ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΟΜΕΑΣ ΜΑΘΗΜΑΤΙΚΩΝ

Γραπτή άσκηση #2

Μάθημα: Δομές Δεδομένων

Διδάσκων: Αντώνιος Συμβώνης

Φοιτήτρια: Ελένη Στυλιανού, ge21708

Email: elenistylianou03@live.com

Άσκηση 1

Για τον υπολογισμό των k μικρότερων από n πραγματικούς αριθμούς σε χρόνο O(nlogk) αρχικά εισάγουμε την λίστα με τους n πραγματικούς αριθμούς και δημιουργούμε μία σωρό μεγίστου έστω H με τους πρώτους k αριθμούς που δίνονται στη λίστα. Ακολούθως, αρχίζοντας από το στοιχείο της λίστας που βρίσκεται στη θέση k+1 έως το τελευταίο στοιχείο της λίστα που βρίσκεται στη θέση n ελέγχουμε το κάθε στοιχείο ξεχωριστά αν είναι μικρότερο από το μέγιστο στοιχείο της H και αν είναι αφαιρούμε την ρίζα από την H, εφόσον η σωρός έχει την ιδιότητα η ρίζα να είναι η μεγαλύτερη μεταξύ των k μικρότερων αριθμών, και βάζουμε το στοιχείο στη θέση που βρισκόμαστε αν δεν είναι προχωράμε στον επόμενο αριθμό μέχρι να φτάσουμε στο τέλος της λίστας. Τέλος, επιστρέφουμε την σωρό μεγίστου H, η οποία θα περιέχει τους k μικρότερους αριθμούς από την λίστα που εισαγάγαμε στην αρχή.

Η χρονική πολυπλοκότητα του αλγορίθμου είναι O(nlogk) γιατί εφαρμόζουμε n εισαγωγές (insertions) στην σωρό μεγίστου H k στοιχείων, που η καθεμιά παίρνει χρόνο O(logk), αυτός ο αλγόριθμος είναι καλύτερος απ' ότι όταν ταξινομήσουμε τα στοιχεία σε χρόνο O(nlogn). Επίσης, η χρήση της σωρού μεγίστου μας επιτρέπει να έχουμε τα k μικρότερα στοιχεία χωρίς να αποθηκεύσουμε όλα τα n στοιχεία.

Άσκηση 2

Για την συγχώνευση k διατεταγμένων λιστών σε μία διατεταγμένη λίστα σε χρόνο O(nlogk), όπου n ο συνολικός αριθμός των στοιχείων αρχικά εισάγουμε τις k διατεταγμένες λίστες και δημιουργούμε μια άδεια σωρό ελαχίστου H και μία λίστα εξόδου L. Στη συνέχεια, για κάθε λίστα από τις k λίστες εισόδου (O(k) φορές) θα ελέγχουμε αν η λίστα δεν είναι άδεια και αν δεν είναι θα εισάγουμε το πρώτο στοιχείο της λίστας στην H (O(logk)). Επίσης, καθώς η H δεν θα είναι άδεια (O(n) φορές), θα αφαιρούμε το μικρότερο στοιχείο της H και θα το βάζουμε στην L (O(1)) και από την λίστα που άνηκε αρχικά αυτό το στοιχείο θα παίρνουμε το επόμενο στοιχείο και θα το βάζουμε στην H (O(logk)) και θα επαναλαμβάνουμε την διαδικασία μέχρι να αδειάσει η λίστα. Τέλος, θα επιστρέφεται η L με όλα τα στοιχεία από τις διατεταγμένες λίστες ταξινομημένα.

Η χρονική πολυπλοκότητα του αλγορίθμου είναι O(n logk) γιατί εκτελούμε k εισαγωγές σε ένα σωρό ελαχίστου, καθεμιά από τις οποίες παίρνει χρόνο O(logk) και εκτελούμε n αφαιρέσεις από ένα σωρό ελαχίστου και n εισαγωγές σε μία λίστα εξόδου, καθεμία από τις οποίες παίρνει σταθερό χρόνο. Επίσης, εκτελούμε το πολύ n εισαγωγές σε ένα σωρό ελαχίστου σε χρόνο O(logk) κάθε μία. Επομένως έχουμε O(k logk +n +n logk)=O(n logk)

