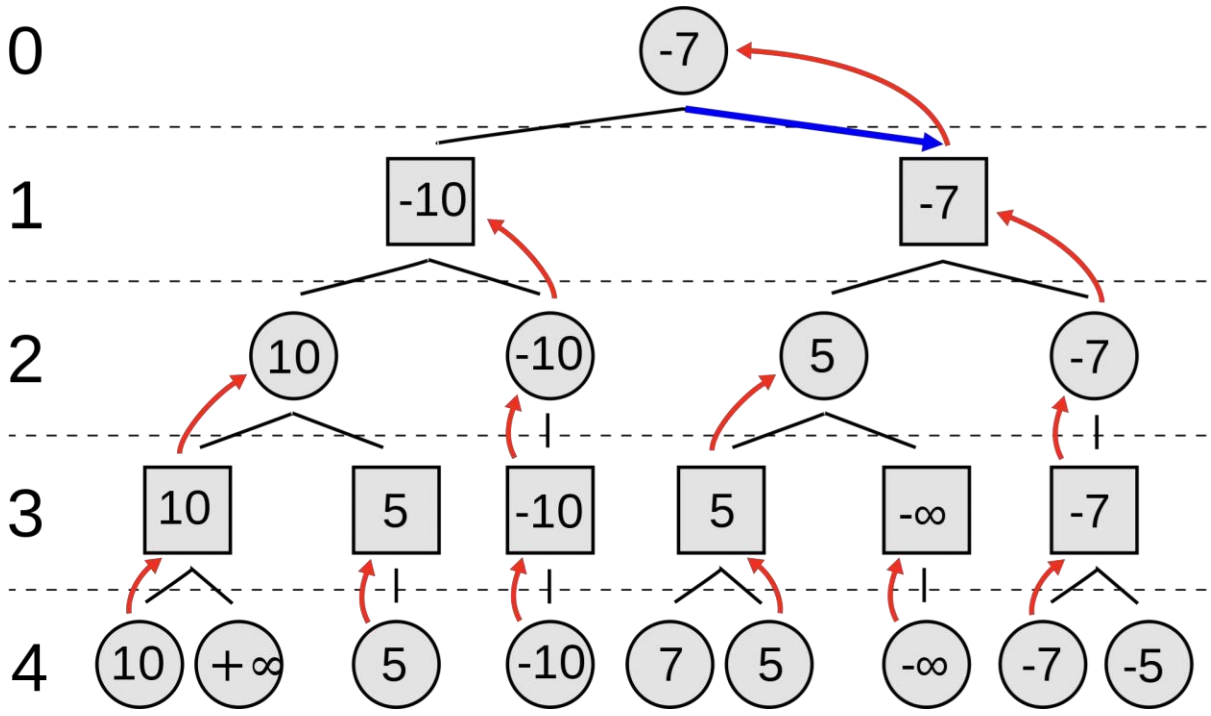
Δομές Δεδομένων

Εργασία 2019-2020

Σπυρίδων Κουνουπίδης ΑΕΜ: 3679

Παπαδημητρίου Θεόδωρος ΑΕΜ: 3851



Γενικά - Main

Υλοποίηση:

Στο πρώτο κομμάτι της εργασίας, ζητήθηκε να υλοποιηθούν:

1. Απλό Δυαδικό Δέντρο Αναζήτησης
2. Δυαδικό Δέντρο Αναζήτησης τύπου AVL
3. Πίνακας Κατακερματισμού με ανοικτή διεύθυνση.

Για την υλοποίηση την παραπάνω δομών χρειαζόμαστε μια main η οποία θα καλεί τις δομές και θα τις τροφοδοτεί με τα σωστά δεδομένα. Ξεκινώντας ορίζουμε πριν από την main την clear (line 19-25) η οποία καθαρίζει τις λέξεις από ειδικούς χαρακτήρες έτσι ώστε να μην εμποδίζουν την ταξινόμηση των λέξεων και την resize (27-43) η οποία αυξάνει την χωρητικότητα του πίνακα που δέχεται ως όρισμα κατά 2500 θέσεις .

