ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΔΕΥΤΕΡΗ ΕΡΓΑΣΙΑ (2019-20)

Στην εργασία αυτή έγινε η υλοποίηση μιας ουράς προτεραιότητας με τη βοήθεια ενός AVL δυαδικού δέντρου αναζήτησης .Οι συναρτήσεις και κλάσεις που χρησιμοποιήθηκαν καθώς και οι τεχνικές είναι οι εξής:

Κλάσεις:

class Node

Η κλάση αυτή έχει ως αντικείμενο κάθε κόμβο του δέντρου . Οι κόμβοι αυτοί διαθέτουν την ακέραια μεταβλητή **data** για τα δεδομένα τους, και τους δείκτες **left**, **right** που "δείχνουν" το αριστερό και αντίστοιχα το δεξί παιδί του κόμβου, αν υπάρχει. Τέλος, διαθέτει έναν κατασκευαστή που αρχικοποιεί κάθε νέο αντικείμενο της κλάσης.

Συναρτήσεις:

Int height(Node* node)

Η συνάρτηση αυτή υπολογίζει το ύψος ενός κόμβου του δέντρου. Αρχικά, η αρχικοποιείται η μεταβλητή **h** δίνοντας την τιμή μηδέν,στη συνέχεια,αν ο κόμβος δεν είναι NULL, υπολογίζει αναδρομικά το ύψος του αριστερού και του δεξιού υποδέντρου του κόμβου **node**, βρίσκει το μεγαλύτερο εξ αυτών προσθέτει ένα. Τέλος επιστρέφει αυτήν την τιμή, αν ο κόμβος δεν ήταν NULL ,αλλιώς επιστρέφει μηδέν.

Int BalanceFactor(Node* node)

Η συνάρτηση αυτή υπολογίζει το δείκτη ισορροπίας bf() του κόμβου **node** που δέχεται σαν όρισμα,αυτό γίνεται με τον υπολογισμό της διαφοράς του ύψους του αριστερού υποδέντρου του από το αριστερό δέντρο του.

Node* Create_Node(int data)

Η συνάρτηση αυτή δημιουργεί και επιστρέφει έναν **νέο κόμβο.**

Node* Insert_Node(Node* root, int data, Node* Max_Node)

Η συνάρτηση αυτή εισάγει έναν νέο κόμβο στο δέντρο. Αρχικά, αν ο το δέντρο είναι κενό απλά δημιουργείται ένας κόμβος-ρίζα, αν όχι τότε αν η τιμή των δεδομένων που πρόκειται να έχει ο νέος κόμβος είναι **μικρότερα ίσα** αυτού ο κόμβος μετακινείται στα **αριστερά** ,αλλιώς αν είναι **μεγαλύτερα ίσα** πάνε στα **δεξιά** του, η διαδικασία επαναλαμβάνεται αναδρομικά μέχρις ότου να καταλήξει ο νέος κόμβος να είναι φύλλο ενός άλλου κόμβου. Όταν η διαδικασία τελειώσει το δέντρο ισορροπείται με τη βοήθεια της συνάρτησης **AVL_Balance()**. Τέλος με τη χρήση ενός if καθορίζεται ο κόμβος με τη μέγιστη τιμή στο δέντρο και επιστρέφεται ο δείκτης-ρίζα.

Node* AVL_Balance(Node* root)

Η συνάρτηση αυτή ισορροπεί το δέντρο, δηλαδή δεν υπάρχει κόμβος με δείκτη ισορροπίας bf>| 1 |. Αν **ο δείκτης ισορροπίας** του κόμβου είναι **μεγαλύτερος του 1 και μεγαλύτερος του 0** έχουμε **left left περιστροφή ,** αν είναι **μεγαλύτερος του 1 και μικρότερος του 0** έχουμε **left right περιστροφή**. Από την άλλη πλευρά αν ο δείκτης ισορροπίας είναι **μικρότερος του -1 και μικρότερος του 0** έχουμε **right right** περιστροφή , αν είναι **μικρότερος του -1 και μεγαλύτερος**

του 0 έχουμε **right left** περιστροφή . Σε όλες τις περιπτώσεις περιστροφής καλείται η ανάλογη συνάρτηση. Τέλος επιστρέφεται ο δείκτης του κόμβου που επεξεργάστηκε.

