ΕΡΓΑΣΙΑ 2 ΕΠΕΞΗΓΗΜΑΤΙΚΟ ΚΕΙΜΕΝΟ

Ανέστης Δήμου (2170052)

ΜΕΡΟΣ Α

Εκτέλεση Προγράμματος από Γραμμή Εντολών:

→ java Covid_k Data/inputfile.txt

→ Η είσοδος του k πραγματοποιείται με την χρήση του scanner

Αλγόριθμος Ταξινόμησης Heap Sort

Ο αλγόριθμος ταξινόμησης που χρησιμοποιήθηκε για την ταξινόμηση των πόλεων είναι ο αλγόριθμος Heap Sort. Η υλοποίηση του έγινε ως εξής:

- **Κλάση HeapSort**

  Δημιουργήσαμε μια κλάση όπου προσθέσαμε όλες τις μεθόδους του αλγόριθμου Heap Sort. Αρχικά ορίσαμε δύο μεταβλητές ένας πίνακας όπου περιέχει αντικείμενα τύπου City (list) και έναν ακέραιο αριθμό (N). Η list αρχικοποιείται μέσω του κατασκευαστή όπου λαμβάνει τον πίνακα προς ταξινόμηση, υποθέτοντας ότι το πρώτο στοιχείο βρίσκεται στην θέση 1, ενώ το N αρχικοποιείται μέσω της στατικής μεθόδου City.getCount() και λαμβάνει τον αριθμό των πόλεων προς ταξινόμηση.

- **Μέθοδος Sink**

  Η μέθοδος Sink στοχεύει στην αποκατάσταση των ιδιοτήτων του σωρού κάτι το οποίο επιτυγχάνει πραγματοποιώντας διαδοχικούς ελέγχους σε κάθε υποδέντρο του σωρού. Έτσι λοιπόν αρχικά αναζητά να βρει το μεγαλύτερο παιδί με την βοήθεια της μεθόδου less(). Στην συνέχεια, συγκρίνει πάλι με την βοήθεια της μεθόδου less() το παιδί με τον πατέρα του, αν τα κριτήρια δεν ικανοποιούνται τότε διακόπτεται η μέθοδος και συνεχίζει στο ανώτερο υποδέντρο, σε αντίθετη περίπτωση γίνεται ανταλλαγή του παιδιού με τον πατέρα του με την μέθοδο swap().

- **Μέθοδος Less**

  Η μέθοδος less συγκρίνει δύο αντικείμενα τύπου City και αν ικανοποιούνται τα κριτήρια και το πρώτο είναι μικρότερο ή ίσο του δεύτερου αντικειμένου επιστρέφει 1 σε αντίθετη περίπτωση επιστρέφει -1.

- **Μέθοδος Swap**

  Η μέθοδος swap ανταλλάσσει τις θέσεις δύο αντικείμενων στον πίνακα.

- **Μέθοδος Sort**

  Η μέθοδος sort πραγματοποιεί όλη την διαδικασία της ταξινόμησης. Αρχικά αποκαθιστά τις ιδιότητες του σωρού καλώντας την μέθοδο sink σε κάθε υποδέντρο ξεκινώντας από το τελευταίο. Στην συνέχεια για να βεβαιωθούμε ότι όλα τα στοιχεία στον πίνακα βρίσκονται ταξινομημένα στην σωστή θέση, ανταλλάσσουμε με την μέθοδο swap την ρίζα με το τελευταίο στοιχείο του σωρού και καλούμε την μέθοδο sink ξεκινώντας από το πρώτο υποδέντρο και μη συμπεριλαμβάνοντας σε κάθε επανάληψη το τελευταίο στοιχείο της σωρού, καθώς γνωρίζουμε ότι βρίσκεται στο σωστό σημείο.

