Αναφορά Εργασίας Δομών Δεδομένων 2020

Υλοποιήθηκε από τους φοιτητές του τμήματος Πληροφορικής Α.Π.Θ. :
Μαζγαλτζίδης Κωνσταντίνος, Τσιροζίδης Ιωάννης.

Σημεία Ιδιαίτερου Ενδιαφέροντος

Μία λέξη θεωρήθηκε ότι αποτελείται από 1 ή περισσότερα συνεχόμενα λατινικά πεζά γράμματα. Κατά τη διαγραφή ενός κόμβου αφαιρείται ολόκληρος μαζί με το περιεχόμενο του.
Για τη διευκόλυνση του περιβάλλοντος χρήστη, κάθε κλάση έχει δύο **(2) set** συναρτήσεων. Ένα set συναρτήσεων παρασκηνίου, στο private μέλος κάθε κλάσης, καθώς και ένα set στο public μέρος για απευθείας χρήση στη main(). Κάθε κλάση διαθέτει επίσης συνάρτηση **DebugInfo(),** η οποία παρουσιάζει πλήρη αναφορά των εκάστοτε στοιχείων.

❑ *κλάση* ***Binary Search Tree:***

//Η κεντρική ιδέα με μια ματιά...

Στην πρώτη δενδρική δομή της εργασίας αξιοποιήθηκε, για την κατασκευή κόμβων, *η ιδέα των pointer σε δομή Struct*. Η σύνδεση κόμβων, έγινε σύμφωνα με τα πρότυπα που ορίζει η δομή των Δυαδικών Δέντρων Αναζήτησης.

//τι υπάρχει στην κλάση...

Οι λειτουργίες **insert(), search(), delete(), inorder(), preorder()** και **postorder(),** καθώς και κάποιες βοηθητικές συναρτήσεις για τη διαγραφή κόμβου.

❑ *κλάση* ***Tree AVL:***

//Η κεντρική ιδέα με μια ματιά...

Για τη δημιουργία κόμβων **AVL,** επίσης αξιοποιήθηκε η ιδέα των pointer σε δομή Struct. Κατά την εισαγωγή και τη διαγραφή ενός κόμβου, γίνεται έλεγχος σε πραγματικό χρόνο στο δέντρο, για εντοπισμό τυχόν λάθους σε κάποιο balance factor. Σε περίπτωση εντοπισμού λάθους, ανιχνεύεται η περιοχή του μη ισορροπημένου κόμβου και η επιμέρους περίπτωση και πραγματοποιείται ένα από τα τέσσερα (4) rotations.

//τι υπάρχει στην κλάση...

Οι λειτουργίες **insert(), search(), delete(), inorder(), preorder()** και **postorder(),** τα τέσσερα είδη περιστροφών **(*rr* rotation(), *ll*_rotation(), *lr* rotation(), *rl* rotation())**, όπως και κάποιες επιπλέον βοηθητικές συναρτήσεις αναπροσαρμογής ύψους και επιδιόρθωσης balance factor κόμβου.

❑ *κλάση* ***Hash Table****:*

//Η κεντρική ιδέα με μια ματιά...

Για την υλοποίηση του πίνακα κατακερματισμού, χρησιμοποιήθηκε πίνακας δεικτών σε struct. Για τη ***συνάρτηση κατακερματισμού*** σχεδιάσαμε τον εξής αλγόριθμο: για είσοδο από τον χρήστη ενός string, θα παράγεται ακέραιος αριθμός, με διαδοχικά ψηφία του τους κωδικούς ASCII των αντίστοιχων γραμμάτων του string και ως τελευταίο ψηφίο το μήκος του string.

π.χ. string s= “dog” ->hash code= 41573

Ωστόσο το μέγεθος του ακέραιου αριθμού που θα παραγόταν θα ήταν εμφανώς μεγάλο και δύσχρηστο οπότε επιστρέφεται το hash code ***mod*** 4.294.967.296 (max int value). Έτσι ο αριθμός περιορίζεται από 0 έως 10 ψηφία.

...

//Η πιθανότητα collision που παρουσιάζει ο αλγόριθμος είναι (0.12... %) (467 λέξεις συνέπιψαν σε ένα δείγμα των 370.000 λέξεων)

...

Στην περίπτωση που υπάρξει collision, εφαρμόσαμε *quadratic probing*.

//τι υπάρχει στην κλάση...

Οι λειτουργίες **insert(), search()** όπως και η συνάρτηση κατακερματισμού **hash algo().**

Τελικές σκέψεις...

Η δενδρική δομή του AVL, ήταν ίσως αυτή που δημιούργησε στην ομάδα το μεγαλύτερο ενδιαφέρον και πρόκληση.. Παρότι οι πρωτεύουσες λειτουργίες ήταν κατά κόρον παρόμοιες με του Δυαδικού Δέντρου Αναζήτησης, η ιδιομορφία που παρουσιάζει το Ισοζυγισμένο Δέντρο AVL, δημιούργησε την ανάγκη χρήσης περιστροφών, στις συναρτήσεις των οποίων χρειάστηκε το μεγαλύτερο debugging.

Ιδιαίτερα ευχάριστο για την ομάδα, ήταν επίσης το brainstorming, στα πλαίσια δημιουργίας ενός αποδοτικού αλγορίθμου κατακερματισμού με όσο το δυνατόν μικρότερο ποσοστό σύγκρουσης.

Ευχαριστούμε για την προσοχή σας,
με εκτίμηση,

Ιωάννης, Κωνσταντίνος