Η main μέθοδος υπάρχει στην κλάση MainClass που περιέχεται στο default πακέτο "project2_2020030050". Επιπρόσθετα, το παραπάνω πακέτο περιέχει και την κλάση η MultiCounter χωρίς καμία μεγάλη αλλαγή, όπως αυτή δόθηκε στο 3ο φροντιστήριο του μαθήματος με μόνη διαφορά ότι πλέον χωράει 15 counters (αντί για 10). Το πρόγραμμα μεταγλωττίζεται με επιτυχία και απαντά σε όλα τα ερωτήματα χωρίς παραγωγή λαθών ή παράλογες χρονικές καθυστερήσεις. Εξωτερικές πηγές πληροφόρησης που χρησιμοποιήθηκαν για διευκόλυνση παρατίθενται στο τέλος της παρούσας αναφοράς.

Δυαδικό Δένδρο Έρευνας

(α) Με χρήση πεδίου αριθμών (array) μεγέθους N: Η ζητούμενη δομή υλοποιείται στην κλάση BSTarray η οποία περιέχει τις μεθόδους: εισαγωγής, διαγραφής και αναζήτησης τυχαίου στοιχείου οι οποίες υλοποιούνται αναδρομικά, αλλά και τις μεθόδους get_node(int pos) , free_node(int pos) όπως αυτές περιγράφονται στην εκφώνηση. Μια ακόμη σημαντική μέθοδος είναι η init_array() η οποία αρχικοποιεί το 3xN δισδιάστατο array ως εξής: η πρώτη και η δεύτερη σειρά του γεμίζουν με την τιμή final int **null_value** = Integer.MIN_VALUE, ενώ η τρίτη σειρά περιέχει τη στοίβα των διαθέσιμων θέσεων (1,2,3….N-1). Η τιμή avail αρχικά είναι ίση με μηδέν και στη συνέχεια ενημερώνεται κατάλληλα μέσω των μεθόδων get_node() , free_node() ανάλογα με το εάν εκτελείται προσθήκη ή διαγραφή. Όλες οι συναρτήσεις που προαναφέρθηκαν αυξάνουν τους αντίστοιχους counters για την μέτρηση των αριθμών συγκρίσεων. Επιπλέον μέθοδοι είναι οι : preorder(), postorder(), inorder() οι οποίες χρησιμοποιούνται για τη διάσχιση του δέντρου.

(β) Με χρήση δυναμικής παραχώρησης μνήμης: : Η ζητούμενη δομή υλοποιείται στην κλάση BST η οποία περιέχει τις μεθόδους: εισαγωγής, διαγραφής και αναζήτησης τυχαίου στοιχείου οι οποίες υλοποιούνται αναδρομικά. Η υλοποίηση της συγκεκριμένης κλάσης βρέθηκε από τη σελίδα: https://www.softwaretestinghelp.com/binary-search-tree-in-java/ , και υπέστη μικρές αλλαγές όπως: η αύξηση των αντίστοιχων counter για την μέτρηση των συγκρίσεων ,και η επιπλέον υλοποίηση των μεθόδων preorder() και postorder() μιας και η προαναφερθείσα πηγή περιείχε μόνο την υλοποίηση της inorder() διάσχισης του δέντρου. Ακόμη προστέθηκε η μέθοδος public void print2D() που δινόταν στην εκφώνηση στο link : https://www.geeksforgeeks.org/print-binary-tree-2-dimensions/.Ας μην παραληφθεί ότι οι κόμβοι που χρησιμοποιούνται σε αυτή την υλοποίηση είναι αντικείμενα της κλάσης Node και έτσι ο κάθε κόμβος φέρει : αναφορές προς το αριστερό παιδί και το δεξί του παιδί αντίστοιχα (Node left, right) ,καθώς και την τιμή του κλειδιού int key.

Στον επόμενο πίνακα ακολουθούν τα αριθμητικά αποτελέσματα που προέκυψαν από την εκτέλεση του κώδικα που έχει παρουσιαστεί μέχρι στιγμής.