Περνώντας στο σώμα της main:

1. (50-67) Δηλώνουμε όλες τις μεταβλητές που θα χρειαστούμε και αρχικοποιούμε τους πίνακες για τις πρωτότυπες λέξεις και τις λέξεις για το hash table με 5000 θέσεις.
2. (73-75) Δημιουργούμε δείκτες για τις τρεις δομές και διαβάζουμε το αρχείο εισόδου το οποίο θα έχει όνομα **file.txt.**
3. (89-123) Εισάγουμε όλες τις λέξεις στα Binary Search Tree (BST) και στο Avl, αφού πρώτα τις έχουμε καθαρίσει από σημεία στίξης με την clear() ,φροντίζοντας παράλληλα να περνάμε τις λέξεις που δεν έχουν εισαχθεί ξανά στον πίνακα words με τις πρωτότυπες λέξεις και στον hashwords με τις λέξεις για τον πίνακα κατακερματισμού φροντίζοντας παράλληλα να κάνουμε resize το μέγεθός τους με την resize().
4. (129-135) Δημιουργούμε το hashtable με μέγεθος αυτό των πρωτότυπων λέξεων και κάνουμε εισαγωγή των λέξεων από τον πίνακα hashwords στο hashtable.
5. (140-153) Για την αναζήτηση στις δομές δημιουργούμε έναν πίνακα τυχαίων λέξεων searchwords με μέγεθος 1/3 των πρωτότυπων λέξεων.
6. (156-223) Έπειτα ανοίγουμε ένα αρχείο results.txt και εκτελούμε αναζήτηση σε κάθε δομή με ερωτήματα από των πίνακα τυχαίων λέξεων και παίρνουμε τον συνολικό χρονο που έκανε η κάθε δομή να απαντήσει στα ερωτήματα τυπώνοντας τον στο results.txt. Τέλος στο results.txt τυπώνουμε και όλες τις λέξεις που κάνουμε αναζήτηση στις δομές και τις φορές που αυτές εμφανίζονται στο κείμενο.

Παρατηρήσεις - Σχόλια:

Για να δημιουργήσουμε τον πίνακα κατακερματισμού με ακρίβεια στο μέγεθος εκμεταλλευτήκαμε την εισαγωγή στις άλλες δομές για να πάρουμε ακριβώς τις εμφανίσεις των πρωτότυπων λέξεων και να ορίσουμε το μέγεθος του hashtable. Θα μπορούσαμε να αρχίσουμε με τυχαίο μέγεθος και να κάνουμε resize αλλά εφόσον η εργασία είναι συνδυαστική εκμεταλλευόμαστε τον χρόνο τις εισαγωγής στις άλλες δομές μιας και αυτός δεν χρονομετρείται

Binary Search Tree (Tree)

Υλοποίηση:

Η Tree περιέχει τις μεταβλητές του αντικειμένου που δημιουργείται όταν εισάγουμε έναν νέο κόμβο (string value για την λέξη που εισάγεται , int duplicates μετρητής για πολλαπλές εμφανίσεις της ίδιας λέξης και δύο δείκτες (Tree* Left , Tree* Right)για το αριστερό και το δεξί παιδί του κόμβου αντίστοιχα.) και τις συναρτήσεις της δομής που είναι οι ακόλουθες.

Insert: Η Insert για αρχή δέχεται σαν όρισμα τον κόμβο της ρίζας και την λέξη που θελουμε να εισάγουμε και επιστρέφει τον κόμβο. Αν ο κόμβος που ελέγχεται είναι κενός (δείκτης κενός) τότε δημιουργείται ένα νέο αντικείμενο κόμβου, αν βρεθεί ίδια λέξη τότε αυξάνουμε το duplicates και σε κάθε άλλη περίπτωση κάνουμε Insert με λεξικογραφικά κριτήρια στο δεξί ή στο αριστερό παιδί του κόμβου αναδρομικά εώς ότου βρούμε κενή θέση ή παρόμοια λέξη στο δένδρο.

