2Η ΕΡΓΑΣΙΑ ΔΟΜΕΣ-ΔΕΔΟΜΕΝΩΝ

Αναστάσιος Χρήστος Χατζής, ΑΜ: 3190216

Θεόδωρος Πάνου, ΑΜ: 3190154

ΜΕΡΟΣ Α

Ο αλγόριθμος ταξινόμησης που χρησιμοποιήσαμε στο μέρος Α είναι ο **heapsort**. Για την υλοποίηση αυτού του αλγορίθμου χρησιμοποιούμε ένα δυαδικό δέντρο , που περιέχει κόμβους αντικείμενα τύπου City και το οποίο στην κορυφή του έχει το αντικείμενο City με την μεγαλύτερη προτεραιότητα. Η προτεραιότητα που υπάρχει μέσα στο δέντρο είναι ο λόγος των ημερήσιων κρουσμάτων covid δια του πληθυσμού της πόλης πολλαπλασιασμένος με το 50.000 για να γίνει αναγωγή σε πληθυσμό 50.000 κατοίκων. Σε κώδικα java αυτό μεταφράζεται σε ένα πίνακα ονόματος heap που περιέχει αντικείμενα τύπου City, ο οποίος πίνακας για λόγους ευκολίας έχει την πρώτη θέση κενή (heap[0]) και είναι κατασκευασμένος έτσι ώστε για μια τυχαία θέση i του πίνακα γνωρίζουμε ότι στην θέση i/2 υπάρχει ο “πατέρας” του κόμβου και αντίστοιχα στις θέσεις 2*i , 2*i+1 υπάρχουν τα “παιδιά” του κόμβου i .

ΜΕΡΟΣ Β

Για την remove του μέρους Β δεν καταφέραμε να την υλοποιήσουμε με πολυπλοκότητα O(logn) , αλλά με πολυπλοκότητα O(N), με τον εξής τρόπο:

Με μια for loop ψάχνουμε, από την θέση 1 μέχρι την θέση size του heap , να βρούμε ποια πόλη έχει το ζητούμενο id (με την μέθοδο getID() της κλάσης City). Μόλις βρούμε την πόλη , έστω ότι ονομάζεται mycity, την αντιγράφουμε σε ένα νέο αντικείμενο City που φτιάξαμε (για να μπορέσουμε να την επιστρέψουμε στο τέλος) και έπειτα κάνουμε swap την mycity με την πόλη που βρίσκεται στο τέλος της λίστας. Μειώνουμε το μέγεθος της λίστας κατά 1 έτσι ώστε να διαγράψουμε την mycity από την heap και τελικά κάνουμε sink την πόλη που ήταν στο τέλος της heap και με την swap βρέθηκε στην θέσης mycity έτσι ώστε να πάει η πόλη αυτή στη σωστή θέση. (σωστή θέση σε έναν μεγιστοστρεφή σωρό είναι όταν ένας κόμβος έχει μικρότερη προτεραιότητα από τον πατέρα του) .

ΜΕΡΟΣ Γ

Για το μέρος Γ τροποποιήσαμε κατάλληλα το μέρος Β με τον εξής τρόπο:

Αυτή την φορά ο πίνακας heap δεν περιέχει στην θέση 1 το αντικείμενο με την μεγαλύτερη προτεραιότητα αλλά αυτό με την μικρότερη προτεραιότητα. Έστω ότι το input του χρήστη είναι το κ=3, και έχουμε ένα αρχείο με 6 πόλεις. Κατά το διάβασμα των 3 πρώτων πόλεων σχηματίζεται ο πίνακας heap οπού στην πρώτη θέση (εννοείται heap[1]) φέρνει το αντικείμενο City με την μικρότερη προτεραιότητα. Όταν διαβάσουμε και την $4^{η}$ γραμμή παίρνουμε την πόλη που διαβάσαμε, αυξάνουμε (προσωρινά) το μέγεθος του πίνακα heap και την τοποθετούμε στην τελευταία θέση του πίνακα. Έπειτα καλούμε swim για αυτό το αντικείμενο έτσι ώστε να τοποθετηθεί στην κατάλληλη θέση. Όταν πια γίνει αυτό καλούμε την getMax() η οποία διαγράφει από τον πίνακα το αντικείμενο που ήταν στην θέση 1 και είχε την μικρότερη προτεραιότητα. Έτσι έχουμε επιστρέψει πάλι στο σωστό μέγεθος του πίνακα και είμαστε σίγουροι ότι στον πίνακα heap έχουμε τις k πόλεις με μεγαλύτερη

