία διαγραφής κόμβου απλού δυαδικού δένδρου αναζήτησης που έχει δύο παιδιά.

Επιπλέον η κλάση BinarySearchTree έχει και

- Constructor χωρίς ορίσματα, που θέτει τον δείκτη της ρίζας σε nullptr (αφού το δένδρο έχει μόλις δημιουργηθεί και δεν έχει κόμβους).
- Destructor που διαγράφει όλα τα στοιχεία (κόμβους) από την μνήμη.

Η κλάση BST_Node:

Η κλάση BST_Node αποτελεί απεικόνιση των κόμβων που περιέχει το δυαδικό δένδρο αναζήτησης και περιέχει ως public μέλη:

- Data, τύπου std::string, όπου αποθηκεύεται η λέξη.
- timesAppeared, τύπου long, όπου αποθηκεύεται το πλήθος των εμφανίσεων της λέξης (που είναι αποθηκευμένη στο μέλος Data).
- leftChild, που είναι δείκτης σε αντικείμενο τύπου BST_Node, ο οποίος δείχνει στο αριστερό παιδί του κόμβου.
- rightChild, που είναι δείκτης σε αντικείμενο τύπου BST_Node, ο οποίος δείχνει στο δεξί παιδί του κόμβου.

Επιπλέον, η κλάση BST_Node έχει δύο constructors και έναν operator:

1. Έναν constructor χωρίς ορίσματα που:
    - Αρχικοποιεί την τιμή του Data με την κενή συμβολοσειρά (“”).
    - Αρχικοποιεί την τιμή του timesAppeared με 1.
    - Αρχικοποιεί τις τιμές των leftChild και rightChild σε nullptr (αφού ο κόμβος έχει μόλις δημιουργηθεί και δεν έχει παιδιά αφού είναι το πιο πρόσφατα δημιουργημένο αντικείμενο).
2. Έναν constructor με όρισμα το std::string a_word που:
    - Αρχικοποιεί την τιμή του Data σε a_word.
    - Αρχικοποιεί την τιμή του timesAppeared με 1.
    - Αρχικοποιεί τίς τιμές των leftChild και rightChild σε nullptr (αφού ο κόμβος έχει μόλις δημιουργηθεί και δεν έχει παιδιά αφού είναι το πιο πρόσφατα δημιουργημένο αντικείμενο).
3. Έναν (friend) ostream operator << που τυπώνει τα περιεχόμενα του κόμβου στο ρεύμα εξόδου ως εξής: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”.

Αναλυτική παρουσίαση των μεθόδων της κλάσης BinarySearchTree

Σημειώνεται εδώ ότι η παρουσίαση των μεθόδων δεν θα γίνει σύμφωνα με τις παραπάνω λίστες αλλά ιεραρχικά με την λογική με την οποία καλούνται (δηλαδή για παράδειγμα εφόσον η insertNode καλεί την internalSearchNode, κρίνεται χρήσιμο να γίνει πρώτα η περιγραφή της insertNode και να ακολουθεί η insertNode)

Η μέθοδος insertNode

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει μία bool τιμή με την κατάσταση της επιτυχίας της εισαγωγής. Έχει ως αποτέλεσμα την εισαγωγή της λέξης που ορίζει το std::string στην δομή (προσθήκη νέας λέξης ή αύξηση του μετρητή εμφανίσεων αν ήδη υπάρχει).

Η λειτουργία της insertNode είναι η ακόλουθη:

Αρχικά, γίνεται έλεγχος αν η ρίζα είναι το nullptr (που σημαίνει κενό δένδρο), όπου απλά δημιουργείται και εισάγεται καινούριο BST_Node ως ρίζα και αυξάνεται η τιμή του πλήθους των κόμβων του δένδρου κατά 1.

Αν το δένδρο δεν είναι κενό (root!=nullptr), τότε γίνεται κλήση της internalSearchNode, με όρισμα το std::string που δόθηκε ως όρισμα στην insertNode, και η τιμή που αυτή επιστρέφει αν είναι != nullptr, σημαίνει ότι η συμβολοσειρά υπάρχει οπότε αρκεί να αυξηθεί ο μετρητής εμφανίσεων του κόμβου της. Διαφορετικά, δημιουργείται καινούριο BST_Node που προστίθεται εκεί που «θα ήταν αν ήδη υπήρχε στο δένδρο» (η διεύθυνση του ανατίθεται στο *found) και αυξάνεται η τιμή του μετρητή πλήθους των κόμβων του δένδρου κατά 1.

Τέλος, οι παραπάνω διαδικασίες βρίσκονται σε block try-catch οπότε σε περίπτωση που εμφανιστεί πρόβλημα κατά την δέσμευση μνήμης, σταματάει η διαδικασία εισαγωγής νέου στοιχείου και επιστρέφεται false. Διαφορετικά, αν δεν έχει διακοπεί η εκτέλεση στα try-catch blocks, επιστρέφεται true.

(Αξίζει να σημειωθεί ότι το found είναι δείκτης σε δείκτη (που είναι μέλος ενός κόμβου έστω Χ) και δείχνει σε κόμβο-παιδί δηλαδή το found δείχνει σε ένα μέλος leftChild ή ένα μέλος rightChild. Αυτό σημαίνει ότι το *found μας δίνει την δυνατότητα να αλλάξουμε το παιδί του κόμβου αυτού (Χ). Δηλαδή πχ. Αν σε εισαγωγή μίας συμβολοσειράς s, ο κόμβος Χ δεν έχει αριστερό παιδί και η συμβολοσειρά s είναι μικρότερη της συμβολοσειράς του Χ τότε η αναζήτηση της s στον δένδρο θα μας επιστρέψει found με τιμή &(Χ.leftChild). Άρα αρκεί να δημιουργήσω έναν κόμβο με συμβολοσειρά s και να θέσω ως τιμή του *found ίση με την διεύθυνση του πρόσφατα δημιουργηθέντα κόμβου. Περαιτέρω λεπτομέρειες της λειτουργίας δεικτών όπως του found βρίσκεται στην συνέχεια στην περιγραφή της internalSearchNode)

Η μέθοδος internalSearchNode.

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει BST_Node** (δείκτη σε δείκτη σε BST_Node).

Έχει δύο τοπικές μεταβλητές, τις:

- currentNode, που δείχνει στον κόμβο που ελέγχουμε στο συγκεκριμένο βήμα και
- pointTo, που δείχνει στον δείκτη (μέλος κλάσης leftChild/rightChild) προγόνου (έστω p) , όπου ο p δείχνει στον κόμβο (τον έχει ως παιδί) που ελέγχουμε στο συγκεκριμένο βήμα.

