ΣΥΡΟΥ ΧΡΙΣΤΙΝΑ – ΟΛΓΑ p3190194

Το αρχείο εισόδου βρίσκεται στο μονοπάτι
"C:\\Users\\user\\Desktop\\p3190194\\covid.txt"

Όπου p3190194 είναι ο φάκελος του project.

Επιπλέον, στην εργασία χρησιμοποιούνται δύο αρχεία της πρώτης εργασίας, τα DLNode.java και DoublyLinkedList.java για την αποθήκευση των αντικειμένων τύπου City. Για τις ανάγκες της εργασίας, τα αρχεία αυτά έχουν αναβαθμιστεί με κάποιες επιπλέον μεθόδους. Συγκεκριμένα, στο DLNode.java έχει προστεθεί η setSize και στο DoublyLinkedList.java έχουν προστεθεί οι: addAtIndex, removeAtIndex και getItem. Αυτές οι μέθοδοι εξηγούνται σε σχόλια στον κώδικα των εν λόγω αρχείων.

ΜΕΡΟΣ Α

Ο αλγόριθμος ταξινόμησης που υλοποιήθηκε (με βάση τον αμ) είναι η heapsort, εξού και το όνομα του αρχείου HeapSort.java στο οποίο έχουν δημιουργηθεί 3 συναρτήσεις (swap, heapify, sort). Από αυτές, μας ενδιαφέρει η sort που κάνει την ταξινόμηση. Συγκεκριμένα, δημιουργεί ένα heap, στο μέγεθος της λίστας στην οποία έχουν εκχωρηθεί τα στοιχεία που θέλουμε να ταξινομήσουμε. Αυτό επιτυγχάνεται με την συνάρτηση heapify, η οποία συγκρίνει έναν γονέα με το αριστερό και δεξιό παιδί του αντιμεταθέτοντας τα όταν χρειάζεται, έτσι ώστε το μικρότερο στοιχείο να βρίσκεται στην θέση του γονέα.

Ξαναγυρνώντας στην sort, έχει δημιουργηθεί σωρός με το μικρότερο στοιχείο ως ρίζα. Έπειτα, αντιμεταθέτει το τελευταίο στοιχείο της λίστας με την ρίζα και μειώνει το μέγεθος του σωρού (total) κατά 1. Χρησιμοποιείται και πάλι η heapify κι έτσι η ρίζα του σωρού είναι και πάλι το μικρότερο στοιχείο από αυτά που απέμειναν. Η διαδικασία αυτή συνεχίζεται για όλο τον σωρό. Έτσι, ο σωρός είναι τώρα ταξινομημένος από το μεγαλύτερο στο μικρότερο.

ΜΕΡΟΣ Β

Η μέθοδος remove της ουράς προτεραιότητας υλοποιήθηκε σε χρόνο O(N) καθώς δυσκολεύτηκα να χρησιμοποιήσω έξτρα μνήμη και να κρατήσω τα στοιχεία σε κάποια δομή εκτός του HashMap (η σκέψη μου ήταν να κρατάω κάθε στοιχείο με την θέση του). Έτσι, η μέθοδος διασχίζει ένα, ένα τα στοιχεία της λίστας. Όταν βρίσκει το ζητούμενο στοιχείο κάνει sink και έπειτα μέσω της μεταβλητής check_sink «τσεκάρει» αν είχε αποτέλεσμα. Αν όχι, τότε η remove καλεί την swim.

Ο constructor αρχικοποιεί τις 2κ θέσεις της DoublyLinkedList με null έτσι ώστε να μπορεί να χρησιμοποιηθεί η μέθοδος addAtIndex της DoublyLinkedList.

Σχετικά με την insert, δεν έχει υλοποιηθεί η resize έτσι ώστε το πρόγραμμα να έχει καλύτερη πολυπλοκότητα. Αντί αυτού, η ουρά προτεραιότητας αρχικοποιείται με 2κ.

ΜΕΡΟΣ Γ

Το τρίτο μέρος της εργασίας δεν είναι ολοκληρωμένο. Δημιούργησα μια ουρά στην οποία προστίθενται αντικείμενα τύπου City. Όπως θα δείτε, όταν τρέχει το πρόγραμμα αυτό τερματίζει με μήνυμα λάθους NullPointerException. Για debugging έφτιαξα ένα μετρητή count_k ο οποίος όταν γίνεται 3 να εκχωρεί στην μεταβλητή

max το μέγιστο της ουράς (μέσω της getMax) και έπειτα, εκτυπώνω το όνομα του μεγίστου και το μέγεθος της ουράς. Υλοποιώ και την remove για να βεβαιωθώ ότι μειώνεται το μέγεθος της (απαραίτητο για την απαίτηση της εργασίας για κ μόνο αντικείμενα στην ουρά). Ενώ το μέγεθος μειώνεται κατά ένα με αυτόν τον τρόπο, το πρόγραμμα τερματίζει (μετά από δύο επαναλήψεις με το δικό μου αρχείο και κ=3).

Το σκεπτικό μου, ήταν να συγκρίνω κάθε επόμενο στοιχείο (από κ+1 και μετά) με τα κ «μεγαλύτερα» προηγούμενα. Δηλαδή, αν το κ+1 στοιχείο είναι μεγαλύτερο του πρώτου, του δεύτερου κ.α να μπαίνει στη σωστή θέση διαγράφοντας το τελευταίο (αν π.χ κ=3 και το στοιχείο είναι το τρίτο μεγαλύτερο) ή μετακινώντας τα μικρότερα του κατά μια θέση (αν π.χ κ=3 και το στοιχείο είναι το δεύτερο μεγαλύτερο, τότε το πρώτο μένει ως έχει, το δεύτερο γίνεται τρίτο και το νέο στοιχείο γίνεται δεύτερο.

Τέλος, οποιαδήποτε πληροφορία σχετικά με το αρχείο DynamicCovid_k_withPQ.java του project μου θα ήταν ιδιαίτερα χρήσιμο. Ευχαριστώ.