project2-report

Μέρος Α

Στο μέρος Α, υλοποιήσαμε τον αλγόριθμο Heap Sort. Η κλάση **MaxPQ** περιέχει την ουρά με προτεραίτητα του μέγιστου, και η μέθοδος **heapSort** της κλάσης ορίζει τον αλγόριθμο.

Ο αλγόριθμος heapSort έχει δύο βασικά στάδια.

1. **Μετατροπή σε σωρό:** Στο πρώτο στάδιο φτιάχνουμε τον σωρό απο τα στοιχεία του array. Η μέθοδος buildHeap εξασφαλίζει οτι κάθε γονικός κόμβος είναι μεγαλύτερος ή ίσος των παιδιών του, δηλαδή την ιδιότητα του σωρού.
2. **Εξαγωγή των στοιχείων και τοποθέτηση στο τέλος:** Στο δεύτερο στάδιο, αφαιρούμε επαναληπτικά το μέγιστο στοιχείο (που βρίσκεται στην ρίζα του σωρού) και το αντικαθιστουμε με το τελευταίο στοιχείο του array. Έπειτα απο κάθε αφαίρεση, η ιδιότητα του σωρού επαναφέρεται μέσω της μεθόδου **sink**, που μετακινεί το στοιχείο που εξετάζουμε στην σωστή θέση.

Ο αλγόριθμος που υλοποίησαμε έχει χρονική πολυπλοκότητα **O(nlogn).**

Μέρος Β

Σκοπός της μεθόδου **remove** του Μέρους Β είναι να αφαιρεί και να επιστρέφει την πόλη με το δοθεν id και ταυτόχρονα να διατηρεί την ιδιότητα του σωρού. Αναλυτικά λειτουργεί ως εξής:

1. **Εύρεση της θέσης του στοιχείου για αφαίρεση (μέθοδος indexOf)**

   Η indexOf επιστρέφει το index του στοιχείου που δέχεται ως argument χρησιμοποιώντας τη μέθοδο **hash** για να έχει πρόσβαση στον πίνακα **indexes.**

   Ο πίνακας indexes αντιστοιχεί τα στοιχεία και τους δείκτες τους στην σωρό. Αυτή η αντιστοίχιση επιτρέπει την άμεση αναζήτηση των δεικτών των στοιχείων και μας βοηθά να αποφύγουμε την πιο κοστοβόρα γραμμική αναζητηση.

2. **Έλεγχος για ύπαρξη του στοιχείου στην ουρα.**

   Αφού γνωρίζουμε το index του στοιχείου είμαστε σε θέση να ελέγξουμε αν βρίσκεται στην ουρά. Αν το indexToRemove είναι -1 τότε το στοιχείο έχει ήδη αφαιρεθεί, ενώ αν είναι μεγαλύτερο του size τότε δεν είναι δυνατό να χωράει στην ουρα. Σε οποιαδήποτε απο τις δύο περιπτώσεις χρησιμοποιούμε το **IllegalArgumentException.**

3. **Περίπτωση αφαίρεσης στο τέλος.**

   Αν το στοιχείο που θέλουμε να αφαιρέσουμε βρίσκεται στο τέλος του array, τότε αφαιρείται μειώνοντας απλώς το size. Πρέπει επίσης να αλλάξουμε το index του αφηρημένου στοιχείου στο **indexes array** σε -1, για να σηματοδοτήσουμε οτι έχει πλέον αφαιρεθεί.

4. **Αντικατάσταση με το τελευταίο στοιχείο.**

   Αν το στοιχείο που θέλουμε να αφαιρέσουμε δεν βρίσκεται στο τέλος, αντικαθιστάται με το τελευταίο στοιχείο του array. Έπειτα ενημερώνουμε το index του μετακινημένου στοιχείου στο **indexes,** και τέλος μείωνουμε το size, αποκλείωντας το στοιχείο που θέλαμε να αφαιρέσουμε.

