ΝΙΚΟΣ ΜΗΤΣΑΚΗΣ - Α.Μ. : 3210122
ΠΑΝΑΓΙΩΤΗΣ ΜΟΣΧΟΣ – Α.Μ. : 3210127

2η ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ ΤΩΝ ΔΟΜΩΝ ΔΕΔΟΜΕΝΩΝ

ΜΕΡΟΣ Ε

Για το μέρος Α:

Δημιουργήσαμε μία κλάση με όνομα Disk που περιγράφει δίσκους χωρητικότητας 1TB. Η κλάση αυτή έχει έναν κατασκευαστή που αρχικοποιεί τα πεδία της κλάσης, τα οποία είναι το id του δίσκου, το μέγεθός του και μία λίστα μονής σύνδεσης (από την 1η εργασία –> ***απαιτείται μία φορά compile με utf-8 encoding στο SinglyLinkedList.java ή στο Disk.java ή στο Test.java ή στο Greedy.java πρώτα***) που δέχεται ακεραίους με όνομα folders που απεικονίζει τους φακέλους που είναι αποθηκευμένοι στον συγκεκριμένο δίσκο ως προς το μέγεθός τους. Υπάρχουν επίσης μέθοδοι που προσθέτουν έναν φάκελο στον δίσκο (σσυγκεκριμένα στην λίστα folders του δίσκου), επιστρέφουν τον ελεύθερο χώρο στον δίσκο σε MB και συγκρίνουν δύο δίσκους με βάση τον ελεύθερο χώρο που έχουν χρησιμοποιώντας την overridden μέθοδο compareTo από τον Comparator τον οποίο κάνει implement η κλάση Disk. Επίσης, υπάρχει μία μέθοδος toString που επιστρέφει τα στοιχεία του δίσκου σε ένα κατάλληλα μορφοποιημένο string.

Επιπλέον δημιουργήσαμε μία κλάση HeapPQ στην οποία έχουμε υλοποιήσει μία ουρά προτεραιότητας χρησιμοποιώντας σωρό, η οποία μας επιτρέπει να απομακρύνουμε το στοιχείο με τη μεγαλύτερη προτεραιότητα σε σταθερό χρόνο (μέθοδος getMax()), (το οποίο εδώ είναι ο δίσκος με τον περισσότερο ελεύθερο χώρο), αλλά και να εισάγουμε (μέθοδος insert) και να αφαιρέσουμε στοιχεία (μέθοδος remove) σε λογαριθμικό χρόνο. Έχουμε φτιάξει και άλλες μεθόδους, όπως η peek η οποία χρησιμοποιείται άμεσα

από εμάς, αλλά οι περισσότερες χρησιμοποιούνται έμμεσα, καθώς καλούνται μέσα στις μεθόδους που αναφέραμε παραπάνω (πχ. sink, swim, swap). Βασισμένοι στα εργαστήρια του μαθήματος, τις διαλέξεις του και τις δικές μας γνώσεις.

Στη συνέχεια για δική μας διευκόλυνση δημιουργήσαμε ουσιαστικά μία wrapper κλάση της HeapPQ με όνομα MaxPQ (αυτή που ζητούσε η εκφώνηση της εργασίας) στην οποία απλά χρησιμοποιούμε συγκεκριμένες από τις μεθόδους της HeapPQ.

Για το μέρος Β:

Στο συγκεκριμένο μέρος φτιάξαμε μία κλάση Greedy στην οποία μέσα γράψαμε 3 στατικές μεθόδους.

Αρχικά μία main στην οποία καλούνται κατάλληλα οι άλλες δύο μέθοδοι και εφαρμόζεται ο αλγόριθμος Greedy στο αρχείο του οποίου θα έχει δώσει ο χρήστης ως όρισμα, όταν θα εκτελέσει το αρχείο Greedy.java .

Αρχικά φτιάξαμε μία read μέθοδο η οποία δέχεται ένα string ως όρισμα και διατρέχει με έναν Buffered Reader ένα συγκεκριμένο αρχείο με την εξής αυστηρή δομή: ένας θετικός ακέραιος ανά γραμμή. Η μέθοδος αυτή διαβάζει τον ακέραιο στη κάθε γραμμή του αρχείου και το τοποθετεί μέσα σε έναν πίνακα από ακεραίους τον οποίο και τον επιστρέφει (Αν πληροί την προϋπόθεση να είναι μεταξύ του 0 και του 1000000).

Τέλος φτιάξαμε και μία ακόμη μέθοδο με όνομα greedy η οποία δέχεται ως όρισμα έναν πίνακα με ακεραίους (στην πράξη θα δέχεται τον πίνακα από ακεραίους που επιστρέφει η read). Η συγκεκριμένη μέθοδος αποτελεί ουσιαστικά την

εφαρμογή του αλγορίθμου Greedy (Greedy Algorithm) στον συγκεκριμένο πρόβλημα.