Μέθοδος	**Μέσος αριθμός συγκρίσεων / εισαγωγή (1.000.000 εισαγωγές)**	**Συνολικός χρόνος για 1.000.000 εισαγωγές**	**Μέσος αριθμός συγκρίσεων / ανά διαγραφή (100 διαγραφές)**	**Συνολικός χρόνος για 100 διαγραφές**
ΔΔΕ με δυναμική παραχώρηση μνήμης	96.154508	841.547 milli-sec	96.18	0.5618 milli-sec
ΔΔΕ με array	98.15451	717.5725 milli-sec	96.02	0.7614 milli-sec

Σχολιασμός του παραπάνω πίνακα: Με μια ματιά στον παραπάνω πίνακα παρατηρείται ότι η εισαγωγή στοιχείου της υλοποίησης με array είναι πιο γρήγορη απ' ότι αυτή της δυναμικής υλοποίησης, αυτό συμβαίνει επειδή στη δεύτερη υλοποίηση η γλώσσα Java θα πρέπει να καλέσει αρκετές φορές τον constructor : new Node(…) για τη δημιουργία αντικειμένου , ενώ στην άλλη, γίνονται απλές αναθέσεις ακεραίων. Η δημιουργία ενός νέου αντικειμένου είναι πιο χρονοβόρα διαδικασία από μια ανάθεση ενός ακεραίου σε μια συγκεκριμένη θέση μνήμης μιας και ένα αντικείμενο χρειάζεται να περιέχει περισσότερες πληροφορίες πέραν της τιμής int key, επίσης στο

array τα στοιχεία βρίσκονται το ένα μετά το άλλο ενώ στην δυναμική υλοποίηση όχι και λ.χ. ο εντοπισμός των παιδιών ενός κόμβου είναι πιο αργός γιατί χρησιμοποιούμε αναφορές αντί για array indexes. Ο αριθμός των συγκρίσεων για την εισαγωγή είναι μεγαλύτερος στην υλοποίηση με array διότι το πρόγραμμα πρέπει ,επιπλέον, α) να διαχειριστεί το AVAIL το οποίο δίνει τις κενές θέσεις του πίνακα και β) να ελέγχει εάν η get_node(int) επιστρέφει έγκυρες θέσεις. Για τις συγκρίσεις διαγραφής μεταξύ των δύο υλοποιήσεων δεν παρατηρείται και πολύ μεγάλη διαφορά, αυτό συμβαίνει διότι εκτελούνται οι ίδιες ακριβώς πράξεις μόνο που στη μία περίπτωση χρησιμοποιούνται αναφορές ενώ στην άλλη δείκτες του πίνακα.

Ουρά προτεραιότητας

(α) Με χρήση πεδίου αριθμών: Η ζητούμενη δομή αναπαρίσταται από την κλάση ArrayHeap και υλοποιεί τις ακόλουθες πράξεις: εισαγωγή τυχαίου κλειδιού, διαγραφή (μέγιστου) κλειδιού , κατασκευή ουράς προτεραιότητας όταν τα κλειδιά δίνονται το ένα μετά το άλλο [int insert(int)] και κατασκευή όταν τα κλειδιά δίνονται όλα μαζί (γίνεται μέσω του constructor). Ο κώδικας είναι σχεδόν ο ίδιος με αυτόν που παρουσιάστηκε στο 6ο φροντιστήριο. Έχουν απλά προστεθεί εντολές αύξησης των counter που μετρούν τις συγκρίσεις.

(β) Με δυναμική παραχώρηση μνήμης: Η δομή αυτή, υλοποιείται στην κλάση DynamicHeap και περιέχει μεθόδους που πραγματοποιούν τις ίδιες πράξεις με την παραπάνω κλάση (ArrayHeap) . Οι κόμβοι της είναι αντικείμενα της κλάσης HeapNode και περιέχουν αναφορές προς: το προηγούμενο «χρονικά» κόμβο, στον επόμενο , στα δύο παιδιά και στον γονέα του εκάστοτε κόμβου. (*HeapNode prev, next, left, right, parent*). Οι παραπάνω αναφορές είναι σημαντικές για την εισαγωγή και τη διαγραφή στοιχείου. Η μέθοδος help_insert(HeapNode, int) [καλείται από την insert(int)], προσθέτει στοιχεία στο δέντρο από πάνω προς τα κάτω και από δεξιά προς τα αριστερά, δημιουργώντας ένα complete binary tree. Η παραπάνω μέθοδος αξιοποιεί/ενημερώνει κατάλληλα τις προαναφερθείσες αναφορές και την HeapNode currrent ,που είναι μεταβλητή της παρούσας κλάσης, ώστε να δημιουργήσει με επιτυχία το complete δυαδικό δέντρο. Στη συνέχεια η bubble_up(HeapNode node) [καλείται στο σώμα της insert αμέσως μετά την help_insert] ,τοποθετεί τον κόμβο που μόλις εισήχθη στο κατάλληλο σημείο του δέντρου ούτως ώστε να ισχύει η συνθήκη της ουράς προτεραιότητας (κλειδί γονέα > κλειδιά παιδιών). Όταν τα στοιχεία εισάγονται όλα μαζί σημαίνει πως έχει κληθεί επανειλημμένα η help_insert για τη δημιουργία ενός πλήρους δέντρου με τα στοιχεία να είναι τοποθετημένα με τη σειρά που δόθηκαν, στη συνέχεια γίνεται (στον constructor) κλήση της heapify(Node current) ούτως ώστε το πλήρες δέντρο να μετατραπεί σε ουρά προτεραιότητας.

