

ΑΣΚΗΣΗ 2

ΝΙΚΟΛΑΣ ΚΟΥΝΤΟΥΡΙΩΤΗΣ 3170195

ΜΕΡΟΣ Α :

Για την υλοποίηση της κλάσης Disk με το Comparable απλά θέσαμε ανάλογες μεταβλητές που θα είναι τα properties του δίσκου , όπως ID , default disk space = 1TB , free disk space , allocated disk space , ένα στοιχείο που θα αποθηκεύει τους φακέλους , και ένα που θα προσδιορίζει το πόσους φακέλους επεξεργάστηκε ο δίσκος . Έχουμε τους ανάλογους getters & setters και μία μέθοδο την addData , που είναι η κύρια μέθοδος σε αυτή τη κλάση . Είναι public αυτή η μέθοδος και θα χρησιμοποιείται στις επόμενες main για να γίνονται allocate οι φακελοι στους δίσκους και να ενημερώνονται αυτόματα οι μεταβλητές του συγκεκριμένου δίσκου . Για την max Priority Queue χρησιμοποίησα τον κώδικα του εργαστηρίου . Η μέθοδος compareTo με την οποία δουλεύει το maxPQ , συγκρίνει τον διαθέσιμο χώρο μέσα σε δίσκους , και ο δίσκος με τον μεγαλύτερο διαθέσιμο χώρο είναι το μεγαλύτερο στοιχείο , δηλαδή η ρίζα στο maxPQ .

ΜΕΡΟΣ Β :

Για την υλοποίηση του Μερους Β έκανα την main στο Greedy.java, χρησιμοποίησα την κλάση Scanner & File να κάνω parse το αρχικό argument (το txt file) , και μετέπειτα το μόνο που έπρεπε να γίνει ήταν έλεγχος στα δεδομένα αν υπερβαίνει το 1TB ή η τιμή του είναι μικρότερη από το 0 , και μετά μπορούμε να τα προσθέτουμε μέσα σε δίσκους . Χρησιμοποιώντας την peek , για να δουλέψει όπως θέλουμε ο αλγόριθμος , όταν διαβάζουμε το data προσπαθούμε να το αναθέτουμε στο δίσκο με τον περισσότερο χώρο , άρα τη ρίζα της maxPQ . Αν το data που διαβάστηκε είναι μεγαλύτερο από το free disk space της ρίζας της maxPQ , τότε δημιουργήσε ένα καινούριο δίσκο , ανέθεσε του το data και πρόσθεσε τον στην maxPQ .

ΜΕΡΟΣ Γ :

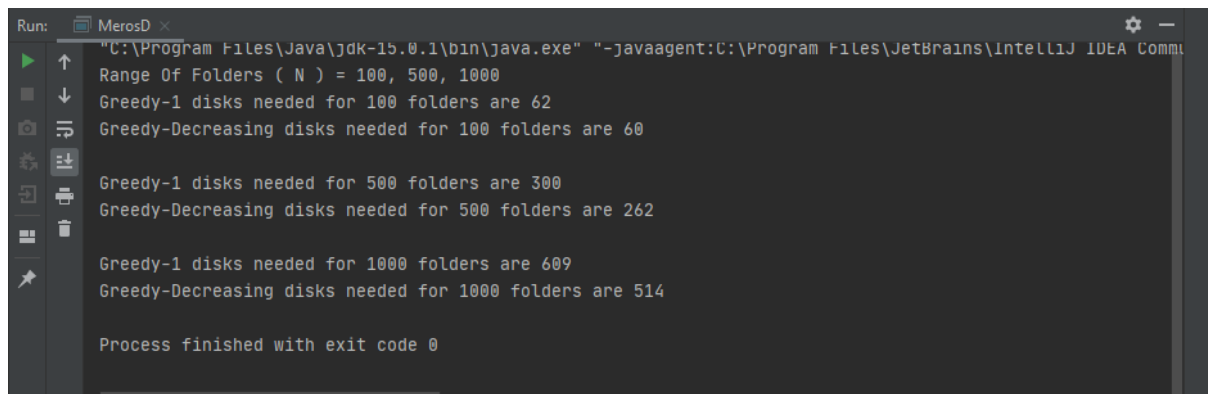
Χρησιμοποιώντας την quicksort και την main της κλάσης Sort, κάναμε sort από μικρότερο σε μεγαλύτερο τα στοιχεία data στον int array και μετέπειτα με την μέθοδο reverseArray , αλλάξαμε τις θέσεις τους έτσι ώστε το μεγαλύτερο στοιχείο να είναι πρώτο . Έτσι για κάθε γραμμή μέσα στο txt file ανατίθεται στον ανάλογο δίσκο σύμφωνα με τον αλγόριθμο .