ΜΕΡΟΣ Β

Σε αυτό το μέρος υλοποιήθηκε η ουρά προτεραιότητας με την χρήση μεγιστοστρεφούς σωρού. Οι μέθοδοι που περιλαμβάνονται στην κλάση PQ είναι οι εξής:

- **isEmpty () :** εξετάζει αν η ουρά προτεραιότητας είναι άδεια και επιστρέφει true ή false
- **size () :** επιστέφει το μέγεθος της ουράς προτεραιότητας
- **insert () :** προσθέτει στοιχεία στην ουρά προτεραιότητας εξασφαλίζοντας την διατήρηση των ιδιοτήτων του σωρού
- **max () :** επιστρέφει το στοιχείο με την μέγιστη προτεραιότητα
- **min () :** επιστρέφει το στοιχείο με την ελάχιστη προτεραιότητα
- **getMax () :** επιστέφει και αφαιρεί το στοιχείο με την μέγιστη προτεραιότητα
- **remove () :** δέχεται ως όρισμα το id ενός στοιχείο και αφαιρεί το συγκεκριμένο στοιχείο από την ουρά προτεραιότητας διατηρώντας της ιδιότητας του σωρού
- **resize () :** σε περίπτωση που υπερβούμε το προκαθορισμένο μέγεθος του πίνακα που είναι αποθηκευμένος ο σωρός, επεκτείνει τον πίνακα με συγκεκριμένο αριθμό θέσεων
- **swim () :** πραγματοποιεί ανάδυση από συγκεκριμένη θέση όπου δίνεται σαν όρισμα για την αποκατάσταση των ιδιοτήτων του σωρού
- **sink () :** πραγματοποιεί κατάδυση από συγκεκριμένη θέση όπου δίνεται σαν όρισμα για την αποκατάσταση των ιδιοτήτων του σωρού
- **swap() :** δέχεται ως ορίσματα τις θέσεις δύο στοιχείων και τις ανταλλάσσει με σκοπό την αλλαγή θέσεων μεταξύ στοιχείων μέσα στον σωρό

❖ **Μέθοδος remove**

Αρχικά έχουμε δημιουργήσει έναν βοηθητικό πίνακα Id με 999 θέσεις, όπου το νούμερο της θέσης κάθε στοιχείου στον πίνακα αντιπροσωπεύεται από το id του και το περιεχόμενο του πίνακα είναι οι θέσεις του εκάστοτε στοιχείου στον σωρό. Μέσα στην μέθοδο σε πρώτη φάση, πραγματοποιούμε έναν έλεγχο για την περίπτωση που η ουρά προτεραιότητας είναι άδεια. Στην συνέχεια, μέσα από τον πίνακα Id βρίσκουμε με το id που μας έχει δοθεί την θέση του στοιχείου στον σωρό και το αποθηκεύουμε σε μια μεταβλητή position τύπου int. Επίσης, αποθηκεύουμε στην μεταβλητή item, όπου είναι τύπου City, το στοιχείο όπου πρόκειται να αφαιρέσουμε για να μπορέσουμε να το επιστρέψουμε στο τέλος. Το επόμενο βήμα είναι να αντιμεταθέσουμε το συγκεκριμένο στοιχείο με το τελευταίο στην ουρά προτεραιότητας με την μέθοδο swap() (η μέθοδος swap() εκτός από την αντιμετάθεση των στοιχείων, αντιμεταθέτει και τις θέσεις στον πίνακα Id). Έπειτα δεν χρειαζόμαστε άλλο το στοιχείο που πρόκειται να αφαιρέσουμε όποτε θέτουμε 0 στον πίνακα Id στην θέση όπου είναι αποθηκευμένο και μειώνουμε το μέγεθος της ουράς κατά ένα μέσω της count. Τέλος, για να αποκαταστήσουμε τις ιδιότητες της σωρού πραγματοποιούμε κατάδυση με την μέθοδο sink() και ανάδυση με την μέθοδο swim() στην θέση όπου έγινε η αντιμετάθεση και επιστρέφουμε το στοιχείο που αφαιρέσαμε μέσω της μεταβλητής item.

ΜΕΡΟΣ Γ

Εκτέλεση Προγράμματος από Γραμμή Εντολών:

→ java DynamicCovid_k_withPQ Data/inputfile.txt

→ Η είσοδος του k πραγματοποιείται με την χρήση του scanner

Πρόγραμμα DynamicCovid_k_withPQ

Σκοπός του συγκεκριμένου προγράμματος ήταν η υλοποίηση ενός αποδοτικότερου προγράμματος από το Covid_k ως προς την πολυπλοκότητα