Search: Η Search δεχεται σαν όρισμα τον κόμβο της ρίζας και την λέξη που θέλουμε να αναζητήσουμε και επιστρέφει τον συγκεκριμένο κόμβο άμα τον βρεί. Η Search επιστρέφει τον δείκτη του κόμβου που αναζητήσαμε ή null ώστε να μπορούμε να εκμεταλλευτούμε τις μεθόδους του κόμβου, πχ στην περίπτωση μας την GetDuplicates() ώστε να εκτυπώσουμε και τις εμφανίσεις της λέξης. Αν η Search φτάσει σε κόμβο που είναι κενός τότε επιστρέφει null. Σε κάθε άλλη περίπτωση κάνουμε Search με λεξικογραφικά κριτήρια στο δεξί ή στο αριστερό παιδί του κόμβου αναδρομικά εώς ότου βρούμε την λέξη που αναζητούμε ή βρεθούμε σύμφωνα με τα λεξικογραφικά κριτήρια σε κενό κόμβο (δείκτη) επομένως δεν υπάρχει η λέξη.

Delete: Η Delete για αρχή δέχεται σαν όρισμα τον κόμβο της ρίζας και την λέξη που θελουμε να διαγράψουμε και επιστρέφει τον κόμβο. Το πρώτο της κομμάτι αφορά την αναζήτηση του κόμβου για διαγραφή αναδρομικά, στη συνέχεια αν βρεί τον κόμβο εξετάζει αν υπάρχουν 0,1 ή 2 παιδιά. Στην περίπτωση ενός παιδιού αποθηκεύει το δεξι ή το αριστερό παιδί, διαγράφει τον κόμβο που είναι η ρίζα και έπειτα επιστρέφει τον δείκτη

που αποθηκεύσαμε (αναδρομικά) στο παιδί του προγόνου του διαγραμμένου κόμβου, πρακτικά ο δείκτης παίρνει την θέση του διαγραμμένου κόμβου. Στην περίπτωση που έχει 2 παιδια καλείται η Inorder_successor η οποία επιστρέφει τον inorder απόγονο του κόμβου που είναι το most left του δεξιού υποδέντρου του κόμβου που θα διαγραφεί και αυτός αποθηκεύεται σε έναν δείκτη temp. Στην συνέχεια αντιγράφει όλα τα στοιχεία του inorder successor στον κόμβο που θα διαγράψουμε οπότε πλέον έχουμε δύο ίδιους κόμβους, και για αυτό τον λόγο ξεκινάμε να διαγράψουμε απο το δεξί παιδί τον κόμβο με value αυτή του temp.

Τέλος υλοποιούνται οι προσπελάσεις postorder ,inorder και preorder.

AVL(Avl)

Υλοποίηση:

Η Avl έχει παρόμοια δομή με την Tree, περιέχει και αυτή τις μεταβλητές του αντικειμένου που δημιουργείται όταν εισάγουμε έναν νέο κόμβο (string value για την λέξη που εισάγεται , int duplicates που είναι μετρητής για πολλαπλές εμφανίσεις της ίδιας λέξης και δύο δείκτες (Tree* Left , Tree* Right) για το αριστερό και το δεξί παιδί του κόμβου αντίστοιχα και καθώς επιπλέον και την int height που τηρεί το ύψος του κόμβου ώστε να εκτελούμε τις περιστροφές.) και τις συναρτήσεις της δομής που είναι οι ακόλουθες:

Βοηθητικές συναρτήσεις:

Bal_calc: επιστρέφει την διαφορά των υψών των παιδιών του κόμβου που περνάει σαν όρισμα.

get_height: επιστρέφει το ύψος του κόμβου που περνάει σαν όρισμα.

Max: οι οποία επιστρέφει το μεγαλύτερο από δύο στοιχεία αντίστοιχα που δέχεται σαν όρισμα.

R_Rotation: δέχεται σαν όρισμα τον κόμβο που θα περιστραφεί. Σε έναν δείκτη top τοποθετούμε το αριστερό παιδί το οποίο θα γίνει η καινούργια ρίζα του υποδέντρου που περιστρέφουμε, σε έναν δείκτη temp τοποθετούμε το δεξί παιδί του κόμβου που θα γίνει ρίζα. Έπειτα εκτελούμε την περιστροφή θέτοντας δεξιά της καινούργιας ρίζας του υποδέντρου τον κόμβο που πήραμε σαν όρισμα (παλιά ρίζα) και τον δείκτη temp αριστερά του κόμβου-όρισμα. Τέλος ανανεώνουμε τα ύψη των κόμβων που περιστράφηκαν βρίσκοντας το μέγιστο από τα ύψη των παιδιών τους και αυξάνοντας κατά ένα.