(Για να γίνει πιο κατανοητή η λειτουργία του pointTo, είναι ωφέλιμο να δοθεί ένα παράδειγμα. Πχ αν το currentNode δείχνει στον κόμβο Υ που είναι αριστερό παιδί του Χ, το pointTo θα δείχνει στο X.leftChild. Στο τέλος της αναζήτησης, το **pointTo δίνει το αντικείμενο που είναι αποτέλεσμα της αναζήτησης)

Σκοπός του pointTo είναι σε εισαγωγή και διαγραφή, να μην αποθηκεύονται ξεχωριστά ο κόμβος, ο γονέας του και η πληροφορία για το αν είναι αριστερό ή δεξί παιδί (και ούτε να γίνονται πολλαπλές αναζητήσεις για να βρεθούν) αλλά όλα αυτά τα δεδομένα να αντιπροσωπεύονται από μία μόνο μεταβλητή, την pointTo, και αν χρειαστεί πρόσβαση στον κόμβο που αναζητήθηκε χρησιμοποιείται το **pointTo, ενώ αν χρειαστεί ο γονέας του αναζητηθέντος στοιχείου να δείξει αντί για αυτό σε ένα άλλο χρησιμοποιείται ανάθεση της μορφής *pointTo = BST_Node_POINTER_NAME.)

Λειτουργία της internalSearchNode:

Αρχικά, το currentNode δείχνει στον ίδιο κόμβο με αυτόν που δείχνει η ρίζα και το pointTo δείχνει στο μέλος (της κλάσης του δένδρου) που δείχνει η ρίζα (ώστε τελικά αν η ρίζα είναι ο κόμβος που ψάχνουμε να επιστραφεί ο αντίστοιχος pointTo). Στη συνέχεια, όσο το currentNode δεν δείχνει σε nullptr (η αναζήτηση συνεχίζεται και δεν έχω φτάσει στο τέλος ενός «κλαδιού») εκτελείται επαναληπτικά η διαδικασία:

Αν η συμβολοσειρά εισόδου είναι λεξικογραφικά μεγαλύτερη από την συμβολοσειρά αποθηκευμένη στον κόμβο που δείχνει το currentNode, τότε γίνεται αναζήτηση στο δεξί υποδένδρο, διαφορετικά γίνεται αναζήτηση στο αριστερό υποδένδρο. Τέλος, αν είναι ίση τότε σημαίνει ότι έχει βρεθεί το στοιχείο και επιστρέφεται η τιμή του pointTo. Η αναζήτηση σε αριστερό/δεξί υποδένδρο αναφέρεται σε ενημέρωση των pointTo και του currentNode να δείχνουν σε αντίστοιχους κόμβους (παιδιά).

Αν κατά την παραπάνω επαναληπτική διαδικασία δεν βρεθεί κόμβος με ίδια συμβολοσειρά, επιστρέφεται nullptr (το στοιχείο που ψάχνουμε δεν υπάρχει στο δένδρο).

Η μέθοδος deleteNode

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει μία bool τιμή με την κατάσταση της επιτυχίας της διαγραφής. Έχει ως αποτέλεσμα την ολοκληρωτική διαγραφή του κόμβου που αποθηκεύει την λέξη που δίνεται ως όρισμα από την δομή.

Αναλυτικά:

Αρχικά, ελέγχεται αν το δένδρο είναι άδειο και αν είναι επιστρέφεται κατάσταση επιτυχίας false και τερματίζεται η συνάρτηση (δεν υπάρχουν κόμβοι για να διαγραφούν).

Έπειτα, ελέγχεται αν το στοιχείο που μας δίνεται για διαγραφή υπάρχει στο δένδρο και γίνεται η αναζήτηση του με το internalSearchNode και αποθηκεύεται η επιστρεφόμενη τιμή. Αν δεν υπάρχει (επιστρέφεται nullptr), επιστρέφεται κατάσταση επιτυχίας false και τερματίζεται η συνάρτηση.

Τέλος, μειώνεται το πλήθος των κόμβων (αφού διαγράφω έναν), εξετάζεται το πλήθος παιδιών του κόμβου που βρέθηκε και καλείται η αντίστοιχη μέθοδος διαγραφής (από αυτές που περιγράφονται παρακάτω) για τον κόμβο που βρέθηκε από το internalSearchNode.

Η μέθοδος deleteNode_NoChildren

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή BST_Node** και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή ενός κόμβου θεωρώντας ότι αυτός δεν έχει παιδιά.

Λειτουργικά, απλά διαγράφει το παιδί χρησιμοποιώντας κατάλληλα τον δείκτη του ορίσματος και να θέτει στο αντίστοιχο μέλος(δείκτη σε παιδί) σε nullptr.

Η μέθοδος deleteNode_OneChild

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή BST_Node** και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή ενός κόμβου θεωρώντας ότι έχει μόνο ένα παιδί.

Λειτουργικά, εφόσον ο κόμβος έχει μοναδικό παιδί αντικαθιστά τον εαυτό του (αντικαθίστανται όλα τα δεδομένα του) με το παιδί του και διαγράφεται το «παλιό» παιδί.

Η μέθοδος deleteNode_TwoChildren

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή BST_Node** και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή ενός κόμβου θεωρώντας ότι έχει δύο παιδιά.

Αναλυτικά, η μέθοδος αυτή:

Αρχικά, ψάχνει το μεγαλύτερο παιδί του αριστερού υποδένδρου με μέθοδο όμοια με την internalSearchNode με την διαφορά ότι εκτός από τις pointTo και currentNode υπάρχει και μία μεταβλητή (BST_Node*) buffer όπου το currentNode «βρίσκεται» πάντα «έναν κόμβο πιο μπροστά» και το buffer «κρατάει» την προηγούμενη τιμή του currentNode, ώστε όταν τελικά ο currentNode «φτάσει στο τέλος» και «γίνει» nullptr να έχουμε δείκτη στον τελευταίο κόμβο πριν το nullptr (που είναι αυτός που ψάχνουμε) και το pointTo να είναι ενημερωμένο κατάλληλα.

Έπειτα, χρησιμοποιώντας τον δείκτη pointTo, αντικαθίσταται (αλλάζουν οι τιμές) του στοιχείου προς διαγραφή με τον κόμβο που μπορώ να πάρω από το pointTo.

Τέλος, ο κόμβος που παίρνω από το buffer πρέπει να διαγραφεί οπότε διαγράφεται με τον κατάλληλο τρόπο (ανάλογα αν έχει ή όχι παιδί).

Η μέθοδος searchNode

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει ένα const BST_Node* που δείχνει στον κόμβο που βρέθηκε ή σε nullptr αν δεν βρέθηκε.

Λειτουργικά είναι όμοια με την internalSearchNode απλά χωρίς την παρουσία του pointTo (γιατί δεν είναι απαραίτητο αφού δεν πρόκειται να κάνουμε αλλαγές στο δένδρο).