- Αρχικά το διάβασμα και η διαχείριση του αρχείου πραγματοποιείται με τον ίδιο τρόπο που γίνεται και στο πρόγραμμα Covid_k.
- Στο συγκεκριμένο πρόγραμμα δημιουργούμε μια ουρά προτεραιότητας με την χρήση μεγιστοστρεφούς σωρού όπως υλοποιήθηκε στο Μέρος Β για την αποθήκευση των πόλεων.
- Κατά την δημιουργία της ουράς δίνουμε ως όρισμα στον κατασκευαστή το k όπου έχει εισάγει ο χρήστης για να εξασφαλίσουμε ότι δεν θα υπερβούμε το προκαθορισμένο μέγεθος της ουράς προτεραιότητας και δεν θα χρειαστεί να κληθεί η resize().
- Ο δεύτερος κατασκευαστής όπου έχουμε δημιουργήσει στην PQ λαμβάνει το συγκεκριμένο όρισμα και δημιουργεί τον πίνακα όπου αναπαριστά τον σωρό σε μέγεθος 2k.
- Στην συνέχεια διαβάζουμε μια προς μια τις σειρές του αρχείου και σε κάθε επανάληψη δημιουργούμε ένα αντικείμενο τύπου City και πραγματοποιούμε έναν έλεγχο όπου σχετίζεται με ένα το μέγεθος της ουράς προτεραιότητάς έχει φτάσει το k.

  1η Περίπτωση: Όταν το μέγεθος της ουράς είναι μικρότερο του k τότε μπορούμε να προσθέσουμε και άλλα στοιχεία στην ουρά προτεραιότητας χωρίς να αφαιρέσουμε, οπότε με την χρήση της insert() η οποία διατηρεί τις ιδιότητες της σωρού προσθέτουμε το εκάστοτε στοιχείο.

  2η Περίπτωση: Όταν το μέγεθος της ουράς έχει προσεγγίσει το k τότε σε περίπτωση προσθήκης επιπλέον στοιχείων, θα έχουμε μέσα στην ουρά προτεραιότητας περιττά στοιχεία όπου εν τέλει δεν θα μας χρησιμεύσουν. Άρα αυτό που κάνουμε είναι μια δομή επιλογής η οποία περιλαμβάνει μια συνθήκη όπου με την χρήση της μεθόδου compareTo() που έχουμε δημιουργήσει συγκρίνει το τρέχων στοιχείο που θέλουμε να προσθέσουμε με το μικρότερο στοιχείο που περιλαμβάνει μέχρι τότε η ουρά προτεραιότητας (το μικρότερο στοιχείο το βρίσκουμε μέσω της μεθόδου min() η οποία επεξηγείται παρακάτω). Σε περίπτωση όπου το τρέχων στοιχείο είναι μεγαλύτερο από το μικρότερο στοιχείο που περιέχει η ουρά τότε θα πρέπει να αφαιρεθεί το μικρότερο στοιχείο της ουράς και να προστεθεί το τρέχων. Αυτό επιτυγχάνεται χρησιμοποιώντας την μέθοδο remove() με όρισμα το id του μικρότερου στοιχείου της ουράς και την μέθοδο insert() με όρισμα το τρέχων στοιχείο.

- Μόλις ολοκληρωθεί η ανάγνωση και επεξεργασία όλων των γραμμών δημιουργούμε μια δομή επανάληψης με k επαναλήψεις όπου σε κάθε επανάληψη

τραβάμε και εμφανίζουμε με την χρήση της getMax() από την ουρά προτεραιότητας το στοιχείο με την μέγιστη προτεραιότητα. Ως αποτέλεσμα να εμφανίσουμε τις k υψηλότερες πόλεις σε πυκνότητα κρουσμάτων.

- **Μέθοδος min()**
  Η μέθοδος min() βρίσκεται στην κλάση PQ και στόχος της είναι να βρει και να εμφανίζει το στοιχείο με την ελάχιστη προτεραιότητα. Αυτό το επιτυγχάνει συγκρίνοντας τα φύλλα του σωρού μεταξύ τους με την χρήση της μεθόδου compareTo(), καθώς σε έναν σωρό το στοιχείο με την μικρότερη προτεραιότητα, αν δεν παραβιάζονται οι ιδιότητες του σωρού, θα βρίσκεται σε κάποιο από τα φύλλα του.

Ανάλυση Πολυπλοκότητας

- Covid_k
  Η πολυπλοκότητα του προγράμματος Covid_k είναι O(NlogN) καθώς χρησιμοποιούμε τον αλγόριθμο heapsort για την ταξινόμηση του πίνακα που περιέχει τις πόλεις.