Άσκηση 3

Πιο κάτω παρουσιάζεται σε μορφή ψευδοκώδικα η υλοποίηση που είναι βασισμένη αποκλειστικά σε δομές σωρού η οποία υποστηρίζει τον αφηρημένο τύπο δεδομένων MinMax_Priority_Queue με πολυπλοκότητα χειρότερης περίπτωσης O(logn) για κάθε λειτουργία της:

```
procedure INSERT(key, value)
    Insert the node (key, value) into both H_min and H_max
```

if key < root(H_{min}) then
Swap (key, value) with the root of H_{min}
if key > root(H_{max}) then
Swap (key, value) with the root of H_{max}
procedure FIND_MIN
return root(H_{min})
procedure REMOVE_MIN
Swap the root of H_{min} with the last node in the heap
Remove the last node
Restore the min heap property by swapping the new root with its smaller child until the
heap property is satisfied
if the removed node is also in H_{max} then
Remove the corresponding node from H_{max}
procedure FIND_MAX
return root(H_{max})
procedure REMOVE_MAX
Swap the root of Hmax with the last node in the heap
Remove the last node
Restore the max heap property by swapping the new root with its larger child until the heap property is satisfied
if the removed node is also in H_{min} then
Remove the corresponding node from H_{min}
function SIZE
return the number of nodes in both H_{min} and H_{max}
function isEMPTY
return True if both H_{min} and H_{max} are empty, False otherwise

Η χρονική πολυπλοκότητα για εισαγωγή νέου κόμβου είναι O(logn), αφού εκτελούμε μέχρι δύο swaps και κάθε swap «κοστίζει» O(logn) χρόνο. Επίσης, η εύρεση μεγίστου και ελαχίστου επιτυγχάνεται σε σταθερό O(1) χρόνο, εφόσον μπορούμε σε σταθερό χρόνο να διαβάσουμε το στοιχείο στην ρίζα του σωρού. Επιπρόσθετα, η χρονική πολυπλοκότητα για διαγραφή μεγίστου και ελαχίστου στοιχείου είναι O(logn), διότι εκτελούμε μέχρι δύο swaps και κάθε swap «κοστίζει» O(logn) χρόνο. Τέλος, έχουμε το μέγεθος και εάν είναι κενός ο σωρός σε σταθερό O(1) χρόνο, αφού τα μεταβάλλουμε κατά την εισαγωγή/διαγραφή στοιχείων.

Άσκηση 4

1)α) Πιο κάτω παρουσιάζεται ψευδοκώδικας για την υλοποίηση της μεθόδου any-fit:
Require: A set of n objects with weights $w_1,w_2,...,w_n$
Ensure: The number of containers required and the objects in each container
Procedure ANYFITAVL($w_1,w_2,...,w_n$)
C← {C_1←Ø} // Δημιουργία λίστας δοχείων με το πρώτο δοχείο

avl←Empty AVL tree //Δημιουργία άδειου ζυγισμένου δυαδικού δέντρου avl
for i=1 to n do
    node←avl.findNextSmaller(1-w_i) //Εύρεση του μικρότερου κόμβου στο δέντρο avl με κλειδί μικρότερο ή ίσο με το 1-w_i
    if node ≠ None then
        $C_{node.id}$←$C_{node.id}$ U i //Προσθήκη αντικειμένου i στο δοχείο με ID node.id
        avl.delete(node.key) //Διαγραφή του κόμβου από το δέντρο avl
        avl.insert(node.key - w_i, node.id) //Εισαγωγή καινούριου κόμβου με κλειδί ίσο με το v υπολειπόμενο χώρο και το ID ίσο με το node.id
    else
        $C_{|C|+1}$←i //Δημιουργία νέου δοχείου και προσθήκη αντικειμένου i σε αυτό
        avl.insert(1-w_i,|C|) //Προσθήκη καινούριου κόμβου με κλειδί ίσο με τον υπολειπόμενο χώρο και ID ίσο με το ID του καινούριου δοχείου
return |C| and C_1, C_2,..., $C_{|C|}$ //Επιστροφή του αριθμού των δοχείων και των αντικειμένων σε κάθε δοχείο