Αναλυτικά, χειρίζεται έναν δείκτη currentNode που αρχικά δείχνει στην ρίζα (τον κόμβο). Στη συνέχεια εκτελεί επαναληπτικά την ακόλουθη διαδικασία όσο το currentNode δεν δείχνει σε nullptr (αν currentNode==nullptr τότε το στοιχείο προς αναζήτηση δεν υπάρχει): Αν η λέξη του «τρέχοντα» κόμβου (αυτόν που δείχνει ο currentNode) είναι λεξικογραφικά μεγαλύτερη από την λέξη του ορίσματος τότε γίνεται αναζήτηση στο δεξί υποδένδρο (το currentNode δείχνει στο δεξί παιδί του κόμβου που δείχνει «τώρα»). Αντίστοιχα, αν η λέξη είναι λεξικογραφικά μικρότερη γίνεται αναζήτηση αριστερά. Η διαφορετική περίπτωση από τις δύο παραπάνω σημαίνει ότι βρέθηκε ο κόμβος προς αναζήτηση άρα και επιστρέφεται ο currentNode.

Αν η παραπάνω επαναληπτική διαδικασία τερματιστεί με currentNode==nullptr (δεν βρέθηκε το στοιχείο), τότε επιστρέφεται nullptr.

Οι μέθοδοι inorder, preorder, postorder

Αυτές οι μέθοδοι έχουν ως όρισμα μία τιμή std::ofstream με αναφορά και δεν επιστρέφουν κάτι. Έχουν ως αποτέλεσμα να τυπώνεται στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του με τρόπο ενδοδιατεταγμένης (ή προδιατεταγμένης ή μεταδιατεταγμένης αντίστοιχα) διαπέρασης.

Η ύπαρξη τους οφείλεται στην ανάγκη για ενθυλάκωση της κλάσης του δένδρου, αφού για να τυπωθούν οι κόμβοι αναδρομικά χρειάζεται πρώτα να δοθεί αρχική τιμή ως όρισμα η ρίζα κάτι που δεν πρέπει να επιτραπεί ως δυνατότητα στον χρήστη γιατί εκείνος χρειάζεται πρόσβαση στην ρίζα, κάτι επικίνδυνο για την δομή.

Λειτουργικά, απλά καλούν την συνάρτηση internalPrintInorder ή internalPrintPreorder ή internalPrintPostorder αντίστοιχα με όρισμα την ρίζα του δένδρου και το ίδιο ρεύμα εξόδου που οι ίδιες δέχτηκαν ως όρισμα.

Η μέθοδος internalPrintInorder

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή BST_Node* και μία τιμή std::ofstream με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα να τυπώνει στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα με τρόπο ενδόδιατεταγμένης διαπέρασης.

Αναλυτικά:

Αρχικά, αν το όρισμα είναι nullptr τερματίζεται η εκτέλεση της συνάρτησης.

Στη συνέχεια, καλείται αναδρομικά η ίδια συνάρτηση για το αριστερό παιδί με ίδιο ρεύμα εξόδου.

Μετά, τυπώνονται τα δεδομένα του κόμβου του ορίσματος με ακόλουθο τρόπο: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”.

Τέλος, καλείται αναδρομικά η ίδια συνάρτηση για το δεξί παιδί με ίδιο ρεύμα εξόδου.

Δηλαδή αυτή η μέθοδος επιτυγχάνει ενδοδιατεταγμένη διαπέραση, αφού τελικά όταν φτάσει να καλεστεί με όρισμα ένα φύλλο αυτό απλά θα τυπωθεί. Αυτές οι εκτυπώσεις γίνονται με σειρά κλήσης, άρα τελικά ενδοδιατεταγμένα.

Η μέθοδος internalPrintPreorder

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή BST_Node* και μία τιμή std::ofstream με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα να τυπώνει στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα με τρόπο προδιατεταγμένης διαπέρασης.

Λειτουργικά είναι σχεδόν πανομοιότυπη με την μέθοδο internalPrintInorder με την διαφορά ότι αλλάζει η σειρά αναδρομικών κλήσεων της. Δηλαδή, πρώτα ελέγχει για δείκτης == nullptr, μετά τυπώνει τα δεδομένα του κόμβου, έπειτα καλείται αναδρομικά για το αριστερό παιδί και τέλος καλείται αναδρομικά για το δεξί παιδί.

Η μέθοδος internalPrintPostorder

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή BST_Node* και μία τιμή std::ofstream με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα να τυπώνει στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα με τρόπο μεταδιατεταγμένης διαπέρασης.

Λειτουργικά είναι σχεδόν πανομοιότυπη με την μέθοδο internalPrintInorder με την διαφορά ότι αλλάζει η σειρά αναδρομικών κλήσεων της. Δηλαδή, πρώτα ελέγχει για δείκτης == nullptr, μετά καλείται αναδρομικά για το αριστερό παιδί , έπειτα καλείται αναδρομικά για το δεξί παιδί και τέλος τυπώνει τα δεδομένα του κόμβου.

Ο κατασκευαστής της BinarySearchTree

Απλά αρχικοποιούνται τα μέλη root σε nullptr (αρχικά έχω κενό δένδρο) και numOfNodes σε 0 (δεν έχω κόμβους)

Ο καταστροφέας του BinarySearchTree

Απλά καλείται η μέθοδος deleteTree με όρισμα την ρίζα. Η ύπαρξη του οφείλεται όπως και με τις μεθόδους για διαπεράσεις, στην ενθυλάκωση.

Η μέθοδος deleteTree

Αυτή η μέθοδος έχει όρισμα μία τιμή BST_Node* και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή από την μνήμη του (υπό)δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα.

Λειτουργικά, είναι πανομοιότυπη με την internalPrintPostorder με τις διαφορές ότι δεν δέχεται ως όρισμα ρεύμα εισόδου και αντί να τυπώνει δεδομένα διαγράφει τον κόμβο από την μνήμη.

Αναλυτικά:

Αρχικά, αν το όρισμα είναι nullptr τερματίζεται η εκτέλεση της συνάρτησης.

Στη συνέχεια, καλείται αναδρομικά η ίδια συνάρτηση για το αριστερό παιδί.

Μετά, καλείται αναδρομικά η ίδια συνάρτηση για το δεξί παιδί.

Τέλος, διαγράφεται από την μνήμη ο κόμβος που δίνεται από το όρισμα.

Δηλαδή αυτή η μέθοδος επιτυγχάνει να διαγράψει κάθε κόμβο του δένδρου από την μνήμη, αφού τελικά όταν φτάσει να καλεστεί με όρισμα ένα φύλλο αυτό απλά θα διαγραφεί. Αυτό όμως θα συμβεί αφού έχουν διαγραφεί τυχών παράγωγοι κόμβοι, άρα δεν υπάρχει κίνδυνος διαρροής μνήμης.