5. **Επαναφορά της ιδιότητας σωρού.**

   Καλώντας τις μεθόδους heapifyDown και heapifyUp, επαναφέρουμε την ιδιότητα της σωρού.

Τέλος, αλλάζουμε το index του αφηρημένου στοιχείου στο indexes σε -1 ώστε να σηματοδοτήσουμε οτι έχει πλέον αφαιρεθεί (αν δεν είχε ηδη γίνει στο βήμα 3) και επιστρέφουμε το στοιχείο.

Μέρος Γ

Αρχικοποιούμε μία ουρά του μέρους Β και ελέγχουμε αν έχουν εισαχθεί σωστά τα command line arguments.

Διαβάζουμε το αρχείο γραμμη γραμμη, οπου κάθε γραμμη είναι μία πόλη. Για κάθε πόλη δημιουργούμε ένα αντικείμενο City με τα στοιχεία της.

Αν η ουρά μας δεν έχει γεμίσει ακόμα, τότε απλώς εισάγουμε την πόλη που δημιουργήσαμε. Αλλιώς αν η ουρά είναι γεμάτη, βρίσκουμε την πόλη της ουράς με τα περισσότερα κρούσματα (top k πόλεις θεωρούμε αυτές με τα λιγότερα κρούσματα) και την συγκρίνουμε με την πόλη που μόλις δημιουργήσαμε. Αν η 'MaxCity' έχει περισσότερα κρούσματα τότε την αφαιρούμε και εισάγουμε τη νέα πόλη, αλλίως αφήνουμε την ουρά ως έχει.

Άφου επαναλάβουμε αυτή τη διαδικασία για κάθε γραμμη του αρχείου, θα έχουμε μία ουρά μονο με τις πόλεις που μας ενδιαφέρουν, στην σωστή σειρά.

Η χρονική πολυπλοκότητα της υλοποίησης μας είναι **O(nlogk),** οπου n είναι οι πόλεις που διαβάζουμε (γραμμές του αρχείου) και k οι πόλεις που θέλουμε να διατηρήσουμε.

Στην περίπτωση που μας ενδιαφέρουν μόνο μερικά στοιχεία, αυτή η υλοποίηση είναι βέλτιστη αφού περιορίζει το χρονικό κόστος της εισαγωγής/αφαίρεσης στην ουρά σε O(logk). Όσο το k πλησιάζει το n, η αποτελεσματικότητα αυτής της υλοποίησης μειώνεται.

Μέρος Δ

Η πολυπλοκότητα του προγράμματος εξαρτάται κυρίως απο τις μεθόδους **insert** και **getMin (remove min)** της ουράς προτεραιότητας. Αυτές οι μέθοδοι, έχουν χρονική πολυπλοκότητα **O(logN),** όπου N το πλήθος των στοιχείων της ουράς προτεραιότητας.

Για τον υπολογισμό του median θα πρέπει να κάνουμε sort τα στοιχεία μας. Αυτό το επιτυγχάνουμε μεταφέροντας όλα τα στοιχεία απο την πρώτη ουρά στην δεύτερη, χρησιμοποιώντας τις μεθόδους insert και getMin. Αφού η μεταφορά κάθε στοιχείου έχει πολυπλοκότητα O(logN), η συνολική πολυπλοκότητα αυτής της διαδικασίας θα είναι O(NlogN).

Αφού έχουμε κάνει sort τα στοιχεία μας στην δεύτερη ουρά, ο υπολογισμός του median εξαρτάται απο το μέγεθος της. Σε κάθε iteration εκτελούμε πράξεις κόστους O(logN)

όπως παραπάνω, και τελικά θα έχουμε συνολικό κόστος O(NlogN), οπου N το μέγεθος της δεύτερης ουράς.

Συνολικά η χρονική πολυπλοκότητα θα είναι O(NlogN), όπου N το πλήθος των στοιχείων.