Η χρονική πολυπλοκότητα της ευρετικής μεθόδου any-fit με ζυγισμένα δυαδικά δέντρα αναζήτησης είναι O(n logn), όπου n το πλήθος των αντικειμένων για πακετάρισμα. Η επανάληψη του for loop εκτελείται για n αντικείμενα και για κάθε αντικείμενο, εκτελεί πράξεις στο ζυγισμένο δυαδικό δέντρο αναζήτησης. Η χρονική πολυπλοκότητα των εντολών findNextSmaller, delete και insert σε ένα τέτοιο δέντρο είναι O(logn) στην χειρότερη περίπτωση, όπου ο αριθμός των κόμβων του δέντρου.
β) Αυτή η μέθοδος δεν επιστρέφει τον βέλτιστο αριθμό δοχείων.
Αντιπαράδειγμα:
Έστω τα βάρη W = {0.5, 0.4, 0.5, 0.6}. Η μέθοδος any-fit επιστρέφει |C| = 3. Ενώ η βέλτιστη λύση είναι |C| = 2 και C_1 = {0.5, 0.5} , C_2 = {0.4, 0.6}.

2) Πιο κάτω παρουσιάζεται ψευδοκώδικας για την υλοποίηση της μεθόδου best-fit:
Require: A set of n objects with weights w_1,w_2,...,w_n
Ensure: The number of containers required and the objects in each container
Procedure ANYFITAVL(w_1,w_2,...,w_n)
T← empty AVL tree //Δημιουργία άδειου ζυγισμένου δυαδικού δέντρου avl
C← {C_1←Ø} // Δημιουργία λίστας δοχείων με το πρώτο δοχείο
for i=1 to n do
    bestNode←findBestNode(T.root, w_i) //Εύρεση του καλύτερου δοχείου για να τοποθετηθεί το αντικείμενο i
    if bestNode ≠ None then
        $C_{bestNode.index}$←$C_{bestNode.index}$ U i //Προσθήκη αντικειμένου i στο δοχείο bestNode.index
        updateNode(bestNode) //Ενημέρωση του κόμβου του δέντρου AVL για το νέο δοχείο
    else
        $C_{|C|+1}$←i //Δημιουργία νέου δοχείου και προσθήκη αντικειμένου i σε αυτό
        insertNode(T.root, |C|) //Προσθήκη καινούριου κόμβου για το νέο δοχείο

```
return |C| and C1, C2,..., C|C| //Επιστροφή του αριθμού των δοχείων και των
αντικειμένων σε κάθε δοχείο
function findBestNode(node, wi)
    if node = None then
        return None
    if wi + remainingSpace(node.container) ≤ 1 then
        if node.left = None then
                return node
        else
                bestLeft← findBestNode(node.left, wi)
                if bestLeft = None or
                remainingSpace(bestLeft.container) > remainingSpace(node.container)
                then
                        return node
                else
                        return bestLeft
    else
        return findBestNode(node.right, wi)
```

Με χρήση του ίδιου αντιπαραδείγματος όπως και στο 1β ούτε αυτή η μέθοδος δεν επιστρέφει τη βέλτιστη λύση.
Η χρονική πολυπλοκότητα της ευρετικής μεθόδου best-fit με ζυγισμένα δυαδικά δέντρα αναζήτησης είναι επίσης O(n logn), όπου n το πλήθος των αντικειμένων για πακετάρισμα. Εξαιτίας του γεγονότος πως κάθε αντικείμενο πρέπει να εισαχθεί στο ζυγισμένο δυαδικό δέντρο αναζήτησης, πράξη που παίρνει O(logn) στην χειρότερη περίπτωση. Στη συνέχεια χρειάζεται να αναζητήσουμε στο δέντρο για να βρούμε το καλύτερο δοχείο για το αντικείμενο που επεξεργαζόμαστε την δεδομένη στιγμή που πάλι παίρνει O(logn) στην χειρότερη περίπτωση. Επομένως, εφόσον η διαδικασία αυτή επαναλαμβάνεται για όλα τα n αντικείμενα έπεται και η ζητούμενη πολυπλοκότητα.