Μέρος δεύτερο: Δυαδικό δένδρο αναζήτησης τύπου AVL

Μία σύντομη παρουσίαση του δυαδικού δένδρου αναζήτησης τύπου AVL

Η κλάση AVL_Tree έχει ως μέλη δεδομένων τα:

- root τύπου AVL_Node*, που είναι ένας δείκτης που δείχνει στην ρίζα του δένδρου (οι λεπτομέρειες υλοποίησης του AVL_Node βρίσκονται παρακάτω)
- numOfNodes τύπου long, που περιέχει το πλήθος των κόμβων του δένδρου
- height τύπου long, που περιέχει το ύψος του δένδρου

Ως μεθόδους public που μπορεί να καλέσει ο χρήστης έχει τις:

- long getheight(). Ως λειτουργία του έχει να επιστρέφει την τιμή του ύψους του δένδρου.
- bool insertNode (const std::string&). Ως λειτουργία του έχει να εισάγει την συμβολοσειρά εισόδου στο δένδρο αν δεν υπάρχει ή να αυξάνει τον μετρητή εμφανίσεων της συγκεκριμένης συμβολοσειράς σε περίπτωση που υπάρχει. Αν επιτύχει η εισαγωγή επιστέφεται true, διαφορετικά false.
- const AVL_Node * searchNode (const std::string&). Ως λειτουργία του έχει να εκτελεί αναζήτηση στο δένδρο και να επιστρέφει const AVL_Node δείκτη στον κόμβο που βρέθηκε ή nullptr σε περίπτωση που δεν το βρει.
- bool deleteNode (const std::string&). Ως λειτουργία του έχει να διαγράφει από το δένδρο τον κόμβο με αποθηκευμένη τιμή την τιμή της δοθείσας συμβολοσειράς. Αν επιτύχει η διαγραφή επιστέφεται true, διαφορετικά false.
- void inorder(std::ofstream &out). Ως λειτουργία του έχει να διαπερνά το δένδρο με ενδοδιατεταγμένη διαπέραση και να εμφανίζει στο ρεύμα εξόδου out τα δεδομένα κάθε κόμβου του δένδρου με μορφή: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”, όπου τα στοιχεία για κάθε κόμβο εμφανίζονται σε ξεχωριστή γραμμή.
- void preorder(std::ofstream &out). Ως λειτουργία του έχει να διαπερνά το δένδρο με προδιατεταγμένη διαπέραση και να εμφανίζει στο ρεύμα εξόδου out τα δεδομένα κάθε κόμβου του δένδρου με μορφή: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”, όπου τα στοιχεία για κάθε κόμβο εμφανίζονται σε ξεχωριστή γραμμή.
- void postorder(std::ofstream &out). Ως λειτουργία του έχει να διαπερνά το δένδρο με μεταδιατεταγμένη διαπέραση και να εμφανίζει στο ρεύμα εξόδου out τα δεδομένα κάθε κόμβου του δένδρου με μορφή: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”, όπου τα στοιχεία για κάθε κόμβο εμφανίζονται σε ξεχωριστή γραμμή.
- int getNumberOfNodes(). Ως λειτουργία του έχει να επιστρέφει τον αριθμό των στοιχείων (κόμβων) του δένδρου.

Ως private μεθόδους που δεν έχει πρόσβαση ο χρήστης έχει τις:

- void deleteTree (AVL_Node *pt). Ως λειτουργία του έχει να διαγράφει από την μνήμη κάθε κόμβο του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt.
- void internalPrintInorder(AVL_Node *pt, std::ofstream &out). Τυπώνει στην έξοδο out τα δεδομένα κάθε κόμβου του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt με ενδοδιατεταγμένη (inorder) διαπέραση.
- void internalPrintPreorder (AVL_Node *pt, std::ofstream &out). Τυπώνει στην έξοδο out τα δεδομένα κάθε κόμβου του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt με προδιατεταγμένη (preorder) διαπέραση.
- void internalPrintPostorder (AVL_Node *pt, std::ofstream &out) . Τυπώνει στην έξοδο out τα δεδομένα κάθε κόμβου του δένδρου με ρίζα τον κόμβο που δείχνει ο δείκτης pt με μεταδιατεταγμένη (postorder) διαπέραση.

- void updateTree_Insert(AVL_Node** routeStack, int top). Ενημερώνει (αυξάνει) τις μεταβλητές ύψους και κάνει περιστροφές, εφόσον χρειαστεί. Χρησιμοποιείται μετά από εισαγωγή.
- void updateTree_Delete(AVL_Node** routeStack, int top). Ενημερώνει τις μεταβλητές ύψους και κάνει περιστροφές, εφόσον χρειαστεί. Χρησιμοποιείται μετά από διαγραφή.
- void leftRotation(AVL_Node* targetNode, AVL_Node** targetPtrPtr). Κάνει μία αριστερή περιστροφή στο παιδί στο οποίο εμφανίζεται πρόβλημα.
- void rightRotation(AVL_Node* targetNode, AVL_Node** targetPtrPtr). Κάνει μία αριστερή περιστροφή στο παιδί στο οποίο εμφανίζεται πρόβλημα.
- AVL_Node** internalSearchNode_WithStack(const std::string &a_word, AVL_Node** &stackOfSerches, int &top). Αναζητά στην δομή τον κόμβο με τιμή Data==a_word και επιστρέφει δείκτη σε μέλος leftchild ή rightchild που δείχνει στον κόμβο που αναζητείται. Επιπλέον, προσθέτει στο array stackOfSerches (που δέχεται δείκτη με αναφορά) τους δείκτες AVL_Node* των κόμβων που συναντά κατά την αναζήτηση σε μορφή στοίβας. Τέλος, ενημερώνει την μεταβλητή top που δέχεται με αναφορά για την θέση της κορυφής της στοίβας.
- void deleteNode_NoChildren(AVL_Node ** targetPtrPtr, AVL_Node** &routeStack, int& top). Διαγράφει τον κόμβο που βρίσκεται στην διεύθυνση ** targetPtrPtr με μεθοδολογία διαγραφής κόμβου δυαδικού δένδρου αναζήτησης που δεν έχει κανένα παιδί ενώ παράλληλα ενημερώνει την στοίβα routeStack.
- void deleteNode_OneChild (AVL_Node ** targetPtrPtr, AVL_Node** &routeStack, int& top). Διαγράφει τον κόμβο που βρίσκεται στην διεύθυνση ** targetPtrPtr με μεθοδολογία διαγραφής κόμβου δυαδικού δένδρου αναζήτησης που έχει ένα μόνο παιδί ενώ παράλληλα ενημερώνει την στοίβα routeStack.
- void deleteNode_TwoChildren (AVL_Node ** targetPtrPtr, AVL_Node** &routeStack, int& top). Διαγράφει τον κόμβο που βρίσκεται στην διεύθυνση ** targetPtrPtr με μεθοδολογία διαγραφής κόμβου δυαδικού δένδρου αναζήτησης που έχει δύο παιδιά ενώ παράλληλα ενημερώνει την στοίβα routeStack.