L_Rotation: παρόμοια με την R_Rotation από την δεξιά μεριά.

Insert_avl: Λειτουργεί παρόμοια με την Insert του απλού δυαδικού, για αρχή δέχεται σαν όρισμα τον κόμβο της ρίζας και την λέξη που θέλουμε να εισάγουμε και επιστρέφει τον κόμβο. Αν η εισαγωγή φτάσει σε κενό κόμβο (δείκτης κενός) τότε δημιουργείται ένα νέο αντικείμενο κόμβου, αν βρεθεί ίδια λέξη τότε αυξάνουμε το duplicates και σε κάθε άλλη περίπτωση κάνουμε Insert με λεξικογραφικά κριτήρια στο δεξί ή στο αριστερό παιδί του κόμβου αναδρομικά έως ότου βρούμε κενή θέση ή παρόμοια λέξη στο δένδρο.

Σε δεύτερο χρόνο με αναδρομή αυξάνει το ύψος των προγόνων κατά ένα και καλεί την balance_calc η οποία ελέγχει αν η διαφορά των υψών των 2 παιδιών του κόμβου έχει γίνει μεγαλύτερη από 1 και επιστρέφει την διαφορά των υψών των δύο παιδιών σε στην int balance.

Εάν η διαφορά είναι μεγαλύτερη από 1 τότε εξετάζει τις περιπτώσεις:

α. Αν η μεταβλητή balance έχει πάρει την τιμή μεγαλύτερη από 1 (get_height(node->Left)> get_height(node->Right)) και η λέξη που εισήχθη είναι μικρότερη από την τιμή του αριστερού κόμβου (άρα είναι αριστερά του αριστερού παιδιού) τότε έχουμε **Left-Left case** οπότε εκτελείται μια δεξιά περιστροφή.

β. Αν μεταβλητή balance έχει πάρει την τιμή μικρότερη από – 1 (get_height(node->Left)< get_height(node->Right)) και η λέξη που εισήχθη είναι μεγαλύτερη από την τιμή του δεξιού κόμβου (άρα είναι δεξιά του δεξιού παιδιού) τότε έχουμε **Right-Right case** οπότε εκτελείται μια αριστερή περιστροφή.

γ. Αν η μεταβλητή balance έχει πάρει την τιμή μεγαλύτερη από 1 (get_height(node->Left)> get_height(node->Right)) και η λέξη που εισήχθη είναι μεγαλύτερη από την τιμή του αριστερού κόμβου (άρα είναι δεξιά του αριστερού παιδιού) τότε έχουμε **Left-Right case** οπότε εκτελείται μια αριστερή περιστροφή(για να πάμε σε Left-Left case) στο αριστερό παιδί και στην συνέχεια μια δεξιά περιστροφή στον κόμβο.

δ. Αν η μεταβλητή balance έχει πάρει την τιμή μικρότερη από -1 (get_height(node->Left)< get_height(node->Right)) και η λέξη που εισήχθη είναι μικρότερη από την τιμή του δεξιού κόμβου (άρα είναι αριστερά του δεξιού παιδιού) τότε έχουμε **Right-Left case** οπότε εκτελείται μια δεξιά περιστροφή(για να πάμε σε Right-Right case) στο δεξί παιδί και στην συνέχεια μια αριστερή περιστροφή στον κόμβο.

Search_avl: Υλοποιήθηκε ακριβώς ίδια με την αντίστοιχη της BST αφού η λειτουργία της είναι ίδια.

Delete_avl: Η Delete για αρχή δέχεται σαν όρισμα τον κόμβο της ρίζας και την λέξη που θέλουμε να διαγράψουμε και επιστρέφει τον κόμβο. Το πρώτο της κομμάτι αφορά την αναζήτηση του κόμβου για διαγραφή αναδρομικά, στη συνέχεια αν βρεί τον κόμβο

εξετάζει αν υπάρχουν 0,1 ή 2 παιδιά. Στην περίπτωση των 2 παιδιών αποθηκεύει σε έναν δείκτη temp τον δείκτη του most left του δεξιού παιδιού και η τιμή του temp αντιγράφεται στο τιμή του κόμβου που θα διαγραφεί. Τέλος διαγράφεται το most left του δεξιού παιδιού. Στην περίπτωση ενός ή κανενός παιδιού τοποθετείται η τιμή του παιδιού που έχει τιμη σε έναν δείκτη temp (αν δεν υπάρχει παιδί η τιμή NULL) και αντιγράφεται η temp στον κόμβο που θα διαγραφεί.