  N = ο αριθμός των πόλεων που θα διαβάσουμε από το αρχείο

- DynamicCovid_k_withPQ
  Η πολυπλοκότητα του προγράμματος DynamicCovid_k_withPQ εξαρτάται από τις παρακάτω μεθόδους του σωρού που δημιουργούμε:

  insert() : Η πολυπλοκότητα της insert O(logk) καθώς διατρέχουμε μόνο ένα μονοπάτι από το τέλος ως την ρίζα.

  min() : Η πολυπλοκότητα της min είναι O(k/2) καθώς εξετάζει μόνο τα φύλλα του σωρού ξεκινώντας από το προτελευταίο, οπότε στην χειρότερη περίπτωση θα κάνει k/2 συγκρίσεις.

  remove() : Η πολυπλοκότητα της remove είναι O(logk) καθώς διατρέχουμε ένα μονοπάτι του σωρού από το σημείο που αφαιρούμε το στοιχείο μέχρι το τελευταίο φύλλο του σωρού.

  getMax() : Η πολυπλοκότητα της getMax είναι O(logk) καθώς έπειτα από την αφαίρεση του στοιχείου με την μέγιστη προτεραιότητα διατρέχουμε ένα μονοπάτι της σωρού για να αποκαταστήσουμε τις ιδιότητες του.

  Επομένως, η συνολική πολυπλοκότητα της DynamicCovid_k_withPQ θα είναι O(k/2), από το οποίο εξάγουμε ότι η πολυπλοκότητα του συγκεκριμένου προγράμματος είναι καλύτερη από το πρόγραμμα Covid_k.

  k = Ο αριθμός των πόλεων που θα αποθηκεύουμε στην ουρά προτεραιότητας και στο τέλος θα χρειαστεί να εμφανίσουμε

ΜΕΡΟΣ Δ

Εκτέλεση Προγράμματος από Γραμμή Εντολών:

→ java Dynamic_Median Data/inputfile.txt

Πρόγραμμα Dynamic_Median

Η ιδέα υλοποίησης του συγκεκριμένου προγράμματος στηρίχτηκε στην χρήση δυο ουρών προτεραιότητας της maxheap και της minheap. Η maxheap αποθηκεύει τις μισές πόλεις που έχουμε διαβάσει την εκάστοτε στιγμή με την μεγαλύτερη πυκνότητα κρουσμάτων και αντίστοιχα η minheap αποθηκεύει τις υπόλοιπες μισές πόλεις που έχουν διαβαστεί με μικρότερη πυκνότητα κρουσμάτων. Με αυτόν τον τρόπο υλοποίησης γνωρίζουμε σε κάθε στιγμή ποιος είναι ο median των γραμμών που έχουν διαβαστεί, στην δική μας περίπτωση ο median είναι το στοιχείο με την μέγιστη προτεραιότητα στην minheap. Ο διαμοιρασμός των πόλεων στις δύο ουρές προτεραιότητας έγινε ως εξής την πρώτη πόλη την τοποθετήσαμε από σύμβαση στην minheap και έπειτα ανάλογα με το αν ο αριθμός της σειρά που διαβάζουμε είναι άρτιος ή περιττός κατανείμαμε τις πόλεις στην εκάστοτε ουρά προτεραιότητας. Τέλος, η ανακατανομή των πόλεων επιτεύχθηκε με την χρήση κατάλληλων λειτουργιών της ουράς προτεραιότητας που υλοποιήσαμε στο μέρος Γ, όπως η insert(), remove(), η getMax() και η min().

Πολυπλοκότητα Εύρεσης Median

Για να τυπώσουμε τον median χρειαζόμαστε να κάνουμε έναν έλεγχο έτσι ώστε να εξασφαλίσουμε ότι τυπώνεται έπειτα από κάθε πέντε γραμμές όπου διαβάζουμε με πολυπλοκότητα O(1) . Επίσης, χρειαζόμαστε να χρησιμοποιήσουμε τις μεθόδους max() και getName() οι οποίες έχουν πολυπλοκότητα O(1) καθώς η πρώτη επιστρέφει την πόλη με την μέγιστη προτεραιότητα, ενώ η δεύτερη επιστρέφει το όνομα της πόλης. Επομένως, η πολυπλοκότητα του υπολογισμού του median είναι O(1).