Επιπλέον η κλάση AVL_Tree έχει και

- Constructor χωρίς ορίσματα, που θέτει τον δείκτη της ρίζας σε nullptr (αφού το δένδρο έχει μόλις δημιουργηθεί και δεν έχει κόμβους). Επίσης, θέτει τον αριθμό κόμβων (numOfNodes) και το ύψος του δένδρου (height) σε 0.
- Destructor που διαγράφει όλα τα στοιχεία (κόμβους) από την μνήμη.

Η κλάση AVL_Node:

Η κλάση AVL_Node αποτελεί απεικόνιση των κόμβων που περιέχει το δυαδικό δένδρο αναζήτησης AVL και περιέχει ως public μέλη:

- Data, τύπου std::string όπου αποθηκεύεται η λέξη.
- timesAppeared, τύπου long όπου αποθηκεύεται το πλήθος των εμφανίσεων της λέξης (που είναι αποθηκευμένη στο μέλος Data).
- leftChild, που είναι δείκτης σε αντικείμενο τύπου AVL_Node, ο οποίος δείχνει στο αριστερό παιδί του κόμβου.
- rightChild, που είναι δείκτης σε αντικείμενο τύπου AVL_Node, ο οποίος δείχνει στο δεξί παιδί του κόμβου.
- leftHeight, τύπου long που αποθηκεύει το ύψος του αριστερού υποδένδρου.
- rightHeight, τύπου long που αποθηκεύει το ύψος του δεξιού υποδένδρου.
- Την μέθοδο long getHeight() που επιστρέφει το ύψος του κόμβου.

Επιπλέον, η κλάση AVL_Node έχει δύο constructors και έναν operator:

1. Έναν constructor χωρίς ορίσματα που:
    - Αρχικοποιεί την τιμή του Data με την κενή συμβολοσειρά (“”).
    - Αρχικοποιεί την τιμή του timesAppeared με 1.
    - Αρχικοποιεί τις τιμές των leftChild και rightChild σε nullptr (αφού ο κόμβος έχει μόλις δημιουργηθεί και δεν έχει παιδιά αφού είναι το πιο πρόσφατα δημιουργημένο αντικείμενο).

    - Αρχικοποιεί τις τιμές των leftHeight και rightHeight σε 0 (αφού ο κόμβος έχει μόλις δημιουργηθεί και δεν έχει παιδιά αφού είναι το πιο πρόσφατα δημιουργημένο αντικείμενο).
2. Έναν constructor με όρισμα το std::string a_word που:
    - Αρχικοποιεί την τιμή του Data σε a_word.
    - Αρχικοποιεί την τιμή του timesAppeared με 1.
    - Αρχικοποιεί τις τιμές των leftChild και rightChild σε nullptr (αφού ο κόμβος έχει μόλις δημιουργηθεί και δεν έχει παιδιά αφού είναι το πιο πρόσφατα δημιουργημένο αντικείμενο).
    - Αρχικοποιεί τις τιμές των leftHeight και rightHeight σε 0 (αφού ο κόμβος έχει μόλις δημιουργηθεί και δεν έχει παιδιά αφού είναι το πιο πρόσφατα δημιουργημένο αντικείμενο).
3. Έναν (friend) ostream operator << που τυπώνει τα περιεχόμενα του κόμβου στο ρεύμα εξόδου ως εξής: “Word : [συμβολοσειρά αποθηκευμένη στον κόμβο], appeared: [αριθμός εμφανίσεων] times”.

Αναλυτική παρουσίαση των μεθόδων της κλάσης AVL_Tree

Σημειώνεται εδώ ότι η παρουσίαση των μεθόδων γίνεται όμοια με την παρουσίαση της BinarySearchTree, ενώ παράλληλα γίνονται αναφορές σε μεθόδους της κλάσης αυτής για να αποφευχθούν άσκοπες επαναλήψεις.

Η μέθοδος insertNode

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει μία bool τιμή με την κατάσταση της επιτυχίας της εισαγωγής. Έχει ως αποτέλεσμα την εισαγωγή της λέξης που ορίζει το std::string στην δομή (προσθήκη νέας λέξης ή αύξηση του μετρητή εμφανίσεων αν ήδη υπάρχει).

Αναλυτικά,

Αρχικά, γίνεται έλεγχος αν η ρίζα είναι το nullptr (που σημαίνει κενό δένδρο), όπου απλώς δημιουργείται και εισάγεται καινούριο AVL_Node ως ρίζα και αυξάνεται η τιμή του πλήθους των κόμβων του δένδρου κατά 1.

Αν το δένδρο δεν είναι κενό (root!=nullptr), τότε δεσμεύεται χώρος δυναμικά για ένα array (με αρκετό χώρο για «χωρέσει» όλους τους δείκτες από την ρίζα μέχρι και το φύλλο, συμπεριλαμβανομένου και του νέου στοιχείου) και γίνεται κλήση της internalSearchNode_WithStack και η τιμή που αυτή επιστρέφει αποθηκεύεται στην μεταβλητή targetPtrPtr. Μετά από αυτό, το array στο οποίο δείχνει το routeStack θα περιέχει τους κατάλληλους δείκτες (κάθε AVL_Node* που συναντήθηκε κατά την αναζήτηση, από την ρίζα μέχρι το κόμβο όπου τερμάτισε η αναζήτηση) που χρειάζονται για εισαγωγή.

Αν η * targetPtrPtr != nullptr, σημαίνει ότι η συμβολοσειρά υπάρχει οπότε αρκεί να αυξηθεί ο μετρητής εμφανίσεων του κόμβου της. Διαφορετικά, δημιουργείται καινούριο AVL_Node και προστίθεται εκεί που «θα ήταν αν ήδη υπήρχε στο δένδρο» (η διεύθυνση του ανατίθεται στο *found) και αυξάνεται η τιμή του μετρητή πλήθους των κόμβων του δένδρου κατά 1.

Στη συνέχεια, προστίθεται στην στοίβα δείκτης AVL_Node* που αντιστοιχεί στο νέο στοιχείο και καλείται η μέθοδος updateTree_Insert για να ενημερώσει τους κόμβους που μπορεί να επηρεάστηκαν (αυτοί που υπάρχουν στην στοίβα). Μετά τον τερματισμό αυτής της μεθόδου θα έχουν γίνει κατάλληλες επεξεργασίες στο δένδρο ώστε αυτό να διατηρεί τις ιδιότητες ενός δένδρου AVL μετά την εισαγωγή του νέου στοιχείου.