Εν συντομία η μέθοδος αυτή για κάθε «φάκελο» στο αρχείο που διαβάσαμε, ελέγχει αν ο δίσκος με τον περισσότερο ελεύθερο χώρο στην ουρά προτεραιότητας χωράει το μέγεθος του εκάστοτε φακέλου, δηλαδή αν ο δίσκος με την υψηλότερη προτεραιότητα χωράει να αποθηκεύσει τον εκάστοτε φάκελο. Αν δεν τον χωράει ή αν αρχικά η ουρά προτεραιότητας είναι άδεια δημιουργούμε ένα νέο δίσκο, στον οποίο και αποθηκεύουμε τον συγκεκριμένο φάκελο και έπειτα τον προσθέτουμε κατάλληλα (δηλ. ανάλογα τον ελεύθερο χώρο του) στην ουρά προτεραιότητας. Αλλιώς, προφανώς αφαιρούμε από την ουρά τον δίσκο , αποθηκεύουμε τον φάκελο και τον ξανατοποθετούμε (κατάλληλα φυσικά) στην ουρά.

Τέλος εκτυπώνουμε τις λεπτομέρειες των πρώτων 100 δίσκων με τον περισσότερο ελεύθερο χώρο σε φθίνουσα σειρά μετά την εκτέλεση της διαδικασίας (αν χρειάστηκαν λιγότεροι, τους εκτυπώνουμε όλους, πάλι όμως σε φθίνουσα σειρά).

Για το μέρος Γ:

Για να εφαρμόσουμε τον αλγόριθμο Greedy-Decreasing αποφασίσαμε να χρησιμοποιήσουμε τον γνωστό αλγόριθμο ταξινόμησης Quicksort, αλλά κάνοντας φθίνουσα ταξινόμηση.

Στο αρχείο Sort.java έχουμε γράψει μία στατική μέθοδο με όνομα quicksortDescending στην οποία ουσιαστικά θα εκτελούμε την ταξινόμηση του πίνακα από ακεραίους που θα επιστρέφει η μέθοδος read.

Η μέθοδος αυτή δέχεται ως ορίσματα, έναν πίνακα από ακεραίους και 2 ακέραιες μεταβλητές, μία low και μία high (όταν καλείται η μέθοδος το low θα είναι ίσο με 0, ενώ το high

θα είναι ίσο με το μέγεθος του πίνακα μείον 1), με βάσει των οποίων θα γίνει η επιλογή του pivot στοιχείου για να ξεκινήσει να δουλεύει ο αλγόριθμος ταξινόμησης. Στη συνέχεια ο αλγόριθμος διαιρεί των δοσμένο πίνακα στα δύο με βάσει το pivot που έχει αποφασιστεί και ψάχνει στον πίνακα στα αριστερά του pivot το πρώτο μικρότερο στοιχείο από αυτό, ενώ στον δεξιό ψάχνει για το πρώτο μεγαλύτερο για να ανταλλάξουν θέσεις. Τέλος, ταξινομεί τον κάθε υποπίνακα καλώντας αναδρομικά τον εαυτό της. (βλ. σχόλια μέσα στον κώδικα)

Για το μέρος Δ:

Στο συγκεκριμένο μέρος για να συγκρίνουμε τους δύο αλγορίθμους ως προς την αποδοτικότητά τους σε τρεις διαφορετικά πλήθη και παράλληλα τυχαία μεγέθη των φακέλων που είναι προς αποθήκευση, δημιουργήσαμε ένα αρχείο Test.java που περιέχει μόνο μία main μέθοδο.

Πρώτιστα χρησιμοποιήσαμε την κλάση Random και την ArrayList για να παράγουμε σε ένα subdirectory του directory της εργασίας μας σε ένα φάκελο data 10 .txt αρχεία για κάθε διαφορετικό πλήθος φακέλων προς αποθήκευση. Οι τρεις διαφορετικές τιμές ήταν 100, 500 ή 1000 φάκελοι προς αποθήκευση. Οπότε για κάθε ένα πλήθος δημιουργήσαμε 10 διαφορετικά αρχεία, τα οποία είχαν σε κάθε γραμμή έναν και μόνο έναν «τυχαίο» ακέραιο από 0 έως 1000000 (το μέγεθος κάθε φακέλου σε MB). Επίσης για να συγκεντρώσουμε τα αρχεία μαζί, ώστε να τα επεξεργαστούμε με ευκολία, φτιάξαμε μία ArrayList<String> και τα προσθέσαμε όλα εκεί.

Στην συνέχεια εκτελούμε τους δύο αλγορίθμους Greedy και Greedy-Decreasing και τους συγκρίνουμε.