προτεραιότητα μέχρι εκείνη την στιγμή , σε ανάποδη σειρά όμως. Ακολουθούμε την ίδια διαδικασία μέχρι να διαβάσουμε όλες τις γραμμές του αρχείου.

Όταν πια έχουμε τελειώσει το διάβασμα του αρχείου έχουμε τον πίνακα heap k μεγέθους ο οποίος περιέχει τις k υψηλότερες πόλεις, σε αντίστροφη σειρά. Για να τις τυπώσουμε δημιουργούμε έναν πίνακα board που περιέχει αντικείμενα City και στον οποίο τοποθετούμε τα στοιχεία του heap ένα προς ένα με την μέθοδο getMax(). Τελικά τυπώνουμε από το τέλος προς την αρχή τον πίνακα board.

Η πολυπλοκότητα του μέρους Γ είναι η εξής:

-O(N) για το διάβασμα του αρχείου(κλάση Read)

-insert():O(logn) για την swim και O(logn) για την getMax()

Η διαφορά σε σχέση με το μέρος Α όσον αφορά την πολυπλοκότητα χρόνου είναι στην μέθοδο insert.Στην οποία καλείται η μέθοδος getMax() η οποία με την σειρά της καλεί την μέθοδο sink(1) η οποία έχει επιπλέον πολυπλοκότητα O(logk).Συνεπώς προβαίνουμε σε μια επιπλέον κλήση σε σχέση με αυτή του μέρους Α .Ωστόσο

οι συγκρίσεις είναι κατά N-k λιγότερες σε σχέση με αυτές του μέρους Α. Αντιλαμβανόμαστε λοιπόν ότι για μικρό k έχουμε και ανάλογα λιγότερες συγκρίσεις(τόσο λιγότερες ώστε το κόστος της sink(1) να είναι αμελητέο).Αν όμως το κ είναι μεγάλο και συγκεκριμένα τείνει προς το N τότε δεν καταφέραμε τίποτα ,αλλά κάναμε την πολυπλοκότητα χειρότερη αφού χρησιμοποιούμε προσεγγιστικά τον ίδιο αλγόριθμο με αυτόν του μέρους Α συν μια έξτρα sink(1) η οποία θα είναι πλέον O(logk) όμως το k θα τείνει στο N.

Συνοψίζοντας η πολυπλοκότητα της δομής στο Μέρος Γ είναι καλύτερη για μικρό k (ακόμα και με μια έξτρα μέθοδο sink(1)) λόγω των αναλογικά πολύ λιγότερων συγκρίσεων που θα πρέπει να προβούμε.

Διαδικαστικές λεπτομέρειες:

- Φτιάξαμε μια κλάση Read η οποία παίρνει ως όρισμα την ουρά προτεραιότητας του Α ή του Γ μέρους. Η κλάση Read περιέχει την μέθοδο read() η οποία διαβάζει το ζητούμενο αρχείο και κάνει insert στην ουρά που δώσαμε ως όρισμα

-Για το input k χρησιμοποιήθηκε η κλάση Scanner(System.in) η οποία εισήχθη μέσω της java.util.Scanner.

-Για την δημιουργία αρχείου (InfoCity.txt) χρησιμοποιήθηκε η κλάση File μέσω της java.io.File

Το path που δόθηκε για την εύρεση και ανάγνωση διαφέρει ανάλογα το windows explorer.

Χρησιμοποιήθηκε Cmd ,Eclipse.