Τέλος, οι παραπάνω διαδικασίες βρίσκονται σε block try-catch όπου σε περίπτωση που εμφανιστεί πρόβλημα κατά την δέσμευση μνήμης, σταματάει η διαδικασία εισαγωγής νέου στοιχείου και επιστρέφεται false. Διαφορετικά, αν δεν έχει διακοπεί η εκτέλεση στα try-catch blocks, επιστρέφεται true.

(Αξίζει να σημειωθεί ότι το targetPtrPtr είναι δείκτης σε δείκτη (leftChild ή rightChild) (που είναι μέλος κόμβου , έστω X) που δείχνει σε κόμβο-παιδί. Αυτό σημαίνει ότι το *targetPtrPtr μας δίνει την δυνατότητα να αλλάξουμε το παιδί του κόμβου αυτού (X). Δηλαδή πχ. Αν σε εισαγωγή μίας συμβολοσειράς s, ο κόμβος X δεν έχει αριστερό παιδί και η συμβολοσειρά s είναι μικρότερη της συμβολοσειράς του X τότε η αναζήτηση της s στον δένδρο θα μας επιστρέψει targetPtrPtr με τιμή &(X.leftChild). Άρα αρκεί να δημιουργήσω έναν κόμβο με συμβολοσειρά s και να θέσω ως τιμή του *targetPtrPtr ίση με την διεύθυνση του πρόσφατα δημιουργηθέντα κόμβου.)

Η μέθοδος updateTree_Insert

Η μέθοδος αυτή, δέχεται ως όρισμα έναν δείκτη AVL_Node** και την θέση της κορυφής της στοίβας και δεν επιστρέφει κάτι. Η λειτουργία του είναι να ενημερώνει τα ύψη των κόμβων που επηρεάζονται (αν χρειάζεται) και να εκτελεί περιστροφές όταν διασπάται η ισορροπία του δένδρου.

Αναλυτικά:

Για κάθε στοιχείο του routeStack (δηλαδή για κάθε στοιχείο στο μονοπάτι από την ρίζα μέχρι το νέο στοιχείο) εκτελείται η επαναληπτική διαδικασία:

Αρχικά, ελέγχεται αν το τρέχον στοιχείο προέρχεται από δεξί παιδί ή αριστερό (για να ενημερωθεί το κατάλληλο ύψος γονέα και να ελεγχθεί η αντίστοιχη περίπτωση ανισορροπίας στην συνέχεια).

Αν είναι δεξί παιδί, τότε αυξάνεται το ύψος του δεξιού υποδένδρου του γονέα του και ελέγχεται αν υπάρχει ανισορροπία στα δεξιά (στα αριστερά δεν μπορεί να υπάρξει ανισορροπία αφού αυξάνεται δεξί ύψος, οπότε και παραλείπεται ο έλεγχος για αριστερά). Στη συνέχεια, ελέγχονται οι πιθανές περιπτώσεις ανισορροπίας και αντιμετωπίζονται κατάλληλα (μία απλή αριστερή περιστροφή για περίπτωση “δεξιά-δεξιά” και πρώτα περιστροφή δεξιά και μετά αριστερά για περίπτωση “δεξιά-αριστερά”) ανάλογα με το αν είναι ρίζα ή όχι (αλλάζουν ελαφρώς τα ορίσματα αλλά η διαδικασία είναι ίδια).

Σε περίπτωση που γίνει κάποια περιστροφή, αποδεικνύεται ότι το δένδρο είναι μετά ισορροπημένο οπότε τερματίζεται η εκτέλεση της μεθόδου. Επιπλέον, αν το νέο δεξί ύψος είναι μικρότερο ή ίσο από το αριστερό ύψος σημαίνει ότι ο γονέας του τρέχοντα (το στοιχείο που εξετάζεται την τρέχουσα επανάληψη) και οι πρόγονοι του κόμβου έχουν ήδη κατάλληλο ύψος, οπότε και τερματίζεται η εκτέλεση της μεθόδου γιατί υπάρχει ήδη ισορροπία.

Αν είναι αριστερό παιδί, γίνονται ανάλογες διαδικασίες (αύξηση αριστερού ύψους γονέα, έλεγχος για ανισορροπίες αριστερά και κατάλληλη αντιμετώπιση τους (μία απλή δεξιά περιστροφή για περίπτωση “αριστερά-αριστερά” και πρώτα περιστροφή αριστερά και μετά δεξιά για περίπτωση “αριστερά-δεξιά”)).

Πάλι, Σε περίπτωση που γίνει κάποια περιστροφή, τερματίζεται η εκτέλεση της μεθόδου. Επιπλέον, αν το νέο αριστερό ύψος είναι μικρότερο ή ίσο από το δεξί ύψος σημαίνει ότι ο γονέας του τρέχοντα (το στοιχείο που εξετάζεται την τρέχουσα επανάληψη) κόμβου και οι πρόγονοι του έχουν ήδη κατάλληλο ύψος, οπότε και τερματίζεται η εκτέλεση της μεθόδου γιατί υπάρχει ήδη ισορροπία.

Η δεξιά και αριστερή περιστροφή στις οποίες γίνεται λόγος παραπάνω γίνονται με την κλήση των μεθόδων rightRotation και leftRotation αντίστοιχα, η αναλυτική περιγραφή των οποίων γίνεται παρακάτω.

Η μέθοδος internalSearchNode_WithStack.

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const), έναν δείκτη AVL_Node** με αναφορά και την θέση της κορυφής της στοίβας με αναφορά και επιστρέφει AVL_Node** (δείκτη σε δείκτη σε AVL_Node). Παράλληλα, ενημερώνει το array του οποίου τον δείκτη δέχεται με αναφορά προσθέτοντας ως νέα στοιχεία τους δείκτες AVL_Node** τους οποίους “συναντά” κατά την διάρκεια της αναζήτησης.

Λειτουργικά είναι όμοια με την αντίστοιχη μέθοδο internalSearchNode του BinarySearchTree, με την διαφορά ότι κάθε φορά που γίνεται αναζήτηση σε υποδένδρο προστίθεται η τρέχουσα τιμή του currentNode στη στοίβα και αντί για BST_Node χρησιμοποιείται AVL_Node.

Έτσι, στο τέλος της αναζήτησης υπάρχει η δυνατότητα άμεσης πρόσβασης στους κόμβους του μονοπατιού που “διασχίσθηκε” κατά την διάρκεια της αναζήτησης και συνεπώς μπορούν να γίνουν εύκολα έλεγχοι και αλλαγές που εξασφαλίζουν την διατήρηση των ιδιοτήτων του δένδρου ως AVL.