Άσκηση 5

Ο αλγόριθμος για το πρόγραμμα ανάλυσης πρόσβασης θα χρησιμοποιεί ΑΤΔ διατεταγμένο χάρτη και μια μεταβλητή count που θα μετρά το πλήθος των επισκέψεων. Η υλοποίηση του χάρτη θα γίνει με χρήση AVL δέντρου. Ο αλγόριθμος θα λειτουργεί ως εξής: Στην αρχή γίνεται αρχικοποίηση με το χάρτη να είναι κενός και η μεταβλητή count ίση με μηδέν. Ακολούθως, θα προστεθεί στο χάρτη ο 1ος κωδικός του αρχείου, δηλαδή τον ακέραιο που αντιστοιχεί στον κωδικό του 1ου χρήστη που έκανε χρήση της υπηρεσίας. Ο κωδικός θα προστεθεί ως κλειδί στη ρίζα του AVL δέντρου. Επιπλέον, η μεταβλητή count θα αυξηθεί κατά 1. Στη συνέχεια, για όσο η μέθοδος getNextAccess() δεν επιστρέφει null, αλλά έναν ακέραιο που αντιπροσωπεύει τον κωδικό ενός χρήστη, η μεταβλητή count θα αυξάνεται κατά 1. Με χρήση δυαδικής αναζήτησης στο χάρτη θα εξετάζεται αν ο τρέχων κωδικός υπάρχει στο χάρτη (O(logk) χρόνος, όπου k ο μέχρι στιγμής αριθμός των χρηστών, δηλαδή το μέγεθος του χάρτη). Εάν ο κωδικός αυτός υπάρχει στο χάρτη, τότε θα καλείται η

μέθοδος getNextAccess() για τον επόμενο κωδικό. Εάν ο κωδικός δεν υπάρχει στο χάρτη, τότε θα γίνεται εισαγωγή ενός νέου κόμβου στο AVL δέντρο που υλοποιεί τον χάρτη. Ως κλειδί του κόμβου θα ορίζεται ο κωδικός αυτός. Η διαδικασία αυτή χρειάζεται O(logk) χρόνο στη χειρότερη περίπτωση, όπου k ο αριθμός των μέχρι στιγμής χρηστών (ισοδύναμα το μέγεθος του διατεταγμένου χάρτη). Έπειτα, θα καλείται ξανά η μέθοδος getNextAccess(). Με τον πιο πάνω αλγόριθμο, πετυχαίνουμε να υπολογίσουμε τον αριθμό των προσβάσεων προς την υπηρεσία και τον αριθμό των χρηστών που χρησιμοποίησαν την υπηρεσία σε χρόνο O(nlogm), αφού για κάθε μία από τις n ”προσβάσεις” που περιέχει το αρχείο χρειάζεται το πολύ O(logm) χρόνος επεξεργασίας, όπου m ο αριθμός των χρηστών που έκαναν χρήση της υπηρεσίας. Εάν το εύρος των κωδικών είναι μικρό, για παράδειγμα από 0 μέχρι 106 , τότε ο αλγόριθμος θα μπορούσε να λειτουργήσει σε χρόνο O(n). Σε αυτή την περίπτωση θα μπορούσαμε, με χρήση μιας δεικτοδοτούμενης λίστας τύπου boolean, να αποθηκεύουμε για κάθε κωδικό αν αυτός έχει χρησιμοποιηθεί. Η λίστα θα είχε μήκος όσο το εύρος των κωδικών και στη θέση i θα αποθήκευε true αν ο i-οστός κωδικός έχει ήδη χρησιμοποιηθεί, αλλιώς false. Στην ουσία θα έπαιζε το ρόλο ενός απλού πίνακα κατακερματισμού. Η μεταβλητή count θα μετρούσε το πλήθος των επισκέψεων. Ο έλεγχος για κάθε είσοδο θα γινόταν σε σταθερό χρόνο, οπότε ο συνολικός χρόνος θα ήταν της τάξης O(n). Η μεταβλητή count θα μετρούσε το πλήθος των επισκέψεων. Εάν όμως το εύρος των κωδικών είναι μεγάλο και απαιτούνται συγκρίσεις, ο βέλτιστος τρόπος να πετύχουμε ”γρήγορη” λύση στη χειρότερη περίπτωση είναι ο διατεταγμένος χάρτης υλοποιημένος με AVL δέντρο. Οι άλλες υλοποιήσεις χάρτη ή και ουράς προτεραιότητας είναι γρήγορες στη μέση περίπτωση, αλλά για τη χειρότερη περίπτωση δεν δίνουν τόσο γρήγορη λύση.