ΑΝΑΦΟΡΑ ΓΙΑ ΤΗΝ ΔΕΥΤΕΡΗ ΕΡΓΑΣΙΑ ΣΤΙΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΧΡΙΣΤΙΝΑ ΑΙΜΙΛΙΑ ΦΛΑΣΚΗ

Όλη η εργασία γράφτηκε μέσω του IDE IntelliJ!

Α' ΜΕΡΟΣ:

Χρησιμοποιήσαμε τον αλγόριθμο ταξινόμησης quicksort και τα στοιχεία αποθηκευτήκαν σε doubly linked list δηλαδή σε διπλά συνδεδεμένη λίστα(παρόμοια με αυτές που φτιάξαμε στην πρώτη εργασία). Τον αλγόριθμο quicksort τον υλοποιήσαμε με 3 μεθόδους στην κλάση **QuickSort_using_Doubly_LinkedList**:

- **partition(Node l, Node h)**: Θεωρεί το τελευταίο στοιχείο ως pivot, τοποθετεί το στοιχείο pivot στο σωστή θέση σε ταξινομημένο πίνακα και τοποθετεί όλα τα μικρότερα (μικρότερα από το αντικείμενο pivot) στα αριστερά του και όλα τα μεγαλύτερα στοιχεία στα δεξιά της μεταβλητής pivot. Μέσα στην partition χρησιμοποιείται **compareTo** (βρίσκεται μέσα στην κλάση City), η οποία παίρνει ως όρισμα ένα αντικείμενο τύπου City που το συγκρίνει με τα κρούσματα της πόλης που καταχωρήθηκε τελευταία με την βοήθεια της μεθόδου **calculateDensity**. Η **calculateDensity** κάνει την αναγωγή σε κρούσματα ανά 50,000 κατοίκους, έτσι βγαίνει ένας πραγματικός αριθμός ο οποίος στρογγυλοποιείται ώστε να έχει 2 δεκαδικά ψηφία. Μετά την σύγκριση η **compareTo** επιστρέφει έναν int αριθμό, το 1 αν τα προηγούμενα είναι περισσότερα, το -1 αν τα κρούσματα της πόλης που στείλαμε στην μέθοδο είναι περισσότερα και 0 για την περίπτωση που ο αριθμός κρουσμάτων των δύο πόλεων είναι ο ίδιος, δηλαδή είναι ίσες οι 2 πόλεις σε κρούσματα.
- **_quickSort(Node l, Node h):** Μια αναδρομική εφαρμογή της quicksort για συνδεδεμένη λίστα.
- **quickSort(Node node):** Η κύρια μέθοδος για να ταξινομηθεί μια συνδεδεμένη λίστα. Καλεί κυρίως την μέθοδο **_quickSort**.

Β'&Γ' ΜΕΡΟΣ:

- **boolean isEmpty():** Ελέγχει το μέγεθος της ουράς κι αν είναι 0 επιστρέφει true.
- **int size():** Επιστρέφει το μέγεθος της ουράς.

- **void insert(City x):** Πρώτα ελέγχει αν το μήκος της ουράς είναι μεγαλύτερο ή ίσο με το 75% του μεγέθους της ουράς που έχει οριστεί και με την βοήθεια της **resize()** (η οποία διπλασιάζει το μέγεθος της ουράς) μεγαλώνει το μέγεθος της και στην επόμενη διαθέσιμη θέση βάζει το καινούργιο αντικείμενο και με την βοήθεια της **swim*** το ανεβάζει στην σωστή θέση.
- **City max():** Επιστρέφει το στοιχείο που βρίσκεται στην θέση 1 της ουράς αφού πρώτα ελέγξει αν η ουρά είναι άδεια με την βοήθεια της **isEmpty().**
- **City getmax():** Αφού ελέγχει αν η ουρά είναι άδεια, παίρνει το στοιχείο που βρίσκεται στην θέση 1 (αφού αυτό είναι με την μεγαλύτερη προτεραιότητα) και στην θέση του βάζει το τελευταίο στοιχείο της ουράς. Έπειτα μειώνει το μέγεθος και με την **sink*** βάζει πίσω στην θέση του το τελευταίο στοιχείο που έβαλε στην θέση του πρώτου προηγουμένως. Και τέλος επιστρέφει το αντικείμενο με την μεγαλύτερη προτεραιότητα.
- **City remove(int id):** Άμα η ουρά δεν είναι άδεια ξεκινάει μια επανάληψη που τρέχει για όλα τα στοιχεία της ουράς και ελέγχει ένα ένα τα id αν είναι ίδια με αυτό που δόθηκε ως παράμετρο στην μέθοδο. Όταν το βρει το αποθηκεύει σε μια μεταβλητή τύπου City ώστε να το επιστρέψει μετά ανταλλάζει θέσεις με το αντικείμενο που βρίσκεται στην τελευταία θέση με την μέθοδο **swap**, κάνει κενό το αντικείμενο στην θέση size (που πλέον βρίσκεται το αντικείμενο που θέλουμε να σβήσουμε) και μειώνει το size και τελειώνει με την χρήση της swim για την θέση i με σκοπό την επαναφορά της ουράς στην κανονική της σειρά.

*Οι μέθοδοι **swim & sink**: Η μέθοδος swim παίρνει την θέση ενός αντικειμένου και το ανεβάζει στην σωστή θέση στην ουρά προτεραιότητας που πρέπει να είναι. Ενώ η μέθοδος sink παίρνει την θέση ενός αντικειμένου και «κατεβάζει» το αντικείμενο στην τελευταία θέση χωρίς να επηρεαστεί η υπόλοιπη ουρά προτεραιότητας.

Μέσα στην main στην **DynamicCovid_k_withPQ** αφού βάλουμε τα στοιχεία μέσα σε μια μεταβλητή τύπου City για να μπορέσουμε να εκχωρήσουμε το αντικείμενο στην ουρά προτεραιότητας ελέγχουμε με το m σε ποια γραμμή βρισκόμαστε. Επειδή θέλουμε να χρησιμοποιούμε καθ' όλη την διάρκεια του προγράμματος k(number) αντικείμενα αν το m είναι μικρότερο από το k απλά με την **insert** προσθέτει το αντικείμενο στην ουρά. Ενώ αν είναι μεγαλύτερο με την **getMin** βρίσκει την πόλη με τα λιγότερα κρούσματα και επιστρέφει την θέση της, έπειτα συγκρίνει τα κρούσματα της πόλης που θέλουμε να προσθέσουμε με αυτά της πόλης με τα λιγότερα. Αν η πόλη που θέλουμε να προσθέσουμε έχει περισσότερα τότε παίρνουμε το id της πόλης με τα λιγότερα κρούσματα (αυτή που πήραμε από την getMin) και το στέλνουμε στην remove για να την αφαιρέσει και μετά προσθέτουμε την καινούρια πόλη. *Όλη αυτή η διαδικασία πραγματοποιείται με πολυπλοκότητα O(n).*