ΝΙΚΟΣ ΜΗΤΣΑΚΗΣ - Α.Μ. : 3210122
ΠΑΝΑΓΙΩΤΗΣ ΜΟΣΧΟΣ – Α.Μ. : 3210127

<<Επειδή η διαδικασία εφαρμογής και των δύο αλγορίθμων είναι σχεδόν ίδια, θα παρατηρήσετε ότι παραλείπουμε κάποια πράγματα για να αποφύγουμε την άσκοπη επανάληψη>>

- Αρχίζουμε με τον αλγόριθμο Greedy, διαβάζοντας κάθε αρχείο με την συνάρτηση read (βλ. μέρος Β) και μετά καλώντας την συνάρτηση greedy περνώντας της αυτόν τον πίνακα, δίχως να ταξινομήσουμε τον πίνακα με τα μεγέθη των φακέλων που περιέχονται στο εκάστοτε αρχείο.

Πριν εφαρμόσουμε στον πίνακα με τους φακέλους την συνάρτηση greedy (βλ. μέρος Β) πρώτα κάνουμε έναν γρήγορο υπολογισμό του συνολικού μεγέθους όλων των φακέλων, εκφράζοντας το αποτέλεσμά μας σε TB (το μέγεθος των φακέλων είναι σε MB), το οποίο και εκτυπώνουμε στην κονσόλα.

Έπειτα καλούμε την μέθοδο greedy, η οποία θα εκτυπώσει το αποτέλεσμα της διαδικασίας (βλ. Μέρος Β και σχόλια στον κώδικα).

Για να σημειώσουμε σε πόσα αρχεία με το ίδιο (και συγκεκριμένο) πλήθος φακέλων εφάρμοσε την συνάρτηση greedy και πόσοι δίσκοι απαιτήθηκαν συνολικά ορίζουμε κάποιες βοηθητικές μεταβλητές. Τις GreedyTotal και GreedyCounter, δύο ακέραιες μεταβλητές.

Με την πρώτη μετράμε πόσοι δίσκοι έχουν χρησιμοποιηθεί συνολικά για αρχεία με ίδιο αριθμό φακέλων και όταν επεξεργαστούμε και τα 10 αρχεία σε κάθε περίπτωση το διαιρούμε το νούμερο αυτό με το 10 για να υπολογίσουμε (και μετά να εκτυπώσουμε) τον μέσο όρο δίσκων που απαιτήθηκε για κάθε 10δα αρχείων. Για αυτό το λόγο κάθε 10 επαναλήψεις μηδενίζουμε την συγκεκριμένη μεταβλητή.

Με την δεύτερη μετράμε πόσα αρχεία έχουμε επεξεργαστεί και τα χωρίζουμε σε 10δες για να εκτυπώσουμε και να υπολογίσουμε τον μέσο όρο δίσκων που απαιτήθηκαν για συγκεκριμένο πλήθος φακέλων. Για αυτό το λόγο έχουμε γράψει 3 if μπλοκς για να χωρίσουμε κάθε 10δα αρχείων.

- Συνεχίζουμε με τον αλγόριθμο Greedy-Decreasing, διαβάζοντας κάθε αρχείο με την συνάρτηση read και πάλι (βλ. μέρος Β) και μετά καλώντας την συνάρτηση greedy περνώντας της αυτόν τον πίνακα, αφού όμως τον έχουμε ταξινομήσει πρώτα με την μέθοδο quicksortDescending (βλ. μέρος Γ) τον πίνακα με τα μεγέθη των φακέλων που περιέχονται στο εκάστοτε αρχείο.

Και πάλι γίνεται χρήση βοηθητικών μεταβλητών και υπολογισμός του συνολικού μεγέθους των φακέλων ακριβώς με τον ίδιο τρόπο όπως περιγράφηκε προηγουμένως για τον αλγόριθμο Greedy.

Από την παραπάνω σύγκριση πηγάζουν τα παρακάτω ενδεικτικά αποτελέσματα:

<u>Αλγόριθμος Greedy</u>

- Μ.Ο. δίσκων 1TB για 100 φακέλους: 59,6 = 60
- Μ.Ο. δίσκων 1TB για 500 φακέλους: 294,7 = 295
- Μ.Ο. δίσκων 1TB για 1000 φακέλους: 582,9 = 583

<u>Αλγόριθμος Greedy-Decreasing</u>

- Μ.Ο. δίσκων 1TB για 100 φακέλους: 53,4 = 54
- Μ.Ο. δίσκων 1TB για 500 φακέλους: 257,4 = 258
- Μ.Ο. δίσκων 1TB για 1000 φακέλους: 504,5 = 505

Μετά από πολλές δοκιμές καταλήγουμε στο εξής συμπέρασμα:

- *<u>Ο αλγόριθμος Greedy-Decreasing είναι πιο αποδοτικός, ειδικά για μεγάλα πλήθη φακέλων.</u>*