Η μέθοδος deleteNode

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει μία bool τιμή με την κατάσταση της επιτυχίας της διαγραφής. Έχει ως αποτέλεσμα την ολοκληρωτική διαγραφή του κόμβου που αποθηκεύει την λέξη που δίνεται ως όρισμα από την δομή.

Αναλυτικά:

Αρχικά, ελέγχεται αν το δένδρο είναι άδειο (δεν υπάρχουν στοιχεία προς διαγραφή) και αν είναι επιστρέφεται κατάσταση επιτυχίας false και τερματίζεται η συνάρτηση.

Έπειτα, δεσμεύεται χώρος για στοίβα με δείκτες που συναντώνται στην αναζήτηση που δίνεται ως όρισμα στην internalSearchNode_WithStack με την οποία ελέγχεται αν το στοιχείο που μας δίνεται για διαγραφή υπάρχει στο δένδρο και γίνεται η αναζήτηση του και αποθηκεύεται η επιστρεφόμενη τιμή. Αν δεν υπάρχει, επιστρέφεται κατάσταση επιτυχίας false και τερματίζεται η συνάρτηση.

Τέλος, μειώνεται το πλήθος των κόμβων (αφού διαγράφω έναν) εξετάζεται το πλήθος παιδιών του κόμβου που βρέθηκε και καλείται η αντίστοιχη μέθοδος διαγραφής (από αυτές που περιγράφονται παρακάτω) για τον κόμβο που βρέθηκε από το internalSearchNode_WithStack. Μετά από κάθε κλήση διαγραφής, καλείται η μέθοδος updateTree_Delete που διορθώνει τα ύψη των κόμβων και κάνει περιστροφές (αν χρειαστεί).

Η μέθοδος deleteNode_NoChildren

Η μέθοδος αυτή, δέχεται ως όρισμα έναν δείκτη AVL_Node** , έναν ακόμη δείκτη AVL_Node** (στοίβα) με αναφορά και θέση της κορυφής της στοίβας (top) με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή ενός κόμβου θεωρώντας ότι αυτός δεν έχει παιδιά.

Λειτουργικά, είναι πανομοιότυπη με την αντίστοιχη μέθοδο του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους) και μειώνει την θέση της κορυφής κατά 1 και θέτει στην νέα κορυφή της στοίβας nullptr (διαγράφηκε ένας κόμβος οπότε μειώνεται το μέγεθος της στοίβας και ο προτελευταίος κόμβος της στοίβας δεν έχει πιά παιδί τον κόμβο που διαγράφηκε άρα βάζω στην νέα κορυφή της στοίβας nullptr).

Η μέθοδος deleteNode_OneChild

Η μέθοδος αυτή, δέχεται ως όρισμα έναν δείκτη AVL_Node**, έναν ακόμη δείκτη AVL_Node** (στοίβα) με αναφορά και θέση της κορυφής της στοίβας (top) με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή ενός κόμβου θεωρώντας ότι έχει μόνο ένα παιδί. Παράλληλα ενημερώνει την στοίβα.

Λειτουργικά, εφόσον ο κόμβος έχει μοναδικό παιδί αντικαθιστά τον εαυτό του (αντικαθίστανται όλα τα δεδομένα του) με το παιδί του και διαγράφεται το «παλιό» παιδί. Μετά από αυτό δίνει στην προτελευταία θέση της στοίβας την τιμή του αντίστοιχου δείκτη του παιδιού που αντικατέστησε τον γονέα. Τέλος, μειώνεται η τιμή του top κατά ένα εφόσον έχει αφαιρεθεί ένα στοιχείο.

Η μέθοδος deleteNode_TwoChildren

Η μέθοδος αυτή, δέχεται ως όρισμα έναν δείκτη AVL_Node**, έναν ακόμη δείκτη σε AVL_Node** (στοίβα) με αναφορά και θέση της κορυφής της στοίβας (top) με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή ενός κόμβου θεωρώντας ότι έχει δύο παιδιά. Παράλληλα ενημερώνει την στοίβα.

Αναλυτικά, η μέθοδος αυτή:

Αρχικά, ψάχνει το μεγαλύτερο παιδί του αριστερού υποδένδρου με μέθοδο όμοια με την internalSearchNode_WithStack με την διαφορά ότι εκτός από τις pointTo (η αντίστοιχη εδώ ονομάζεται pointToRightMost) και currentNode υπάρχει και μία μεταβλητή (AVL_Node*) buffer όπου το currentNode «βρίσκεται» πάντα «έναν κόμβο πιο μπροστά» και το buffer «κρατάει» την προηγούμενη τιμή του currentNode, ώστε όταν τελικά ο currentNode «φτάσει στο τέλος» και «γίνει» nullptr να έχουμε δείκτη στον τελευταίο κόμβο πριν το nullptr (που είναι αυτός που ψάχνουμε). Παράλληλα, σε κάθε εκτέλεση του βρόχου προσθέτω τις τιμές του currentNode στην στοίβα ενώ αυξάνω την τιμή του top κατά ένα.

Έπειτα, χρησιμοποιώντας τον δείκτη pointToRightMost, αντικαθίσταται (αλλάζουν οι τιμές) του στοιχείου προς διαγραφή με τον κόμβο που μπορώ να πάρω από το pointToRightMost.

Τέλος, ο κόμβος που παίρνω από το buffer πρέπει να διαγραφεί οπότε διαγράφεται με τον κατάλληλο τρόπο (ανάλογα αν έχει ή όχι παιδί)

Η μέθοδος updateTree_Delete

Η μέθοδος αυτή, δέχεται ως όρισμα έναν δείκτη AVL_Node** και την θέση της κορυφής της στοίβας και δεν επιστρέφει κάτι. Η λειτουργία του είναι να ενημερώνει τα ύψη των κόμβων που επηρεάζονται και να εκτελεί περιστροφές όταν διασπάται η ισορροπία του δένδρου.

Λειτουργικά, είναι όμοια με την updateTree_insert αλλά με κάποιες διαφορές:

Για κάθε στοιχείο του routeStack (δηλαδή για κάθε στοιχείο στο μονοπάτι από την ρίζα μέχρι το νέο στοιχείο) εκτελείται η επαναληπτική διαδικασία:

Αρχικά, ενημερώνονται οι τιμές του αριστερού και δεξιού ύψους του τρέχοντος στοιχείου χρησιμοποιώντας την μέθοδο getHeight() του AVL_Node.

