2η ΕΡΓΑΣΙΑ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΝΙΚΟΛΕΤΑ-ΚΛΕΙΩ ΠΑΤΑΤΣΗ 3180266

ΠΛΑΤΩΝ ΚΑΡΑΓΕΩΡΓΗΣ 3180068

Μέρος Α΄

Στο μέρος Α, όπως υπέδειξε η εκφώνηση και οι αριθμοί μητρώου μας, χρησιμοποιήσαμε τη Heapsort ως αλγόριθμο ταξινόμησης. Αφού φτιάξαμε όλες τις μεθόδους στη κλάση City, φτιάξαμε τη main για να διαβάζει το αρχείο με τέτοιο τρόπο, ώστε να γίνει πιο εύκολη η υλοποίηση. Στη συνέχεια όπως μας προτείνεται στην εκφώνηση, φτιάχνουμε μία μέθοδο calculate density, η οποία φτιάχνει ένα αντικείμενο τύπου City για κάθε κλήση της και το κάνει μετά add στον πίνακα στη θέση count που είναι ένας μετρητής. Έτσι, όταν έχει διαβαστεί όλο το αρχείο έχουμε έναν πίνακα με όλες της πόλεις και τα στοιχεία τους και το μόνο που μένει είναι να τα ταξινομήσουμε με τη Heapsort. Φτιάχνουμε λοιπόν μία τέτοια νέα κλάση η οποία καλείται μέσω αντικειμένου στη main, με ένα όρισμα τύπου πίνακα, δηλαδή τον πίνακα που πρέπει να ταξινομηθεί.

Η Heapsort λειτουργεί ως εξής: Αποτελείται από 2 μεθόδους, τη sort και την heapify. Αρχικά καλείται η sort μέσω της main, βλέπει το μέγεθος του πίνακα και φτιάχνει ένα ανάλογο for loop. Μέσα σε αυτό καλεί την heapify. Η χρησιμότητα αυτού του loop είναι ουσιαστικά να «χτιστεί» η σωρός. Η heapify έχει 3 ορίσματα, τον πίνακα, το μέγεθος του και μία μεταβλητή i που αρχικοποιείται « i =(n/2)-1» (μειώνεται κατά 1 ανά loop) και φροντίζει πόσες επαναλήψεις θα χρειαστούν εντός της heapify για να κατασκευαστεί η σωρός (όσο μεγαλύτερος ο πίνακας, τόσο περισσότερες επαναλήψεις χρειαζόμαστε, καθώς χρειάζονται περισσότερες συγκρίσεις). Μόλις βγεί από το loop, το πρόγραμμα μπαίνει σε ένα νέο loop, αλλά αυτή τη φορά έχει σκοπό να κάνει pop κάθε φορά το μεγαλύτερο στοιχείο, το στοιχείο που βρίσκεται στο root της σωρού που φτιάξαμε προηγουμένως. Αυτό γίνεται με κλήση της heapify ξανά μέσα στο loop, αλλά αυτή τη φορά στο όρισμα του μεγέθους του πίνακα δεν βάζουμε το σταθερό n, αλλά το i που μειώνεται ανά κλήση κατά 1. Επίσης, αντί γιά i που βάζαμε πρίν στο 3ο όρισμα, τώρα βάζουμε το μηδέν, γιατί η σωρός έχει ήδη κατασκευαστεί και δεν χρειαζόμαστε περισσότερες συγκρίσεις. Έτσι, ανεξαρτήτως μεγέθους του πίνακα, θα βγάζουμε το κορυφαίο στοιχείο και θα το τοποθετούμε πρώτο στον πίνακα, μέχρι να τελειώσουν τα στοιχεία και ο πίνακας να έχει γίνει sort πλήρως. Όταν τελειώσει η διαδικασία και βγεί το πρόγραμμα και από αυτό το loop, θα γίνει return ο πίνακας, αφού η sort είναι τύπου array. Το σορταρισμένο Array αποθηκεύεται σε μία μεταβλητή στη main και η main έπειτα καλεί την μέθοδο Covid που κάνει print τα k πρώτα στοιχεία του πίνακα ανάλογα με το k που έδωσε ο χρήστης.

Μέρος Β΄ και Μέρος Γ’

Το κομμάτι της main και του print παραμένουν παρόμοια με το Α'. Εδώ, ασχοληθήκαμε κυρίως με το πως να φτιάξουμε τις μεθόδους με τις ζητούμενες πολυπλοκότητες. Το να καταφέρουμε να κάνουμε την remove σε O(logn) ήταν εύκολο σαν ιδέα, αλλά δύσκολο σαν υλοποίηση. Το πρώτο πράγμα που σκεφτήκαμε, ήταν ότι αφού έχουμε σαν όρισμα το id μόνο, δεν μπορούμε να κάνουμε search τον πίνακα, γιατί έχει n στοιχεία και επομένως O(n) πολυπλοκότητα. Οπότε, κατευθείαν συμφωνήσαμε να βρούμε ένα τρόπο, να έχουμε αποθηκευμένα τα indices όλων των στοιχείων, ώστε να αποφύγουμε search και να ξέρουμε κατευθείαν τη θέση που βρίσκεται το στοιχείο που πρέπει να γίνει remove. Έχοντας αναλογιστεί τα παραπάνω, για εμάς το HashMap ήταν μονόδρομος και αυτό υλοποιήσαμε ώσπου ανανεώθηκε η εκφώνηση και απαγορεύτηκε. Παρόλα αυτά θα αφήσουμε σε έναν αστερίσκο, την εξήγηση της υλοποίησης του, που εμείς θεωρούμε ότι είναι πολύ καλή. Η λύση που ακολουθήσαμε μετά την αλλαγή της εκφώνησης ήταν η εξής. Φτιάξαμε ένα πίνακα 999 θέσεων, όσο ήταν το όριο στα ID που δεχόμαστε στην ανανεωμένη εκφώνηση (στην πρώτη ήταν 9999). Κάθε φορά που διαβάζεται μία νέα πόλη, αποθηκεύουμε στη θέση int = ID στον πίνακα, η οποία είναι μοναδική γιατί 2 πόλεις δεν μπορούν να έχουν το ίδιο ID. Έτσι, κρατάμε τα indices που βρίσκεται η κάθε πόλη, τα οποία φυσικά ανανεώνονται όταν γίνονται αλλαγές. Οπότε η αφαίρεση γίνεται σε O(1) και επειδή μετά πρέπει να κάνουμε sink η πολυπλοκότητα καταλήγει σε O(logn).