Μέθοδος	**Συνολικός χρόνος κατασκευής όταν τα κλειδιά δίνονται όλα μαζί**	**Συνολικός χρόνος κατασκευής όταν τα κλειδιά δίνονται το ένα μετά το άλλο**	**Μέσος αριθμός συγκρίσεων / εισαγωγή (1.000.000 εισαγωγές)**	**Μέσος αριθμός συγκρίσεων / διαγραφή (100 διαγραφές)**	**Συνολικός χρόνος για 100 διαγραφές**
Ουρά προτεραιότητας με Array	33.0807 milli-sec	41.6901 milli-sec	2.563234	403.1	0.2313 milli-sec
Ουρά προτεραιότητας με δυναμική παραχώρηση μνήμης	409.106 milli-sec	420.1059 milli-sec	18.563231	347.28	0.7851 milli-sec

Σχολιασμός του παραπάνω πίνακα: Κοιτώντας τον παραπάνω πίνακα παρατηρείται ότι η εισαγωγή στην ουρά προτεραιότητας που έχει υλοποιηθεί με array είναι σημαντικά πιο γρήγορη από την εισαγωγή στην ουρά προτεραιότητας η οποία έχει υλοποιηθεί με δυναμική παραχώρηση μνήμης και στην περίπτωση που τα στοιχεία δίνονται το ένα μετά το άλλο αλλά και στην περίπτωση που τα στοιχεία δίνονται όλα μαζί. Αυτό συμβαίνει επειδή στην στατική υλοποίηση η συνάρτηση προσθήκης στοιχείου (και δημιουργίας της ουράς) για να προσθέσει ένα στοιχείο στη δομή χρησιμοποιεί συγκεκριμένες θέσεις μνήμης τις οποίες αντλεί μέσα από τους δείκτες (indexes) του πίνακα int Heap[] με χρήση μαθηματικών πράξεων και άμεση πρόσβαση στην μνήμη. Ένα απλό παράδειγμα είναι αυτό της εύρεσης του πατέρα ενός στοιχείου το οποίο βρίσκεται στη θέση pos : int parent=(pos-1)/2 , ο

αριθμός parent είναι η θέση στον πίνακα όπου βρίσκεται ο γονέας , για να αποκτήσει κανείς πρόσβαση στον γονέα αρκεί να γράψει : Heap[parent] . Ο χρόνος για την προσθήκη ενός στοιχείου στη δυναμική ουρά προτεραιότητας είναι μεγαλύτερος εφόσον εκεί, πρέπει να γίνεται εντοπισμός του γονέα ή του επόμενου στοιχείου κλπ. μέσω αναφορών, επίσης απαιτείται συχνή ενημέρωση των αναφορών αυτών. Ακόμη κατά την προσθήκη απαιτείται η δημιουργία νέου αντικειμένου (κλήση constructor new HeapNode(...)). Ο χρόνος διαγραφής είναι πιο μικρός για τη στατική υλοποίηση και η εξήγηση δε διαφέρει και πολύ από την παραπάνω. Σχετικά με τις συγκρίσεις για τις εισαγωγές και τις διαγραφές, τα αποτελέσματα είναι αναμενόμενα.

Εξωτερικές πηγές πληροφόρησης που με βοήθησαν στην άσκηση:

- https://www.softwaretestinghelp.com/binary-search-tree-in-java/
- https://www.geeksforgeeks.org/print-binary-tree-2-dimensions/