ΜΕΡΟΣ Δ :

Για να τρέξει το πρόγραμμα πρέπει να αλλαχτεί το string path και να δείχνει εκεί που παραχθηκαν τα αρχεία , το project directory δηλαδή .

Για το μέρος Δ έχουμε την main κλάση MerosD.java όπου πρώτιστα , εισάγουμε την κλάση Random για την παραγωγή τυχαίων αριθμών . Χρησιμοποιώντας την μέθοδο της κλάσης Random.nextInt(int bound) , το bound που δέχεται ως παράμετρος η μέθοδος αυτή δεν συμπεριλαμβάνει το άνω όριο , άρα με τη τιμή bound = 1000001 μπορούμε να έχουμε τις τιμές 0-1000000 όπως μας ζητάει η άσκηση . Δημιουργούμε ένα for loop το οποίο θα εκτελεστεί 30 φορές , για την παραγωγή 30 txt files . Για κάθε εύρος τιμών 1-10 , 11-20,21-30

δημιουργούμε αρχεία με διαφορετικές τιμές φακέλων . Για τα πρώτα 10 αρχεία έχουμε 100 φακέλους , για τα επόμενα 10 έχουμε 500 , και για τα επόμενα 10 έχουμε 1000 φακέλους . Μετά υλοποιήσαμε σε κάθε κλάση Sort & Greedy ανάλογα , τις μεθόδους sort(File file) , greedy(File file) . Κατασκευάζουμε ένα αντικείμενο για κάθε εύρος τιμών για κάθε αλγόριθμο , έτσι θα μπορούμε να έχουμε private στοιχεία κλάσης για να συγκρίνουμε αποτελέσματα αλγορίθμων μετά . Για τα 30 txt αρχεία τρέχουμε τον αλγόριθμο και για κάθε ένα από αυτά καλούμε τις μεθόδους που προανέφερα . Μέσω των μεθόδων sort και greedy γίνονται οι αναλογες εκχωρήσεις φακέλων σε δίσκους για κάθε αλγόριθμο ξεχωριστά . Μετά παίρνουμε τα αντικείμενα και τα συγκρίνουμε μεταξύ τους αν είναι στο ίδιο εύρος τιμών . Χρησιμοποιώντας μια μέθοδο getNumOfdisks() και getDisksUsed() παίρνουμε ακριβώς τον αριθμό δίσκων που χρειάστηκαν για την εκχώρηση φακέλων . Ένα στιγμιότυπο αποτελεσμάτων είναι το εξής :



```
Run: MerosD x
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Commu
Range Of Folders ( N ) = 100, 500, 1000
Greedy-1 disks needed for 100 folders are 62
Greedy-Decreasing disks needed for 100 folders are 60

Greedy-1 disks needed for 500 folders are 300
Greedy-Decreasing disks needed for 500 folders are 262

Greedy-1 disks needed for 1000 folders are 609
Greedy-Decreasing disks needed for 1000 folders are 514

Process finished with exit code 0
```

Όπου βλέπουμε πως σε όλες τις περιπτώσεις ο greedy-Decreasing χωρίζει τους ίδιους φακέλους σε λιγότερους δίσκους . Είναι πιο οικονομικός σε χώρο παρά τον πρώτο αλγόριθμο. Το μέρος Δ δεν δέχεται παραμέτρους , απλά παράγει τα txt files και συγκρίνει δίσκους που χρησιμοποιήθηκαν από κάθε αλγόριθμο .

ΤΕΛΟΣ