Node* Right_Right(Node* parent)

Η συνάρτηση αυτή πραγματοποιεί **right right** περιστροφή του κόμβου που δέχεται σαν όρισμα. Αρχικά δημιουργείται μια μεταβλητή τύπου Node* η **temp** , στη συνέχεια αυτή "δείχνει" το **δεξί παιδί** του κόμβου-ορίσματος. Έπειτα το **δεξί παιδί** του κόμβου-ορίσματος "δείχνει" στο **αριστερό παιδί** του δείκτη **temp**, και στη συνέχεια το **αριστερό παιδί** του δείκτη **temp** "δείχνει" τον **κόμβο-όρισμα.** Τέλος η συνάρτηση επιστρέφει τον δείκτη **temp**.

Node* Left_Left(Node* parent)

Η συνάρτηση αυτή πραγματοποιεί **left left** περιστροφή του κόμβου που δέχεται σαν όρισμα. Αρχικά δημιουργείται μια μεταβλητή τύπου Node* η **temp** , στη συνέχεια αυτή "δείχνει" το **αριστερό παιδί** του κόμβου-ορίσματος. Έπειτα το **αριστερό παιδί** του κόμβου-ορίσματος "δείχνει" στο **δεξί παιδί** του δείκτη **temp**, και στη συνέχεια το **δεξί παιδί** του δείκτη **temp** "δείχνει" τον **κόμβο-όρισμα.** Τέλος η συνάρτηση επιστρέφει τον δείκτη **temp**.

Node* Left_Right(Node* parent)

Η συνάρτηση αυτή πραγματοποιεί **left right** περιστροφή του κόμβου που δέχεται σαν όρισμα. Αρχικά δημιουργείται μια μεταβλητή τύπου Node* η **temp** , στη συνέχεια αυτή "δείχνει" το **αριστερό παιδί** του κόμβου-ορίσματος. Έπειτα το **αριστερό παιδί** του κόμβου-ορίσματος "δείχνει" στην τιμή που επιστρέφει η Right_Right(**temp**). Τέλος επιστρέφεται η τιμή που επιστρέφει η Left_Left(parent).

Node* Right_Right(Node* parent)

Η συνάρτηση αυτή πραγματοποιεί **right left** περιστροφή του κόμβου που δέχεται σαν όρισμα. Αρχικά δημιουργείται μια μεταβλητή τύπου Node* η **temp** , στη συνέχεια αυτή "δείχνει" το **δεξί παιδί** του κόμβου-ορίσματος. Έπειτα το **δεξί παιδί** του κόμβου-ορίσματος "δείχνει" στην τιμή που επιστρέφει η Left_Left(**temp**). Τέλος επιστρέφεται η τιμή που επιστρέφει η Right_Right(parent).

Node* Delete_Node(Node* root,int key)

Η συνάρτηση αυτή βρίσκει το κόμβο με τιμή key και στην συνέχεια τον διαγράφει και επιστρέφει το δέντρο χωρίς αυτόν. Αρχικά ,γίνεται έλεγχος για το αν το δέντρο είναι κενό.

Node* Smallest_Node(Node* root)

Η συνάρτηση αυτή βρίσκει το κόμβο με την μικρότερη τιμή διασχίζοντας το δέντρο με προδιάταξη, αν βρεθεί μια τιμή μικρότερη από την **temp→data** γίνεται η νέα **temp→data .**

Node* Delete_Node(Node* root, int key)

Η συνάρτηση αυτή χρησιμοποιείται για την διαγραφή ενός κόμβου από το δέντρο. Αρχικά ,αν το δέντρο δεν είναι κενό, βρίσκει αν υπάρχει τον κόμβο που έχει την τιμή **key** που έχει σαν όρισμα η συνάρτηση. Όταν η τιμή αυτή βρεθεί ,αν ο κόμβος αυτός είναι φύλλο ή έχει μόνο ένα παιδί διαγράφεται και γίνεται η σύνδεση του δέντρου με το παιδί αν υπάρχει, αν έχει 2 παιδιά βρίσκεται ο μικρότερος απόγονος στο δεξιό υποδέντρο του κόμβου με τη συνάρτηση Smallest_Node αντικαθίσταται η τιμή του κόμβου που διαγράφεται με αυτήν και ο κόμβος που είχε αυτή την τιμή διαγράφεται. Τέλος επιστρέφεται το δέντρο χωρίς τον κόμβο που διαγράφθηκε.

Int main()

Στη συνάρτηση αυτή καλούνται οι παραπάνω μέθοδοι , δημιουργείται μια ουράς προτεραιότητας με τη βοήθεια δυαδικού δέντρου ,πραγματοποιούνται εισαγωγές ,διαγραφές και δίνεται και το μέγιστο στοιχείο του δέντρου σε χρόνο O(1).