Το μέρος Γ' είναι σε πολύ μεγάλο βαθμό ίδιο με το μέρος Β΄. Κάναμε copy όλες τις κλάσεις σε ένα νέο project (χρησιμοποιούμε Apache NetBeans αλλά τα παραδοτέα αρχεία τα μετατρέπουμε ώστε να τρέχουν σε cmd για δική σας ευκολία) και έπειτα διαβάσαμε την εκφώνηση. Συνειδητοποιήσαμε ότι η υλοποίηση μας στο μέρος Β΄ τηρούσε και τις απαιτήσεις του μέρους Γ΄, με εξαίρεση την πιο σημαντική, η ουρά να έχει μέχρι k στοιχεία τα οποία να είναι μάλιστα και τα k κορυφαία μέχρι εκείνη τη στιγμή. Εδώ σκεφτήκαμε να κάνουμε τη σωρό να λειτουργεί ανάποδα και να ταξινομεί τα στοιχεία με το μικρότερο πάνω. Όμως αυτό δεν αλλάζει ότι η σωρός είναι μεγιστοστρεφής. Η σωρός παραμένει απαράλλαχτη και το μόνο που αλλάζουμε, είναι ότι αντιστρέφουμε τη λειτουργία του comparator. Η σωρός είναι μεγιστοστρεφής, γιατί τηρεί τον ορισμό της, που είναι κάθε κόμβος να είναι μεγαλύτερος ή ίσος από τα παιδιά του. Απλά εμείς θεωρούμε μεγαλύτερα τα μικρότερα στοιχεία με τον comparator. Η πολυπλοκότητα του Γ είναι ίδια με του Α (O(nlogn)), αν αφήσουμε το πρόγραμμα να τρέξει και να διαβάσει όλο το αρχείο και k=n γιατί είναι O(n) μόνο για το διάβασμα (ένα 2^{ο} for loop που χρησιμοποιούμε στη main κάνει πάντα πολύ λίγες επαναλήψεις όποτε το θεωρούμε σαν να μην υπάρχει) και O(logn) η διαδικασία ταξινόμησης. Ωστόσο αν θεωρητικά θέλουμε να σταματήσουμε σε ένα k αριθμό πόλεων το διάβασμα τότε είναι O(klogk) η πολυπλοκότητα. Σε γενικές γραμμές δεν έχουμε μεγάλο κέρδος από όλη τη διαδικασία στην πολυπλοκότητα από άποψη χρόνου, έχουμε όμως μεγάλο κέρδος σε χώρο.

*Ήταν το ιδανικό εργαλείο αποθήκευσης που χρειαζόμασταν, θα του βάζαμε σαν key το id της κάθε πόλης και σαν value το index που είναι αποθηκευμένη στον πίνακα. Έπειτα, μόλις καλούνταν η remove θα βρίσκαμε μέσω του id που παίρνει σαν όρισμα, τη θέση της πόλης στον πίνακα (O(1) πολυπλοκότητα το search στο HashMap) και θα την σβήναμε από τον πίνακα. Το δύσκολο κομμάτι όμως ήταν να ανανεώνονται τα indices. Δηλαδή, αφού μία πόλη

έμπαινε πχ στην 1η θέση και άρα γινόταν καταγραφή στο HashMap id πόλης και index 1, όταν διαβάσουμε μετά μία πόλη που είναι μεγαλύτερης προτεραιότητας και θα πρέπει εκείνη να πάει στην πρώτη θέση, τότε πρέπει το index της πόλης που βρισκόταν πριν σε εκείνη τη θέση να γίνει 2. Αυτό απαιτούσε πολλούς ελέγχους οι οποίοι όμως είναι πάντα O(1). Αφού γίνουν όλα αυτά, κάνουμε sink για να φτιαχτεί ξανά η σωρός, χωρίς το σβησμένο στοιχείο πλέον.

ΓΙΑ ΤΟ ΔΙΑΒΑΣΜΑ ΤΩΝ ΑΡΧΕΙΩΝ

Τα αρχεία τρέχουν με τον εξής τρόπο. Για να βλέπει το cmd το package πάμε στο subfolder που έχει τα .java αρχεία. Τρέχουμε πχ την εντολή javac A/Covid_k.java όπου A το folder που περιέχει τα αρχεία αλλά και το όνομα του package. Μετά τρέχουμε την εντολή java A.Covid_k και κατευθείαν το πρόγραμμα ζητάει το path του αρχείου input.txt. Του το δίνουμε και έπειτα ζητάει τον αριθμό k και ξεκινά το πρόγραμμα. Επίσης είναι πιθανό να χρειαστεί σε κάποιες κλάσεις η εντολή javac B/Covid_k_PartB.java -Xlint:unchecked καθώς αρνείται να κάνει compile λόγω warnings.