Αφού εκτελεστεί η διαγραφή ανανεώνουμε τα ύψη και γίνεται έλεγχος για την ισορροπία των παιδιών του κόμβου στον οποίο βρισκόμαστε με την μεταβλητή balance που χρησιμοποιήθηκε και στην εισαγωγή. Στην περίπτωση της διαγραφής αφού διαπιστωθεί ότι η balance ότι έγινε -1<balance<1, το δεύτερο κριτήριο για να δούμε σε ποια περίπτωση βρισκόμαστε είναι:

(1) Να καλέσουμε την Bal_calc στο αριστερό παιδί στην περίπτωση που η balance έχει πάρει την τιμή μεγαλύτερη από 1 (get_height(node->Left)> get_height(node->Right)) και να εκτελέσουμε δεξιά περιστροφή αν το αποτέλεσμα της Bal_calc είναι >=0 (γιατί έχουμε Left-Left) ή αν το αποτέλεσμα της Bal_calc < 0 μία αριστερή περιστροφή στο αριστερό παιδί για να πάμε σε Left-Left και μετά δεξιά περιστροφή στον κόμβο.

(2) Να καλέσουμε την Bal_calc στο δεξί παιδί στην περίπτωση που η balance έχει πάρει την τιμή μικρότερη από -1 (get_height(node->Left)< get_height(node->Right)) και να εκτελέσουμε αριστερή περιστροφή αν το αποτέλεσμα της Bal_calc είναι <=0 (γιατί έχουμε Right-Right) ή αν το αποτέλεσμα της Bal_calc > 0 μία δεξιά περιστροφή στο δεξί παιδί για να πάμε σε Right-Right και μετά αριστερή περιστροφή στον κόμβο.

Open Address Hash Table (Hash Table)

Υλοποίηση:

Για την υλοποιήση του πίνακα κατακερματισμού με ανοιχτή διεύθυνση δημιουργήσαμε δύο κλάσεις, μια για αντικείμενα κελιού (Cell, που περιέχει ένα string value και μια int duplicates) και μία για την δομή του πίνακα κατακερματισμού. Συγκεκριμένα η κλάση Hash Table έχει ως ιδιότητες έναν διπλό δείκτη σε Cell με όνομα HashMap, μια int wordcount για το σύνολο των λέξεων που έχουμε μέσα στον πίνακα και

μία int size για το μέγεθος του πίνακα. Ο διπλός δείκτης χρησιμοποιήθηκε ώστε να κάνουμε πίνακα δεικτών σε Cell αντικείμενα και να εκμεταλλευτούμε το NULL που έχουν οι δείκτες.

Αρχικά έχουμε τον constructor του Hash Table ο οποίος δέχεται σαν όρισμα το πλήθος τον μοναδικών λέξεων που υπάρχουν στο αρχείο κειμένου μας. Έτσι δημιουργεί δυναμικά έναν πίνακα δεικτών με διπλάσιο μέγεθος απο το σύνολο των λέξεων. Στην συνέχεια θέτει όλους τους δείκτες σε NULL και ορίζει το μέγεθος του πίνακα σε 2*I (όπου I το σύνολο των λέξεων) και το wordcount σε μηδέν.

Έπειτα έχουμε τρείς συναρτήσεις οι οποίες υλοποιούν λειτουργείες του πίνακα κατακερματισμου, όπως:

Hash Function: Συγκεκριμένα η συνάρτηση κατακερματισμού που χρησιμοποιήσαμε δέχεται μια συμβολοσειρά και επιστρέφει το hash % size ώστε να περιορίσουμε τις τιμές στο έυρος [0, size). Στην αρχή αρχικοποιεί την τιμή hash = 5381 και έπειτα σε μία λούπα κάνει πράξεις με το κάθε γράμμα της λέξης που δέχθηκε ώστε να δημιουργήσει ξεχωριστά κλειδιά για κάθε λέξη που δέχεται. Η συγκεκριμένη συνάρτηση έχει πολλή καλή κατανομή και είναι γρήγορη, για αυτό και επιλέχθηκε.