Στη συνέχεια, ελέγχεται αν υπάρχει δεξιά ή αριστερή ισορροπία:

Αν υπάρχει δεξιά ανισορροπία, ελέγχονται οι πιθανές περιπτώσεις ανισορροπίας και αντιμετωπίζονται κατάλληλα (μία απλή αριστερή περιστροφή για περίπτωση “δεξιά-δεξιά” και πρώτα περιστροφή δεξιά και μετά αριστερά για περίπτωση “δεξιά-αριστερά”) ανάλογα με το αν είναι ρίζα ή όχι (αλλάζουν ελαφρώς τα ορίσματα αλλά η διαδικασία είναι ίδια).

Αν υπάρχει αριστερή ανισορροπία, γίνονται ανάλογες διαδικασίες: μία απλή δεξιά περιστροφή για περίπτωση “αριστερά-αριστερά” και πρώτα περιστροφή αριστερά και μετά δεξιά για περίπτωση “αριστερά-δεξιά”.

Τέλος, αξίζει να σημειωθεί ότι σε αντίθεση με την updateTree_insert, εδώ δεν διακόπτεται η εκτέλεση σε κάποιο σημείο παρά μόνο μετά το τέλος του βρόχου (δηλαδή μετά τον έλεγχο κάθε στοιχείου της στοίβας).

Η μέθοδος leftRotation

Η μέθοδος αυτή δέχεται ως ορίσματα έναν δείκτη AVL_Node *targetNode και έναν δείκτη AVL_Node** targetPtrPtr και δεν επιστρέφει ορίσματα. Έχει ως λειτουργία να προκαλεί μία αριστερή περιστροφή στον κόμβο targetNode.

Αναλυτικά, χρησιμοποιεί τον κόμβο που δείχνει το targetNode ως κόμβο στον οποίο θα γίνει η περιστροφή και τον δείκτη targetPtrPtr ως τον δείκτη σε δείκτη σε παιδί (δείκτη σε leftChild/rightChild) του γονέα του targetNode.

Αρχικά, θέτω το *targetPtrPtr (το παιδί του γονέα του targetNode) στο δεξί παιδί του targetNode.

Στη συνέχεια, αποθηκεύω το δεξί παιδί του targetNode σε προσωρινό δείκτη (temp).

Μετά, θέτω το δεξί παιδί του targetNode στο αριστερό παιδί του δεξιού παιδιού του targetNode και ενημερώνω το ύψος.

Τέλος, θέτω ως αριστερό παιδί του temp στο targetNode και ενημερώνω το ύψος.

Η μέθοδος rightRotation

Η μέθοδος αυτή δέχεται ως ορίσματα έναν δείκτη AVL_Node *targetNode και έναν δείκτη AVL_Node** targetPtrPtr και δεν επιστρέφει ορίσματα. Έχει ως λειτουργία να προκαλεί μία δεξιά περιστροφή στον κόμβο targetNode.

Αναλυτικά, χρησιμοποιεί τον κόμβο που δείχνει το targetNode ως κόμβο στον οποίο θα γίνει η περιστροφή και τον δείκτη targetPtrPtr ως τον δείκτη σε δείκτη σε παιδί παιδί (δείκτη σε leftChild/rightChild) του γονέα του targetNode.

Αρχικά, θέτω το *targetPtrPtr (το παιδί του γονέα του targetNode) στο αριστερό παιδί του targetNode.

Στη συνέχεια, αποθηκεύω το αριστερό παιδί του targetNode σε προσωρινό δείκτη.

Μετά, θέτω το αριστερό παιδί του targetNode στο δεξί παιδί του αριστερού παιδιού του targetNode και ενημερώνω το ύψος.

Τέλος, θέτω ως δεξί παιδί του temp στο targetNode και ενημερώνω το ύψος.

Η μέθοδος searchNode

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή std::string με αναφορά (const) και επιστρέφει ένα const AVL_Node* που δείχνει στον κόμβο που βρέθηκε ή σε nullptr αν δεν βρέθηκε.

Λειτουργικά, είναι πανομοιότυπη με την αντίστοιχη μέθοδο του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους).

Οι μέθοδοι inorder, preorder, postorder

Αυτές οι μέθοδοι έχουν ως όρισμα μία τιμή std::ofstream με αναφορά και δεν επιστρέφουν κάτι. Έχουν ως αποτέλεσμα να τυπώνεται στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του με τρόπο ενδοδιατεταγμένης (ή προδιατεταγμένης ή μεταδιατεταγμένης αντίστοιχα) διαπέρασης.

Λειτουργικά, είναι πανομοιότυπες με τις αντίστοιχες μεθόδους του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους).

Η μέθοδος internalPrintInorder

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή AVL_Node* και μία τιμή std::ofstream με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα να τυπώνει στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα με τρόπο ενδοδιατεταγμένης διαπέρασης.

Λειτουργικά, είναι πανομοιότυπη με την αντίστοιχη μέθοδο του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους).

Η μέθοδος internalPrintPreorder

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή AVL_Node* και μία τιμή std::ofstream με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα να τυπώνει στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα με τρόπο προδιατεταγμένης διαπέρασης.

Λειτουργικά, είναι πανομοιότυπη με την αντίστοιχη μέθοδο του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους).

Η μέθοδος internalPrintPostorder

Η μέθοδος αυτή, δέχεται ως όρισμα μία τιμή AVL_Node* και μία τιμή std::ofstream με αναφορά και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα να τυπώνει στο ρεύμα εξόδου που δίνεται ως όρισμα τα δεδομένα του κάθε κόμβου του δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα με τρόπο μεταδιατεταγμένης διαπέρασης.

Λειτουργικά, είναι πανομοιότυπη με την αντίστοιχη μέθοδο του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους).

Ο κατασκευαστής της AVL_Tree

Απλά αρχικοποιούνται τα μέλη root σε nullptr (αρχικά έχω κενό δένδρο) και numOfNodes και height σε 0 (δεν έχω κόμβους)

Ο καταστροφέας του AVL_Tree

Απλά καλείται η μέθοδος deleteTree με όρισμα την ρίζα. Η ύπαρξη του οφείλεται όπως και με τις μεθόδους για διαπεράσεις στην ενθυλάκωση.

Η μέθοδος deleteTree

Αυτή η μέθοδος έχει όρισμα μία τιμή AVL_Node* και δεν επιστρέφει κάτι. Έχει ως αποτέλεσμα την διαγραφή από την μνήμη του (υπό)δένδρου με ρίζα τον κόμβο του οποίου ο δείκτης δίνεται ως όρισμα.

Λειτουργικά, είναι πανομοιότυπη με την αντίστοιχη μέθοδο του BinarySearchTree με την διαφορά ότι χρησιμοποιείται AVL_Node αντί για BST_Node (βλέπε στο τμήμα αναλυτικής παρουσίασης του πρώτου μέρους).