Insert Value: Στην Insert Value έχουμε τις μεταβλητές k όπου θα χρησιμοποιηθούν για το hash της λέξης που θέλουμε να εισάγουμε, την flag η οποία είναι για όταν αποθηκευτεί η λέξη μας και η Start η οποία είναι το αρχικό κλειδί. Συγκεκριμένα έχουμε μια λούπα η οποία τερματίζει όταν γίνει flag == true ή όταν προσπελάσουμε όλο τον πίνακα και δεν βρούμε κενή θέση (Start == k) . Μέσα στην λούπα ελέγχουμε άμα το κελί που μας δείχνει το κλειδί k είναι κενό, όπου δημιουργούμε ένα νέο αντικείμενο Cell, αποθηκεύουμε την συμβολοσειρά και θέτουμε flag = true ώστε να λήξει η λούπα. Άμα δεν είναι κενό ελέγχουμε αν η λέξη που θέλουμε να εισάγουμε είναι ίδια με την λέξη που είναι στο κελί που δείχνουμε. Αν δεν είναι ίδιες κάνουμε k+=1 αλλά με % size ώστε όταν φτάσουμε στο τελευτάιο κελί το επόμενο να είναι το πρώτο (άν κ = 499 με size = 500 στην επόμενη λούπα θα πάει k = 0) . Τέλος αυξάνουμε το wordcount κατά ένα.

Search Value: Στην Search Value έχουμε πάλι τις μεταβλητές k όπου θα χρησιμοποιηθούν για το hash της λέξης που θέλουμε να αναζητήσουμε, την flag η οποία είναι για όταν βρεθεί η λέξη μας και η Start η οποία είναι το αρχικό κλειδί. Συγκεκριμένα έχουμε μια λούπα η οποία τερματίζει όταν γίνει flag == true ή όταν προσπελάσουμε όλο τον πίνακα και δεν βρούμε την λέξη (Start == k) . Μέσα στην λούπα ελέγχουμε άν το κλειδί που μας έδωσε η συνάρτηση κατακερματισμού δείχνει σε κενό κελί, αν ναι πηγαίνουμε στο επόμενο με τον ίδιο τρόπο που κάναμε στην Insert Value, αν όχι τότε ελέγχουμε άν

μέσα στο κελί βρίσκεται η λέξη που αναζητούμε, αν ναι τότε κάνουμε flag = true , αν όχι τότε πάμε στο επόμενο κελί.

Βοηθητικά έχουν δημιουργηθεί και οι παρακάτω:

IsEmpty: Η συγκεκριμένη δέχεται ένα κλειδί και ελέγχει άν το κελί του κλειδιού είναι κενό ή όχι.

PrintHash: Τυπώνει στην οθόνη τις πληροφορίες όλων των *μη κενών* κελιών.

Παρατηρήσεις - Σχόλια:

Για να μπορέσουμε να παρατηρήσουμε οπτικά ότι το AVL έχει όλα τα φύλλα με διαφορά επιπέδου =<1 δημιουργήσαμε την print2d η οποία εκτυπώνει τα BST και AVL. Η εκτύπωση γίνεται αναδρομικά από τα δεξιά προς τα αριστερά το οποίο στην κονσόλα μετατρέπεται σε right most->αρχή της εκτύπωσης , κέντρο->ρίζα και left most -> τέλος της εκτύπωσης . Το δέντρο δηλαδή έχει περιστρέφει αριστερά 90 μοίρες με την ρίζα να βρίσκεται στο πιο αριστερό μέρος και τα φύλλα στο δεξιό. Για λίγες εισαγωγές για να μην γεμίζει η κονσόλα φαίνεται εμφανώς η διαφορά ανάμεσα σε BST και AVL. Η η κλίση της print2d(root) για το BST και print2d(root1) για το AVL αφήνεται στην διάθεσή σας για κλίση αν επιθυμείτε.

Ευχαριστούμε για τον χρόνο σας.